NATIONAL TECHNICAL UNIVERSITY OF ATHENS
SCHOOL OF RURAL AND SURVEYING ENGINEERING
POST-GRADUATE PROGRAM IN GEOINFORMATICS
CARTOGRAPHY LABORATORY

Comparative Evaluation of Servers and Libraries for Web Mapping Applications

Master Thesis

Lydia D. Gouta
Rural and Surveying Engineer NTUA

Supervisor:
Lysandros Tsoulos
Professor Emeritus NTUA

Athens, October 2018

NATIONAL TECHNICAL UNIVERSITY OF ATHENS
SCHOOL OF RURAL AND SURVEYING ENGINEERING
POST-GRADUATE PROGRAM IN GEOINFORMATICS
CARTOGRAPHY LABORATORY

Comparative Evaluation of Servers and Libraries for Web Mapping Applications

Master Thesis

Lydia D. Gouta
Rural and Surveying Engineer NTUA

Supervisor:
Lysandros Tsoulos
Prof. Emeritus NTUA

Certified and approved by the three-member committee:

(Signature)	*(Signature)*	*(Signature)*
…….	…….	…….
Lysandros Tsoulos	Vasileios Veskoukis	Nikolaos Doulamis
Professor Emeritus	Associate Professor	Associate Professor
NTUA	NTUA	NTUA

Athens, October 2018

(Signature)

……..

LYDIA D. GOUTA
Rural and Surveying Engineer NTUA

© 2018 - All rights reserved

Acknowledgements

First of all, I would like to truly thank my supervisor, *Lysandros Tsoulos*, Professor Emeritus of NTUA for his guidance, instructions and understanding through all the time of this effort. His help and advices were very valuable for me and our cooperation was excellent.

Furthermore, I would like to thank *Andelina Skopeliti*, Dr. Engineer of NTUA, for her help and support, as she was always there when I needed her help. Our cooperation was also excellent.

I also would like to thank my family and all the people that supported and helped me to complete this work, each one with its own way.

Finally, I dedicate this thesis to my father, Dimitrios T. Goutas, who would be really proud of me. I thank him for everything.

Abstract

With the advanced development of technology and the emergence of World Wide Web (WWW), the situation in Cartography changed. This was achieved with the development of Cartography in Internet interface (Web Mapping).

In order to produce maps in such interfaces, and more specifically *interactive maps*, certain components are necessary, which consist the architecture of this system. On its simplest version, a database in which the data is stored as well as a GIS Server which will serve the data to the Internet is required. Through various web services the map can be published to internet provided that the symbolism has been applied properly and subsequently and on the client side now, the cartographic web application is taking place. This application is enriched with further operations such as the configuration of a friendly intercommunication environment, the interactive involvement of the user and the interactive attribution tools, like the zooming in/out, the panning etc. These last operations are accomplished by using web mapping library software such as the JavaScript libraries OpenLayers, Leaflet etc. Finally, for the structure of the presentation site the programming languages HTML, CSS and JavaScript are exploited. These tools combined are giving to the application its final form at a browser environment.

In the present thesis the analysis and comparison of two of the most basic parts of this system is attempted. In chapter 2, the term *web mapping* is briefly analyzed as well as its architecture and its basic components. Subsequently (chapter 3), some of the most widely used *web map servers* for serving geospatial data through the internet are examined. Some of them are free and open-source, like GeoServer and MapServer and some are proprietary, namely the ESRI's ArcGIS Server. Secondly (chapter 4), the most widely used JavaScript libraries for web map applications such as OpenLayers and Leaflet are also analyzed and compared.

By publishing the web map on the internet each time by using a different map server and trying a different JavaScript library by creating a web map application for the Internet (chapter 5), useful and important results for the capabilities and offered operations of each server and library are derived. The advantages and disadvantages which accrued when publishing the map on the internet are analyzed as well as each one's technical characteristics, the evaluation of the final product and the level of each one's usability by using specific criteria (chapter 6).

Lastly, in chapter 7, the conclusions of the comparison and evaluation of this web mapping software are presented.

Περίληψη

Με την ανάπτυξη της τεχνολογίας αλλά κυρίως με την εμφάνιση του Παγκόσμιου Ιστού (World Wide Web), η κατάσταση στην κλασική Χαρτογραφία μετεβλήθηκε. Πλέον οι χάρτες δεν παρουσιάζονται μόνο σε αναλογική μορφή, αλλά και σε ψηφιακή. Εκτός από τα Συστήματα Γεωγραφικών Πληροφοριών όπως είναι το ArcGIS και το QGIS, τα οποία μπορούν συν τοις άλλοις να παράγουν χάρτες σε ψηφιακή μορφή, τα οποία αφορούν χρήση από έναν μεμονωμένο χρήστη ο οποίος εμφανίζει τα αποτελέσματα (τον χάρτη δηλαδή) μόνο τοπικά, υπάρχει πλέον και η δυνατότητα διάχυσης αυτής της γεωγραφικής πληροφορίας στο διαδίκτυο. Για να επιτευχθεί αυτό, απαιτούνται ορισμένα συγκεκριμένα συστατικά στοιχεία τα οποία συνδυαστικά συνθέτουν την αρχιτεκτονική των διαδικτυακών χαρτογραφικών εφαρμογών. Αυτά τα συστατικά μέρη είναι εν συντομία η *βάση (database) ή ο φάκελος* δεδομένων που περιέχει τα προς χρήση γεωγραφικά δεδομένα τα οποία θα δημοσιοποιηθούν στο διαδίκτυο, ο *εξυπηρετητής γεωγραφικών δεδομένων (web map server),* ο οποίος δημοσιοποιεί μέσω διάφορων προτύπων τη γεωγραφική πληροφορία στο διαδίκτυο και φροντίζει για την οπτικοποίηση και τον συμβολισμό της, ο εξυπηρετητής εφαρμογής ο οποίος παραλαμβάνει τα ερωτήματα μέσω HTTP request και τα μεταφέρει στον εξυπηρετητή χωρικών δεδομένων και τέλος, ο *χρήστης (client),* ο οποίος στέλνει τα ερωτήματα στον εξυπηρετητή και λαμβάνει την απάντηση μέσω HTML (HyperText Markup Language). Χρήστης για παράδειγμα μπορεί να είναι ένας φυλλομετρητής (browser), λ.χ. ο Mozilla Firefox ή ο Google Chrome.

Για να είναι δυνατή η δημοσιοποίηση των γεωγραφικών δεδομένων στο διαδίκτυο με την μορφή χάρτη, έχουν θεσπισθεί κάποια πρότυπα από τον OGC (Open Geospatial Consortium). Ένα από αυτά είναι και το WMS, το οποίο σε συνδυασμό με το WFS (Web Feature Service) είναι αυτά που χρησιμοποιούνται συχνότερα σε τέτοιου είδους εφαρμογές. Με τον όρο WMS (Web Map Service), εννοούμε το πρότυπο το οποίο χρησιμοποιείται για τη διάχυση της γεωγραφικής πληροφορίας στο διαδίκτυο. Το παραγόμενο αποτέλεσμα είναι ο προς θέαση χάρτης σε μορφότυπο εικόνας (π.χ JPEG, PNG κτλ), με δεδομένες διαστάσεις πλαισίου. Έτσι, επιλέγονται τα δεδομένα που θέλουμε να απεικονιστούν από τον εξυπηρετητή χωρικών δεδομένων και οπτικοποιούνται. Αυτό πετυχαίνεται δίνοντας ένα συγκεκριμένο URL με στοιχεία που αφορούν τον χάρτη που θέλουμε να σχεδιαστεί. Άρα τελικά ο εξυπηρετητής γεωγραφικών δεδομένων είναι αρμόδιος για την οπτικοποίηση και διάχυση στο διαδίκτυο της γεωγραφικής πληροφορίας. Οι εξυπηρετητές γεωγραφικών δεδομένων που επελέγησαν να εξετασθούν σε αυτήν την εργασία ήταν οι: GeoServer, MapServer και ArcGIS Server. Οι δύο πρώτοι αποτελούν ελεύθερο και ανοικτού κώδικα λογισμικό, ενώ ο τρίτος είναι ένα λογισμικό κλειστού τύπου που διατίθεται στην αγορά επί πληρωμή.

Η πληροφορία αυτή, ο χάρτης με άλλα λόγια, για να φτάσει στον τελικό χρήστη, πρέπει να διαμορφωθεί σε ένα συγκεκριμένο ψηφιακό περιβάλλον. Αυτό αποκαλείται *περιβάλλον διεπαφής* (user interface). Με αυτόν τον όρο εννοούμε το περιβάλλον το οποίο είναι κατάλληλα φτιαγμένο ώστε να περιέχει τον χάρτη καθώς επίσης και όλη την επιπρόσθετη πληροφορία που θέλουμε να προσδώσουμε στην τελική εφαρμογή. Είναι λοιπόν το αντίστοιχο «αρχιτεκτονικό» κομμάτι του έντυπου χάρτη και ονομάζεται αλλιώς και *ιστοσελίδα*. Η ιστοσελίδα αυτή, λόγω του ότι πρόκειται για περιβάλλον που απευθύνεται διαδικτυακά στον χρήστη, έχει συγκεκριμένο τρόπο δόμησης. Προκειμένου λοιπόν να δημιουργηθεί η ιστοσελίδα, στην πλευρά του χρήστη πλέον, είναι απαιτούμενες κάποιες γλώσσες προγραμματισμού και σήμανσης. Οι εν λόγω γλώσσες που απαιτούνται είναι η HTML, η CSS και η JavaScript. Η HTML είναι η κύρια γλώσσα σήμανσης για τις ιστοσελίδες, και τα στοιχεία της είναι τα βασικά δομικά στοιχεία των ιστοσελίδων. Η CSS είναι μια γλώσσα υπολογιστή που ανήκει στην κατηγορία των «γλωσσών φύλλων ύφους», που χρησιμοποιείται για τον έλεγχο της εμφάνισης ενός εγγράφου που έχει γραφτεί με μια γλώσσα σήμανσης. Χρησιμοποιείται δηλαδή για τον έλεγχο της εμφάνισης ενός εγγράφου που γράφτηκε στις γλώσσες HTML και XHTML, δηλαδή για τον έλεγχο της εμφάνισης μιας ιστοσελίδας και την αναπτύσσει στυλιστικά. Τέλος, η γλώσσα προγραμματισμού JavaScript είναι διερμηνευμένη γλώσσα προγραμματισμού για ηλεκτρονικούς υπολογιστές. Αρχικά αποτέλεσε μέρος της υλοποίησης των φυλλομετρητών Ιστού, ώστε τα σενάρια από την πλευρά του πελάτη (client-side scripts) να μπορούν να επικοινωνούν με τον χρήστη, να ανταλλάσσουν δεδομένα ασύγχρονα και να αλλάζουν δυναμικά το περιεχόμενο του εγγράφου που εμφανίζεται (*Wikipedia*). Επομένως η γλώσσα αυτή ορίζει την συμπεριφορά της ιστοσελίδας και την αλληλεπίδραση με τον χρήστη. Οι τρεις αυτές τεχνολογίες συνδυάζονται μαζί και παράγουν το τελικό αποτέλεσμα στην πλευρά πλέον του χρήστη.

Όσον αφορά την τεχνολογία JavaScript, έχουν αναπτυχθεί επίσης χαρτογραφικά λογισμικά τα οποία ονομάζονται *χαρτογραφικές βιβλιοθήκες* (web mapping libraries). Αυτές είναι λογισμικά τα οποία αποτελούνται και περιέχουν σενάρια (scripts) γραμμένα σε γλώσσα JavaScript και χρησιμοποιούνται στην χαρτογραφική εφαρμογή για να προσδώσουν διαδραστικότητα στον χρήστη. Οι βιβλιοθήκες αυτές «τρέχουν» στην πλευρά του χρήστη, αφορούν δηλαδή τεχνολογία client-side. Μέσω αυτών των βιβλιοθηκών είναι δυνατή η επιλογή εργαλείων διαδραστικότητας όπως είναι η πλοήγηση (pan), η μεγέθυνση και η σμίκρυνση του χάρτη (zoom in/out), η αναγραφή των συντεταγμένων, η χρήση pop-up windows, το μενού αλλαγής θεματικών επιπέδων κτλ. Με αυτόν τον τρόπο είναι δυνατή η επίτευξη αλληλεπίδρασης μεταξύ του τελικού χρήστη και της εφαρμογής. Έτσι, ο χάρτης παύει να είναι στατικός, αλλά γίνεται διαδραστικός. Δύο είναι οι ευρύτερα χρησιμοποιούμενες βιβλιοθήκες JavaScript για διαδικτυακές χαρτογραφικές εφαρμογές: η OpenLayers και η Leaflet, οι οποίες αποτελούν και οι δύο ελεύθερο λογισμικό ανοικτού κώδικα (ΕΛ/ΛΑΚ). Εξετάζονται λοιπόν και οι δύο αυτές βιβλιοθήκες όσον αφορά τα χαρακτηριστικά τους, την ευκολία/δυσκολία τους και τις διαφορές και ομοιότητές τους.

Η σύγκριση και αξιολόγηση τόσο των εξυπηρετητών όσο και των βιβλιοθηκών έγινε στα πλαίσια μίας χαρτογραφικής εφαρμογής για το διαδίκτυο. Στόχος της εφαρμογής αυτής ήταν να αναδείξει διαφορές και ομοιότητες μεταξύ των τριών εξυπηρετητών χαρτογραφικών δεδομένων και των δύο βιβλιοθηκών και κατόπιν βάσει ορισμένων κριτηρίων να αξιολογηθούν.

Για τον σκοπό αυτόν δημιουργήθηκε ένας χάρτης της νήσου Σύρου ο οποίος αποτελούνταν από τα εξής θεματικά επίπεδα: ακτογραμμή, πολύγωνο ξηράς και πρωτεύουσας, ισοϋψείς καμπύλες, τριγωνομετρικά και υψομετρικά σημεία, υδρολογικό και οδικό δίκτυο, και εν συνεχεία κάποια σημεία ενδιαφέροντος (POI-Points of Interest) όπως οι παραλίες, τα θαλάσσια σπορ, τα λιμάνια, τα μνημεία, οι εκκλησίες, οι μονές και τα εξωκκλήσια, τα σημεία κάμπινγκ, το αεροδρόμιο, το ελικοδρόμιο, το νοσοκομείο, τα βενζινάδικα και οι οικισμοί. Επειδή ο χάρτης αυτός θα απεικονισθεί σε τρεις διαφορετικές κλίμακες σε κάθε εξυπηρετητή, τα ως άνω χωρικά δεδομένα (τα οποία ήταν μορφοτύπου shapefile) υπέστησαν χαρτογραφική γενίκευση προκειμένου να ανταποκρίνονται στις δεδομένες κλίμακες. Εκτός από τα προαναφερθέντα, προσετέθησαν και τα τοπωνύμια, όπως είναι η πρωτεύουσα, τα ακρωτήρια, τα λιμάνια και οι όρμοι. Οι κλίμακες απόδοσης ήταν οι 1:50.000, 1:75.000 και 1:150.000. Ειδικά για την μικρότερη κλίμακα, την 1:150.000, το ανάγλυφο δεν αποδόθηκε μέσω των ισοϋψών καμπύλων, αλλά μέσω της σκίασης αναγλύφου που δημιουργήθηκε σε κανονικοποιημένη (raster) μορφή από το ψηφιακό μοντέλο εδάφους της περιοχής.

Η δημιουργία των χαρτών για χρήση σε κάθε έναν από τους εξυπηρετητές ήταν διαφορετική. Για τον ArcGIS server ο χάρτης δομήθηκε εξ ολοκλήρου στο λογισμικό ArcMap και στη συνέχεια μεταφορτώθηκε στον εξυπηρετητή ενώ για τον GeoServer, δημιουργήθηκε η σύνδεση με τον φάκελο ο οποίος περιέχει τα αρχεία και μεταφορτώθηκαν στον GeoServer. Στη συνέχεια, σειρά είχε ο συμβολισμός των δεδομένων. Ο συμβολισμός στον GeoServer δομείται μέσω της χρήσης ενός προτύπου θεσπισμένου από τον OGC, το SLD (Styled Layer Descriptor). Αυτό αποτελεί έναν κώδικα γραμμένο σε γλώσσα XML (eXtensive Markup Language) ο οποίος περιέχει όλη την πληροφορία για τον συμβολισμό κάθε θεματικού επιπέδου και ο οποίος πρέπει να αντιστοιχισθεί με το εκάστοτε θεματικό επίπεδο ώστε να γίνει δυνατή η οπτικοποίησή του. Όλα αυτά γίνονται εφικτά μέσω του φιλικού περιβάλλοντος διεπαφής (user interface) που παρέχει ο GeoServer. Αντιθέτως, ο MapServer λειτουργεί τελείως διαφορετικά. Δεν διαθέτει περιβάλλον διεπαφής χρήστη, επομένως η εισαγωγή των δεδομένων και του συμβολισμού τους ενσωματώνεται σε ένα ενιαίο αρχείο που περιέχει όλη την πληροφορία προκειμένου να απεικονισθεί ο χάρτης. Το αρχείο αυτό ονομάζεται *mapfile* και είναι ο θεμέλιος λίθος του MapServer. Αφού δημιουργηθεί το εν λόγω αρχείο, καλείται μέσω αιτήματος WMS έτσι ώστε να απεικονιστεί ο χάρτης στο διαδίκτυο χρησιμοποιώντας το κατάλληλο URL.

Ο χάρτης όπως αναφέρθηκε και παραπάνω, δημιουργήθηκε τρεις φορές με κάθε έναν εξυπηρετητή προκειμένου να συγκριθούν τα αποτελέσματα, να διαπιστωθούν

διαφορές και να αξιολογηθούν με βάση ορισμένα κριτήρια. Ως προς το τελικό οπτικό αποτέλεσμα που παράγουν, βρέθηκε ότι και οι τρεις εξυπηρετητές ανταποκρίθηκαν εξίσου καλά και ήταν ικανοί να παράγουν υψηλού επιπέδου χαρτογραφικά προϊόντα. Ένα άλλο κριτήριο σύγκρισης ήταν ο τρόπος που δομούν τον συμβολισμό όπου διαπιστώθηκε ότι ενώ είναι διαφορετικός σε κάθε περίπτωση, το παραγόμενο αποτέλεσμα διαφέρει ελάχιστα έως καθόλου. Στη συνέχεια συγκρίθηκαν και αξιολογήθηκαν βάσει των τεχνικών χαρακτηριστικών τους και τον χρόνο τον οποίον χρειάζεται κανείς για να υλοποιήσει μια ολοκληρωμένη διαδικτυακή χαρτογραφική εφαρμογή χρησιμοποιώντας του αντίστοιχους εξυπηρετητές και διαπιστώθηκε ότι το κλειστού κώδικα λογισμικό, δηλαδή ο ArcGIS Server ήταν ο ταχύτερος αλλά και ο ευκολότερος στην υλοποίηση, πράγμα το οποίο ήταν αναμενόμενο λόγω του ότι δεν απαιτείται γνώση συγγραφής κώδικα ή κατανόηση της αρχιτεκτονικής του. Η δυσκολία του GeoServer έγκειται στο γεγονός ότι χρειάζεται χρόνος ώστε να εξοικειωθεί κανείς με το τρόπο συμβολισμού, με την προδιαγραφή SLD δηλαδή, καθώς επίσης απαιτείται επίσης χρόνος ώστε να συγγραφεί, ειδικά εάν το πλήθος των δεδομένων είναι μεγάλο ή έχουν πολύπλοκο συμβολισμό (όπως τα τοπωνύμια με την μέθοδο annotation). Τέλος, η δυσκολία με τον MapServer προκύπτει από την απουσία περιβάλλοντος διεπαφής με τον χρήστη, έτσι ώστε για έναν αρχάριο να χρειάζεται αρκετός χρόνος για την εξοικείωση με αυτόν. Ένα επιπλέον σημαντικό στοιχείο που αποτελεί κριτήριο σύγκρισης και αξιολόγησης είναι το ποιούς μορφοτύπους δεδομένων εισόδου και εξόδου μπορεί να υποστηρίξει ο κάθε εξυπηρετητής. Αυτοί οι μορφότυποι δεδομένων αφορούν διανυσματικά δεδομένα, κανονικοποιημένα δεδομένα καθώς και διάφορες βάσεις δεδομένων οι οποίες είναι δυνατόν να συνδεθούν με τους εξυπηρετητές. Λόγω της ευρείας κοινότητας χρηστών των ελεύθερων λογισμικών ανοικτού κώδικα (ΕΛ/ΛΑΚ) και της διαλειτουργικότητας την οποία πρεσβεύουν, τα λογισμικά αυτά συνεχώς εξελίσσονται και βελτιστοποιούνται και έχουν φτάσει σε ένα βαθμό τέτοιον, ώστε να μην υστερούν σε σχέση με τα αντίστοιχα λογισμικά κλειστού κώδικα.

Ταυτόχρονα με τους εξυπηρετητές, έγινε δοκιμή και στις δύο από τις πιο συχνά χρησιμοποιούμενες βιβλιοθήκες JavaScript για χαρτογραφικές εφαρμογές, την OpenLayers και την Leaflet. Όσον αφορά τις βασικές τους λειτουργίες, και οι δύο βιβλιοθήκες έχουν την δυνατότητα πρόσθεσης επιπλέον θεματικού επιπέδου διανυσματικών δεδομένων (vector overlay) και θεώρηση βασικού επιπέδου (base layer) όπως επίσης επιτελούν και τις βασικότερες λειτουργίες που απαιτούνται σε μία διαδραστική χαρτογραφική εφαρμογή, δηλαδή εργαλεία αλλαγής κλίμακας, για παράδειγμα μεγέθυνση/σμίκρυνση (zoom in/out), πλοήγηση (pan) κ.ο.κ αλλά και δημιουργία πινακιδών (tiles). Επιπλέον, υποστηρίζουν εκτός από το πρότυπο WMS, και το πρότυπο WFS με την προϋπόθεση ότι στα δεδομένα μπορούν να εφαρμοστούν ερωτήματα (queries). Μέσω αυτού λοιπόν είναι δυνατή η επιλογή υποσυνόλου των δεδομένων μέσω του ποντικιού και η αναγραφή πληροφοριών (χωρικών ή μη) για αυτά. Οπτικά αυτό μπορεί να αποδοθεί μέσω ενός pop-up παραθύρου. Κάτι το οποίο δεν υποστηρίζει η Leaflet, αλλά επιτελεί με ευκολία η

OpenLayers είναι η αλλαγή της χαρτογραφικής προβολής (reprojection). Τέλος, το γεγονός ότι η Leaflet αποτελεί ελαφρύτερο λογισμικό, συμβαίνει επειδή πολλές από τις λειτουργίες της είναι διαθέσιμες μόνο μέσω επεκτάσεων (Plugins). Ασφαλώς και οι δύο βιβλιοθήκες έχουν περιθώρια βελτίωσης ούτως ώστε να μπορούν στο μέλλον να συμπεριλάβουν και άλλες λειτουργίες ως επί το πλείστον πιο χαρτογραφικές, όπως για παράδειγμα η γενίκευση σε πραγματικό χρόνο (real-time generalization) (Ντζελέπης, 2014). Και τα δύο λογισμικά αυτά διαθέτουν έναν υψηλό αριθμό απασχολούμενων (κοινότητα) η οποία λαμβάνει υπ΄όψην τις ανάγκες των χρηστών και συνεχώς βελτιώνει τις επιδόσεις του κάθε λογισμικού και διορθώνει τυχόν λάθη στον κώδικα.

Παρατηρήθηκε λοιπόν ότι για απλές χαρτογραφικές εφαρμογές, τόσο η OpenLayers όσο και η Leaflet μπορούν να ανταποκριθούν εξίσου καλά. Η διαφορά τους εντοπίζεται όταν πρόκειται για πιο απαιτητικές εφαρμογές, όπου η OpenLayers παρέχει περισσότερες δυνατότητες από την Leaflet, όμως είναι και πιο απαιτητική στην εκμάθησή της. Από την άλλη όμως, η Leaflet ενδείκνυται περισσότερο για χαρτογραφικές εφαρμογές που απευθύνονται σε κινητά, tablets και smartphones, αφού είναι ελαφρύτερη. Το καλύτερο οπτικό αποτέλεσμα πετυχαίνεται με χρήση της OpenLayers, λόγω της καλύτερης ανάλυσης που προσφέρει. Επιπλέον, εξετάστηκαν και άλλα κριτήρια τα οποία αφορούσαν όχι τόσο τις επιτελούμενες λειτουργίες που προσφέρουν, αλλά το θέμα της διαχείρισής τους μέχρι να επιτευχθεί το τελικό αποτέλεσμα. Το πρώτο από αυτά αφορούσε τον απαιτούμενο χρόνο υλοποίησης για την ολοκλήρωση του κώδικα JavaScript της εκάστοτε βιβλιοθήκης. Ο χρόνος που απαιτούταν για τη δόμηση του κώδικα στην Leaflet, ήταν πολύ λιγότερος από ότι ο χρόνος που χρειάστηκε για την OpenLayers. Ένα δεύτερο σημαντικό κριτήριο για την επιλογή της κατάλληλης βιβλιοθήκης αποτελεί το εάν υπάρχει διαθέσιμο επιμορφωτικό υλικό (tutorial) το οποίο θα βοηθάει τους χρήστες να αντιμετωπίσουν δυσκολίες που θα προκύψουν, καθώς επίσης και επαρκείς οδηγίες χρήσης (documentation). Οι γνώσεις σε προγραμματισμό που απαιτούνται είναι ελάχιστες. Αρκεί μία υποτυπώδης γνώση της γλώσσας JavaScript μόνο και μόνο ώστε να γίνει αντιληπτή η δομή της και να αναγνωρίζεται εύκολα. Τελευταία κριτήρια σύγκρισης αποτελούν ο βαθμός δυσκολίας εκμάθησης και υλοποίησής τους. Γενικότερα, η OpenLayers είναι πιο δύσκολη στην εκμάθησή της καθώς επίσης και στην υλοποίησή της.

Από τα προαναφερθέντα γίνεται φανερό ότι η τάση στην Διαδικτυακή Χαρτογραφία πλέον είναι η στροφή προς τα ελεύθερα και ανοικτά λογισμικά (ΕΛ/ΛΑΚ), όχι μόνο όσον αφορά τους εξυπηρετητές χαρτογραφικών δεδομένων και των βιβλιοθηκών, αλλά όλων των επιμέρους συστατικών στοιχείων που απαρτίζουν ένα τέτοιο σύστημα. Με την πάροδο των χρόνων και την ενασχόληση όλο και μεγαλύτερου αριθμού χρηστών τα λογισμικά αυτά εμπλουτίζονται, ανανεώνονται και συμβαδίζουν με όλες τις νέες τεχνολογίες. Επιπλέον έχουν μηδενικό κόστος αγοράς και η εγκατάστασή τους μπορεί να γίνει πολύ εύκολα. Τα λογισμικά αυτά βέβαια, εκτός από το είδος της άδειας που έχουν, διαφοροποιούνται πολύ και ως προς τον τρόπο με τον οποίο επεξεργάζονται και παράγουν τα δεδομένα καθώς και ως προς

τις λειτουργίες που επιτελούν. Γίνεται αντιληπτό λοιπόν, ότι η επιλογή του κατάλληλου εξυπηρετητή ή/και βιβλιοθήκης εξαρτάται από την χαρτογραφική εφαρμογή και τις απαιτήσεις της. Και τα τρία λογισμικά εξυπηρετητών ήταν ικανά να παράγουν υψηλής ποιότητας χάρτες παρά τις μεγάλες τους διαφορές αλλά και οι δύο βιβλιοθήκες ανταποκρίθηκαν εξίσου καλά, με ένα μικρό προβάδισμα στην OpenLayers. Φυσικά, πολλές φορές, δεν υπάρχει αντικειμενικό κριτήριο επιλογής και σε αυτές τις περιπτώσεις η επιλογή γίνεται αποκλειστικά από τον χρήστη, είτε για λόγους προτίμησης, συνήθειας και ευκολίας.

Table of Contents

Tables

Figures

Diagrams

1. INTRODUCTION

For decades, digital geographic data were analyzed, displayed and used for cartographic purposes through desktop-based PCs only and data sharing was limited. It required specialized and often proprietary software to handle the data and this resulted in narrowing the audience that could benefit from them.

After the mass uptake of Internet in the mid-1990s, people came up with the idea of how those geographic or more commonly spatial data or even *geodata* could be shared within organizations but also with the public (Walgrun).

With the wide spread of spatial data through the Internet, web mapping was also developed. With the term 'web mapping' we mean the process of constructing and using maps delivered by geographic information systems, known as GIS (Wikipedia).

The advent of web mapping can be regarded as a major development in cartography. Until recently, cartography was restricted to a few companies, institutes and mapping agencies, requiring relatively expensive and complex hardware and software as well as skilled cartographers and geomatics engineers.

It is easily understandable that web mapping goes beyond classic web cartography, and that happens because a web map on the World Wide Web is both served and displayed. Therefore, it consists of a service which involves some very important hardware parts such as a client and a server and the associated software. This architecture will be analyzed later in this document.

The first step towards web mapping was the posting of static images of maps on the Internet through HTML pages. HTML (HyperText Markup Language) is a markup language for creating web sites and web applications.

This was very simple and soon the need for interactive maps came to surface. At the beginning, the maps requested were simple and servers had issues to be solved such as speed and scalability. Scalability is the ability to handle simultaneously many users. This resulted in the accommodation (from the server) of a limited number of map requests at a time before slowing down and/or crashing. However, this was a huge step in the history of web mapping, even though now it may seem a little primitive. In addition, more services were added.

Recently, after the deployment of the smartphones, mobile-based applications have also been developed. Therefore, the development of mobile-GIS began, such as Google Maps, Foursquare etc.

Nowadays, the technology has been developed very rapidly, so new technologies have shown up. This can have positive effects to the future of web mapping as nowadays cartographers can do more things than ever. It also has some

disadvantages though, as it is very difficult to keep in pace with these new technologies because they evolve rapidly (Roth et. al., 2013). These technologies even include cloud web mapping, Intelligent web mapping. These technologies are relatively new and they are not presented in this thesis.

The history of web mapping is presented below (Veendaal et. al, 2017) to help readers see how rapidly it evolves:

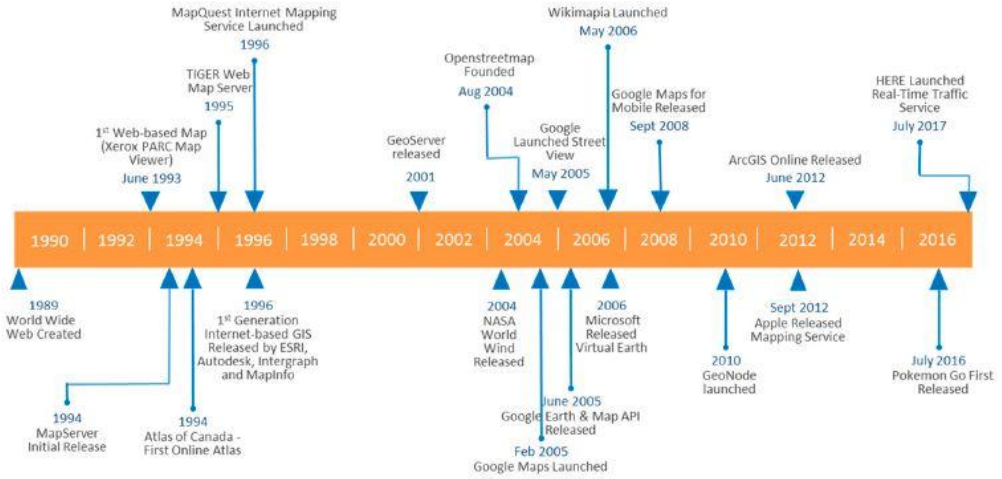

Figure 1.1: History of important steps in web mapping (Veendaal et.al., 2017)

This web-GIS architecture consists of several components which all together build this complicated system. The last years it has also begun a turnover on using free and open-source software for building such web map applications. In this thesis are compared two of the most important components of this system, the map servers and the JavaScript libraries and they are also evaluated. For the comparison it was considered appropriate to compare not only two of the most powerful free and open source map servers like GeoServer and MapServer but also to add to the comparison one proprietary, the ArcGIS Server. The addendum of the proprietary service was necessary so as to compare the free and open–source software (FOSS) with the closed proprietary and finally decide whether the FOSS can be as worthy as the proprietary and therefore be used instead of them or not. The criteria chosen concern not only the technical characteristics of each software, but also their usability, their handling and they quality of maps that they produce.

2. WEB MAPPING

This chapter deals with the architecture of web-GIS systems and the further analysis of their components. Finally, the most important and widely-used geospatial web services are presented and described as defined by the Open Geospatial Consortium (OGC)[1].

2.1 General

Web Mapping is a relatively new trend in cartography which became more popular after the uptake of the internet and the development of free and open-source Geographic Information System software (GIS) such as QGIS. Until recently, cartography was restricted to a few companies, institutes and mapping agencies, requiring relatively expensive and complex hardware and software as well as skilled cartographers and geomatics engineers.

Web mapping is the process of constructing and using maps delivered by geographic information systems (GIS). A web map on the World Wide Web is both served and utilized, thus web mapping is more than just web cartography, it is a service by which consumers may decide what the map will show.

On the other hand, web-GIS emphasizes on geodata processing aspects which are involved with design aspects such as data acquisition and server software architecture.

Web mapping utilizes both free geographical datasets (for example OSM - www.opestreetmap.org) and proprietary ones. A range of free software to generate maps has also been conceived and implemented alongside proprietary tools like ArcGIS. As a result, the barrier for serving maps on the web has been lowered.

Another factor that helped the spread of those systems has been the availability of free (open) spatial data through the internet. Nowadays, spatial data are available from many international and local organizations and are intended for many purposes.

Geospatial data can be used in numerous applications combined with GIS such as:

[1] www.opengeospatial.org

- ✓ Cadastre
- ✓ Environmental Management
- ✓ Cartography
- ✓ Geology and Hydrogeology
- ✓ Urban Planning
- ✓ Transportation
- ✓ Telematics
- ✓ Construction and Infrastructure projects
- ✓ National Defense and Governmental projects
- ✓ Statistics and Analysis
- ✓ Crime analysis

and many other more.

Types of web maps

Two are the most distinctive types of web maps. The first one is the *static map,* where the image of a map is placed on a web page, and the *interactive map*.

Static maps are view-only without animation or interactivity. These files are created once, often manually, and are rarely updated. Typical graphics formats for static web maps are PNG, JPEG, GIF, or TIFF for raster files and SVG, PDF or SWF for vector files.

Interactive maps offer the capability of interaction with the user and the client interface through some very useful tools such as pan, zoom in/zoom out, clickable areas, layers on/off etc. These functionalities demand more specialized tools such as JavaScript libraries (e.g. OpenLayers, Leaflet).

2.2 Web-GIS Architecture

Though terms as Web-GIS and Web Mapping are used interchangeably, they do not actually mean the same. Web mapping enables the design and visualization of geographically referenced data through a web interface. Web-GIS enables the communication of all components through the web, including data and algorithm analysis. Thus, web mapping is a subset of Web-GIS, as shown in figure 2.1 and usually involves client-server interaction (IJARCEE- Heda & Chikurde).

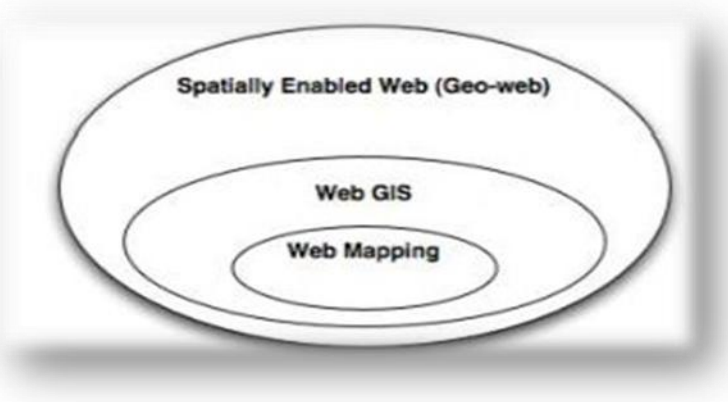

Figure 2.1: WebGIS and its subsets (IJARCEE- Heda & Chikurde)

Web-GIS has a complicated architecture which consists of several structural elements. The model that is mostly applied is the *client-server model*. Based on this model, there is typically the **client,** which receives and displays the final products, **a web server**, an **application server** and one or more **data servers** and **GIS servers**. Therefore, the main components of a typical Web-GIS system are:

- The client
- The web server and the application server
- The map server
- The data server

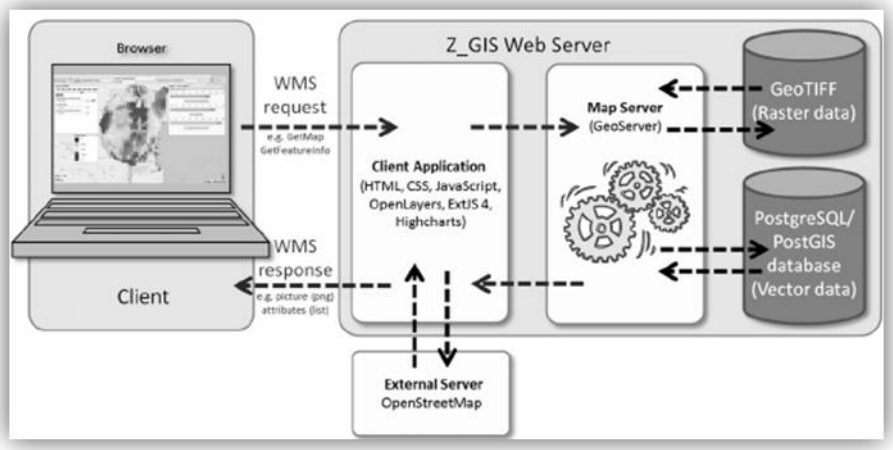

Figure 2.2: WebGIS architecture, (https://www.researchgate.net/)

2.2.1 Client

A client is a computer or a program that, as part of its operation, relies on sending a request to another computer program (which may or may not be located on another

computer). For example, web browsers are clients that connect to web servers and retrieve web pages for display. A client is part of a client–server model, which is still used today.

Clients and servers may be computer programs running on the same machine and are connected via inter-process communication techniques. Combined with Internet sockets, programs may connect to a service operating on a remotely located system through the Internet protocol suite. Servers wait for potential clients to initiate connections that they may accept.

In few words, a client is the site where the final products are presented. The client sends a request through HTTP to the server and gets the answer through HTML. While traditional desktop-based GIS use graphic user interfaces as their clients lying on installed software, web-GIS applications mostly use web browsers as clients together with several web mapping APIs, like OpenLayers.

2.2.2 Web Server and Application Server

The second component is the web server and the application server. The web server is responsible for answering the requests initiated from clients through HTTP protocol. There are several ways for these requests to be answered, like:

• By sending existing HTML files to the clients (such as JPEG static map images), or
• By transferring the requests to other programs called CGI programs, which process the request.

In order for the requests to be transferred, the web server activates the application server. Mainly, the application server acts like a compiler and proxy of those requests and interpretates them.

2.2.3 Map Server

The map server is a very significant component for web-GIS and web mapping. The map server is responsible for executing all spatial requests, spatial analysis and produces and distributes requested maps at various formats. Outputs of a map server may be either:

• Data, vector or raster, or

• Map images, which can be formatted as JPEG, TIFF, etc.

The role of the map server can play several open source servers, most notable and widely used are GeoServer and MapServer and the proprietary software, ArcGIS Server.

2.2.4 Data Server

The data server is the environment where the spatial and non-spatial information is stored. Usually, they are stored into a relational or non-relational database. Through the map server, the client is able to perform spatial or non-spatial queries to the database by composing SQL queries to it. Open source databases are MySQL, Oracle Spatial, Postgres/PostGIS etc.

2.3 Implementation patterns for building Web-GIS applications

There are several categories of patterns for implementing Web-GIS. Most important of them are the three below:

- those based on the server (**server-side**)
- those based on the client (**client-side**)
- **hybrid**, which takes advantages of both server and client for distributing its functionalities.

Regarding the server-side systems, the user asks for some data and the request is passed to the server, where the processing is made. The server then, sends the answer to the user. Therefore, the whole process is executed within the server.

On the contrary, client-based Web-GISs are based on the computing power of the client. The processes and calculations are mainly executed by the client and the server is used only to get new data or search from the database.

Hybrid method offers a balanced distribution of the processions to both server and client. In this way, the capabilities of both technologies are exploited.

2.3.1 Server-side Web-GIS techniques

As mentioned above, server-side techniques are based mainly on the server operations and are using the web browser as their user interface. More specifically, according to this architecture, the scripts are executed by the server and the results are sent to the client's web browser in the form of an HTML page.

These pages - in contrast with those pages that are produced at the client's side using script languages - contain neither code nor embedded scripts, but only tags which determine the structure of the page. Furthermore, server-side scripting

languages (e.g. PHP) are used to connect and communicate with external databases from which they get data.

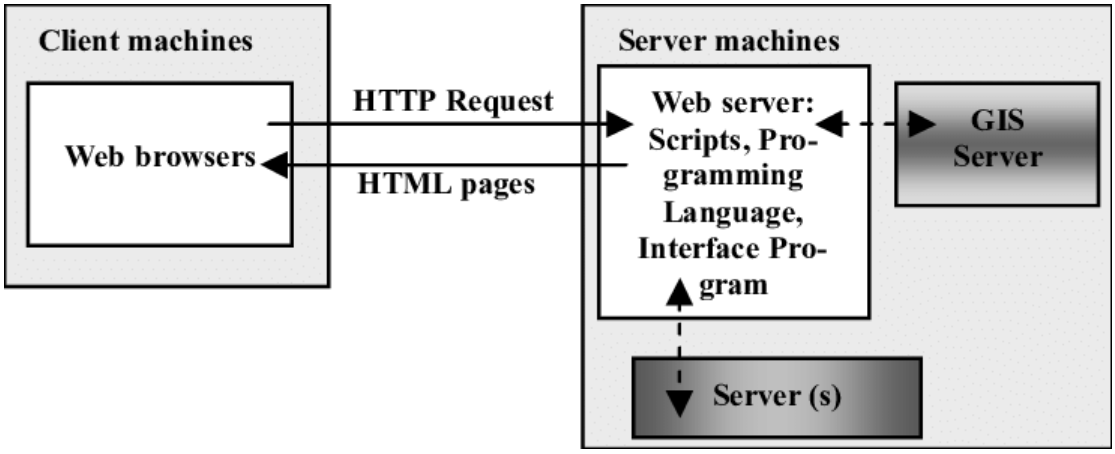

Figure 2.3: Server-side architecture (https://www.researchgate.net/)

The advantage of this method is that users can have access to complex and bulky data, which would neither be easily available locally nor easily transferred via internet and secondly the fact that users can execute even complex geographic analysis processes using systems of relatively low specification.

On the contrary, the main disadvantage of this method is that every task has to be transferred through Internet in order to be executed and that is a factor that slows up the whole process. From this is deducted that the efficiacy depends on the server's characteristics and the connection it can achieve. A second disadvantage is that the capabilities of the client are not exploited.

2.3.2 Client-side Web-GIS techniques

A way to overcome the disadvantages of server-side techniques is to build a Web-GIS system with a new, different architecture, like client-side Web-GIS. With this type of architecture, most operations are executed on the web browser of the user's system and the server is used mainly to obtain new data. Client-side techniques are divided into two main categories; Those which require software installation and those which all of its capabilities are included within a program (usually written in Java) that is "downloaded" on the user's machine through an HTML page. This program is called *applet*.

The main characteristic of this method is that the applications take advantage of all client's capabilities and lighten the server, while the main disadvantage might be the fact that transferring large amount of data may cause serious delays.

2.3.3 Hybrid Web-GIS architecture

Through this architecture a balanced distribution of the operations is achieved. The server is used to obtain data from databases, while the client is used for the simple operations that require continuous intervention from the user. In this case, server and client are in permanent communication to each other.

With client-side and server-side scripting using programming languages, users are given the opportunity to interact with web pages and modify content. This approach is also used at this thesis.

The scheme below shows how hybrid architecture works, showing the various technologies used for each step of the process of building a map for the internet.

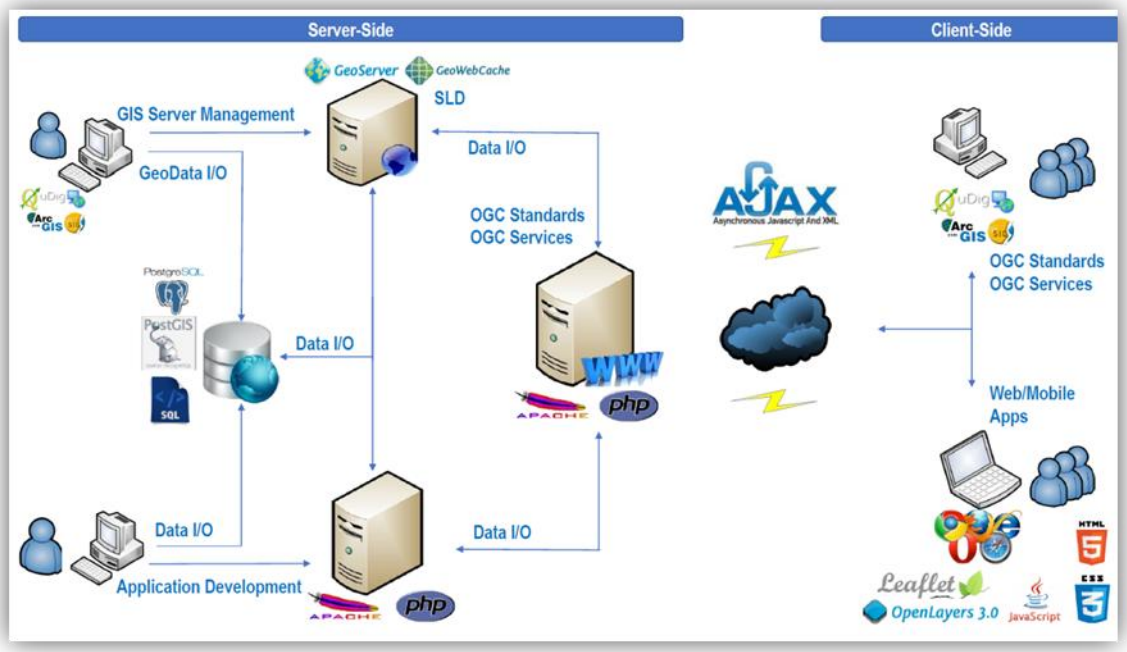

Figure 2.4: Hybrid architecture (Tsoulos & Antoniou, 2017, National Technical University of Athens)

2.4 Geospatial Web Services

Geospatial web services are services providing standard procedures and tools for sharing geospatial data on the World Wide Web. These services perform operations such as:

- Providing access to geographic data stored in a remote database or in a remote store in general.
- Executing calculations, like distance measurement between two points or calculating polygon areas.
- Returning messages containing geographic information that can be delivered as text or numeric data.
- Returning map images in various formats for displaying or further editing from another web service.
- Executing queries on spatial data and return results as text or image.

In order to implement geospatial web services into GIS, specifications and standardization have been developed. These specifications were developed by organizations such as the Open Geospatial Consortium (OGC) and the International Organization for Standardization.

OGC is an international non - profit organization committed to making quality open standards for the global geospatial community. These standards are made through a consensus process and are freely available for anyone to use to improve sharing of the world's geospatial data. Some of OGC's standards for spatial data sharing through the internet are described below.

2.4.1 Web Map Service

The OpenGIS® Web Map Service Interface Standard (WMS) provides a simple HTTP[2] interface for requesting geo-registered map images from one or more distributed geospatial databases.

A WMS request defines the geographic layer(s) and area of interest to be processed. Clients, through a simple HTTP interface, request a map from a map server who is

[2] HTTP: The **Hypertext Transfer Protocol** (**HTTP**) is an application protocol for distributed, collaborative, and hypermedia information systems. HTTP is the foundation of data communication for the World Wide Web. Hypertext is structured text that uses logical links (hyperlinks) between nodes containing text. HTTP is the protocol to exchange or transfer hypertext. (*en.wikipedia.org/wiki/Hypertext_Transfer_Protocol*)

compliant with WMS service (e.g. GeoServer, MapServer) and the response to the request is one or more geo-registered map images (returned as JPEG, PNG or SVG etc) that can be displayed in a browser application. The interface also supports the ability to specify whether the returned images should be transparent so that layers from multiple servers can be combined or not.

Therefore, WMS clients can request images from multiple WMS servers, and then combine them into a single view for the user. The request is accompanied by a series of parameters which define the extent of the map (bounding box), the coordinate system (projection), the image format etc.

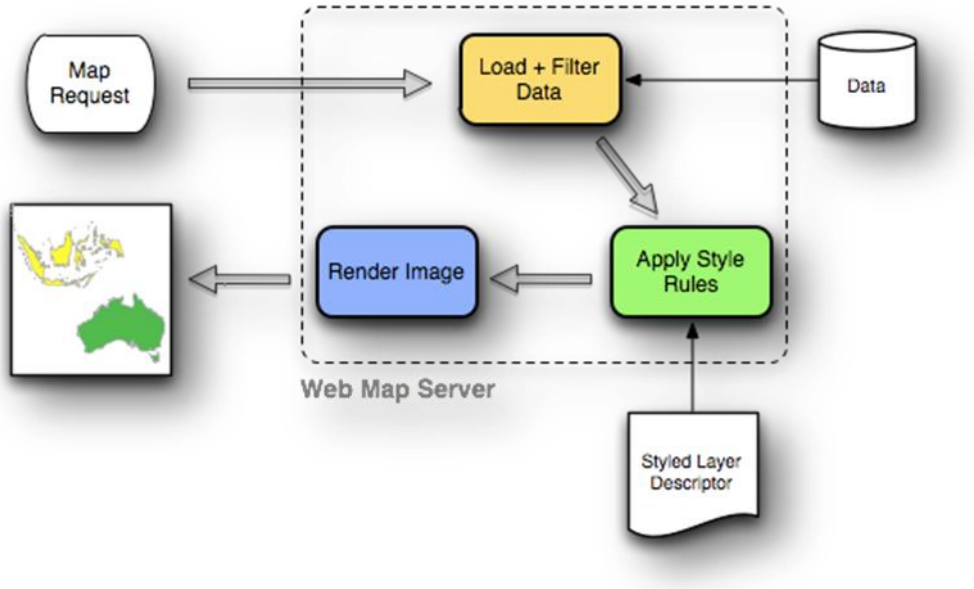

Figure 2.5: A diagram showing how a WMS turns data into a map image (opengeo.org)

The most widely used WMS request is **Get Map**. Through GetMap the client retrieves a map image at various formats such as JPEG, PNG etc, for a specific area and content. A typical example of a WMS GetMap request includes the following parameters:

Parameter	*Description*
REQUEST=*GetMap*	name of request
VERSION=version	version of service
LAYERS=layers_list	list of selected layers to display
STYLES=styles_list	style list

CRS or SRS=namespace:identifier	coordinate system
BBOX=[minx,miny,maxx,maxy]	extent of the output image
WIDTH=output_width	image width in pixels
HEIGHT=output _height	image height in pixels
FORMAT=output_format	output format (PNG,JPEG etc)

Table 2.1: Parameters of a WMS GetMap request

WMS requests can also perform some other operations, which are mentioned briefly below:

• **GetCapabilities**: Requests metadata about the operations, services, and data ("capabilities") that are offered by a WMS server. The parameters for the GetCapabilities operation are:

Parameter	*Required?*	*Description*
service	Yes	Service name. Value is WMS.
version	Yes	Service version. Value is one of 1.0.0, 1.1.0, 1.1.1, 1.3.0.
request	Yes	Operation name. Value is GetCapabilities.

Table 2.2: GetCapabilities Parameters

• **GetFeatureInfo**: The ***GetFeatureInfo*** operation requests the spatial and attribute data for the features at a given location on a map. It is similar to the WFS *GetFeature* operation, but less flexible in both input and output. The one advantage of GetFeatureInfo is that the request uses an (x,y) pixel value from a returned WMS image. This is easier to use for a naive client that is not able to perform true geographic referencing. The standard parameters for the GetFeatureInfo operation are same with GetMap, but additionally, layers have to be queryable.

• **GetLegendGraphic:** The ***GetLegendGraphic*** operation provides a mechanism for generating legend graphics *as images*, beyond the LegendURL reference of WMS Capabilities. It generates a legend based on the style defined on the server, or alternatively based on a user-supplied SLD.

2.4.2 Web Feature Service

The Open Geospatial Consortium (OGC) Web Feature Service (WFS) is a protocol for serving geographic features across the Web. Feature information that is encoded

and transported using WFS includes both feature geometry and feature attribute values. Basic Web Feature Service (WFS) supports feature query and retrieval.

WFS defines a standard for exchanging vector data over the Internet using HTTP. With a compliant WFS, clients can query both the data structure and the source data. Advanced WFS operations also support feature locking and edit operations. A WFS encodes and transfers information of spatial data in Geography Markup Language (GML), a subset of the standard web language XML.

The basic operations of a WFS request are:

• **GetCapabilities:** Requesting the capabilities using the GetCapabilities request to a WFS server returns an XML document showing what layers and projections are available.
• **DesribeFeatureType:** Returns a description of feature types supported by a WFS service.
• **GetFeature:** Returns a selection of features from a data source including geometry and attribute values.

WFS returns features and feature information in a number of formats. Most common of them are GML, ESRI Shapefile, JSON, CSV.

2.4.3 Web Coverage Service

The Web Coverage Service (WCS) is a standard created by the OGC that refers to the receiving of geospatial information as 'coverages': digital geospatial information representing space-varying continuous phenomena. WCS provides few options for changing coverage functionality. While various elements can be configured for WFS and WMS requests, WCS allows only metadata information to be edited. This metadata information, entitled *Service Metadata*, is common to WCS, WFS and WMS requests.

One can think of WCS as the equivalent of Web Feature *Service* (WFS), but for raster data instead of vector data. It lets you get at the raw coverage information, not just the image.

An important distinction must be made between WCS and WMS. They are similar, and can return similar formats, but a WCS is able to return more information, including valuable metadata and more formats. Additionally, it allows more precise queries, potentially against multi-dimensional backend formats.

WCS provides a standard interface on how to request the raster source of a geospatial image. While a WMS can return an image it is generally useful only as an image. The results of a WCS can be used for complex modeling and analysis, as it

often contains more information. It also allows more complex querying - clients can extract just the portion of the coverage that they need.

2.4.4 Web Processing Service

The Web Processing Service (WPS) Interface Standard provides rules for standardizing how inputs and outputs (requests and responses) for geospatial processing services, such as polygon overlay. The standard also defines how a client can request the execution of a process, and how the output from the process is handled. It defines an interface that facilitates the publishing of geospatial processes and clients' discovery and binding to those processes. The data required by the WPS can be delivered across a network or they can be available at the server.

2.4.5 Web Feature Service Transactional

The basic Web Feature Service allows for the query and retrieval of features. A transactional Web Feature Service (WFS-T extension) allows modification of the data, including creation, deletion and updating of features. Changes are stored directly to the database.

3. WEB MAP SERVERS

In this chapter, the web map servers used for comparison are described. This thesis examines three of the most-used web map servers for web mapping applications, GeoServer and MapServer which are free and open source (FOSS) and ArcGIS Server, which is a proprietary one.

3.1 General

Web mapping servers act as a framework to publish a map or a GIS application online. Generally, these servers include functionality to query spatial DBMS, projection support, integration with other geographic libraries as well as vector and raster support. Moreover, interoperable web standards have been developed by OGC in support of web mapping, including WMS, WFS, WCS, Geography Mark-up Language (GML) and Styled Layer Descriptor (SLD). These OGC services allow numerous projects to be published online.

3.2 GeoServer

3.2.1 About GeoServer

GeoServer[3] is an open - source server written in Java that allows users to share, process and edit geospatial data. Designed for interoperability, it publishes data from any major spatial data source using open standards. GeoServer is an OGC compliant implementation of numerous open standards such as Web Feature Service (WFS), Web Map Service (WMS) and Web Coverage Service (WCS).

Additional formats and publication options are available including Web Map Tile Service (WMTS) and extensions for Catalogue Service (CSW) and Web Processing Service (WPS).

GeoServer can handle a wide variety of formats including various spatially enabled DBMS, such as Oracle Spatial, Postgres/PostGIS, ArcSDE, DB2, My SQL, as well as ESRI shapefiles, GeoTIFF, GTOPO30 and much more.

Through standard protocols, it can also produce data in many formats such as KML, GML, ESRI Shapefile, GeoJSON, PDF, JPEG, PNG, GIF, SVG and many more. In

[3] www.geoserver.org

addition, someone can also edit data by using the WFS-T (Web Feature Service – Transactional) profile. GeoServer also includes an integrated OpenLayers client for previewing data layers.

GeoServer additionally supports efficient publishing of geospatial data to Google Earth through network links, using KML. Advanced features for Google Earth output include templates for customized pop-ups, time and height visualizations, and "super-overlays".

Regarding its architecture, GeoServer uses Restlet as a framework for the REST services it provides. It packages Jetty (web server) as an embedded server, but supports any common servlet container. GeoWebCache, a Java-based caching component similar to TileCache, is bundled with GeoServer, but it is available separately. Similarly, GeoServer packages GeoTools as a Java library, but it is also available separately. GeoServer is packaged as a standalone servlet for use with existing application servers such as Apache Tomcat and Jetty.

GeoServer allows for the display of spatial information to the world. Implementing the Web Map Service (WMS) standard, GeoServer can create maps in a variety of output formats. OpenLayers, a free mapping library, is integrated into GeoServer, making map generation quick and easy. GeoServer is built on Geotools, an open source Java GIS toolkit.

There is much more to GeoServer than nicely styled maps, though. GeoServer also conforms to the Web Feature Service (WFS) standard, which permits the actual sharing and editing of the data that is used to generate the maps. Others can incorporate our data into their websites and applications, freeing our data and permitting greater transparency.

Through years, GeoServer became more powerful and continues to make spatial data more accessible to everyone. For example, it can now connect to a free and open spatial database such as PostGIS and through the implementation of OpenLayers (an open source browser - based map viewing utility), the functionality of GeoServer has become more powerful than ever.

GeoServer can be installed as a service or can run manually. When run manually, GeoServer is run like a standard application under the current user. When installed as a service, GeoServer is integrated into Windows Services, and thus is easier to administer. If running on a server, or to manage GeoServer as a service, the right selection is to be run as a service.

It has a very friendly user interface (GUI) which allows users to manage data easily and perform actions such as layer preview, styling and layer grouping.

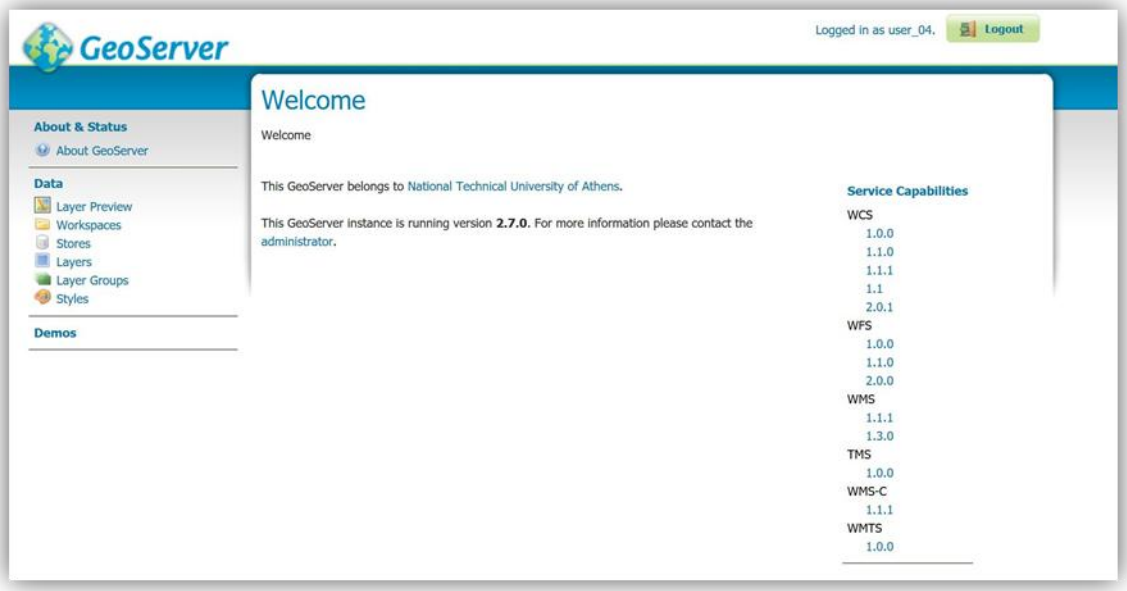

Figure 3.1: The interface of GeoServer

While installing, it is required to enter the port that GeoServer will respond on. This affects the location of the GeoServer Web administration interface, as well as the endpoints of the GeoServer services such as Web Map Service (WMS) and Web Feature Service (WFS). The default port is 8080, though any valid and unused port will work.

As mentioned above, one of the major advantages of GeoServer is the fact that has a very handy user interface which makes it easy for beginners to manage their data.

In order to start working with GeoServer, a workspace has to be created. A workspace is an iconic container which helps organizing projects. In GeoServer, a workspace is often used to group similar layers together.

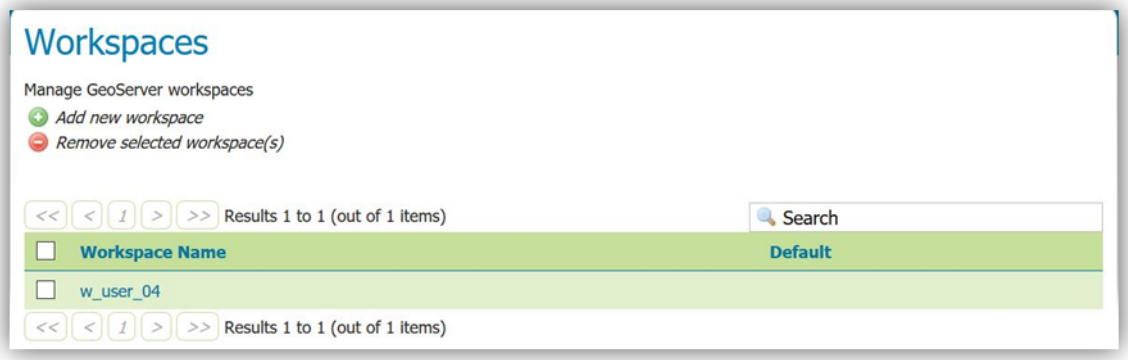

Figure 3.2: The Workspace environment of GeoServer

Separate workspaces can be added, one for every different project that are worked on GeoServer.

A store is also needed to connect with the data source that contains the vector or raster data. A data source can be a file or group of files, a table in a database, a single raster file or a directory. As we can see below, there are many different data sources that the GeoServer connects to as well as publishes, divided in vector and raster data sources.

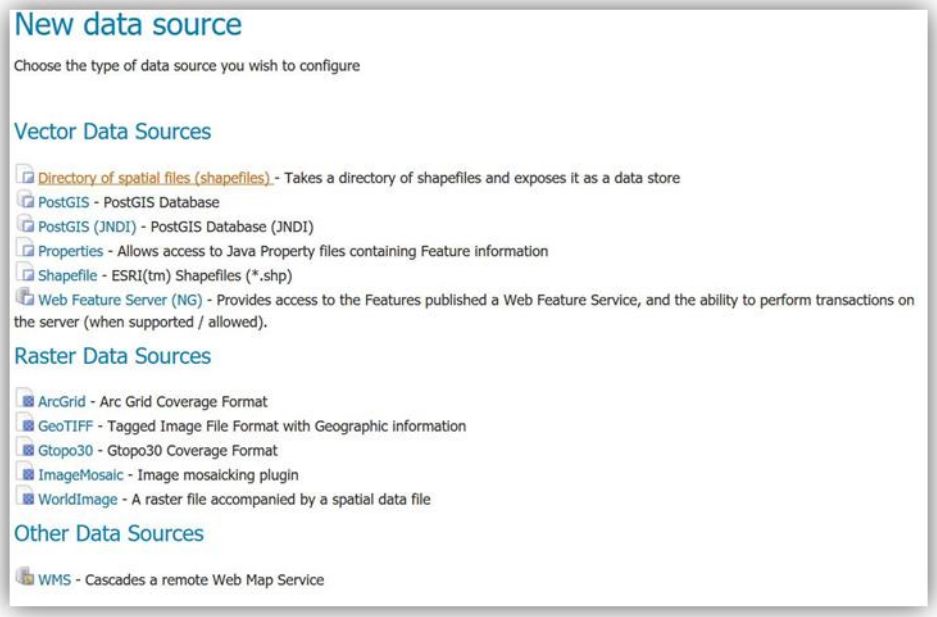

Figure 3.3: The connected data sources

Notice that if someone wants to add raster data, a store for each of the raster data has to be created. As a result, multiple stores can be created. The stores are referred to the workspace used.

In GeoServer, the term “layer” refers to a raster or vector dataset that represents a collection of geographic features. Layers must be published to GeoServer in order to be viewed. When editing a layer, its SRS (Source Reference System) as well as its boundaries must be determined.

The list below shows how layers are shown within GeoServer.

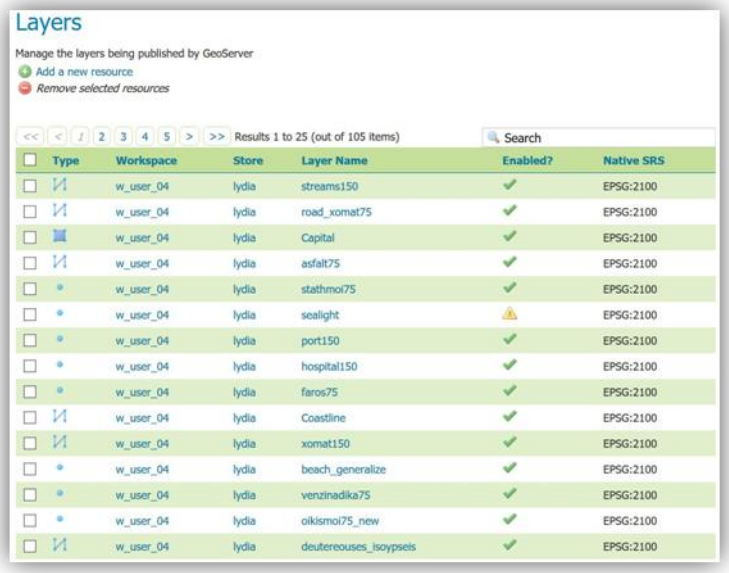

Figure 3.4: The layer menu

Layers can be of four types; point, line, polygon and raster (grid). If someone wants to preview layers together as a map, he has to put them in the same *layer group* at a particular order, depending on how he wants them to be visualized.

There are several ways to publish or preview data from GeoServer. In order to preview data, GeoServer allows preview in a wide variety of output formats. Layers must be enabled before the preview occurs.

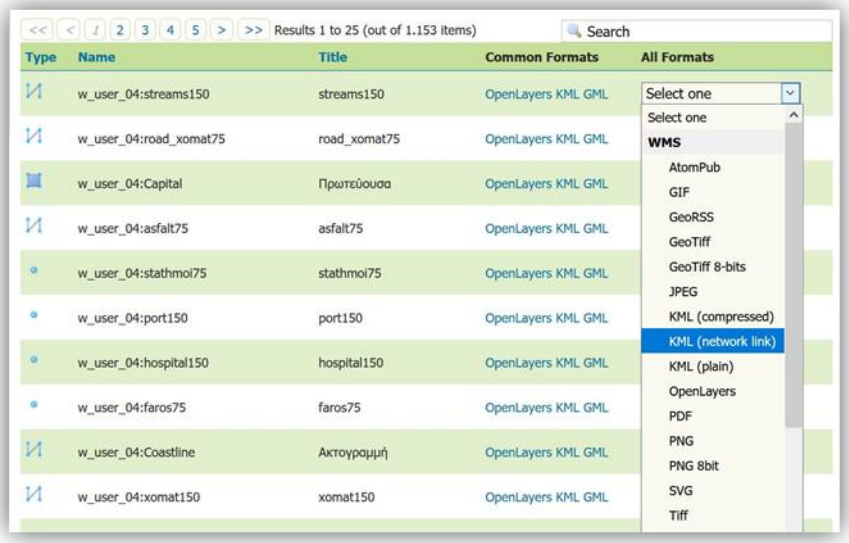

Figure 3.5: Possible output formats

As it can be seen, there are many available output formats. Most common are OpenLayers, KML, GML, JPEG, TIFF/GeoTIFF, PDF, PNG, SVG, CSV, GeoJSON, GIF.

3.2.2 Styling

Geospatial data applied in GeoServer cannot be visualized without a style. So, a style is applied so as data can be displayed.

Styling is a major issue for GeoServer and can be applied in several ways. Most common is the SLD (Styled Layer Descriptor). This is an XML-based markup language and is very powerful, although somewhat complex. Therefore, SLD uses tags, such as HTML or XML.

SLD is specified by the Open Geospatial Consortium (OGC) for describing the appearance of map layers. It is capable of describing the rendering of vector and raster data. A typical use of SLDs is to instruct a Web Map Service (WMS) how to render a specific layer. SLDs are basically XML documents and can be very simple, i.e. describing a simple point layer or can be very complex, i.e. describing a layer that has many attributes and each attribute has to be visualized in a different way than the others. Thus, an SLD document can be very long, containing many lines.

In more complex SLDs one or more **rules** can be applied; rules control how styling is applied based on feature attributes and zoom level. In addition, rules select applicable features by using **filters**, which are logical conditions containing **predicates**, **expressions** and **filter functions**. To specify the details of styling for individual features, rules contain any number of **symbolizers**. Symbolizers specify styling for **points**, **lines** and **polygons**, as well as **rasters** and **text labels**. An SLD is created for every layer we want to publish in GeoServer and it has to be associated with it.

A typical example of a simple SLD document which represents a simple line layer is shown below.

```
<FeatureTypeStyle>
  <Rule>
    <PointSymbolizer>
      <Graphic>
        <Mark>
          <WellKnownName>triangle</WellKnownName>
          <Fill>
            <CssParameter name="fill">#009900</CssParameter>
            <CssParameter name="fill-opacity">0.2</CssParameter>
          </Fill>
          <Stroke>
            <CssParameter name="stroke">#000000</CssParameter>
            <CssParameter name="stroke-width">2</CssParameter>
          </Stroke>
        </Mark>
        <Size>12</Size>
      </Graphic>
    </PointSymbolizer>
  </Rule>
</FeatureTypeStyle>
```

Figure 3.6: Example of an SLD file

As we can see, numerous tags can be applied, depending on what we want to show. Most important tags are described below:

•*Point Symbolizer*: This tag is used to declare the styling of point data. In point symbolizers *well known symbols* can be applied (circle, triangle, star etc) or *graphics*.

Well known symbols can be furthermore specified through *Fill* tag where the color in hexadecimal string and the opacity are declared. *Stroke* tag is also applied to determine the outline and its width. Finally, *Size* is used to determine the size of the symbol in pixels.

Graphics use a graphic instead of a simple shape to render the points. In SLD, this is known as an `<ExternalGraphic>`, to distinguish it from the commonly-used shapes such as squares and circles that are "internal" to the renderer. The path and the file are needed, as well as its format (PNG, SVG etc).

•*Line Symbolizer:* For linear symbols the *color* and the *width* in pixels are initially needed. It is also possible to draw a dashed line, an offset line, line with an outline and more complex symbols that involve graphics as well as labels.

•*Polygon Symbolizer:* It is similar to the Line Symbolizer. By using *Fill*, color is specified and by *stroke* the outline and width are specified.

•*Raster Symbolizer:* Raster Symbolizer is the simpler of all mentioned above. It uses the *ColorMap* tag to specify the color for each range of pixel values.

•*Text Symbolizer:* This tag is used for data labeling. Text labels are positioned either at points or along linear paths derived from the geometry being labeled. Labeling is a complex operation, and effective labeling is crucial to obtaining legible and visually pleasing cartographic output. For this reason SLD provides many options to control label placement. A text symbolize contains the following elements:

i. *<Geometry> :* Specifies the geometry to be labeled (point or line).
ii. *<Label>:* The text content for the label.
iii. *<Font>*: The font information for the label.
iv. *<LabelPlacement>* : Sets the position of the label relative to its associated geometry.
v. *<Halo> :* Creates a colored background around the label text, for improved legibility.

vi. *<Fill>* : The fill style of the label text.

Additional tags are often used to improve the display of the data such as:

- **Rules and filters,** which are used to assign an attribute-based styling to the objects of the same layers (for example national, local and suburban road).
- **Opacity** is used to make a layer transparent (values from 0 to 1, with 1 being 100% transparent), and
- **Min** and **Max Denominators**, which are used to specify the scale range in which the data are desirable to be displayed.

Other styling options available in GeoServer are YSLD (an SLD extension), CSS styling which uses a CSS-derived language instead of SLD and MapBox Style Specification, all available through extensions.

Another element that is very important for the map display is *antialiasing.* Anti-aliasing is the software technique for smoothing the jagged appearance of curved or diagonal lines caused by poor resolution on a display screen. Methods of anti-aliasing include surrounding pixels with intermediate shades and manipulating the size and horizontal alignment of the pixels. This technique is very important especially for the annotation as well as for the symbols of a map. GeoServer supports antialiasing and this helps the visual result of the map to be of a better quality and also adds more realism to the map image (Tsoulos & Skopeliti).

Additionally, as mentioned above, GeoServer has a component named GeoWebCache. GeoWebCache is used to add tiles to the maps created by GeoServer. When a map is published, it is published as a raster file (JPEG, PNG etc). These types of data usually have a big size and therefore the drawing of the image may slow down, especially when the map contains many layers with complicated symbolism. In order to fix this, map tiles. With this technique, the image “breaks” in pieces so as they can be drawn faster, as they have now a smaller size. This technique is very useful as it improves the speed of the image rendering at the server, but also can cause a problem at annotation labels. If annotation labels are placed near to the “breaking” of the image, at the redrawing they might appear as duplicates.

3.2.3 OGC Services

GeoServer serves data using standard protocols established by the Open Geospatial Consortium. Most common of these protocols are:

- The **Web Map Service** (WMS), which supports requests for map images (and other formats) generated from geographical data. GeoServer supports WMS 1.1.1, the most widely used version of WMS, as well as WMS 1.3.0.
- The **Web Feature Service** (WFS), which supports requests for geographical feature data (with vector geometry and attributes) and
- The **Web Coverage Service** (WCS), which supports requests for coverage data (rasters).

3.3 ArcGIS Server

3.3.1 General

ArcGiS Server is a proprietary software that makes the geographic information available to anyone who is using an Internet connection. As with open source servers, data are available through web services, which allow a powerful server computer to receive and process requests for information sent by other devices.

ArcGIS Server is available for the Microsoft Windows .NET Framework and the Java Platform. ArcGIS Server ships in three functional editions, Basic, Standard, and Advanced, with the Advanced edition providing the most functionality. ArcGIS Server Basic edition is used primarily to manage multiuser geodatabases and geodata services. Both ArcGIS Server Standard and Advanced editions support the following types of Web services: Feature (for Web editing), Geodata (for geodatabase replication), Geocode (for finding and displaying addresses/locations on a map), Geometry (for geometric calculations such as calculating areas and lengths), Geoprocessing (for scientific modeling and spatial data analysis), Globe (for 3D and globe rendering), Image (for serving raster data and providing control over imagery delivery, such as satellite imagery or orthophotos), Keyhole Markup Language (KML), Map (for cached and optimized map services), Mobile (for running services on field devices), Network Analyst (for routing, closest facility location, or service area analysis), Search (for enterprise search of GIS assets), Web Coverage Service (WCS), Web Feature Service (WFS) and Transactional Web Feature Service (WFS-T), and Web Map Service (WMS).

In addition, ArcGIS Server editions are available at two levels, scaled according to capacity: Workgroup and Enterprise. ArcGIS Server Workgroup can be deployed on a single machine to support a maximum of 10 simultaneous connections to a multiuser geodatabase. With Workgroup, the multiuser geodatabase storage capacity cannot exceed ten gigabytes. ArcGIS Server Enterprise supports distributed deployment of ArcGIS Server components, unlimited simultaneous connections to a multiuser geodatabase, and unlimited multiuser geodatabase storage capacity.

ArcGIS Server is also used to manage multiuser geodatabases. Multiuser geodatabases leverage ArcSDE technology, implemented on a relational database management system (RDBMS). ArcGIS Server Enterprise supports IBM DB2, IBM Informix Dynamic Server, Microsoft SQL Server, Oracle, and PostgreSQL. ArcGIS Server Workgroup supports Microsoft SQL Server Express R1 and R2.

ArcGIS Server is used by the software developer and Web developer to create Web, desktop, and mobile applications. ESRI provides developers with application development framework (ADF) and application programming interface (API) including, ArcGIS API for JavaScript, ArcGIS API for Flex, ArcGIS API for Microsoft Silverlight/WPF, ArcGIS API for iOS, BAO API, BAO for iOS, as well as the ArcGIS Mobile software development kit (SDK), and ArcGIS Server REST and SOAP APIs.

In order to publish services, various components of ArcGIS are to be used. For example, if you want to publish a map, this map will be created with ArcMap. The connection with the data is the simplest it could be. Via ArcCatalog, the connection with the folder that contains the data or the geodatabase is possible.

3.3.2 ArcGIS Server Architecture and Components

As mentioned above, a server's task is to receive requests to the services, fulfill them and send the results back to client applications. ArcGIS Server provides a set of tools that allows managing the services; for example, the ArcGIS Server Manager can be used in order to add and/or remove services.

The architecture of an ArcGIS Server site is presented below:

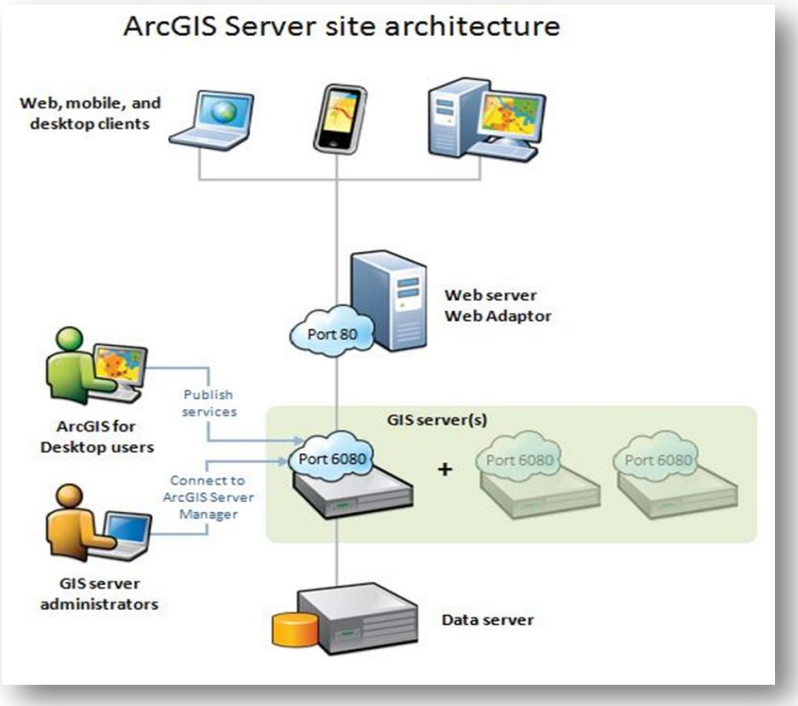

Figure 3.7: ArcGIS Server Architecture (ESRI)

As it can be seen from this figure, the components of ArcGIS Server site are mainly five;

• **ArcGIS Server** - The ArcGIS Server machine fulfills the requests to the web services. It draws maps, run tools, queries data and performs any other action that can be done with a service. The ArcGIS Server can consist of one machine or many machines working together. The machines all have access to the same data and configuration information, so the number of participating machines can be grown or shrinked on demand.

The ArcGIS Server machine exposes the services through the common web protocol HTTP. Once ArcGIS Server is installed, there is the possibility of using a set of web services in the apps.

• **Web Server** – A web server can host web applications and provide optional security and load balancing benefits to the ArcGIS Server site.

• **Data Server** – Data can be placed directly on each GIS server, or be accessed it from a central data repository, such as a shared network folder or an enterprise geodatabase. This service is optional.

Apart from the software, there is the need of the human factor in order for the system to function. So, there are people who author the data, maintain the services and finally, use the services. Regarding the human factor, there is some types of people components which are presented below;

• **ArcGIS Server site administrators** – These people install the software, configure web applications and tune the site for the best performance. They can also use ArcGIS Desktop or ArcGIS Server Manager to administer the site.

• **ArcGIS Desktop content authors and publishers** - The GIS resources that are going to be published to the site, such as maps and geodatabases are created by ArcGIS Desktop content authors and publishers.

• **Client application users** – Web, mobile and desktop applications can connect to services. The end users of these applications depend on the ArcGIS Server for data and analysis.

3.3.3 Main functionalities of ArcGIS Server

As part of a server-based GIS, ArcGIS Server offers a big variety of tools, which make it a really powerful software, not only for publishing web maps but also for performing analysis on geographic data. Most basic operations of ArcGIS Server are presented below:

• **Web Service Publishing –** As soon as ArcGIS Server is installed, web services from ArcGIS such as maps, imagery and geoprocessing models can be published. These services can be exposed through REST[4] and SOAP[5] and can be invoked by ESRI and non-ESRI clients.

• **GeoAnalytics tools –** This is a geoprocessing service which is used to support big data analysis.

• **Geometry Service –** This service can perform geometric calculation such as buffering, simplifying, calculating areas and lengths and projecting.

• **Raster Analysis Tools –** This service supports distributed raster analysis in ArcGIS Image Server.

• **Spatial Analysis Tools –** This tool is used to perform spatial analysis. The server does the work of processing analysis requests, storing the results in ArcGIS Data Store and returning results to members in the Portal for ArcGIS website.

---

[4] **REST,** or REpresentational State Transfer, is an architectural style for providing standards between computer systems on the web, making it easier for systems to communicate with each other. REST-compliant systems, often called RESTful systems, are characterized by how they are stateless and separate the concerns of client and server. We will go into what these terms mean and why they are beneficial characteristics for services on the Web.

[5] **SOAP** (originally **Simple Object Access Protocol**) is a messaging protocol specification for exchanging structured information in the implementation of web services in computer networks. Its purpose is to induce extensibility, neutrality and independence. It uses XML Information Set for its message format, and relies on application layer protocols, most often Hypertext Transfer Protocol (HTTP) or Simple Mail Transfer Protocol (SMTP), for message negotiation and transmission.

3.3.4 OGC Web services

With the term “service” we are referring to a server which takes the data or the GIS resources, such as a map, tool or even a geodatabase and makes it available (shares it) to a wider audience through ArcGIS Server. In order for this to be achieved, other parts of ArcGIS are involved, such as ArcMap or ArcCatalog. The way for the service to be published, depends on the GIS resource type.

ArcGIS Server supports a wide variety of services to be published, but at this thesis we are focused on the *OGC* Services, analyzed on previous chapter.

As mentioned before, Open Geospatial Consortium, Inc. (OGC), web services provide a way that maps and data can be made available in an open, internationally recognized format over the web. OGC has defined specifications for making maps and data available on the web to anyone with a supported client application. All developers are free to use the OGC specifications to create these supported clients. In some cases, the client can be as simple as a web browser. In other cases, it can be a rich client such as ArcMap.

ArcGIS Server allows several kinds of services to be published through ArcGIS Desktop.

• **Web Map Service (WMS)** – This service is used for serving collections of layers as map images. This is useful if someone wants to make maps available online in an open, recognized way across different platforms and clients. Any client built to support the WMS specification can view and work with the service. Four versions of the WMS specification have been published so far. They *are v1.0.0, v1.1.0, v1.1.1, and v1.3.0* (most recent).

Client applications work with a WMS service by appending parameters to the service's URL. WMS services published to ArcGIS Server support the following operations:

- Requesting metadata about the service (GetCapabilities)
- Requesting a map image (GetMap)
- Requesting information about features in the map (GetFeatureInfo [optional])
- Requesting user-defined styles (GetStyles)
- Requesting legend symbols (GetLegendGraphic)

<u>Publishing a WMS service</u>

There are two ways for a WMS service to be published:

- Publish a map service with the WMS capability enabled. To publish a map service, you first need to create a map document.
- Publish an image service with the WMS capability enabled. To publish an image service, you need to have a raster dataset, a mosaic dataset, or a layer file referencing a raster dataset or mosaic dataset.

The number of layers in the map directly affects the amount of time it takes to initially create the WMS service or start it after it has been stopped. The startup time can be reduced by removing or consolidating layers in the service.

WMS services take advantage of a map service cache if one is available. The map service cache is only used when requesting the original projection, layer order, and layer visibility of the service. The cached tiles are resampled to fit the scale requested by the client, which can take some processing time and reduce the image quality. The benefit using a cache can be seen when a WMS service has many layers or sophisticated symbology that would otherwise take a long time to draw dynamically. If there are just a few layers and uncomplicated symbology, better performance without a cache can be get.

• **Web Map Tile Service (WMTS) -** Web Map Tile Service (WMTS) specification is an international specification for serving digital maps over the web using cached image tiles. When a cached map or image service is created using ArcGIS Server, the service and its tiles are automatically accessible using the WMTS specification. Clients built to support the *WMTS 1.0.0* specification and RESTful or KVP encoding can view and work with WMTS services. SOAP encoding is not supported.

• **Web Feature Service (WFS) -** This is an open specification for serving geographic features over the web. Serving your data through a WFS service allows any application that can work with web services to access geographic features from your map or enterprise geodatabase. The WFS services you create are compliant with the WFS 1.1.0 and 2.0 specifications. They also support the WFS 1.0.0 specification on a read-only basis.

WFS services use Geography Markup Language (GML) to encode the feature data. GML is just a way of using XML to represent geographic information. The GML used by ArcGIS Server WFS services uses the Simple Features profile.

• **Web Coverage Service (WCS) –** It provides an open specification for sharing raster datasets on the web. ArcGIS Server allows publishing WCS services from imagery collections, maps, or geodatabases that contain rasters. Any client built to support the WCS *1.0.0, 1.1.0, 1.1.1, 1.1.2, and 2.0.*1 specifications can view and work with this service. WCS 2.0.1 services also support the following OGC extensions:

- Service extensions : Scaling, Interpolation, Range Subsetting, and CRS

- Protocol extensions: KVP/Get and XML/Post
- Format encoding extensions: GeoTIFF

• **Web Processing Service (WPS) -** It is an international specification for serving and executing geospatial processing on the web. Client applications work with a WPS service by appending parameters to the service's URL. WPS services published to ArcGIS Server support the following mandatory operations:

- Requesting metadata about the service (GetCapabilities)
- Requesting detailed information about the processes that can be run on the service (DescribeProcess)
- Requesting to run a process implemented by the WPS service (Execute)

Other types of services that can be published through ArcGIS Server except for OGC Services are KML, Schematics, Vector Tile, Geometry, Geoprocessing, Network Analysis etc services.

3.3.5 Sharing maps through web using ArcGIS Server – How It Works

At the very moment the map is ready, it can be published as service using ArcGIS Server. Because ArcGIS Server is proprietary software, is much easier to use than similar open source software. That happens because all the required codes at proprietary software are embedded and hidden, so the users find them ready to use and they don't need to write down their own code, which may contain bugs or even doesn't work at all.

It is widely accepted that using open source software can be difficult, especially when the user has no prior experience on programming. Thus, ArcGIS Server might be an ideal solution for beginners on web mapping, although proprietary software costs a lot.

Publishing a web map through ArcGIS Server is very different than using GeoServer or MapServer. First of all, there is no need of using the server for the map composition. The map synthesis is made on ArcGIS Desktop, more specifically on ArcMap. ArcCatalog is used to obtain connection with the data.

Figure 3.8 shows the operations that ArcMap is responsible for as well as the ArcGIS Server responsibilities, to help us understand the differences between the operations they achieve:

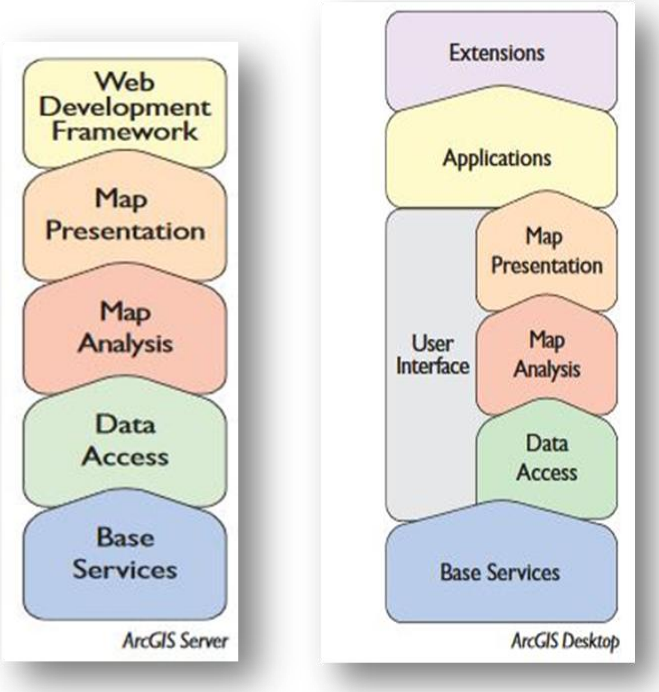

Figure 3.8: Differences between ArcGIS Server and ArcGIS Desktop (enterprise.arcgis.com)

Once the map is ready to be published, the connection with the server is made by making the following simple steps:

- Open the ArcCatalog and choose *Add ArcGIS Server*
- A window opens and asks what connection you want (publish, add or administer ArcGIS services)
- A next window appears so as to fill out the connection parameters, as follows:

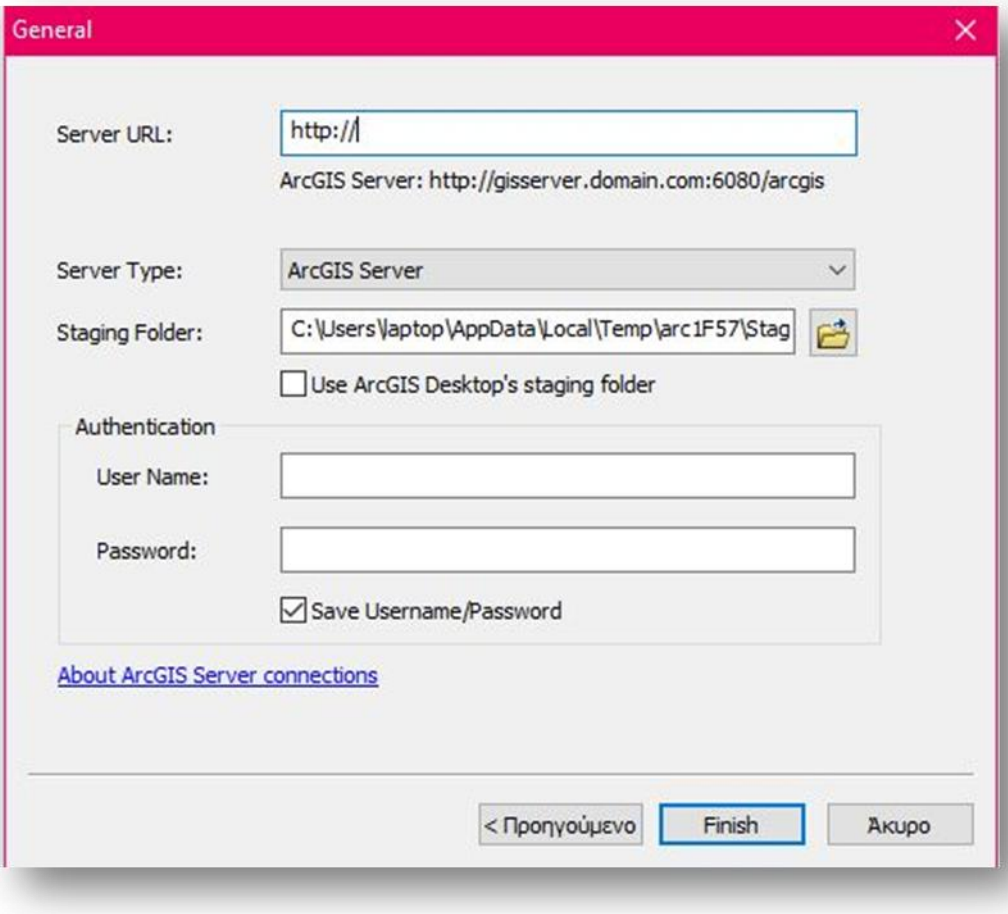

Figure 3.9: Connection parameters

The connection parameters are the server URL and username and password to obtain access to the server. In our case the server URL is *http://aris.survey.ntua.gr:6080/arcgis/* which is located at National Technical University of Athens.

The interface of ArcGIS Server is simple and easy to use, even for a beginner. As it can be seen, it has a friendly environment which has what is needed for publishing a web map through the internet. As it can be seen from the figure that follows, the site of the server shows where the data is stored, what saved projects are there and how to publish the service we want.

Figure 3.10: ArcGIS Server interface

Once a service is published it can be consumed in web, desktop or mobile applications. Services have web addresses or URLs that client applications use to access them. When an application is used or developed, the URLs of the services that we want to use must be provided. The application goes to the URLs and begins working with the services to bring in the maps that are requested.

3.3.6 Symbology

While regarding to open source software such as GeoServer and MapServer, symbology is a very complicated procedure that demands knowledge of programming and at some cases (GeoServer) knowing the basics of XML, with ArcGIS Server everything is simpler. The step of data symbology is made on ArcMap, before data is uploaded to ArcGIS Server.

In ArcMap, styling the data can be done just by choosing the symbol we want from a very wide list of symbols available. Then, the chosen symbol can become bigger or smaller or even can be combined with another (for more complex symbols, such as lighthouses).

3.3.7 Other approaches for publishing services with ArcGIS

Except for publishing services with ArcGIS Server, ArcGIS offers two more approaches for sharing information with others through web services. That gives the flexibility to deploy our services on a server whose size, scope and cost most closely match the corresponding needs. The other two options are:

• ArcGIS Online

• Portal for ArcGIS (using ArcGIS Server as a back end).

- ArcGIS Online

With this option, it is possible to publish GIS web services to an ESRI-administrated cloud environment. The difference here is that there is no need to install anything on the computer. The only thing needed is to sign in to an ArcGIS Online account. There are two types of services to be deployed:

- Feature services expose the geometry, attributes and symbol information for vector GIS features. They are appropriate for displaying, querying and editing data on top of web basemaps.
- Tiled map services, which expose a set of pregenerated map images (known as a map cache) that can be viewed as basemaps in a web mapping application. When the service is published, it is possible to create and store a cache of tiles. This cache of tiles can be brought into the web map by accessing the service's URL.

Using a combination of tiled map services and feature services in the application allows fast mapping while supporting query and editing operations.

The main advantage of ArcGIS Online is that there is no need for installing any server software or tune the services. The services run in an ESRI-administered cloud environment in which the server automatically scales up to meet demand.

- Portal for ArcGIS

The Portal for ArcGIS offers the same tiled map services and feature services as ArcGIS Online. The difference is that the portal has to be installed on our own network infrastructure. Our own ArcGIS Server implementation has to be connected to the portal to act as the web service hosting engine.

Portal for ArcGIS requires more setup than ArcGIS Online, but it is an appropriate choice for someone who is not connected to the internet, cannot send data off-premises or need full control over the hardware running the portal.

This approach allows a broad segment of users to publish feature services and tiles map services, while leaving the administration of more advanced services (e.g. geoprocessing services) to a narrower group of ArcGIS Server publishers.

3.3.8 Other capabilities

ArcGIS has its own web mapping API which is used for building the most demanding web applications. This is the proprietary equivalent of OpenLayers and Leaflet and it is called *ArcGIS API for Javascript*. It can create 2D or even 3D maps in one application. It is ideal for sophisticated and complicated data visualization, and this is an element that is not applied on open source libraries, such as OL and Leaflet.

Another difference between ArcGIS API for Javascript and the other open source web mapping APIs is that its code cannot be modified. This API is powerful; performance has been enhanced in the ArcGIS API for JavaScript to enable 3D scenes in the browser of a smartphone without a plugin. All the end users need is a URL to the app and they can then interact with the data in 3D.

Developers can build full-featured 3D applications powered by Web Scenes that can include rich information layers such as terrain, basemaps, imagery, features, and 3D objects that can be streamed via tile, feature, image, and scene services.

The ArcGIS API for JavaScript is available to developers in several ways. The most common way is to use the CDN version (by referencing the API which is on a hosted version).

Some of the most impressive operations this API can perform are:

- Query statistics client-side
- Request data from remote server
- Point styles for 2D as well as 3D buildings
- Switch view from 2D to 3D
- Project points of a CSV file on-the-fly
- Display point-cloud layers as well as change their density
- Geoprocessing analysis
- Look around camera positions, etc

It can be noticed that these operations go beyond classic 2D web mapping and focus on the visualization of any kind of data and especially 3D visualizations. These visualization tools concern of course classic cartography maps but they can also be applied to any kind of data that can be visualized in some way, for example statistic data or weather data.

3.4 MapServer

At this subsection, MapServer is presented and the way it works is described. It is analyzed the way it serves data and how they are symbolized as well as its main functionalities.

3.4.1 General

MapServer[6] is another open source development environment for building spatially enabled internet applications. It can run as a CGI[7] application with Apache and Microsoft web servers or via MapScript[8] which supports several programming languages. Its scope is primarily to display dynamically enabled spatial maps over the Internet. It was developed by the University of Minnesota — therefore it is often and more specifically referred as "UMN MapServer", to distinguish it from commercial "map servers". Originally, MapServer is released under an MIT-style license, and runs on all major platforms (Windows, Linux, Mac OS X). MapServer is not a full-featured GIS system, nor does it aspire to be.

MapServer is one of the founding projects of the OSGeo foundation, and is maintained by a growing number of developers (nearing 20) from around the world. It is supported by a diverse group of organizations that fund enhancements and maintenance, and administered within OSGeo by the MapServer Project Steering Committee made up of developers and other contributors. All source code is openly available via GitHub. However, it does not have the friendly user interface that GeoServer has. Hence, it is much more difficult for someone to use, especially if he does not have a similar experience.

3.4.2 Map Server Architecture

MapServer's architecture is a lot different than GeoServer's. In its simplest form, a MapServer application consists of several files that all together build an integrated result.

These files are:

---

[6] http://mapserver.org/

[7] **Common Gateway Interface** (**CGI**): offers a standard protocol for web servers to execute programs that execute Console-like applications (also called Command-line interface programs) running on a server that generates web pages dynamically. Such programs are known as *CGI scripts* or simply as *CGIs*. The specifics of how the script is executed by the server are determined by the server. In the common case, a CGI script executes at the time a request is made and generates HTML.

[8] **MapScript:** This is language agnostic documentation for the MapScript interface to MapServer generated by SWIG. This document is intended for developers and to serve as a reference for writers of more extensive, language specific documentation located at Mapfile.

• ***Map File (.map*).** Map File is a structured text configuration file. This file contains information about the layers of the map, the coordinate system, the data symbology, the data source, the extent and the units of the map. In order to be recognizable from MapServer, it must have a *.map* ending.

• ***Geographic data*.** Data of various formats can be recognized by MapServer, most used is the ESRI shapefile format. However, a wide range of data formats are supported both for vector and raster data.

• ***HTML files***. These files are the interface between the user and MapServer. Interactive maps must be placed in an HTML form on a page. A simple MapServer CGI application may include two HTML pages:

- *Initialization File*. This file uses a form with hidden variables to send an initial query to the web server and MapServer.

- *Template File.* This file is almost always an HTML file, although it can also be a URL that contains special characters that are replaced by *mapserv* each time the template is processed. Templates are used to define the look of the MapServer CGI application interface, create a customized output and present the results of a query. Also, simple pan/zoom interfaces use a single template file while complicated queries often require many templates.

• ***Web/HTTP server***. It serves up the HTML pages when hit by the user's browser.

• ***MapServer CGI***. This is the binary or executable file that receives requests and returns images, data etc. It is located in the *cgi-bin* directory of the web server. By default, this program is called *mapserv.*

The architecture of a MapServer application is shown aggregated below:

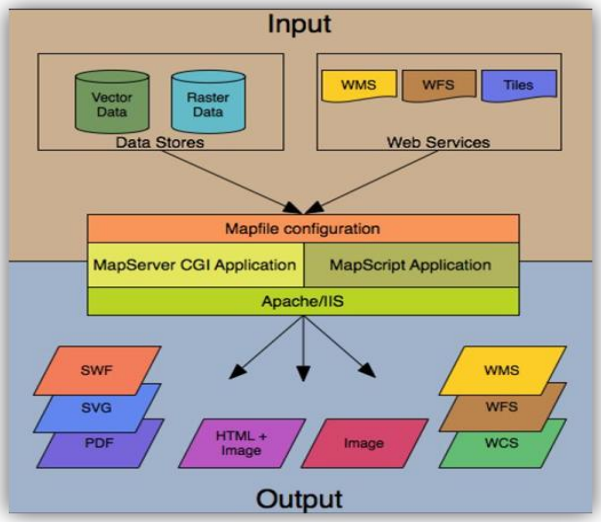

Figure 3.11: Anatomy of a MapServer application, source: MapServer Documentation, 2017

3.4.3 The MapFile

MapServer's most basic component is the *Map File.* Map File or *Mapfile* is a file with the *.map* extension which contains all the important information that the server needs in order to produce /draw the requested map, such as styling and data access.

This file comes with a *.map* extension and it can easily be created by opening a text editor (e.g *Notepad++*). It is an ASCII text file and made up of different objects, each one of them has a variety of parameters available for it.

Mapfile follows the principles of object-oriented programming, as it focuses on objects which have attributes, classes, subclasses etc. It has a hierarchical structure in which some objects contain some others and inherit their attributes.

On top of the hierarchy lies the MAP object, where all the other objects are contained. Therefore, every mapfile starts with the word MAP and every section ends by writing the word END.

The hierarchical structure of a mapfile is shown below:

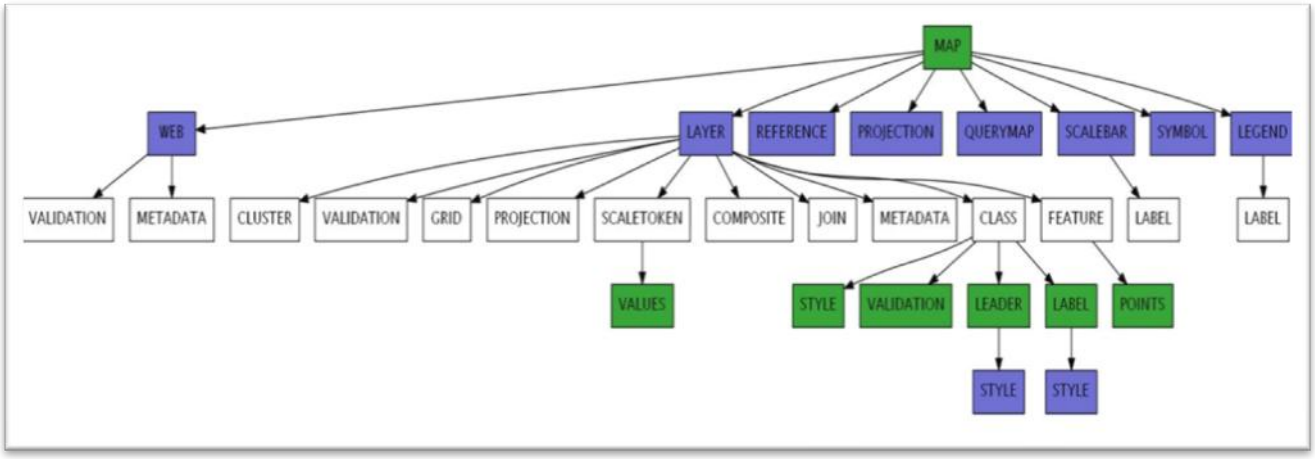

Figure 3.12: Mapfile structure, Source: MapServer Documentation, McKenna et al, 2017

Simple maps with a few layers and simple symbology do not include all of these elements. Of course, most of them are mandatory, such as the LAYER element, the PROJECTION element and the WEB element. More complicated maps built for cartographic purposes need a complex symbolization and therefore a mapfile can be very long and can contain many embedded elements.

Below is a typical example of a very simple mapfile:

```
MAP
    NAME "sample"
    STATUS ON
    SIZE 600 400
    SYMBOLSET "../etc/symbols.txt"
    EXTENT -180 -90 180 90
    UNITS DD
    SHAPEPATH "../data"
    IMAGECOLOR 255 255 255
    FONTSET "../etc/fonts.txt"

    #
    # Start of web interface definition
    #
    WEB
        IMAGEPATH "/ms4w/tmp/ms_tmp/"
        IMAGEURL "/ms_tmp/"
    END # WEB

    #
    # Start of layer definitions
    #
    LAYER
        NAME 'global-raster'
        TYPE RASTER
        STATUS DEFAULT
        DATA bluemarble.gif
    END # LAYER
END # MAP
```

Figure 3.13: Example of a mapfile

The mapfile can be placed everywhere, as long as it is accessible to the web server. As it can be seen, it consists of many different objects, which are described below:

• **MAP Object.** This is the main body of the mapfile. Contains general information, mostly regarding the output. These are:

- *SIZE.* Specifies the size of the map box in pixels.
- *EXTENT.* This parameter defines the boundaries (in coordinates) of the map are defined (bottom left and upper right corner - [minx] [miny] [maxx] [maxy]).
- *SHAPEPATH.* Here the path in the server is defined where input data (i.e. shapefiles) is stored.
- *IMAGECOLOR.* The color of the background of the map in RGB values or as hexadecimal string.
- *UNITS.* The units of the map regarding the projection, e.g. meters.
- *IMAGETYPE.* Format of the output map (jpeg, png).

• **Projection**. In order to display data in the right way, mapfile needs a definition of the data coordinate system. Projection can be declared with two ways, for example

if you wish to use Greek Grid, you can declare it by adding all the parameters of the projection[9] as

```
PROJECTION "proj=tmerc"
"lat_0=0"
"lon_0=24"
"k=0.9996"
"x_0=500000"
"y_0=0"
"ellps=GRS80"
"towgs84=-199.87,74.79,246.62,0,0,0,0"
"units=m"
"no_defs"
  END
```

or by simply using the proper EPSG[10] code:

```
PROJECTION
            "init=epsg:2100"
  END
```

• **WEB Object.** The WEB element contains the metadata needed to publish the data as service. Metadata also are needed on each LAYER element, respectively.

• **LAYER Object**. Here lies information about the data of the layer, its name and its type (vector, raster). It also provides the path to the data.

Within the **LAYER** Object, the following parameters are available:

- ✓ *NAME.* The name of the layer is declared here.
- ✓ *TYPE*. The kind of the data of the layer (point, line, polygon, raster, annotation)
- ✓ *STATUS.* On/off status shows whether the layer is visible or not. Default means that LAYER is always displayed.
- ✓ *DATA.* Here is cited the name of the shapefile (the .shp extension is not necessary)
- ✓ *CLASS.* This is a subsection of the LAYER object as mentioned above. Here the style is applied (color, outline color etc) for every layer and expressions used to differentiate objects of the same layer with different attributes.

[9] http://spatialreference.org
[10] www.epsg.io

• ***CLASS and STYLE Objects.*** Here information about the styling of the layer is stored s, where the symbol, its color and its size are defined. Symbols can be stored in a separate file than the mapfile or be defined directly in the mapfile. When the separate file is called by the mapfile the SYMBOLSET parameter in the MAP Object must be used.

Classes are used to style same layer's data differently regarding a specific attribute, i.e. primary roads and secondary roads. ***LABELs*** also defined within the class object are used to place a label and can be customized by size, font, color and position. Comments can be included by simply using the "#" symbol before writing. All objects, including LAYER, CLASS and STYLE end with the keyword END.

MapServer also supports antialiasing, just like GeoServer and ArcGIS Server do. In order to enable antialiasing, it is necessary for every layer to add ANTIALIASING TRUE. This is important especially for the annotation (labels) of each map because it smoothens the edges of the letters and makes the map look prettier. Finally, it also supports map tiling by using *CGI's Tile Mode* for creating map tiles for a faster image rendering.

Some other very useful functionalities that a mapfile has are the ability to create a map legend specified by the evolving layers, a reference map as well as a scale bar.

- A *Legend* can be created by using the LEGEND Object and specify the color, the size and the format. The name of the layers in the legend is specified in NAME subclass of LAYER Object.
- A *Reference map* is also very easy to draw by using the REFERENCE Object and specify some parameters like the path to the image, the extent, the color of the square and the size.
- A *Scalebar* is drawn same way, by using the SCALEBAR Object and specify parameters such as the color, the label, the units and the interval.

3.4.4 Styling – Symbology

MapServer offers a wide variety of complex cartographic symbolizations for layers mostly by multiple rendering and overlay. The method used to visualize layers in MapServer is completely different than the symbology applied by GeoServer.

While in GeoServer an SLD document needs to be formed and applied for each layer in the map (and it can contain too many lines - in some cases more than 1000), with MapServer the style for each layer is embedded within the mapfile and more specifically within the LAYER Object.

Therefore, when putting a layer in the mapfile, next step is to form its style. If the style is complex, e.g. when you want to display a highway road with a black outline, a

red line and a dashed white line on top, it is preferable to use multiple style within the same layer (combining symbols).

Notice that because the mapfile is read from top to bottom, the style we want to be on top has to be applied last. That applies also to layers.

Depending on the type of the data (point, polygon and line), the construction of the symbols differs. So, we have the three subcategories below:

• *Point symbols*. Point symbols can be of TYPE *truetype*, *pixmap*, *ellipse* and *vector*.
Symbols of TYPE *vector* and *ellipse* determine the shape of the symbol by setting X and Y values in a local two-dimensional coordinate system. *Ellipse* is used to draw ellipses and circles, while *vector* is used to define advanced vector symbols.

Truetype symbols are also used to display point data. The character of the ASCII number of the character is specified in the CHARACTER parameter.

Finally, *pixmap* symbols are also used to draw point data. Pixmaps are practically small raster images. Their name is specified in the IMAGE parameter of the SYMBOL element. MapServer only supports the raster formats GIF and PNG for pixmaps.

• *Line symbols*. For displaying line geometries the width of the line is specified through the WIDTH parameter as well as COLOR parameter to define the color of the line. Color can be given in RGB values or as hexadecimal string.
There are also more parameters defining a line symbol that apply to more complicated symbolization. These parameters are PATTERN for dashed lines, LINECAP, LINEJOIN and LINEJOINMAXSIZE to determine the ending of the lines (round, miter, bevel) and OUTLINECOLOR to draw a outline.

• *Area symbols.* Styling for area symbols is similar to the styling applied on line symbols. Moreover though, on this type of symbols, hatching can also be applied.

There are also more complex styling symbols and these concern *raster* symbols and *label* symbols. Especially for label symbols which are used for instance to put names on settlements, it is required to determine the exact position of the label, the size and type of the font and the direction (straight, curved etc). Therefore, it is required more complex scripts into the mapfiles for such label symbols. It is possible to also use expressions for labeling, because each label can be put not in the same position with the others.

Moreover, what concerns raster data like TIFF or JPEG, their styling is made by using pixel ranges. This applies for pixel range 0-255 of course. Generally, the symbolism of raster data is much simpler than vector data. Additionally, transparency can also be applied on raster layers by using the parameter *opacity,* which takes values from 0 (fully transparent) to 100 (zero transparency).

Finally, there is the possibility of using logical expressions to symbolize data at MapServer. However, this action can only be applied to shapefiles.

3.4.5 Supported input and output formats

MapServer supports a wide range of data formats, including vector and raster formats as well as connection with spatial databases.

Most commonly supported vector formats are ESRI Shapefile, GML, GPS Exchange format (GPX) and Keyhole Markup Language (KML), MapInfo files, GeoJSON and OGC WFS.

MapServer also supports rendering a variety of raster file formats in maps. Raster input support is possible only through the GDAL raster library. Therefore, all supported formats can be found at GDAL - Supported Raster Format List. Most widely used are GeoTIFF, JPEG (.jpeg), PNG (.png), BITMAP (.bmp) and ERDAS Imagine (.img).

Finally, MapServer supports connection with numerous spatial databases such as MSSQL (Microsoft SQL), MySQL, Oracle Spatial, PostGIS/PostgreSQL and SpatiaLite. Moreover, MapServer supports connection with ESRI File Geodatabase (.gdb) and Personal Geodatabase (.mdb) which contain *feature classes*.

A variety of outputs is also available with MapServer. Typical examples are 24-bit png, 24-bit jpeg and png output with number of colors reduced with quantization. An aggregated table containing the main data formats that all three servers support is shown at chapter 6.

3.4.6 Other Operations

MapServer supports some geoprocessing functionalities available via map file which produce new layers which mostly concern geographic transformations. These are *Contour rendering*, *Simplify Line* and *Smooth Line*.

- ✓ *Contour rendering*

MapServer can also compute and render a contour line on-the-fly from a raster source. The raster source has to be an one-band raster data, which represents a

digital elevation model (DEM). Contour interval is added as parameter and the output is a vector line layer.

✓ *Simplifying a line*

Another geometric transformation supported by MapServer is the simplification of a line. This transformation simplifies a line or an area using the standard Douglas-Peucker algorithm. Tolerance is mandatory and is a specification of the maximum deviation allowed for the generalized line compared to the original line.

✓ *Smoothing a line*

Geometric transformation *smoothsia* returns a smoothed version of a line. It uses the following parameters: *shape*, *smoothing_size*, *smoothing_iterations* and *preprocessing* which adds more vertices to the geometry prior to smoothing. The last one is used so as important for the line change of direction isn't lost.

✓ *Charting*

It is possible through MapServer to automatically draw pie or bar graphs whose values are taken and adjusted from attributes of a data source. To achieve this, a layer of type "chart" has to be added in the mapfile.

3.4.7 OGC Services

MapServer supports numerous OGC standards, allowing users to publish and acquire data. Below the available OGC specifications are listed as well as their available versions from MapServer.

• Web Map Service (OGC WMS)

Server: 1.0.0, 1.0.7, 1.1.1, 1.3.0
Client: 1.0.0, 1.0.7, 1.1.1

• Web Feature Service (OGC WFS) 1.0.0, 1.1.0, 2.0
• Web Coverage Service (OGC WCS) 1.0.0, 1.1.0, 2.0.0, 2.0.1

• WMS with Time Support

Especially what concerns WMS, it is mandatory to install a specific program called *mapserv CGI* program, because WMS requests are handled by this program. Because not all versions support WMS, a check is necessary. This check is made by posting at the command line *"–v"* and look for "SUPPORTS=WMS_SERVER" and the following text should come up:

```
C:\apache\cgi-bin> mapserv -v
MapServer version 6.3-dev OUTPUT=GIF OUTPUT=PNG OUTPUT=JPEG OUTPUT=KML
SUPPORTS=PROJ SUPPORTS=GD SUPPORTS=AGG SUPPORTS=FREETYPE SUPPORTS=CAIRO
SUPPORTS=ICONV SUPPORTS=FRIBIDI SUPPORTS=WMS_SERVER SUPPORTS=WMS_CLIENT
SUPPORTS=WFS_SERVER SUPPORTS=WFS_CLIENT SUPPORTS=WCS_SERVER
SUPPORTS=SOS_SERVER SUPPORTS=GEOS INPUT=JPEG INPUT=POSTGIS INPUT=OGR
INPUT=GDAL INPUT=SHAPEFILE
```

Figure 3.14: WMS Settings Parameters

Thing is that this is not the only step required in order for the server to be able to perform the WMS Get Map action. In order for a map synthesis to be available to the server to be distributed through WMS is to add to the mapfile the necessary parameters. These parameters are mandatory and constitute the so named *metadata.* So, for each one of the layers have in the mapfile, it is mandatory to add some information, like:

Layer Metadata:

- *wms_title*
- *wms_crs (*optional, since the layers inherit the MAP's CRS)
- *wms_onlineresource*
- *wms_enable_request*

These parameters have to be added at the MAP object too, at the WEB section. Only by this way, data are available to the server through WMS.

4. WEB MAPPING LIBRARIES

4.1 General

The second component of a web GIS-based application discussed in this thesis is *Javascript libraries.* Javascript libraries for web mapping are client-side tools used to display dynamic maps in desktop as well as in mobile applications. This is achieved by posting the proper JavaScript code in the HTML page of the application.

The term **library** declares a collection of non-volatile resources used by computer programs, often to develop software. These may include configuration data, documentation, help data, message templates, pre-written code and subroutines, classes, values or type specifications.

Hence, a library in computing is a collection of implementations of behavior, written in terms of a language that has a well-defined interface by which the behavior is invoked. For instance, people who want to write a higher level program can use a library to make system calls instead of implementing those system calls over and over again. In addition, the behavior is provided for reuse by multiple independent programs. A program invokes the library-provided behavior via a mechanism of the language.

Developed for web mapping applications, two are the most powerful and commonly used open source JavaScript libraries, ***OpenLayers*** and ***Leaflet***. Spatial libraries like them undertake the management of the spatial part of the map application.

4.2 OpenLayers

OpenLayers[11] is an open source (provided under the 2-clause BSD License) JavaScript library for displaying map data in web browsers as slippy maps without server-side dependencies. It was developed by MetaCarta in 2005. It provides a client-side JavaScript API for building rich web-based geographic applications and is also an Open Source Geospatial Foundation project.

OpenLayers aims to separate spatial visualization and manipulation tools from spatial data and makes it easy to put a dynamic map in any web page. It can display map tiles, vector data and markers loaded from any source. OpenLayers has been developed to further the use of geographic information of all kinds.

[11] http://openlayers.org/

OpenLayers supports GeoRSS, KML (Keyhole Markup Language), Geography Markup Language (GML), GeoJSON and map data from any source using OGC standards as Web Map Service (WMS) and Web Feature Service (WFS). It uses AJAX (Asynchronous JavaScript and XML) to communicate with map servers.

The code of the library is carefully organized in folders and subfolders to facilitate access and search. OpenLayers' latest versions are OpenLayers 4 and 5, which together with Openlayers 3 were entirely revised to incorporate new technologies that HTML5 introduced like Canvas and WebGL JavaScript API for 2D and 3D graphic representation.

Below is a schema showing how OpenLayers communicates through several protocols.

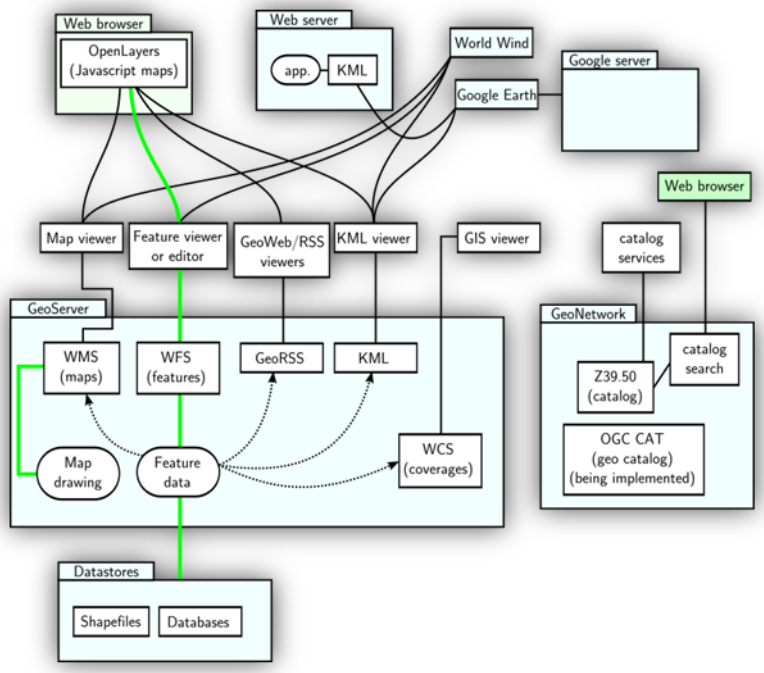

Figure 4.1: OpenLayers communication (ortheast.org)

OpenLayers except for the necessary code needed for data loading and display, provides a number of tools that can be added in a map application, such as:

- Interaction tools (pan, zoom in/out)
- Pop-up windows including information about the clicked data
- Layer overlay with the ability to switch on/off
- Placing markers

- Draw circles and other shapes
- Tools for dragging and rotating the map
- Freehand drawing
- Marker animation
- Arithmetic and graphic scale of the map
- Layer clipping

and many more.

In order to include OpenLayers in a web site, it can be referenced directly from *www.openlayers.org* by posting:

<script src="http://www.openlayers.org/api/OpenLayers.js"></script>,

including the version used.

4.2.1 Map Object

The fundamental variable in OpenLayers is the `map` variable. With this JavaScript code, a map object is created:

var map = new ol.Map({…})

To attach the map object to the `<div>`, the map object takes a `target` into arguments. The value is the `id` of the `<div>`:

```
target: 'map'
```

The `layers: [ ... ]` array is used to define the list of layers available in the map. Layers in OpenLayers are defined with a type (Image, Tile or Vector) which contains a source.

The next part of the `Map` object is the `View`. The view allows specifying the center, resolution, and rotation of the map. The simplest way to define a view is to define a center point and a zoom level. Note that zoom level 0 is zoomed out.

```
View: new ol.View({
      center: ol.proj.fromLonLat([…,…]),
      zoom: …
    })
```

4.2.2 Layers

Layers are lightweight containers that get their data from sources, such as OSM and WMS. OpenLayers provides two types of layers, *Base Layers* and *Overlays*, each one of them is displayed in a different way.

- Base Layer: Base layers determine the projection of the data and the available to the user zoom levels. Only one base layer can be on at one time, which means that if someone has two base layers, they cannot be displayed simultaneously.

To determine whether a layer is base layer or not, is stated from the following line:

```
isBaseLayer:true
```

Base layers are always displayed on bottom of the map. Usually they are raster images.

- Overlays: Overlays are displayed differently than base layers. More than one can be activated simultaneously. They cannot control the zoom level of the map, but they can be set to on or off depending on the scale.

Some overlays support reprojection so as they can have the same coordinate system with the base layer, while loading on the map (on-the-fly reprojection). Overlays are always displayed on top of the map.

There are some different subclasses of layers regarding their source. Most common of these are:

- `OpenLayers.Layer.WMS` (or `ol.source.ImageWMS` for OpenLayers 4) for WMS services,
- `ol.layer.Tile and ol.layer.TileWMS` for tiled images and tiled WMS images (sources that provide pre-rendered, tiled images in grids that are organized by zoom levels for specific resolutions), and
- `ol.layer.Vector` to import vector data that is rendered client-side, like GeoJSON.

4.3 Leaflet

4.3.1 General

Leaflet[12] is a widely used open source JavaScript library used to build web mapping applications. First released in 2011, it supports most mobile and desktop platforms, supporting HTML5 and CSS3. Along with OpenLayers it is one of the most popular JavaScript mapping libraries and is used by major web sites. It is suitable for building mobile-friendly interactive maps and is very lightweight. Leaflet is designed with simplicity, performance and usability in mind. It works efficiently across all major desktop and mobile platforms and can be extended with lots of plugins. Latest version is 1.3.3.

Leaflet allows developers without a GIS background to very easily display tiled web maps hosted on a public server, with optional tiled overlays. It can load feature data from GeoJSON files, style it and create interactive layers, such as markers with pop-ups when clicked.

Apart from GeoJSON files, Leaflet supports Web Map Service (WMS) layers, vector layers and Tile layers natively. Many other types of layers are supported via plugins.

Like other web map libraries, the basic display model implemented by Leaflet is one basemap, plus zero or more translucent overlays, with zero or more vector objects displayed on top.

The major Leaflet object types are:

- Raster Types (TileLayer and ImageOverview)
- Vector Types (Path, Polygon and specific types such as Circle)
- Grouped Types (LayerGroup, GeoJSON)

Leaflet also supports various *Controls*, such as zoom in/out, layer switcher and scale.

In contrast with OpenLayers, Leaflet is designed with emphasis on the simplicity. It works efficiently across all major desktop and mobile platforms, has an easy to use and well-documented API and a simple, readable source code. At this code, whoever wants to, can easily contribute.

Main object in Leaflet is the *map* object. Within the map object (which is created in the map), is also defined the center (lon, lat) and the zoom (z). The code needed to achieve this, is very simple, just a line.

```
Var map = L.map('map').setView([lon, lat], z);
```

[12] http://leafletjs.com/

The disadvantage here is that the view has to be in geographical coordinates, even though someone wants to display a map at a specific projection. Also, like OpenLayers 4 and 5, Leaflet supports scales as zoom. This means that specific scales are referred as a number which determines the zoom. This has happened because zoom matches better with parameters of digital maps such as screen resolution.Consequence of this is slight differences between the desirable scale and the zooming.

In addition to the above insertion, when someone wants to display a map from another source, for example Openstreetmap, this can also be easily done by placing a code as following:

```
L.tileLayer('https://{s}.tile.openstreetmap.org/{z}/{x}/{y}.png', {
    attribution:                    '&copy;                    <a
href="https://www.openstreetmap.org/copyright">OpenStreetMap</a>
contributors'
}).addTo(map);
```

There is a plethora of sites using Leaflet for their applications, such as Github, Marinetraffic, Foursquare, Pinterest, Flickr and many more.

4.3.2 Mobile Applications

Due to its ease and lightness, Leaflet is ideal for building mobile applications that can run on iPhone, iPad or Android phones. Additionally, the extra here that has also to be taken into account is the ability to turn into full screen as well as the ability to detect the location of the phone (detect and use current user location).

Turning the map in full screen is very easy and it can be done by modifying the CSS code, specifically by putting height and width:

```
body {
padding: 0;
margin: 0;
}
html, body, #map {
height: 100%;
width: 100vw;
}
```

Also, we need to tell the mobile browser to disable unwanted scaling of the page and set it to its actual size by placing the following line in the head section or our HTML page:

```
<meta name="viewport" content="width=device-width, initial-scale=1.0,
maximum-scale=1.0, user-scalable=no"/>
```

Another very important factor for building mobile applications is the geolocation. Leaflet has a very handy shortcut for zooming the map view to the detected location — by replacing the usual *setView* method in the code:

```
map.locate({setView: true, maxZoom: Z});
```

Z is specified as the maximum zoom when setting the map view automatically. As soon as the user agrees to share its location and it is detected by the browser, the map will set the view to it. Now we have a working fullscreen mobile map.

Additionally, to make the map fancier, we can add a marker to the current user location as well as a text like “You are here”. If the point is a circle, it is needed to determine just the coordinates and the radius, just like this:

```
function onLocationFound€ {
    var radius = e.accuracy / 2;

    L.marker(e.latlng).addTo(map)
        .bindPopup(“You are within “ + radius + “ meters from this
point”).openPopup();
  L.circle(e.latlng, radius).addTo(map);
}
map.on(‘locationfound’, onLocationFound);
```

4.4 Comparison of the basic scripts used to display a map

There is a wide variety of coding available on both libraries, depending on how complicated we want the application to be. Nevertheless, there are some fundamental algorithms that need to be used in order to display a simple map through an application on internet. These algorithms concern the map display, the panning and the zooming, showing coordinates while panning, the scalebar, the projection, the map box and the switching between layers.

They are presented on the following table, to enable the direct comparison of the two libraries regarding their complexity at coding.

	OpenLayers	Leaflet
Map object	*var map = new ol.Map ({ interactions: ol.interaction.defaults(). extend([new ol.interaction.DragRotateAndZoom()]), layers: [layer], target: 'xartis', controls: ol.control.defaults({attributionOptions: /** @type {olx.control.AttributionOptions} */*	*var map = new L.map('xartis', {center: [37.44,24.92], zoom: 11.25, maxBounds: bounds, scales: myscales, minZoom: 11.25, maxZoom: 13, zoomSnap: 0*
Boundaries	*var view = new ol.View({projection: projection, center: [581000,4142500], zoom: 1, minZoom:1, maxZoom:3, extent: [572000, 4130000, 590000, 4155000] });*	*var northeast = L.latLng(37.3, 24.6), ortheast = L.latLng(37.7, 25.2), bounds = L.latLngBounds(ortheast, ortheast);*
WMS layer insertion	*var layer = new ol.layer.Image({ extent: extent, source: new ol.source.ImageWMS({ url: 'http://127.0.0.1/cgi-bin/mapserv.exe?map=/ms4w/apps/mapserv-demo/syros.map&', params: {'LAYERS': 'SYROS', 'FORMAT' : 'image/png'}, ratio: 1, serverType: 'mapserver'}) });*	*var ms = L.tileLayer.wms('http://127.0.0.1/cgi-bin/mapserv.exe?map=/ms4w/apps/mapserv-demo/syros.map&', { layers: 'SYROS', format: 'image/png', transparent: true, attribution: 'MapServer© 2017 Lydia', }).addTo(map);*

Scalebar	*var scaleLine = new ol.control.ScaleLine();*	*L.control.scale({maxWidth : 100, metric : true, position: 'bottomleft'}).addTo(map);*
Toggle full screen	*var fs = new ol.control.FullScreen();*	–
Zooming	*Integrated in ol.view*	*L.control.zoom({zoomInTitle: "zoom in",zoomOutTitle: "zoom out"});*
Coordinates while panning	*var pontiki = new ol.control.MousePosition({ coordinateFormat: ol.coordinate.createStringXY(4), projection: 'EPSG:2100', target: document.getElementById('mouse-position') });*	*L.control.mousePosition().addTo(map);*

Table 4.1: Comparison of the common tools of the libraries in scripts

As it can be seen from the table above, the main functionalities are the same for both libraries. They both perform the necessary processions to make available a web map on internet through an application.

It is obvious that the complexity of OpenLayers is bigger because it needs many more lines of coding. However, the capabilities of OpenLayers are chaotic comparing with Leaflet's. Leaflet is the perfect library for building a simple web application as well as for a beginner with no prior knowledge of programming. Furthermore, Leaflet is lighter comparing to OpenLayers and that makes it suitable for mobile applications.

There are also some similarities regarding to the code, but still OpenLayers has more extensive coding. That makes it a little more difficult to elaborate with it, especially when someone is beginner. It takes more time for someone to understand how the code works when it is so extended. Lest we forget that they are JavaScript-coded libraries.

Another basic difference between the two JavaScript libraries is the fact that Leaflet is using plugins. That means that it contains only the basic operations and that is why it is so light. When someone wants to build a more sophisticated application, then additional reference to these plugins has to be made into the .js file. With

OpenLayers this does not happen. Each version of OpenLayers contains all packages of coding, and that is why it is heavier than Leaflet.

The problem here is to know from the beginning whether or not the one we want to use is included to the main version of Leaflet or it has to contain the appropriate plugin each time.

5. IMPLEMENTATION OF THE APPLICATION

In this chapter the main application is presented. In order for an application to take its final scheme, many steps are required. Therefore, building an application may sometimes be complicated.

5.1 Scope of the application

The objective purpose of the application is to present the different ways of how geographic data are spread through internet in the shape of a *cartographic map*- namely by using the two mostly and widely used open source map servers, GeoServer and MapServer and of course the proprietary one, ArcGIS Server. Secondarily are presented the two open source web mapping APIs, OpenLayers and Leaflet, which are JavaScript libraries used to display maps in websites offering a plethora of useful tools.

Through this application it is attempted to draw some conclusions on the differences of displaying the map, which map server offers the most advantages regarding to tools as well as facilities and convenience and which one is eventually easier to use.

The map presented is a topographic map of Syros Island, Cyclades, Greece in multiple scales and it is judged exclusively from a cartographic point of view.

5.2 Description of the application

In this sub-chapter the application and its components are described. Such types of map applications require some specific and necessary components. These components are presented below:

• *Data.* Data are the most important component. Without them, the application cannot exist. Regarding this attempt, the kind of data used is *geographic data.* That means data with a geographical reference. This type of data requires a very special handling related to simple data. This kind of data comes to several forms, known as formats. In this particular application, the data being used are ESRI *shapefiles* for vector and GeoTiff images as raster.

• A *map server*, in order to “serve” the data through the internet.

• *Styling.* This is about how data are visualized. Styling on maps follows some strict rules in order to build a map which is visually proper and shows correctly all the information that contains.

• A Javascript *library* which helps the application to be displayed on the internet. It contains tools written in Javascript which enable the interaction between the user and the map.

• An *interface* for the website. This is achieved with the combination of three very important components, which cooperate and build together the frame in which the whole application is displayed. These are: *HTML, CSS and JavaScript*. HTML stands for HyperText Markup Language and is the standard markup language for creating web pages and web applications. With Cascading Style Sheets (CSS) and JavaScript, it forms a triad of cornerstone technologies for the World Wide Web.

CSS is used to form the styling of the webpage (colors, frames etc), while Javascript is a high-level, interpreted programming language which enables building interactive web pages and thus is an essential part of web applications. The vast majority of websites use it, and all major web browsers have a dedicated JavaScript engine to execute it.

5.2.1 Data preparation

As mentioned above, a high percentage of the data used are of vector format, namely shapefiles. Shapefiles are nowadays of an open format. The shapefiles used were points, as well as lines and polygons.

Line shapefiles include roads, coastline and contours, polygon shapefiles include the polygon of the shape of the island, while point shapefiles are used to represent point objects such as churches, beaches, various POI, Z points etc. Raster images are used to display a digital terrain model (DTM) of the surface and the hillshade effect.

In order for the data to be ready to be imported to the application, some processions are requisite. Initially, because the map is of three different scales, specifically 1:50.000, 1:75.000 and 1:150.000, the data had to be generalized. Generalization is a complex and mainly subjective cartographic method for deriving a smaller-scale map from a larger scale map or map data. Whether done manually by a cartographer or by a computer or set of algorithms, generalization seeks to abstract spatial information at a high level of detail to information that can be rendered on a map at a lower level of detail.

Generalization reduces the complexity of a map by making less important objects of the map “disappear” and at the same time preserves the logical and topological relations between the objects of a map, while also maintains its aesthetic quality. According to the International Cartographic Association (1973), a formal definition of

generalization is the following*: "Selection and simplified representation of detail appropriate to the scale and/or the purpose of the map".*

In order to make the maps of the smaller scales (1:75.000 and 1:150.000), generalization was needed. More specifically:

For point data, the method of deduction was applied. This method deletes objects from each layer depending on their necessity. There also is an empirical law which suggests the proper number of objects depending on the scale of the final map. This law is called *Töpfer's Radical Law (Töpfer and Pillewiser 1966*) and is the only quantitative rule in the selection of the features. It yields the number of features to be displayed, *but does not reveal which of the features should be chosen.* This law is expressed by the following formula:

$$n = n_0 \sqrt{\frac{S}{S_0}},$$

in which, n is the number of the objects at the smaller scale, n_0 the initial number of objects and S, S_0 the corresponding scales.

Apart from deduction, some of the remaining objects were moved so as to take a more proper place in the map.

Concerning line objects, more than one action were performed. For the *coastline,* the necessity of the simplification was obvious. The algorithm used was the *Simplify line,* of course with a tolerance which matches the desirable scales. 50 meters tolerance was used for 1:75000 and 100 meters for 1:150000.

Contours only needed smoothing, as they do not have a metric value.

On the contrary, roads and streams needed both simplification and smoothing.

Polygon layers such as capital and coastline polygon underwent the same actions as the coastline.

Our region of study is Syros Island, Greece. Now our data are ready to be consumed through the map servers.

5.3 Building the application using Geoserver

5.3.1 Introduction

The current version of GeoServer used is *2.11.5* and it concerns the GeoServer located at National Technical University of Athens (NTUA) with URL: *http://atlas.geocenter.survey.ntua.gr:8080/geoserver/web/*

In order to use GeoServer an account has to be made. If someone wants to, he can download GeoServer locally on the computer and activate it when he wants to use it. In this way, his computer becomes the server.

Our data consist of vector ESRI shapefiles and GeoTiff images. The *Store* in Geoserver is the place where the data is stored and can be accessible from GeoServer. Vector and raster data cannot be put together in the same folder. Therefore, it is created a folder which contains all the shapefiles together.

Second of all, one store can contain only one raster image. That is why DTM and hillshade are stored on separate folders within the GeoServer Store, as it can be seen.

Figure 5.1: GeoServer Store

After the data are imported to GeoServer and the Workspace is created, this data can be uploaded and viewed. Each data now is called *layer* and can contain only one type of data (point, line or polygon).

From the information that the shapefile contains itself, the boundaries and the SRS are automatically filled. Available list of uploaded layers can be viewed on the field *Layers.*

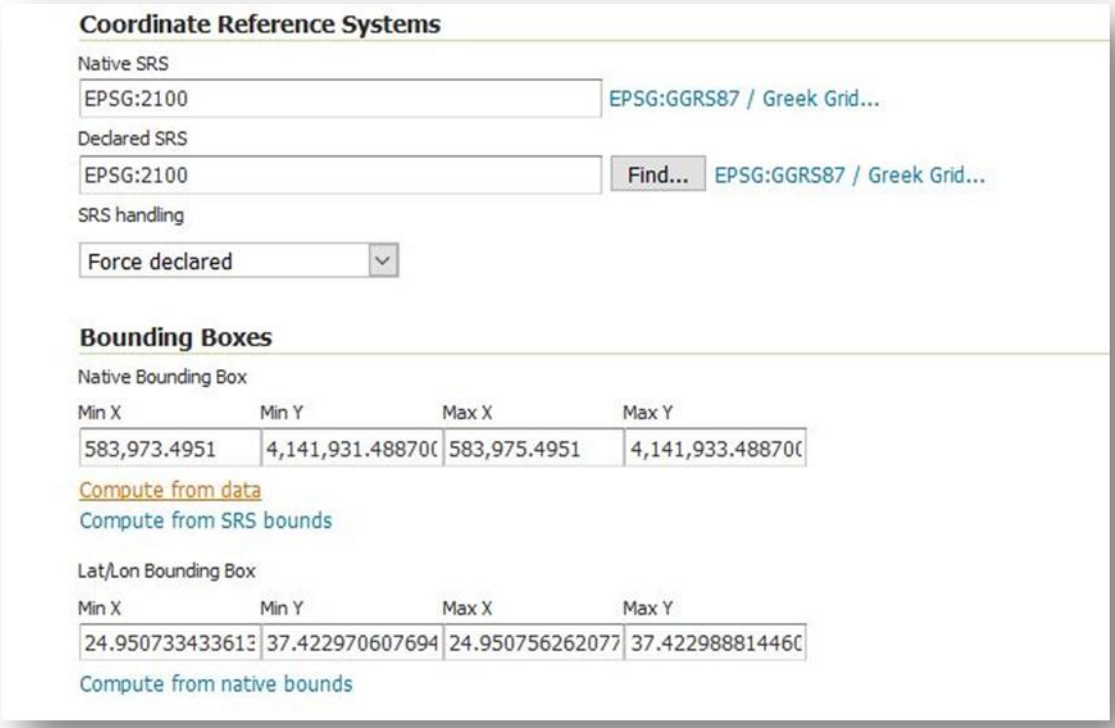

Figure 5.2: Layer Parameters

It has also to be filled the basic info of the layer, as well as its WMS settings and finally and most important, the layer has to be connected with a specific style.

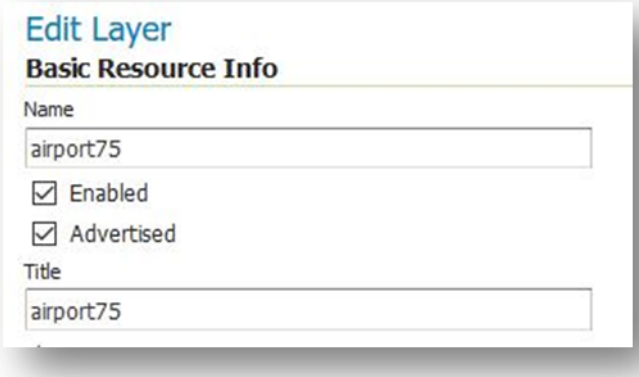

Figure 5.3: Layer Info

The style is very important as it concerns the way our data are visualized. As mentioned on previous chapter, GeoServer has a complicated method for applying symbology to data, the SLD specification.

So, in order for a layer to be viewed, it has to be connected to a specific SLD file that already exists in the GeoServer. This specific SLD file contains all the necessary information for drawing the layer.

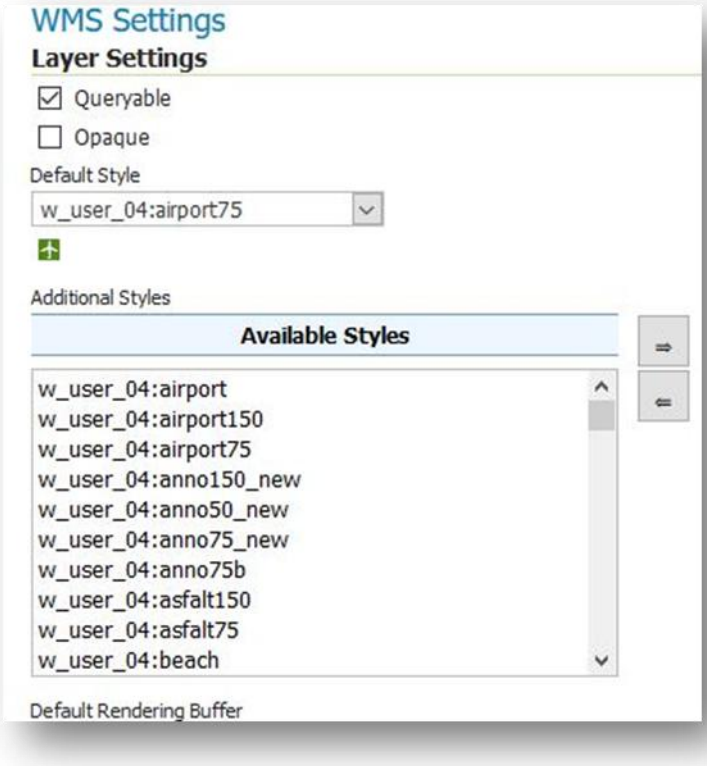

Figure 5.4: Connection with Style

5.3.2 Styling

The next step is about symbology. Styling in GeoServer is applied with a special way, discriminating it from MapServer and ArcGIS Server. Here, the so named SLD specification is used. A separate SLD file has to be created for every layer.

Thus, for example, the layer airport has its own SLD file, layer contours its own, etc. The SLD file then, has to be connected with the corresponding layer, so as to be symbolized and displayed properly through *Layer Preview*.

All SLD files are stored within GeoServer, at the field *Styles*.

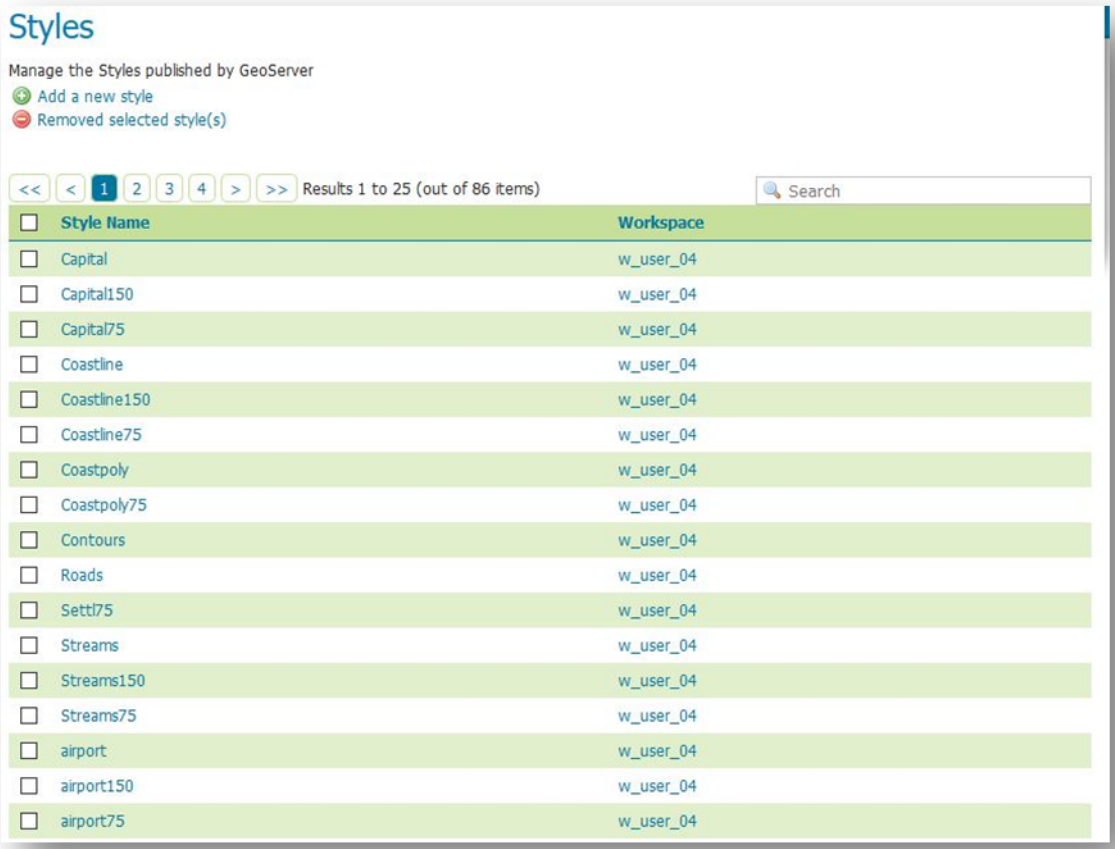

Figure 5.5: SLD files

As mentioned on a previous chapter, SLD files follow the structure of XML documents. That means that it practically is a text which contains all the necessary information for the style in it. These SLD files are similar with minor differences, when it comes to point, line or raster data.

For example, an SLD file for a point ESRI shapefile layer is like the following:

```
<?xml version="1.0" encoding="UTF-8"?>
<StyledLayerDescriptor version="1.0.0"
 xsi:schemaLocation="http://www.opengis.net/sld
http://schemas.opengis.net/sld/1.0.0/StyledLayerDescriptor.xsd" xmlns=http://www.opengis.net/sld
xmlns:ogc="http://www.opengis.net/ogc" xmlns:xlink="http://www.w3.org/1999/xlink"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
 <NamedLayer>
<Name>beach</Name>
   <UserStyle>
<Name>beach</Name>
<Title>beach</Title>
<FeatureTypeStyle>
<Rule>
<Title>beach</Title>
<MinScaleDenominator>25000</MinScaleDenominator>
<MaxScaleDenominator>65000</MaxScaleDenominator>
<PointSymbolizer>
<Graphic>
<ExternalGraphic>
<OnlineResource xlink:type="simple"
xlink:href="paralia.png" />
<Format>image/png</Format>
</ExternalGraphic>
<Size>
 <ogc:Literal>14</ogc:Literal>
 </Size>
</Graphic>
</PointSymbolizer
</Rule>
 </FeatureTypeStyle>
 </UserStyle>
 </NamedLayer>
</StyledLayerDescriptor>
```

Figure 5.6:Example of an SLD file

When an image needs to be used, this is also stored within the GeoServer and it is called from the SLD file. These images are mostly PNG images or SVG images. Their size, the scales that will be displayed and all the styling options, they are all determined within this SLD file.

The SLDs are the same number as the layers. It is understandable that when someone has never used this specification before, it is a little bit hard to build all these SLD files for each layer. The perfect combination of colors, sizes and symbols depends on the map scale, the aesthetics of the cartographer and the final display of the map application. Therefore, this procedure may be laborious, difficult and multi-hour.

This is a reason and an explanation why GeoServer may be difficult for a beginner. Apart from styling, GeoServer in general is a very easy-to-use server for publishing

the map through the web. Its ease comes from the fact that GeoServer has a very friendly user interface (GUI), which some servers may not have (e.g. MapServer). The user interface is very important, especially for a beginner, because it sets out the basic directions for what to do as well as facilitates the procedure for someone who is not familiar with programming.

5.3.3 Presentation of the application

In this subsection, the main application is presented. Aim of the application is to present a complete and flawless map through an internet application using the OGC WMS Service. The application consists of a map of three different scales and fulfills all the specifications of a cartographically correct map.

In order for a map to be spread to internet, the collaboration between three different programming languages is needed. These are HTML, JavaScript and CSS.

HTML is used to build the website, it is about the architecture of the page. It determines where the box that will contain the map will be placed as well as where the rest of the information will be placed. The HTML code used is included in the appendix. CSS concerns the styling of the page. It is about the colors and the sizes of the boxes mentioned above. This is also included in the appendix. JavaScript is the most important and complex file of all three. With the Javascript file, the behavior of the application and the page is determined. In this file is also put the URL that calls the map through WMS. The URL needed for calling the WMS Service can be found on GeoServer.

In the JavaScript code is contained all the necessary information from the libraries. The URL of the corresponding library is put so it can be called, and then the code is put. This file follows of course the structure of JavaScript programming language and so do HTML and CSS. Moreover, here lies the URL used for WMS request too. This was:

http://atlas.geocenter.survey.ntua.gr:8080/geoserver/wms

Before we end up to the final application, it is possible to view the data directly from GeoServer, if someone wants to. This is possible through GeoServer's *Layer Preview.* Through OpenLayers, which is integrated to GeoServer the data display is available. The disadvantage here is that every layer is shown separately. In order to solve that, we can put together all the layers we want that consist a map, but carefully, in the right order we want them to be shown. This is called *Layer Group* and it is very convenient if someone has many different maps for publishing because it puts the necessary layers in order.

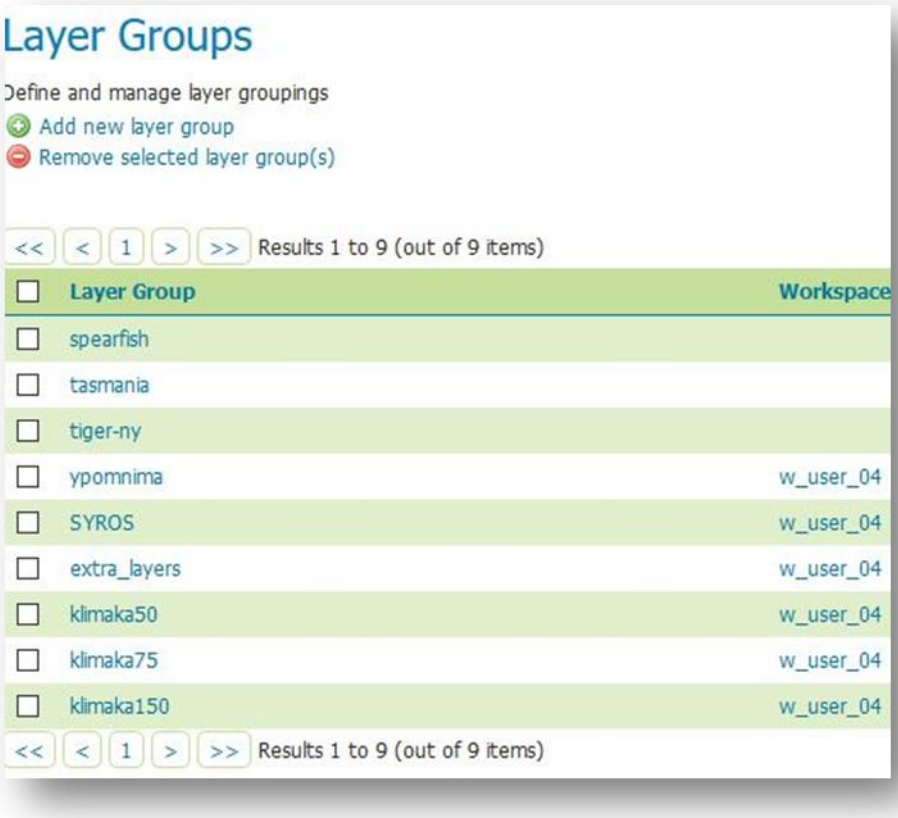

Figure 5.7: Layer Groups

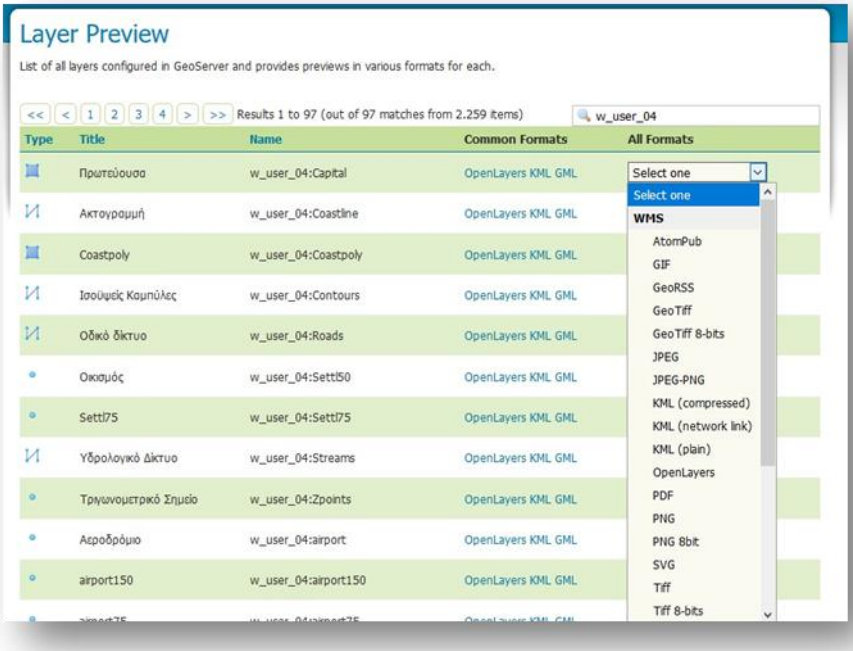

Figure 5.8: Layer Preview

It can be noticed that there are available many types of output formats for WMS as well as WFS services, such as PNG, TIFF, SVG, JPEG, BMP etc.

The application runs by using all kinds of web browsers available. These are Google Chrome, Mozilla Firefox, Opera, and Safari. Generally, there will be no differences to the display between these browsers, but this sometimes has to do with the version or the bugs that it contains.

The map was created on three different scales and the result is presented below:

Figure 5.9: Scale 1:150.000

The next scales presented are 1:75.000 and 1:50.000. Here a snippet of the map is shown, as these scales cannot fit entirely into one page.

Figure 5.10: Scale 1:75.000

Figure 5.11: Scale 1: 50.000

The results shown indicate that GeoServer is able of producing high-quality maps and can also handle label annotation and symbols with great success.

5.3.4 Results and commenting

The effort to make a map available on the internet using the WMS specification has succeeded. GeoServer has done its job with great success. It served the data properly applying the styles that were determined. In cooperation with OpenLayers and the final application, has made a very nice result.

When using open source software, such as GeoServer or OpenLayers, we must be prepared that the way to the final product might be hard. This is not necessarily bad; it makes people realize why they are doing every single step and what impacts it has to the whole process. So, when someone uses FOS software, becomes more interactive and takes a leading role to the whole procedure, because it is less automaticated than using proprietary software.

5.4 ArcGIS Server web map

The procedure here is a lot different than using free and open-source software. When using ArcGIS Server, the whole process is simplified. There is no need for someone to have programming knowledge; it is enough to have just cartographic knowledge. Let's check on how the implementation is done using ArcGIS Server.

5.4.1 Preparation

The current version used is ArcGIS Server 10.5. In order for the map(s) to be uploaded to ArcGIS server and therefore to become available for everyone connected to Internet, they have to be drawn using ArcMap and ArcCatalog. ArcCatalog is used to connect to the data, which are locally on a folder. ArcMap (also proprietary), is the software that is used to create the map(s).

As discussed previously, styling here is applied by using the already existing symbols of ArcMap. So styling is not of major reference as it is when working with open source servers, such as GeoServer and MapServer.

Layers on different scales –as in our attempt-, have to be drawn within the same .mxd document. To achieve this, is set at which scales we want them to be visible by the *Layer Properties*. Assuming we want a map to be displayed at three different scales; the layers that constitute the first map, in our case the scale 1:150.000, they

must all have the same scale range. That means that they must have a minimum scale and a maximum scale in which the data will be visible so as there is not overlapping with the data of the other scales. This is determined from the *Properties* of each layer, as following:

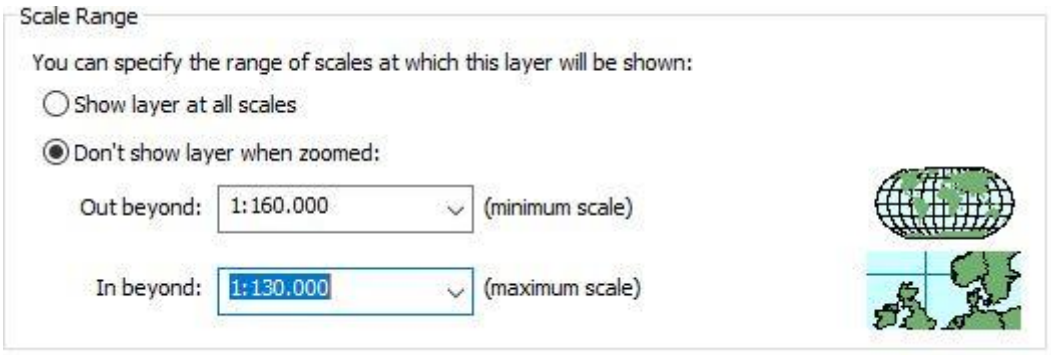

Figure 5.12: Scale ranges in ArcMap

Therefore, layers of the next scale must have another range so as they do not overlap.

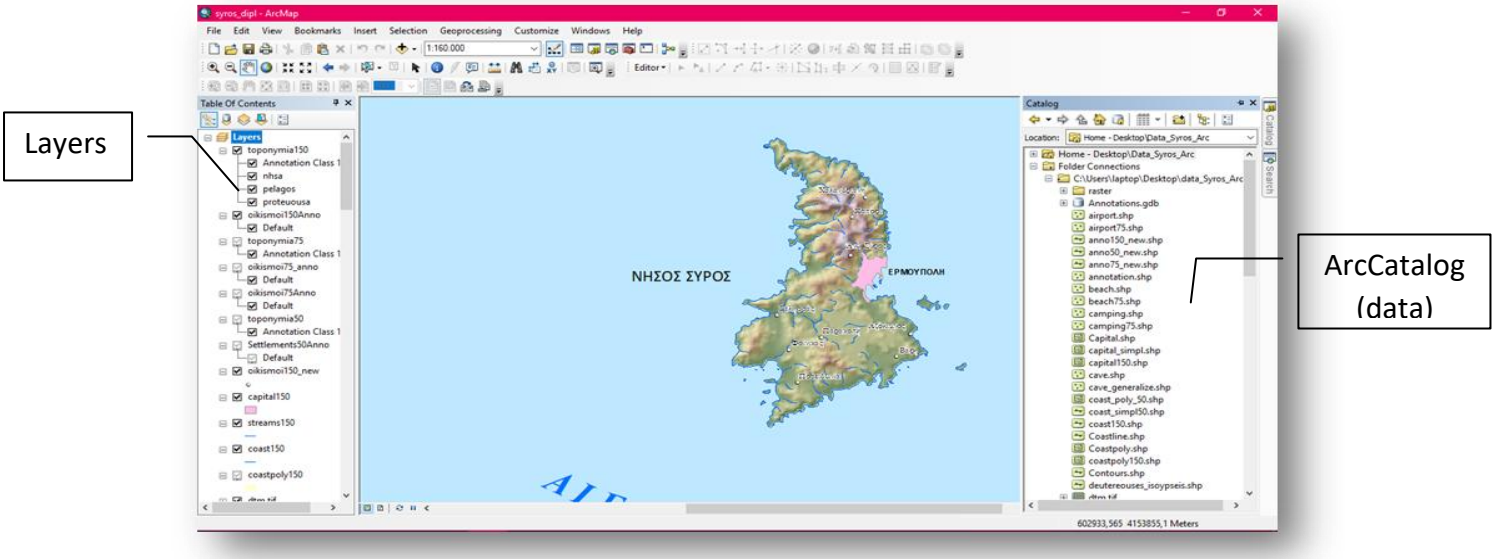

Figure 5.13: ArcMap Interface

5.4.2 Uploading to ArcGIS Server

Whenever the map synthesis is ready, the connection to the ArcGIS Server has to be established. This is achieved by right-clicking within ArcCatalog, *Add ArcGIS Server*.

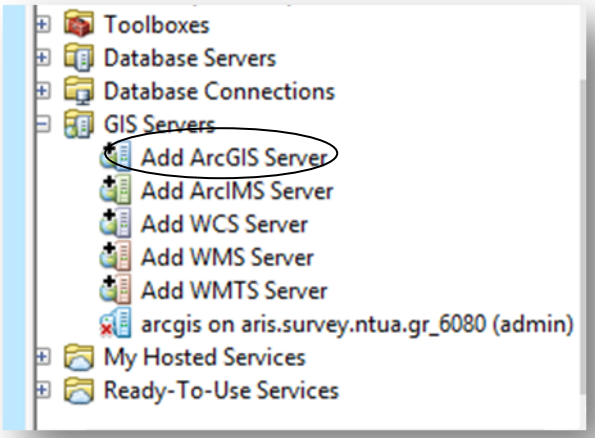

Figure 5.14: Connection to ArcGIS Server

When the connection to the ArcGIS Server is established, the next step is an analyzing check, which controls and checks whether there are errors that prevent the map to be published or not. If the field *Errors* does not contain any registrations, the map is ready to be published to *ArcGIS Server*. In that way, it is secured that the map will be shown with no errors or omissions.

Subsequently, all the data that are stored locally and which are used to draw the map, are uploaded to ArcGIS Server. This can take some minutes, depending on what data they are or how many they are.

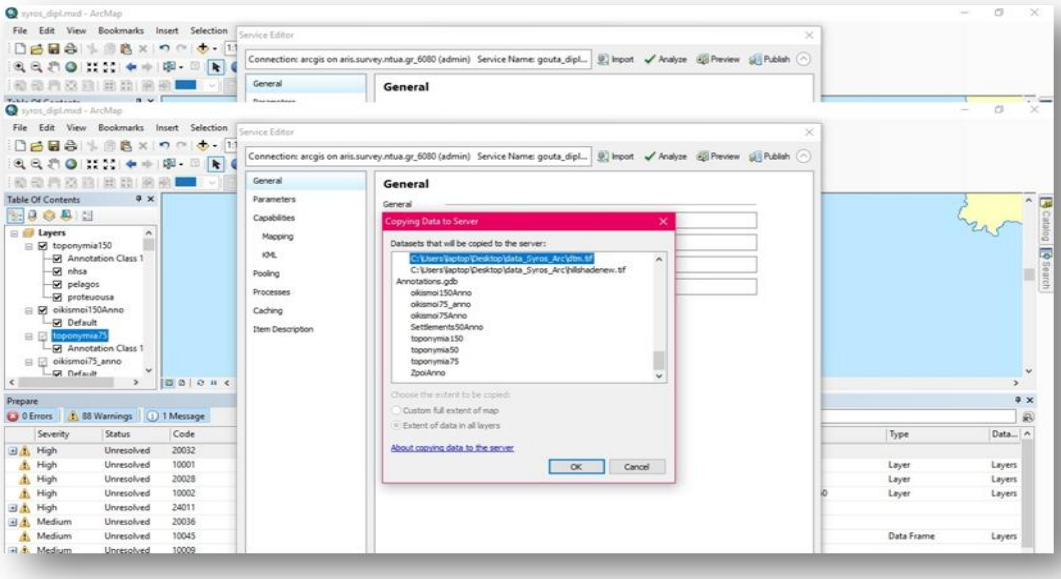

Figure 5.15: Data upload to ArcGIS Server

Before data are uploaded, ArcGIS Server performs some checks. If the check is completed and errors will not occur, the data are successfully uploaded to the server. However, there might be cases where errors occur. In that case, they have to

be solved or else the map cannot be uploaded to the server. Except for the errors, the server also shows warnings. These warnings do not prevent the uploading, but it is better to solve all isues before uploading, so as the data are visualized properly. The check is performed by clicking *Analyze* on the main menu before uploading.

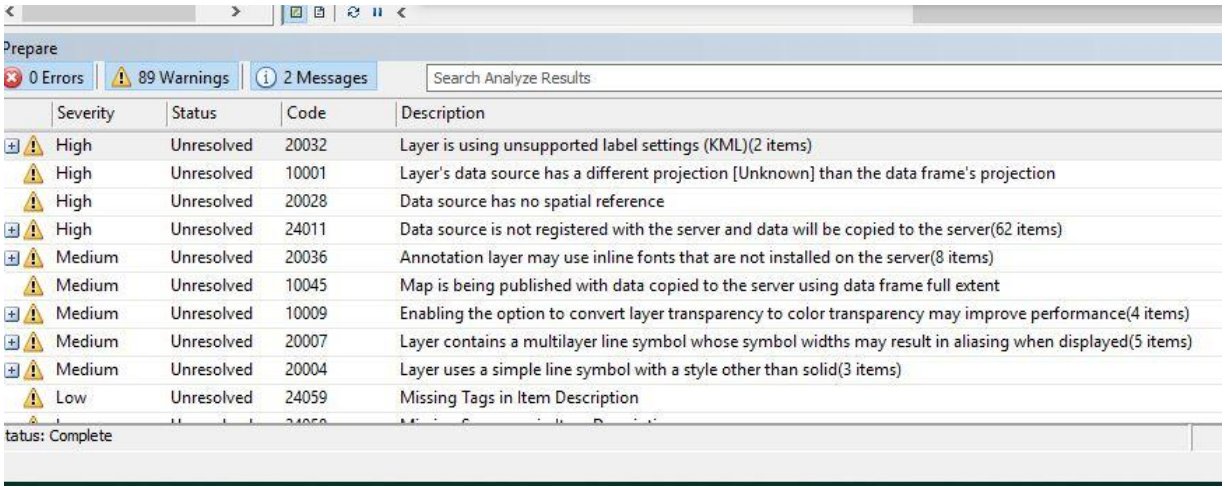

Figure 5.16: Checks performed by ArcGIS Server before uploading

Checks concern mainly the display of the data and the user can either fix them or ignore them (if they are not of high severity or marked as errors). Some of the most common errors are about the name of the layers (layers of the map document must not have same names), the spatial reference, the annotation fonts and problems at anti-aliasing.

When the upload ends successfully, the map is stored within ArcGIS Server in its own folder and is ready to be viewed and later published using WMS.

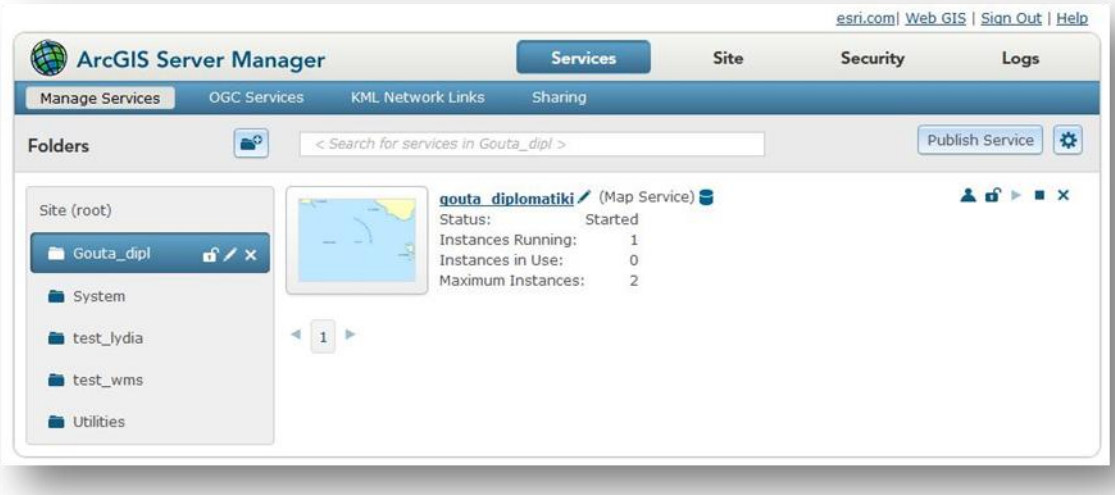

Figure 5.17: Folders at ArcGIS Server per project

In order for the data/map to be viewed within ArcGIS Server, ArcGIS Server has its own Javascript API, the *ArcGIS Server Javascript API* and it is the proprietary equivalent to OpenLayers and Leaflet.

It must be understood that when the synthesis is ready, then it has to be uploaded to ArcGIS Server. There are uploaded the data of course, as well as the styling (symbols) which accompany them. Therefore, the .mxd document is published to the Server and that is why it is asked which data we want to copy to the server while uploading.

This is how data were visualized by using ArcMap and ArcGIS Server:

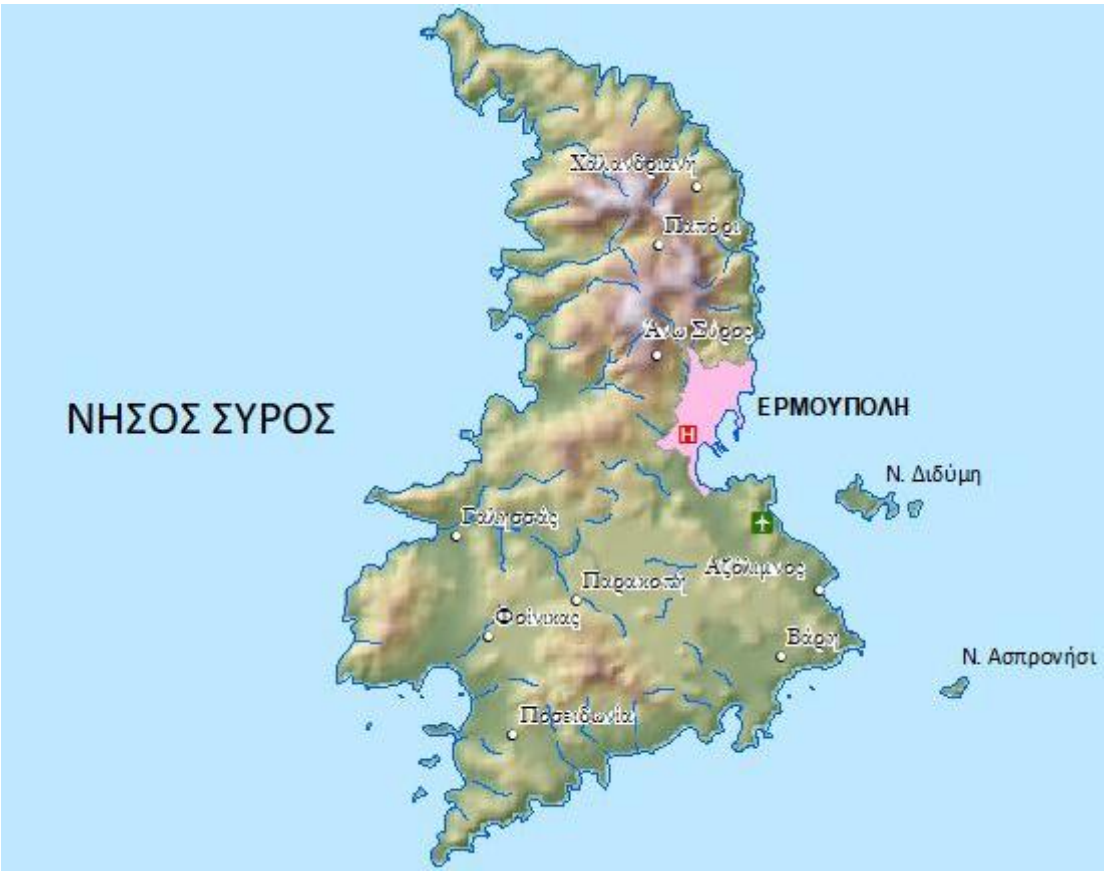

Figure 5.18: Scale 1:150.000-ArcGIS Server

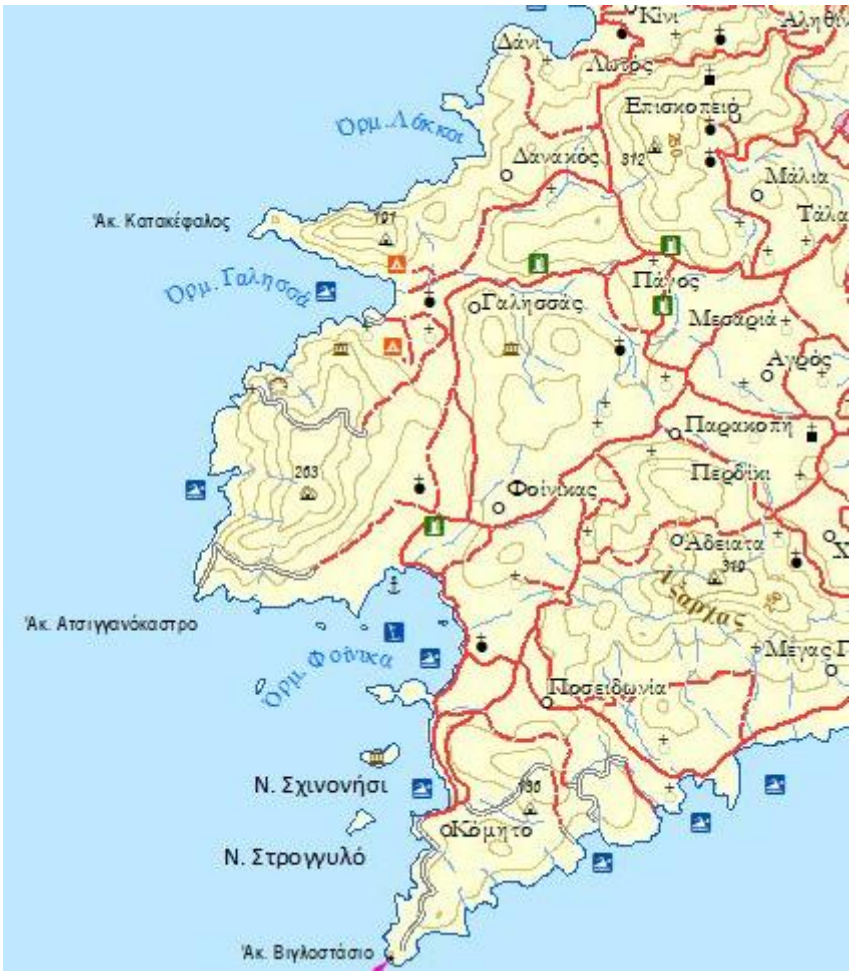

Figure 5.19: Scale 1:75.000-ArcGIS Server

Figure 5.20: Scale 1:50.000-ArcGIS Server

In ArcMap, layers can be of point, line, polygon or raster data, like as all map applications. However, at GeoServer and MapServer the annotation is not separated from the other types of layers. It is concerned as a line layer which displays only its labels.

Also here, annotation is a different kind of feature. It has a slight difference which luckily does not affect the upload and publishing on ArcGIS Server; these annotation layers cannot be of a shapefile format. They have to be stored into an ArcGIS File or Personal Geodatabase and of course at the same coordinate system with the rest of the map (Greek Grid). This is mentioned, because at all three servers the data are of SHP format and so are for ArcGIS Server, except for Annotation Layers.

This is also mentioned here to show how data from different data sources, e.g. from a file that contains shapefiles and data that are stored in an ESRI Geodatabase can be integrated to the same MXD and therefore in the same application as a single map synthesis. This improves the functionality and promotes interoperability.

Except for the possibility to fix occuring errors, there is also the possibility to preview the map, namely to view the map and see how it is displayed to the Server. This is possible while uploading the data to the Server. When the option *Preview* is enabled,

it opens a new window which displays our synthesis, but also calculates the time in seconds it took to be drawn. This is a sign of how fast the server on data rendering is. Of course, it also depends on the data volume are and their format and size and how complex their styling is. Usual rendering time was 0.2 seconds.

5.4.3 Results and commentation

The attempt to publish a web map through ArcGIS Server was a very different process than building the same application with open source map servers, like GeoServer or MapServer. Whereas with MapServer and GeoServer data is uploaded from the very first moment on the Servers and the styling is applied there, with ArcGIS Server the synthesis is made entirely on ArcMap, namely before anything is uploades to the Server.

ArcGIS Server offers a very convenient user interface, as well as GeoServer, but it is much easier to publish a web map with ArcGIS Server. If someone knows how to draw a map on ArcMap, then the whole process is simplified. There is also no need to write a single line of code, such as MapServer's Mapfile or SLD specification that is needed for GeoServer.

ArcGIS Server offers a calculation of time that is needed in order to draw a map each time. This varies, from 0.23 seconds to 1 second. In any case, the time needed is diminutive. It gives a measure to understand the speed that ArcGIS Server offers.

Although when the map was drawn on ArcMap, it was added a light blue font to all the synthesis which concerned the sea. This font was not transferred to ArcGIS Server during uploading. That happened because this font was not a single layer itself. In order to display a font, it has to be drawn as a single polygon layer with the desirable color.

Despite its ease and efficiency, ArcGIS Server is undermined by a disadvantage. When a change to the data or the style has to be made, or even an insertion or deletion of data, then there is not a refresh to ArcGIS Server enough, like it happens with MapServer and ArcGIS Server. It has to be uploaded all the content with the new changes again from ArcMap to ArcGIS Server.

This sadly, makes the application not so interactive, because uploading the data to the Server takes some minutes to complete. So, it is suggested that the whole synthesis has to be completed in order for it to be uploaded to the Server.

5.5 Web Mapping using MapServer

The next attempt was to publish the very same data as a map using MapServer to serve them through internet. The procedure using MapServer was different than ArcGIS Server, but also different than using GeoServer. The results though, were similar.

5.5.1 The role of Mapfile

As discussed on a previews chapter, mapfile is the core of MapServer. That also applies here. The mapfile contains all the necessary information for the map to be drawn properly and has to be initially created. It is also included in the appendix.

Since MapServer does not have a user interface, like the other servers, it was very difficult to begin with it. Through a URL the previewing of the data was possible. This URL is the following (concerning the MapServer that it is installed locally on the computer):

http://127.0.0.1/cgi-bin/mapserv.exe?map=/ms4w/apps/mapserv-demo/syros.map&LAYERS=SYROS&VERSION=1.1.1&SERVICE=WMS&REQUEST=GetMap&FORMAT=application/openlayers&WIDTH=1000&HEIGHT=600&SRS=EPSG:2100&BBOX=572000.000,4130000.000,590000.000,4155000.000

In that way, viewing the map while building it is possible, through a version of OpenLayers application integrated in MapServer.

This URL requests the mapfile (*syros.map*) and its layers and makes it available via WMS request, GetMap at predefined dimensions, giving specific boundaries and coordinate system.

The mapfile contains the directory where the data is stored. Data is stored at MapServer's *bin* file. It also contains the path to the fonts that are used, as well as the PNG files needed for the symbology. All mentioned above, are stored in MapServer's program files.

In contrast with GeoServer, MapServer (due to its lack of UI), has all layers and styles in ONE mapfile. This one-piece architecture makes things cumbersome. For example, if someone wants to make changes to one layer or one style, the whole mapfile needs to be opened and modified. On the other hand, this can have and a positive side; styling is much simpler than GeoServer. Here, there is no need of creating some lines of XML-based code for each layer. The styling is integrated for every layer in the mapfile in a much simpler form.

For example, for line and polygon layers it is enough to write the color and the width or stroke. For point layers, it is enough to refer to the directory where the PNG files are stored and the name of the corresponding PNG, and finally, its size. All of the above have to be in the right order structurally and by using the proper tags. It is understandable that everything that is displayed in the map is included in the mapfile. That is why the mapfile needed for this attempt had more than 2000 lines.

Anyone could say that this way of data serving is cumbersome. For people who don't occupy with programming, it may be. But at the same time, the simplicity and plainness of the mapfile's structure, facilitates things.

5.5.2 Presentation of the application

As mentioned above, the mapfile is "called" from the URL and in that way the server knows how to display the appropriate data into a map. The application, as well as data is the same. It was attempted the same effort in order for the results to be comparable.

The map is presented below; the map synthesis consists of a map of three different scales which is requested via WMS. In this case too, JavaScript, HTML and CSS where pulled together to create the application. The maps that MapServer produced were the following:

Figure 5.21: Scale 1:150.000-MapServer

Figure 5.22: Scale 1:75.000-MapServer

Figure 5.23: Scale 1:50.000-MapServer

5.5.3 Results and commentation

As with the other two attempts, the final request was to create a web map and distribute it using MapServer on the internet using the WMS specification. Considering not only the final visual result but also the whole course of the process, were drawn various conclusions for each one of the servers and the way they work to build web map applications. MapServer specifically, is an open-source software that is used to display maps online; it is not a GIS-System. The final visual result was similar to GeoServer's and ArcGIS Server's maps. But we also have to take into consideration the procedure that led to the final result as well as its functionality, convenience and capabilities.

MapServer is the oldest server of the three mentioned above and that is reflected to its way of handling the map building procedure. On the other hand, its oldness reflects also its simplicity. However, the display of the map is equivalent to the other two attempts. It takes long time for someone to familiarize with MapServer, but once this is achieved, the result compensates us. Furthermore, MapServer comes with a very detailed and informative documentation, which also facilitates things.

5.6 Presentation of the JavaScript Libraries

Apart from the comparison of the three web map servers, it was attempted to compare the two JavaScript libraries, OpenLayers and Leaflet. For this purpose, the maps where integrated to the application once by using OpenLayers and the other by using Leaflet. The JavaScript code for both libraries is located at the Appendix.

5.6.1 OpenLayers

OpenLayers was used to add interactivity to the map. Once the map was served to the Internet using WMS specification and the application was structured, OpenLayers was used to add tools to enrich the application. The map application with use of OpenLayers is presented below:

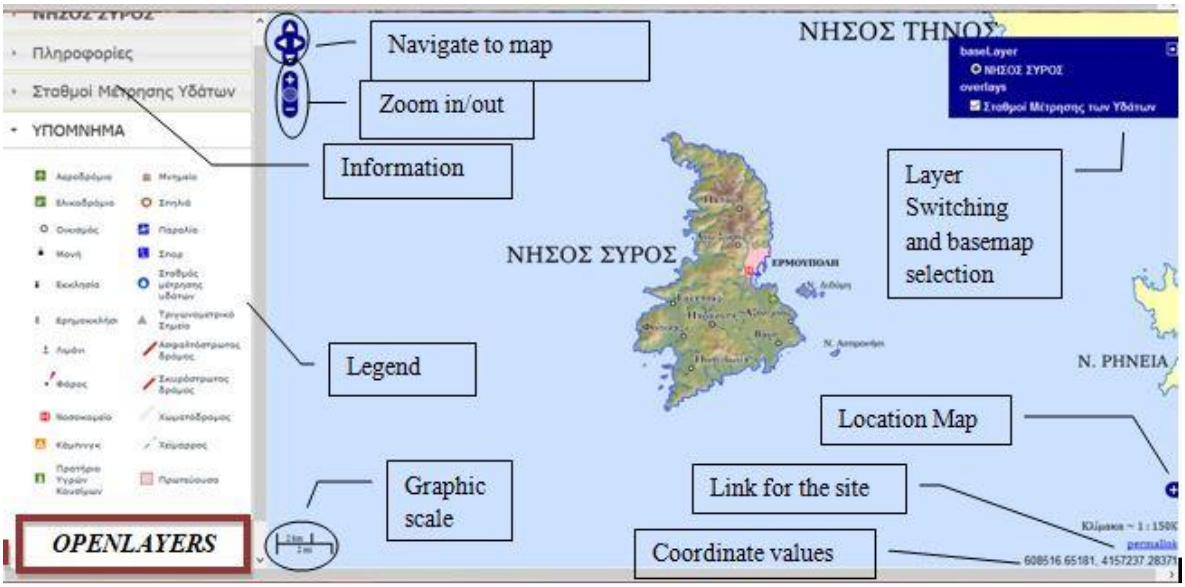

Figure 5.24: OpenLayers Tools

It is worth mentioning that these tools are a subset of all tools that OpenLayers provides. These are the most widely used and necessary for a simple web map application. Additionally, there are some other tools that are Implemented by pressing specific buttons on the user's computer, for example, if someone presses *Ctrl+Enter* simultaneously, the map is being rotated. These buttons are not shown on the screen, though.

5.6.2 Leaflet

The same map is presented, this time by using Leaflet. Indicative tools of Leaflet are presented:

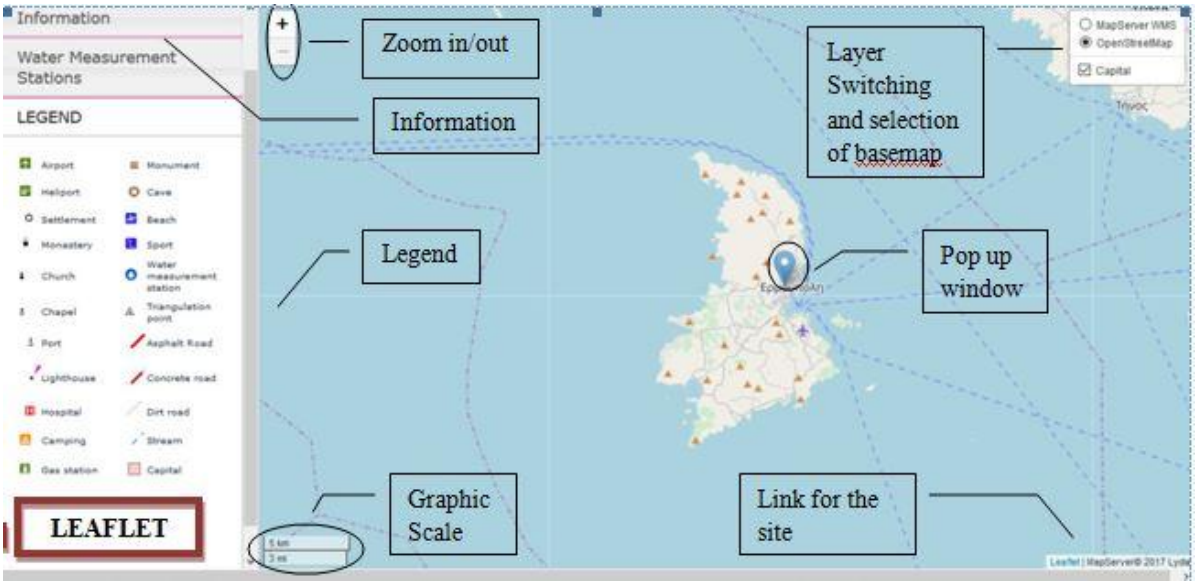

Figure 5.25: Leaflet tools

As it can be seen, both libraries offer the essential tools to customize and add interactivity to such map applications. They also can cooperate perfectly with the other standards like HTML, CSS and JavaScript. Their differences lie on their difficulty and complexity as well as the offered functionalities they can accomplish. Their strengths and weaknesses are discussed on the next chapter. Additionally, they are both FOS software, which means that they can also be easily customizable. There are many tools that are available at Leaflet through plugins. These plugins are created by developers based on the demand of the users. This means that behind FOS Software is a big community that is getting involved with such software and improves their performances and skills. This helps them to become even more popular within the years and people who are occupied with web mapping are leaning eventually towards FOSS.

6. CRITERIA-BASED COMPARISON AND EVALUATION

In this chapter, the three servers and the two JavaScript libraries are examined, compared and evaluated using some criteria that concern their usability and their products from the beginning – their approach, until the end – the final product(s). The criteria concerning the servers are not entirely the same with the criteria concerning the JavaScript libraries. However, most of them are the same.

6.1 Criteria for server evaluation

In this section, the criteria for server comparison and evaluation are presented and described. Concerning the servers, they are further divided into three general categories; general features, server handling and how they handle data and symbolism, namely how they behave and how the map composition is finally achieved through them.

Manufacturing Technology

GeoServer is an open source software written in Java. GeoServer uses the Spring framework for Java application, providing request dispatch architecture for modules implementing OGC services. The web administration application uses Apache Wicket, allowing extensions to contribute additional configuration screens. The application provides a REST API implemented using the spring-mvc-framework.

GeoServer is a web application, supporting any common servlet container (a standalone distribution is available with the Jetty (web server) as an embedded server). GeoWebCache, a Java-based caching component similar to TileCache, is bundled with GeoServer, but available separately. Similarly, GeoServer packages GeoTools as a Java library, which is also available separately. GeoServer is a longstanding application and has undergone several architectural changes. GeoServer 1.0 was built around the STRUTS framework, with the migration to Spring and Wicket taking place for GeoServer 2.0. Early versions of the REST API used restlet before migration to spring-mvc-framework.

MapServer is an open source development environment for building spatially enabled internet applications. It can run as a CGI program or via MapScript which supports several programming languages (using SWIG). It is written in C/C++ and can run as a cross-platform[13] software.

ArcGIS Server is typically deployed on-premises within the organization's service-oriented architecture (SOA) or off-premises in a cloud computing environment. All ArcGIS software is written in C++.

Operating System

By the term operating system (OS), we mean the system software that manages computer hardware and software resources and provides common services for computer programs. It is understandable that the map servers have to be supported by the OS that we work in order to be utilizable. Thus, GeoServer can run on all the major operating systems like Microsoft Windows (it is necessary to have the Java Runtime Environment or *JRE* installed on the computer, though), Mac OS X and Linux.

In addition to this, MapServer also runs on all major operating systems such as Microsoft Windows, Mac OS X, Linux, Solaris and more. Concerning the demand on hardware, minimum hardware specifications are needed for MapServer applications. Therefore a medium machine works just fine. That applies to GeoServer as well.

On the contrary, ArcGIS Server is only compatible with Microsoft Windows and Linux. With regard to MS Windows, minimum operating requirements concern 64-bit versions. 32-bit version is not supported. Notice that basic versions of MS Windows such as 10, 8.1 and 7 are supported for basic testing and application development use only. They are not recommended for deployment in a production environment. The ones recommended for this are:

Windows 10 Pro and Enterprise
Windows 8.1 Pro and Enterprise
Windows 7 Ultimate, Professional, Enterprise, and Home Premium

Also, minimum hardware requirements for ArcGIS Server are 8 GB per unique license role, unless someone wants to use e.g. the GeoAnalytics Server. Then, minimum RAM required is 16 GB.

When it comes to Linux version of ArcGIS Server, the same requirements also apply.

[13] In computing, **cross-platform software** (also **multi-platform software** or **platform-independent software**) is computer software that is implemented on multiple computing platforms. Cross-platform software may be divided into two types; one requires individual building or compilation for each platform that it supports, and the other one can be directly run on any platform without special preparation, e.g., software written in an interpreted language or pre-compiled portable bytecode for which the interpreters or run-time packages are common or standard components of all platforms.

First/last version and Update Frequency

The first version that came out can show us the age of the server and therefore gives us a measure of its stability and consistency. Combined with the last version within years, information for its frequency of updates is elicited. Software updates imply that the software becomes better and fixes errors or bugs.

GeoServer first came out in 2001 by The Open Planning Project (TOPP), a non-profit technology incubator based in New York. Now, 17 years later, it continues to evolve and add many more operations compatible with many different data formats and specifications. Latest version at this time of writing is 2.13. It is worth mentioning that when this thesis was about to start, the latest version of GeoServer was 2.10 and almost a year after, the latest version is 2.13. This means that within a year, GeoServer has released many updates, with no significant differences, but surely each one better than the last. The update frequency here is high.

MapServer's initial release was in 1994. It is the oldest from the other two servers. Current stable release is 7.2.0 and was released at the 27th July 2018. The version used at this thesis was MapServer 7.0.7. As it can be seen, MapServer's updates are more rare and sparse. This has probably to do with the number of community developers supporting the server.

ArcGIS Server was initially released in 2004; therefore is the youngest server of all three. The last stable release was released two years ago, in February 2016 and the current version is 10.6. In table 6.1, the history of ArcGIS version is presented:

Version	Released
9.0	2004-5-01
9.0.1	2005-1-01
9.1	2005-5-01
9.2	2006-11-01
9.3	2008-6-01
9.3.1	2009-4-01
10.0	2010-6-30
10.1	2012-6-11
10.2	2013-7-30
10.2.1	2014-1-07
10.2.2	2014-4-15
10.3	2014-12-10
10.3.1	2015-5-13
10.4	2016-2-18

Table 6.1: ArcGIS Server version history, wikipedia

It can be noticed that first release starts with 9.0; this is the same numbering as the ArcGIS version that existed then. Update frequency is almost yearly. ArcGIS Server has to be updated in order to be also compatible with the corresponding updates of ArcGIS (ArcMap, ArcCatalog etc).

Users/ Community developers

The number of users occupied with the development and improvement of these servers is of major importance. Through communities, many issues are proposed and discussed in order to improve and enrich the already existing services, as well as add new. Proper documentation also helps new users to evolve, learn and -why not- contribute to.

GeoServer fully embraces an open source development model that does not see a split between user and developer, producer and consumer, but instead sees everyone as a valuable contributor. Developers usually start with bug fixes and other small patches, and then move into larger contributions as they learn the system. GeoServer's developers try to keep the code clean and well documented.

Another crucial way for somebody to help out is with documentation. Whether it's adding tutorials or just correcting mistakes, every contribution serves to make the project healthier. And the best part is that someone does not need to be a developer in order to contribute.

The official documentation is contained as part of the official code repository. As part of the GitHub model, anyone can submit patches as pull requests, which will be evaluated by the team. Finally, there is an online list[14] of the most valuable contributors to GeoServer available. As it can be seen, the number is not of minor importance.

MapServer is one of the founding projects of the OSGeo foundation, and is maintained by a growing number of developers (nearing 20) from around the world. It is supported by a diverse group of organizations that fund enhancements and maintenance, and administered within OSGeo by the MapServer Project Steering Committee made up of developers and other contributors. All source code is openly available via GitHub.

MapServer is developed and supported by a rich ecosystem of businesses and individuals around the world. It is also provided a list of service providers who can assist in getting the best out of the MapServer investment. Their services can range from training and technical support to help get started, all the way to specialized development and support services to advance the software and support the organization's mission-critical applications.

[14] http://old.geoserver.org/Contributors.html

MapServer also has a list of requests made by people who use this server (its community in general). These requests concern error fixes but mostly addenda and extensions that they would like to see integrated to MapServer in the future, for example support for curved labels, raster color corrections, label priority etc. A complete list of these suggestions can be found on: *https://mapserver.org/development/index.html*.

These suggestions aim to make MapServer better and easier to use as well as more powerful. They are discussed and it is decided whether they will be finally applied or not. The results lead to the update of the current version available. MapServer finally, has a very detailed, instructive and enlightening documentation which is necessary for a beginner as well as for the more engaged with web mapping.

ArcGIS Server has its own website for information that concerns the server and its services. It has not a concentrated documentation, but the site is helpful. Users that use ArcGIS Server are mostly organizations who work with GIS services and techniques and want to make available their geographic information through internet. The server is maintained by ESRI.

Cost/Expenses

Nowadays, with the growing number of developers occupied with free and open source software, their abilities have enormously increased. It has adopted a new trend that has also affected web mapping to turn to FOSS. MapServer and GeoServer are free and open-source[15] software. With the free spread of those technologies, the number of people which start to evolve with them has increased. That means that the accessibility to these open source servers has aroused the interest of novices and has developed their creativity. This is very important, because as the number of people that occupy with web mapping grows, the results are becoming more remarkable and this field evolves. It also shows that after all, anyone can build good-looking web map applications and does not have to be a programmer to do it.

On the other hand, ArcGIS Server is a proprietary software. This means that its code cannot be modified. It also means that in order for someone to use it, he has to pay money. This money can reach a number of tens thousands of Euros. Lest we forget that if someone wants to use ArcGIS Server, he has to have also ArcGIS Desktop installed. This doubles expenses, as ArcGIS Desktop is also proprietary. The only reason for someone to do this is to be a big commercial organization which produces and probably sells its own web map applications. Therefore, this is not a solution for individuals, which can use it for just a hobby after all.

[15] **Open-source software** (**OSS**) is a type of computer software whose source code is released under a license in which the copyright holder grants users the rights to study, change, and distribute the software to anyone and for any purpose. Open-source software may be developed in a collaborative public manner. According to scientists who studied it, open-source software is a prominent example of open collaboration.

Scalability

Scalability is the capability of a system, network, or process to handle a growing amount of work, or its potential to be enlarged to accommodate that growth. For example, a system is considered scalable if it is capable of increasing its total output under an increased load when resources (typically hardware) are added.

Scalability, as a property of systems, is generally difficult to define and in any particular case it is necessary to define the specific requirements for scalability on those dimensions that are deemed important. It is a highly significant issue in electronics systems, databases, routers, and networking. A system, whose performance improves after adding hardware, proportionally to the capacity added, is said to be a *scalable system.*

Talking about scalability with regard to the servers which are practically software, it goes to the potential extensions they might have. These extensions also concern software at this case and no hardware, as it is in the strict sense of the term.

Extensions in GeoServer add an extra functionality to the server. They are installed as add-ons to the basic installation of GeoServer. These extensions mostly concern extra and more uncommon and rare output formats such as DXF format for WFS, Excel Output format, Vector Tiles, Monitoring etc. These functions are not used often, they concern more complicated operations.

MapServer does not seem to have such extensions or any other type of extensions that promote scalability.

ArcGIS Server as another kind of software (proprietary and not open source) has its own forms of scalability and these concern software as well as hardware. It goes without saying that when it comes to this server, it is assumed that it is a very powerful tool. ArcGIS Server offers expansion from one ArcGIS Server machine to multiple machines. This can be achieved by sharing the corresponding paths and move the new machines to the same site.

In other words, ArcGIS Server has a scalable architecture that allows deployment sizes ranging from one to many machines. It may be needed to consider a distributed installation of ArcGIS Server so that an acceptable level of performance for the number of users accessing the system can be achieved.

All machines in an ArcGIS Server site have the ArcGIS Server component installed. On the first ArcGIS Server machine the configuration happens, it is needed to create the site. Subsequently, additional ArcGIS Server machines to the site can be added or joined. Each ArcGIS Server in the site must be at the same version number and be licensed exactly the same. This concerns of course, again, big organizations which handle big amounts of geospatial data.

The above information concerns the general characteristics of each server. The next category is about some more qualitative criteria which are about the performances of each server and their characteristics concerning the map composition.

Time

Time is an important factor for comparing this kind of software and especially to obtain valuable results for the convenience that they provide. It is reasonable to assume that when using proprietary software the knowledge needs are limited; that happens because everything that someone needs is provided straight from the software without needing to write lines of code.

GeoServer does not take much time to implement, provided that someone gets familiar with the SLD specification. Its interface can help even and beginners to comprehend its functionalities. MapServer on the other hand, is much more difficult to comprehend. However, the time required is as similar to GeoServer's.

ArcGIS Server is the simplest of all three. That happens probably due to the fact that it is proprietary. This means that no coding is required, the user interface is very friendly and simple to use, even for a beginner. With basic knowledge of how to compose a map in ArcMap, the whole process until the map is uploaded to the ArcGIS Server is very fast and easy. It takes almost only one week to finish the map composition and just a few minutes for the uploading to the server.

Easiness and convenience in learning

By this term it is meant how easy and quick it is to "learn" the software. All three are of similar philosophy and architecture. Therefore, when someone attempts to learn the first of these technologies, then it is easier to learn the others. Of course that depends on each learner's personal ability. This last one is called learning curve. The learning curve depends exclusively on the learner and not the software. It depends on how much the user comprehends and how many repetitions he does in order to understand it.

GeoServer is easy at learning. In this, crucial role play the tutorials, the documentation and the community. By community here it is meant, people who had in the past similar problems, post the solution they used in order to solve the problem each time through answering questions on the internet. That depends on how many users there are that use this software. GeoServer has many of them.

MapServer is more difficult to learn. That happens not only because there are fewer users available for questioning, but also because MapServer has a different architectural structure which is more complicated than GeoServer's. It introduces users in a more "programming environment". In this, the lack of interface advocates for it.

ArcGIS Server is very simple and handy. There is almost no need of learning anything. Whatever is needed, it can easily be accomplished. That happens with all proprietary software, they are more practical.

Easiness and convenience for implementation

This criterion is a consequence of the previous one. When someone learns something in theory, then the implementation follows. Sometimes it is hard to segue from theory into practice. Whether this is achieved or not, depends more on the way the software is structured, rather than on the learner's ability. Because there was a very detailed documentation available and many articles online as well as wish of learning, the implementation at all three was not difficult. Of course, again, ArcGIS Server is the easier to learn of all three. This also happens because the user is not obliged to write scripts or code. Whenever the map composition is ready, it is just uploaded to the server with no need of building symbolization.

Need in programming knowledge

Sometimes a little knowledge in programming for web mapping applications is required. Apart from the knowledge needed in order to form the final application (e.g. HTML, CSS etc), programming knowledge is also required at the step of managing and publishing the data through the server.

More specifically, it was needed to write multiple lines of code when building the symbolization and the styling of GeoServer. This happened, because Geoserver uses the SLD specification for the styling of geospatial data. In addition to this, it is necessary for every layer to be associated to its own SLD file, so the more the layers are, the more need in coding is required. SLD specification uses XML (eXtensive Mark-Up Language).

MapServer requires a whole different approach; that is also affected by the needs in programming. MapServer's core file, the Mapfile, is on its own a file written in a type of "programming language". It is not written in an actual programming language, but it has all of its elements; the structure, the sequence, the start and the corresponding end etc. The styling in MapServer is included into the mapfile. Therefore, it can be told that styling also here needs a minor knowledge in programming.

On the other hand, ArcGIS Server does not require programming at all. As it happens with all proprietary software, its approach is just this: not to bother the users with a single line of programming.

Available tutorials and documentation

This factor is of major importance, especially when it comes to free and open-source software such as GeoServer and MapServer. Some of the coding comes from volunteers, namely simple users who like to contribute to open source software. Therefore, documentation is important to understand the philosophy and the steps required to accomplish the target each time.

MapServer offers a very handy and comprehensive documentation of approximately 200 pages. This documentation can be found online at MapServer's official site and it can also be downloaded in a PDF format. The documentation is very useful, even for advanced users, as it contains simple issues as well as more complicated issues concerning higher–level applications. At MapServer's official web page can also be found a few tutorials. These tutorials are not as enlightening as they concern the simpler of all operations that MapServer does, for example single vector layer viewing.

GeoServer also offers a very detailed user manual that is available only online. Nevertheless, it contains everything a novish or an experienced user needs in order to handle this software. It also provides some tutorials but contrary to MapServer, they concern more complicated actions. GeoServer also provides examples for the styling, namely SLD files that can be downloaded or XML code that can be copied. These examples are about all kinds of symbols, for vector data such as polygon, line, label or point as well as for raster data. These examples are very helpful to the understanding of how SLD is structured.

Finally, ArcGIS Server also provides some instructions on the official site of ESRI Enterprise that concern ArcGIS Server, but mainly they refer to its architecture and which operations it is capable of. It also provides some tutorials which are about the publishing of the different services it can handle. There is no need for extra documentation as this server is very simple to use.

Convenience (easiness to use)

Based on the above described characteristics, it is easy to come to conclusions about the easiness and convenience of each server. The convenience in using depends on many factors; most of them are discussed above. Easiness and convenience at handling does not always come with the capabilities. In order of easiness, ArcGIS Server would be classified as first; following by GeoServer and finally, less convenient would be MapServer.

The previous criteria that concern the servers themselves are presented in a summarized table that follows:

	Map Servers		
	MapServer	GeoServer	ArcGIS Server
Programming language	*C/C++*	*Java*	*C++*
Proprietary or open source	*free and open source*	*free and open source*	*proprietary*
Supported operating systems	*all*	*all*	*MS and Linux*
Initial release	*1994*	*2001*	*2004*
Number of developers	*a few*	*more than MapServer*	*ESRI's developers*
Cost	*free*	*free*	*too expensive*
Scalability	*no*	*yes*	*yes*
Time to complete a project	*takes a lot*	*takes a lot*	*very little*
Easiness at learning	*no*	*yes*	*yes*
Easiness at implementing	*yes*	*yes*	*yes*
Needs in programming	*yes*	*yes*	*no*
Tutorials and documentation	*yes*	*yes*	*a few*

Table 6.2: Server characteristics

The last category of comparison is about *Data and Symbolism*, namely how the server handles and finally displays geospatial data to the final user (client).

Quality of the map synthesis

This criterion concerns the final image of the map which is displayed to the client. With all three servers the result was of higher standards than expected. There are differences when someone uploads a map on the internet with a map that is printed on paper. However the quality of the map, as it can be seen, was very satisfactory with all three servers. ArcGIS Server offers the same quality with ArcMap, and GeoServer also provided a high-quality map synthesis like MapServer also accomplished. Of course, graphics play major role to this result. What is significant is that all three servers carried it out equally good.

A parameter that plays major role to a map composition when it is displayed at a screen is *antialiasing* as mentioned in a previous chapter. Antialiasing is a software technique for smoothing the jagged appearance of curved or diagonal lines caused by poor resolution of a display screen. These curved or diagonal lines can also be found at annotation labels, too. This happens because the differences between high value pixels (annotation, usually dark) and low value pixels (sea or terrain) are big. This causes high contrast at these edges so this effect has to be reduced. Result of antialiasing is that the aesthetic quality of the map is better and also realism to the image is added (Tsoulos & Skopeliti). GeoServer supports this technique as well as MapServer and ArcGIS Server. Especially for MapServer, antialiasing is added by simply setting at every layer the parameter antialiasing as “TRUE”.

A second parameter that plays important role to the quality of the web map is the tiling. Tiling is responsible for the fast rendering of the map image. In that way image rendering takes less time. This improves the performances of the server and it is supported from all the servers. According to surveys though (Ballatore et al.), MapServer has a better performance than GeoServer as well as greater stability and scalability.

Symbolization structure

Another element that is worth comparing is the way the three servers build the symbolization of geospatial data. It has been mentioned that each one of these servers uses a different way for styling the data. That luckily does not affect the visual result. The same symbols were used by all three servers. GeoServer, as it is already known, uses the SLD specification for styling. It can also support YSLD (an extension of SLD). MapServer builds symbolization into the mapfile by including some lines which concern the styling parameters and finally ArcGIS Server is not associated with styling. Styling is applied on ArcMap at the phase of map creation. ArcMap has an embedded library of symbols which are used to apply styling to the data.

Other elements that can be compared are the supported data formats and coordinate systems, and if they can perform actions beyond just map publishing, like querying and tiling.

Both GeoServer and MapServer, but also ArcGIS Server supports a plethora of coordinate reference systems. For GeoServer, this number reaches the 4000 different coordinate reference systems and projections. MapServer relies on the PROJ.4[16] library for projecting map data. It is used also by GDAL and a multitude of other Open Source GIS libraries. ArcMap also supports the most common as well as less known CRSs.

As for data formats, they also offer a wide variety of input and output formats. All ordinary data formats such as ESRI's shapefile, GeoTIFF, GeoJSON etc are supported, as well as all known open source geodatabases. ArcGIS Server also supports ESRI's File Geodatabase and can also connect to other proprietary Geodatabases, as well as simple relational databases, such as the open source one, Altibase.

These servers also offer the ability of querying data. GeoServer supports the use of both CQL and ECQL in WMS and WFS requests, as well as in GeoServer's SLD dynamic symbolizers. Whenever the documentation refers to CQL, ECQL syntax can be used as well. CQL (Common Query Language) is a query language created by the OGC for the Catalogue Web Services specification. Unlike the XML-based Filter Encoding

[16] https://proj4.org/

language, CQL is written using a familiar text-based syntax. It is thus more readable and better-suited for manual authoring.

However, CQL has some limitations. For example it cannot encode id filters, and it requires an attribute to be on the left side of any comparison operator. For this reason, GeoServer provides an extended version of CQL called ECQL. ECQL removes the limitations of CQL, providing a more flexible language with stronger similarities with SQL.

In MapServer, map layers can be queried to select features using spatial query methods or the attribute query method. They use a pseudocode that is consistent with the language independent API reference and each line is a statement. It has not been examined in depth how this works. ArcGIS Server also provides querying operations since version 10.1.

Lastly, all three offer anti-aliasing and Tile Caching as mentioned earlier. Especially for MapServer, it can feed tile-based map clients directly using the CGI "tile mode". Tile-based map clients work by dividing the map of the world up into a discrete number of zoom levels, each partitioned into a number of identically sized "tiles". Instead of accessing a map by requesting a bounding box, a tile client builds a map by accessing individual tiles. Tile requests are handled by the '*mapserv*' CGI program. In order to return tiles in the correct projection, MapServer must be built with the –*use-proj* option turned on. That depends on the version used.

6.2 Overall evaluation based on the criteria

In the previous paragraphs the criteria that were used in order to compare the various map servers were presented. From the above criteria we came to the conclusion that each one of these servers is capable of publishing a very good web map. Apart from the visual result, these servers are capable of putting through other operations too. When someone wants to make a web map application for the internet, all three are appropriate. Of course, they also have some differences, which mostly concern the way they handle data and the process that someone has to follow in order to reach the final result. Moreover, these three servers show another major difference; they are not all free and open-source (FOSS), ArcGIS is proprietary. These arises another question on how useful it finally is to use a proprietary software which costs a lot of money than using a well-built and well-documented FOSS which practically performs the same or even a little few operations.

Finally, apart from the way they work and their technical characteristics, there are also differences on how easy it is to learn them and implement them as well as how they handle the data symbolism. The offered services and supported data formats specifically are presented below in an aggregated table to help readers compare the

servers easier. The blue dot (•) indicates that the service is available only by using extensions.

As it can be seen, for implementing a simple web map application, all three servers can function equally well. They offer almost the same services and data formats. Their real differences refer to the way they behave and operate, not really on the final result they produce.

		GeoServer	MapServer	ArcGIS Server
Coordinate Reference Systems (CRS)		~4000	PROJ.4 library	all ESRI's ArcMap
Vector data	ESRI shapefile	●	●	●
	Java Properties file	●		
	GML	●	●	
	VPF	●		
	GPX	●	●	
	KML	●	●	●
	DGN (cad)		●	●
	SVG	●		
	DXF, DWG		●	●
	GeoJSON	●	●	
	MS excel (.xls, .xlsx)	●	●	●
	OSM data		●	
	Annotation			●
	Text files (.txt)			●
	MOSS			●
Raster data	GeoTIFF (.tif)	●	all formats supported by GDAL library	●
	GTOPO30	●		
	WorldImage	●		
	ESRI ArcGrid	●		●
	ERDAS (.ecw, .img)	●		●
	JPEG			●
	Oracle Spatial Georaster	●		
	TIN			●
	National Imagery Transmission Format (NITF) (.ntf)			●
	Hierarchical Data Format (HDF)			●
	DTED Level 0, 1, and 2 (.dt*)			●
Spatial DBs	PostGIS	●	●	
	H2	●		
	ESRI ArcSDE	●		
	DB2	●		
	MySQL	●	●	
	Oracle Spatial	●	●	
	MS SQL	●	●	
	Teradata	●		●
	JNDI	●		
	SpatiaLite		●	
	ESRI mdb		●	
	ESRI File Geodatabase(.gdb)		●	●
	ESRI Personal Geodatabase (.mdb)			●
OGC Services	WMS	●	●	●
	WFS	●	●	●
	WCS	●	●	●
	WFS-T (Transactional)	●		●
	WMTS(web map tile service)	●	●	●
	WPS	●		●
	CSW	●	●	
Styling	SLD	●	within the mapfile	ArcMap integrated symbols
	CSS	●		
	YSLD	●		
Tiling		GeoWebCache	via CGI's tile mode	ESRI Tile Caching
Query		Supported	Supported	Supported
Anti-aliasing		Supported	Supported	Supported

Table 6.3: Supported functionalities and data formats per server

Due to the fact that the criteria used to make the comparison and the evaluation of the servers are not quantitative but qualitative, it might be difficult for the reader to understand the advantages and disadvantages of each software. For this reason, it was considered as appropriate to quantify the most important criteria and present them on a table as well as on a spider diagram. By this attempt, the results are better understandable. The comparison scale is arbitrary and has a gradation from 0 to 5, with 0 means none or nothing and 5 much or many (for example for the criterion *low cost,* 0 will mean expensive and 5 no cost at all) and for more qualitative characteristics such as convenience, 0 means no convenience and 5 very convenient.

Criterion / ***Server***	**GeoServer**	**MapServer**	**ArcGIS Server**
Frequency of updates	5	3	4
Community	5	4	3
Low Cost	5	5	0
Scalability	5	2	3
Time needed	4	4	1
Convenience at learning	4	2	5
Convenience at implementing	4	2	5
Minor needs in programming knowledge	3	1	5
Tutorials-Documentation	5	4	4
Overall convenience	4	2	5
Quality of the map synthesis	5	5	5
Symbolization (easy or difficult to implement)	1	3	5
Supported data formats	5	5	4

Table 6.4: Quantification of criteria

The criteria are shown for each server graphically through as spider diagram, so as to be more comprehensible.

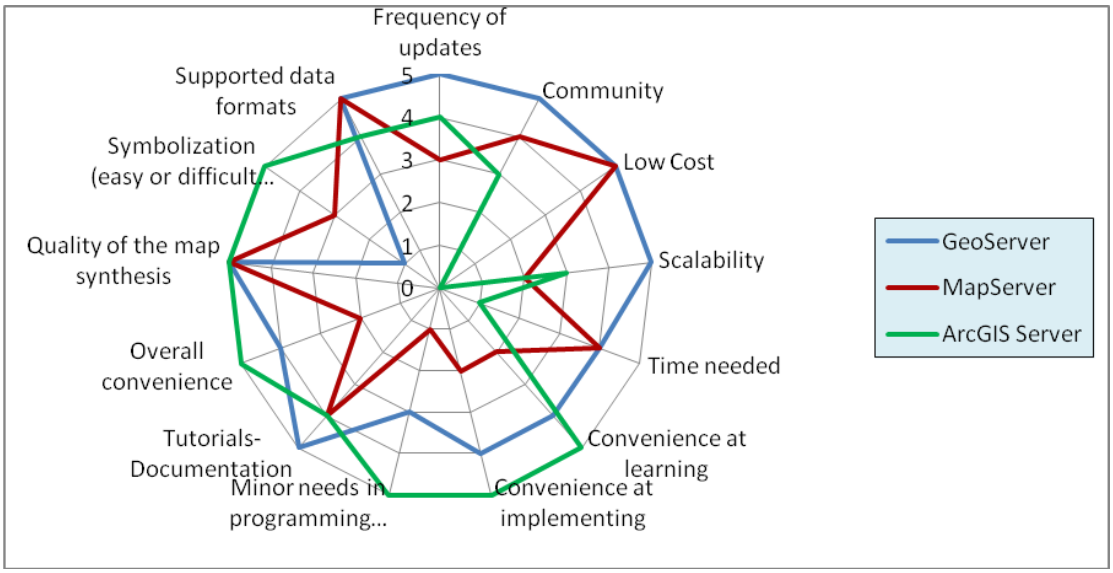

Diagram 6.1: Server characteristics

6.3 Criteria for the comparison of JavaScript libraries

As with the map servers, there are also some criteria regarding JavaScript libraries. Some of them are the same with the servers'; some other though are slightly different. The libraries compared here are OpenLayers and Leaflet, both FOS software. The criteria are divided into two subcategories; criteria concerning their handling and criteria which concern their functionalities. Regarding their handling, the criteria are the following:

Time needed

As with the servers too, this is an important clue for choosing the appropriate tool. It is not the most important, though. There was no huge difference between the two libraries for writing the code. OpenLayers is a richer software which has more scripts that can be added to enrich a web map application. Therefore, more time was needed in order to delve into its capabilities and choose the appropriate scripts. Leaflet needed less time to build the code because its offered capabilities are limited.

Another fact that extended the time needed to create the application using OpenLayers was that at the time of beginning this thesis, the first attempt with GeoServer, used OpenLayers 2. On September 2017, OpenLayers 4 had already been released and was used to detect possible differences. Unlikely, OpenLayers 4 is completely different than the other older versions. Its structure is different and somebody who has learned working with OL2 will have a little difficulty to get familiar with OL4. In the meantime (September 2018), OpenLayers 5 was released. This shows that updates at OL are more frequent than Leaflet's. Leaflet has been

updated to version 1.3.4, while the version used was 0.4. That means that more developers are occupied with OpenLayers, rather than with Leaflet.

Available Tutorials and Documentation

Both OpenLayers and Leaflet have a detailed documentation which explains in depth the operations that they offer. They also have tutorials for the most common tasks, in order to help beginners start from simple things and move to more fancy staff.

Demands on programming

Both libraries are written in JavaScript. JavaScript is of course a programming language. But the nature and purpose of this software, namely the JavaScript libraries', is to minimize someone's needs to programming. They provide scripts, each one for performing a specific action, and the user just needs to put them in an order in a specific file. It is not recommended that the user has zero relation to JavaScript language. The user must know to recognize when a code is written to JavaScript as well as to be able to make modifications depending on the needs of the particular application.

Convenience at learning and implementing

The fact that both libraries offer a very good/detailed documentation quickens the procedure of learning. It is acceptable that OpenLayers code is more complex than Leaflet's. It also requires more lines of coding. However, the convenience at learning is comparable; the paradigms and tutorials are very helpful and detailed. As for the implemention, it was easier to implement OpenLayers to the application, than Leaflet. That probably happened because Leaflet is not as detailed as OpenLayers and has only the basic operations, which they may not be always compatible. Some scripts did not work at first, specifically what concerns the coordinate reference systems.

The second subcategory is about their *functionality*, as mentioned above. That has to do with the offered operations they are capable of perform, namely the operations they are capable of performing on a web map application. These characteristics are analyzed as detailed below.

Tools for scale switching and panning

First of all, there are basic operations that these libraries perform and they are nothing else but the tools for building an *interactive* or *slippy* map. Maps of the same area in different scales are presented at the same window on the screen. In order for the different be shown up on demand, there are buttons that do it. These are the *zoom in* and *zoom out* buttons which switch the map display regarding the desirable scale. There is also another operation, *panning*, which moves the map on the desirable position each time. These operations are to be found at OpenLayers as well as at Leaflet. These operations are the most basic for all web map applications as

they add interactivity between the map(s) and the user. OpenLayers supports more tools than Leaflet. With OL is possible the rotation of the map and extending the map to full screen. These functions are supported only to the latest version of OpenLayers, namely OL4 and OL5. The common operations are the navigation buttons, the scale line, the pop-ups and the bounding box. OpenLayers also supports some operations such as the bounding box selection, the sea level and the overview map.

Vector layer overlay

There are instances where the cartographer doesn't want a layer with specific data to be displayed along with the basic data. This basic data often consist the *base layer* and the overlays are data that are complementary. For example, the map of Greece can have as an overlay a specific layer depending on the application, which shows NATURA regions. This layer is seperated for the rest map and it is "called" through WMS seperately as vector overlay. These vector overlays can be switched on or off on demand and their difference from the base layer (the map) is that the map is published as a raster image (JPEG, PNG etc) and these overlays are of vector data. Both libraries support this operation, which is very significant, as there are many times that the space for the map display is small, especially when it comes to computer screens. The information that a map offers can sometimes be too much, so a good solution is to switch on and off the secondary layers (the overlays), so as the user is not confused.

Using a basemap from another server

Another operation that is very helpful and handy and makes a very beautiful visual result is the addition of a certain basemap. This basemap can be found on other external map sites such as OpenStreetMap (OSM) etc. Subsequently, it can be attached to the main map application and create the so named *cartographic mashup*. By this term it is meant a map synthesis which consists of data from different sources, all requested by WMS specification which are all put together to produce an integrated map synthesis. The basemaps can be requested from other servers that provide free data to the Internet, such as OpenStreetMap (OSM) or Greek Cadastre (http://gis.ktimanet.gr). With a basemap like this, it is possible to add from GeoServer or MapServer our own data as an overlay to these basemaps. This function is also supported from both JavaScript libraries.

Change of the cartographic projection (Reprojection)

Reprojection is the operation with which the coordinates of data of a certain map are transformed in another coordinate system. It is another useful operation but is supported only by OpenLayers. Leaflet can project geographical coordinates into a 2D point. Leaflet also has some pre-defined projections to use, like the Elliptical or Spherical Mercator projection. Disadvantage of Leaflet is that Leaflet projects only geographical coordinates. OpenLayers is easier to reproject; the only thing needed is to specify the CRS which we want the data to be reprojected.

Map Layout and Marginalia

This is another important factor which should be taken into consideration regarding web maps. The architectural elements of the map should all fit to the screen, like in printed maps. Therefore, basic elements which must accompany the map such as the scale (numerical or graphic), the location map or even the cartographic canvas, should be also put in the application. These elements are put by the JavaScript libraries, around the bounding box of the map. The most basic are the scale (graphic is preferred) and the location map. Both libraries support these elements and they automatically place them at the corners of the box that contains the map.

Retrieval of certain elements of the map / Selection of a subtotal of data

The most basic operation for distributing maps over the internet is the WMS and WFS request. Through WMS we can request the data from the server and use JavaScript libraries to display them into an application and when using WFS, specific data can be requested or queried. That means that when the user clicks on a specific element on the map, information for this can be retrieved. This information is available only if the layer(s) are queryable using *WFS GetFeatureInfo*. These operations are performed also by the JavaScript libraries. Information for the specific element is shown within a pop-up window which contains information about the data. This information can be either spatial or descriptive. In addition to these, OpenLayers also support WMTS. Especially, when it comes to OpenLayers, it has also the ability to export the map as a PDF locally on the computer if someone wants so. Finally it is also possible to download data directly locally on the computer. In order to do so, the data have to be geoJSON.

Quality of visual result

Another factor that affects the comparison of the libraries mentioned above is the quality of the visual result they offer. With Openlayers, and specifically with OpenLayers 4, the appearance of the map is very good-looking; the buttons were also located on the four sides of the mapping box. Some other functions that could not be put extra on the corners because there would be overlapping, where available by pressing buttons on the computer. This last capability of OL is very handy and practical. On the other hand, Leaflet presented the less appealing result, without meaning that it was bad. This criterion is more subjective rather than objective, so it is not to be taken into consideration.

Mobile Support

Mobile support is another element that must be taken into account for choosing the proper library according to the application. If the web map application is aimed at mobile telephones and Smartphones or tablets, this factor must be examined. OpenLayers offers specific controls for mobile applications, as their software is different. Here is a table that shows how different mobile browsers handle the capabilities that OpenLayers offers:

Mobile Browser Support

Browser	Touch events	Multiple touches	Accelerometer	geolocation
iOS (4.x)	yes	yes	yes	yes
iOS (3.x)	yes	yes	no	yes
iOS (2.x)	yes	yes	no	?
iOS (1.x)	yes	no	no	?
Android	yes	no(2)	no	yes
Opera Mobile	no	no	no	yes
Symbian	no	no	no	no
IE7 (WP7)	no	no	no	no
Firefox 4(1)	no	no	no	yes

Table 6.5: Mobile Browser Support for OpenLayers (OpenLayers documentation)

Leaflet also offers an interface for mobile applications and tools like *geolocation* (this is: detecting the location of the phone and shows it on the map). It can also create a full-screen map tuned for mobile devices. More specifically, it can create the map at the proper width and length to fit the screen of the phone. It is supported by iPhone, iPad or even Android phones. Leaflet may be more suitable for mobile applications because it is more lightweight than OpenLayers and performs the basic operations that a mobile application may need. This means that except for being lightweight software, it also has to be simple to implement. Thus, Leaflet is probably more appropriate for this purpose.

	Leaflet	
Browser support	**Desktop**	**Mobile**
	Chrome	*Safari for iOS 7+*
	Firefox	*Android Browser 2,2+*
	Safari 5+	*Chrome for mobile*
	IE 7-11	*Firefox for mobile*
	Edge	*IE for Win8+ devices*
	Opera 12+	

Table 6.6: Browser support for Leaflet

Compatibility with other specifications

Both JavaScript libraries can cooperate equally well with other programming languages such as HTML and CSS and being combined so as to produce the final interface of the web map application. HTML is used to build the architectural structure of the web page, CSS is used to determine the styling (colors etc) of the web page and finally JavaScript determines the "behavior" of the site. This last sentence means that JavaScript adds interactivity with the user and determines which functions have to be done on a certain mouse-click.

6.4 Overview of the JavaScript libraries

The libraries used, namely OpenLayers and Leaflet, are functioning in a similar way. They are both client-side libraries written in JavaScript and are used for adding interaction to web map applications for the internet. OpenLayers is a "lower level API[17]", which means that it requires more JavaScript coding to initialize and set up the map. This can be convenient when there are customized and complex requirements, as the API allows more control over the map and data. Nevertheless, with a more robust API comes a steeper learning curve, so more time may be needed. On the other hand, Leaflet is very easy on handling the most common mapping tasks like consuming base map tiles, panning, and zooming, and its API is easy to understand and simple to use. Once it goes beyond the common tasks, however, there is a need of using Leaflet Plugins and this is where things become more complicated. There might be functionality that unfortunately doesn't exist. However both libraries are well documented and work solidly on mobile devices.

Below (Table 6.4) is presented a table which shows some of the most important characteristics of each library.

	Libraries	
	OpenLayers	**Leaflet**
Programming language	*JavaScript*	*JavaScript*
Proprietary or open source	*free and open source*	*free and open source*
Supported operating systems	*all*	*all*
Initial release	*2006*	*2011*
Number of developers	*many*	*less than OpenLayers*

[17] In computer programming, an **application programming interface (API)** is a set of subroutine definitions, communication protocols, and tools for building software. In general terms, it is a set of clearly defined methods of communication between various components. A good API makes it easier to develop a computer program by providing all the building blocks, which are then put together by the programmer. (Wikipedia.org)

Cost	*free*	*free*
Scalability	*yes*	*yes*
Time to complete a project	*takes more than Leaflet*	*takes a little*
Easiness at learning	*yes*	*yes*
Easiness at implementing	*yes*	*yes*
Needs on programming	*yes*	*yes*
Tutorials and documentation	*yes*	*yes*
Mobile support	*yes*	*yes*
Scripts	*big*	*small*
Reprojection	*yes*	*no*
WMS & WFS Support	*yes*	*yes*
Compatibility with other specifications	*yes*	*yes*

Table 6.7: Summarized table of libraries' main characteristics

As with servers, libraries' main criteria where also quantified and presented on a table using the same scale as before.

Library / ***Criterion***	**OpenLayers**	**Leaflet**
Time needed	5	3
Tutorials-Documentation	5	5
Demands on programming	4	4
Convenience at learning	3	4
Convenience at implementing	5	3
Supported operations	5	2
Quality of Visual result	5	4
Compatibility with mobile devices	5	5
Lightweight software	2	5
Compatibility with browsers	5	5
Compatibility with other specifications	5	5

Table 6.8: Quantified main criteria of libraries

The corresponding diagram is presented as follows:

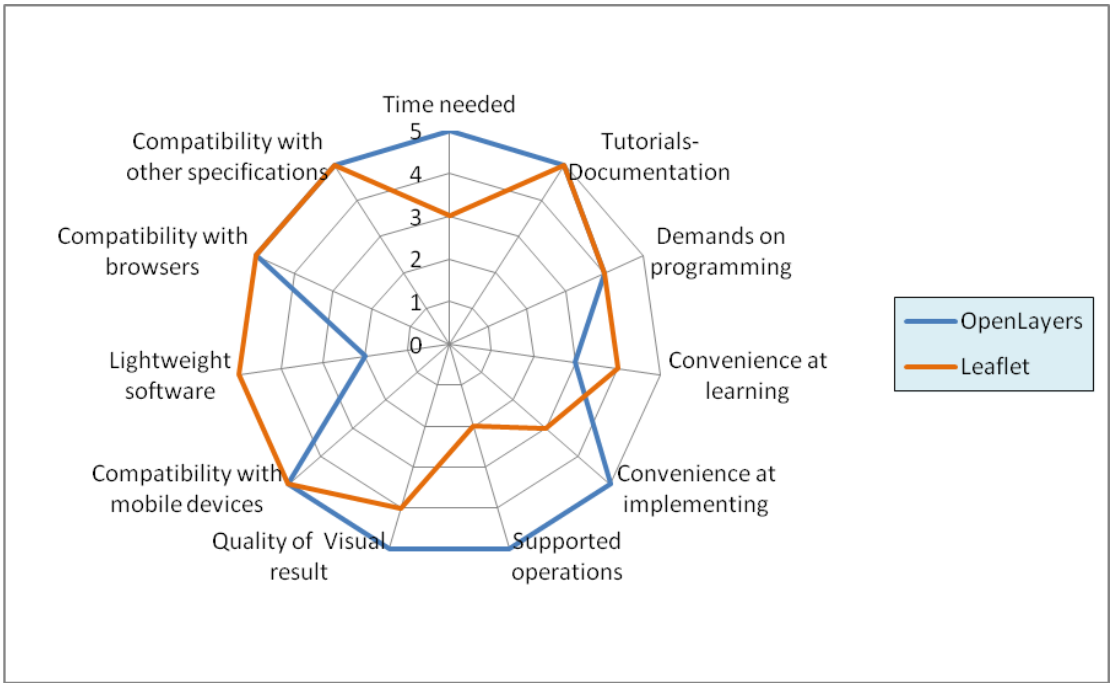

Diagram 6.2: Spider-diagram for libraries

7. REVIEW AND CONCLUSIONS

In this thesis, three of the most well-known web map servers used for building web map applications are compared and evaluated. Two of the most widely used JavaScript libraries are also compared. The comparison was made by creating a web map application of a map for the Internet at three different scales using these specific servers and libraries. The servers chosen are GeoServer, MapServer and ArcGIS Server. It is worth mentioning that the last web map server is neither free nor open-source software, but proprietary. The question then arises; is it worthy to have that kind of software where at the same moment there are many other free and open-source (FOSS) that do the same job? Cost is a very important factor when choosing the appropriate software. Another important factor though, is the number and kind of functionalities that each software offers. For mapping applications, all three servers can respond very well. They may not have similarities on the way they handle and visualize data, but they can all produce high level maps. Especially what concerns visualization, namely the symbolization of the data, GeoServer uses an OGC standard - the SLD (Styled Layer Descriptor), MapServer integrates styling into the mapfile and ArcGIS Server contains a library of pre-existed symbols within ArcMap and this is their biggest difference. These software components were also compared with respect to the time needed for the application to be accomplished, their characteristics and the type of available formats and functionalities that they can support. GeoServer, as well as MapServer, have a wide community of users and that has helped them to develop with very rapid steps. It also helps them to be frequently updated and to keep pace with the contemporary technologies. These FOS softwares are gaining even more space regarding web mapping applications. It is very important, because it also makes non-professionals to occupy with such applications and the community of web cartographers increases. Therefore, its spreading becomes bigger with the years. Interoperability is increasingly becoming a focus point for organizations that distribute and share data over the Internet. Therefore, the support of OGC Services, such as WMS, WFS, WPS, is mandatory and fortunately exists at all three servers.

What has to be taken into consideration is that FOS software as GeoServer and MapServer focus mainly on the map synthesis and display whereas proprietary software like ArcGIS Server can handle more complex tasks such as data analysis.

Concerning the JavaScript mapping frameworks, it was concluded that both libraries can respond to simple web map applications equally good and effective. Leaflet though is simpler to adjust, because it requires less code than OpenLayers as it is a higher-level API. On the other hand, OpenLayers is more complicated but it also contains more functions that concern more complex applications. Regarding the libraries, we can say that they have more similarities than differences. They are both

written in JavaScript, they have the fundamental tools to support a nice web application, they can cooperate with other standards such as HTML and CSS but also they can respond very well to mobile map applications with great success. So, the choice of the more appropriate library clearly depends on the type and level of complexity of the application exclusively. However, there are still more interactivity tools that have to be taken into consideration when making up such applications. These concern cartographic tools such as real-time generalization, alternative ways of data rendering, etc. This form of interactivity should be taken into consideration for the future of web mapping and especially when they are going to be used for educational purposes (Ntzelepis et al., 2014)

To summarize, it was concluded that the main component for creating web map applications is the web map server because it specifies the number/kind of capabilities that are offered, the types of spatial data that can be used and the services that can be supported. Nowadays the volume of produced data has increased and includes sources such as aerial and satellite imagery, crowdsourced data, online databases, sensors. This means that each server has to be able to support all these new types of data, handle their volume and also be able to handle their heterogeneity based on standards. It also determines the convenience of implementing that they provide, and finally and most important, they determine the quality of the final produced map by implementing the data symbolism. Therefore, the selection of the map server consists of several criteria which must be taken seriously into consideration combined and thus to make the optimal selection for each application.

Web Mapping libraries also play an important role to the web map applications but they do not concern the map synthesis, but contribute to the interaction with the user client-side. Map interaction is an element that exists only to web map applications and makes the application more interesting than posting a simple static image of a map to the internet. The choice of the appropriate JavaScript library lies on the needs and complexity of each application.

With time, new technologies are also being developed and the community of users who are occupied with such applications is getting even bigger and the rhythm of development is enormous.

REFERENCES

Agrawal, S., Gupta, R.J., 2014, *Development and Comparison of Open source based web GIS frameworks on Wamp and Apache Tomcat Web Services,* The International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences, Volume XL-4, 2014, ISPRS Technical Commission IV Symposium, 14 – 16 May 2014, Suzhou, China

Aime, A., & Giannecchini, S., *GeoServer in Production: We do it, here it is how!,* GeoSolutions, FOSS4G 2017, Boston, August 17th-19th 2017

Arca, D., Alkan, M., Bayik, C., Seker D.Z., *Web Based GIS for Safranbolu Historical City, Turkey,* FIG Working Week 2012, Knowing to manage the territory, protect the environment, evaluate the cultural heritage Rome, Italy, 6-10 May 2012

ArcGIS Server Administrator and Developer Guide, 2009, *The ArcGIS Server Architecture*, web.pdx.edu

Ballatore, A., Tahir, A., McArdle, G., Bertolotto, M., 2011, *A comparison of open source geospatial technologies for web mapping,* Int. J. Web Engineering and Technology, Vol. X, No. Y.

Bastin, L., Buchanan, G., Beresford, A., Pekel, J.F., Dubois, G., Gross, D., 2013, *Open-source mapping and services for Web-based land-cover validation,* Ecological Informatics, 9 - 16.

Bedini, E., 2017, *Creation of Touristic Web Map for Tinos Island*, Cartography Laboratory, School of Rural and Survey Engineering, Master's Programme in Geoinformatics, National Technical University of Athens

Brewer, C. A., Hanchett, C. L., Buttenfield B.P, Usery E. L., 2010, *Performance of map Symbol and Label Design with format and display resolution options through scale for the national map,* symposium of ISPRS Technical Commission IV & AutoCarto in conjunction with ASPRS/CaGIS, Orlando, Florida

Brock, A., Deoliveira, J., *WMS Performance Tests! Mapserver & Geoserver*, FOSS4G 2007, www.refractions.net

Brovelli M.A., 2012, *Free and Open Source Web Mapping,* Politecnico di Milano – Como Campus – Italy

De la Beaujardiere, J., 2006, *OpenGIS® Web Map Server Implementation Specification*, Open Geospatial Consortium Inc., 15-03-2006, version 1.3.0

GeoServer Documentation – *User Manual*, 2017

Heda, M.R., Chikurde, S.V., 2016, *A Review: Geo-Information Technology for Web-Mapping Application,* International Journal of Advanced Research in Computer and Communication Engineering Vol. 5, Issue 3, March 2016

Maclean, M., 2013-2014, *Leaflet Tips and Tricks – Interactive Maps made Easy*, www.learnpub.com

Morisette, D., McKenna, J., Ramsey P., 2009, *WMS Project Status*

Kapoukakis, D., 2016, *Map clustering web applications for rating venues,* Master thesis, University of Piraeus, Department of Informatics

Skopeliti A., 2017, *Map Development for the Internet using Open software,* Programming and Geospatial Applications course, Lecture 10, Technological Institute of Athens

Steiniger, S., Hunter, A.J.S., 2012, *Free and Open Source GIS software for Building a Spatial Data Infrastructure,* Department of Geomatics Engineering, University of Calgary, 2500 University Drive NW, Calgary, AB T2N 1N4, Canada

The MapServer Team- McKenna et al., 2017, *MapServer Documentation*, Release 7.0.6, www.mapserver.org

Tsou , M. & Kemp, K., 2008, *Encyclopedia of Geographic Information Science*, p.511-516.

Veenendaal, B., Brovelli, M.A., Li, S., 2017, *Review of Web Mapping: Eras, Trends and Directions*, International Journal of Geo-Information, ISPRS, 2017, 6, 317

Walgrun, J.O., *GEOG:585 – Open Web Mapping*, Online course, PennState College of Earth and Mineral Sciences, Department of Geography (https://www.e-education.psu.edu/geog585/node/508)

Zheng, K., Soomro., T.R, Pan, Y., 2000, *Web GIS: Implementation issues*, Chinese Geographical Science, Volume 10, Number 1, pp. 74-79, 2000, Science Press, Beijing, China

Andrakakou M., 2017, *Web Maps for urban routes of Cultural Heritage*, School of Rural and Surveying Engineering, NTUA. *(in Greek)*

Astyakopoulos, A., 2009, *Design and Implementation of an integrated information system for managing urban area data using ArcGIS Server: Implementation at Kifissia Municipality*, School of Rural and Surveying Engineering, NTUA. *(in Greek)*

Gatzoflias, D., 2007, *Geographic Information Systems at Web*, Master's Program in Geoinformatics, School of Rural and Surveying Engineering, NTUA. *(in Greek)*

Kladis, D., 2016, *Carto-Tools - Web Application for the provision of Geographic Services*, Master's Program in Geoinformatics, School of Rural and Surveying Engineering, NTUA. *(in Greek)*

Kontopoulos, G., 2010. *Development of Web-GIS Applications using ope-source software (GeoServer)*, Post-Graduate Program on Technical Systems, University of Macedonia. *(in Greek)*

Kyriakoulias K., Ntelis E., 2006. *Web Services Technology*, School of Management and Economics, Technological Institute of Mesolonghi. *(in Greek)*

Papapostolou, A., 2013, *Web and Dynamic Maps for the graphic rendering of the Asia Minor Expedition*, Post-Graduate Program in Geoinformatics, School of Rural and Surveying Engineering, NTUA. *(in Greek)*

Stamou L., Skopeliti, A., 2014. *Maps on the Internet: Common Cartographic Practics or a new Cartographic Culture*, 13th Cartography Conference, Argyri Market, Patras., 22-24 October 2014. *(in Greek)*

Tzelepis N., Krassanakis V., Nakos B., 2014, *Employment of free and open-source software for creating Web Maps at Education*, Records of the 13th Cartography Conference, pp. 297-311, Patras, Ziti Publications. *(in Greek)*

Tsiougou N., 2015*, Interactive Map of island Melos*, Department of Geography, Harokopio University. *(in Greek)*

Tsoulos, L., Skopeliti, A., Stamou, L. 2015a. "*Publishing of maps and cartographic data on the Internet*". At Tsoulos L., Skopeliti, A., Stamou, L., 2015. "Cartographic Composition and Rendering on Digital Environment", Athens. Available at: http://hdl.handle.net/11419/2518. *(in Greek)*

Tsoulos, L., Skopeliti, A., Stamou, L. 2015b. "*Map Composition for the Internet*". At Tsoulos L., Skopeliti, A., Stamou, L., 2015. "Cartographic Composition and Rendering on Digital Environment", Athens. Available at: http://hdl.handle.net/11419/2519. *(in Greek)*

Online references

GeoServer Documentation – User Manual, docs.geoserver.org

Coordinate System Worldwide, www.epsg.io

Open Geospatial Consortium, http://www.opengeospatial.org/

http://blog.thinkgeo.com

http://json.org

Boundless, https://boundlessgeo.com/

W3Schools, https://www.w3schools.com/

Wikipedia, www.wikipedia.org

WordReference, www.wordreference.com

W3C, www.w3.org

OpenLayers, www.openlayers.org

Leaflet, http://leafletjs.com

The Apache software foundation, https://www.apache.org

https://gis.stackexchange.com

http://spatialreference.org

https://enterprise.arcgis.com/

APPENDIX

I. *JavaScript Code for OpenLayers*

```
// OpenLayers 4
$(function()
{
$( "#accordion" ).accordion();
});
function init()
{
var extent = [572000, 4130000, 590000, 4155000];
//projection
var projection = new ol.proj.Projection({
code: 'EPSG:2100',
units: 'm',
extent: extent
});
// view
var view = new ol.View({
projection: projection,
center: [581000,4142500],
zoom: 1,
minZoom:1,
maxZoom:3,
extent: [572000, 4130000, 590000, 4155000]
});
//define WMS layer
var layer =  new ol.layer.Image({
extent: extent,
```

```
source: new ol.source.ImageWMS({
url: 'http://127.0.0.1/cgi-bin/mapserv.exe?map=/ms4w/apps/mapserv-demo/syros.map&',
params: {
'LAYERS': 'SYROS',
'FORMAT' : 'image/png'
},
ratio: 1,
serverType: 'mapserver'
})
});

//syntetagmenes me pan
var pontiki = new ol.control.MousePosition({
coordinateFormat: ol.coordinate.createStringXY(4),
projection: 'EPSG:2100',
target: document.getElementById('mouse-position')
});
//scalebar
var scaleLine = new ol.control.ScaleLine();

//full screen
var fs = new ol.control.FullScreen();
//overview
var overview = new ol.control.OverviewMap({
layers: [
new ol.layer.Image({
extent: extent,
source: new ol.source.ImageWMS({
```

```
url: 'http://127.0.0.1/cgi-bin/mapserv.exe?map=/ms4w/apps/mapserv-demo/syros.map&',
params: {
'LAYERS': 'SYROS',
'FORMAT' : 'image/png'
},
ratio: 1,
serverType: 'mapserver'
})
})
],
collapseLabel: '\u00BB',
label: '\u00AB',
collapsed: true
});
//map object
var map = new ol.Map ({
interactions: ol.interaction.defaults().extend([
new ol.interaction.DragRotateAndZoom()
]),
layers: [layer],
target: 'xartis',
controls: ol.control.defaults({
attributionOptions: /** @type {olx.control.AttributionOptions} */ ({
collapsible: false
}),
}).extend([pontiki,scaleLine,fs,overview]),
view: view
});
};
```

II. JavaScript Code for Leaflet

```
//javascript code gia Leaflet

$(function()
{
$( "#accordion" ).accordion();
});

function init()
{
var southWest = L.latLng(37.3, 24.6),
northEast = L.latLng(37.7, 25.2),
bounds = L.latLngBounds(southWest, northEast);

//scales
var myscales= [50000, 75000, 150000];

//projection
//var crs = new L.Proj.CRS('EPSG:2100', '+proj=tmerc +lat_0=0 +lon_0=24
+k=0.9996 +x_0=500000 +y_0=0 +ellps=GRS80 +towgs84=-
199.87,74.79,246.62,0,0,0,0 +units=m +no_defs',
//);
var  map = new L.map('xartis', {
center: [37.44,24.92],
zoom: 11.25,
maxBounds: bounds,
scales: myscales,
minZoom: 11.25,
maxZoom: 13,
zoomSnap: 0
});

L.control.zoom({
zoomInTitle: "zoom in",
zoomOutTitle: "zoom out"
});


var ms = L.tileLayer.wms('http://127.0.0.1/cgi-
bin/mapserv.exe?map=/ms4w/apps/mapserv-demo/syros.map&', {
layers: 'SYROS',
format: 'image/png',
transparent: true,
```

```
attribution: 'MapServer© 2017 Lydia',
}).addTo(map);

var ypo =
L.tileLayer('http://{s}.tile.openstreetmap.org/{z}/{x}/{y}.png?{foo}', {foo:
'bar'}).addTo(map);

L.control.scale({
maxWidth : 100,
metric : true,
position: 'bottomleft'
}).addTo(map);

var marker = L.marker([37.445,24.937], {opacity: 0.7}, {title: 'pata
to!'}).addTo(map);

marker.bindPopup("<b>Ermoupolis</b><br>Capital of Cyclades</br>");

var popup = L.popup();

var baseLayers = {
"MapServer WMS": ms,
"OpenStreetMap" : ypo
};

var overlays = {
"Capital": marker
};

L.control.layers(baseLayers, overlays).addTo(map);

};
```

III. *HTML code*

```
<!DOCTYPE html>

<html>

<head>

<meta charset='utf-8' />

<title>My OpenLayers 4 MapServer Map</title>

<script src="http://code.jquery.com/jquery-1.10.2.js"></script>

<script src="http://code.jquery.com/ui/1.11.4/jquery-ui.js"></script>

<link rel="stylesheet"
href="http://code.jquery.com/ui/1.11.4/themes/smoothness/jquery-ui.css">

<link rel="stylesheet" href="/resources/demos/style.css">

<link rel="stylesheet" href="https://openlayers.org/en/v4.4.2/css/ol.css"
type="text/css">

<script src="https://openlayers.org/en/v4.4.2/build/ol.js"></script>

<script type='text/javascript' src='geo07.js'></script>

<link rel="stylesheet" href="mystyle.css" type="text/css">


</head>

<body onload='init();'>

<div id="main" name="main">

<div id="menu" name="menu">

<div id="accordion">

<h3><b>ΝΗΣΟΣ ΣΥΡΟΣ</b></h3>

<div>

<p style="font-size:85%;"

style="text-align:center;">

Terrain Rendering with Shading <br> Scale <b>1:150000 </b></br>

</p>

<p style="font-size:85%;"

style="text-align:center;">

Terrain Rendering with Contours <br> Scales <b>1:75000</b> and <b>1:50000 </b></br>
```

<p style="font-size:80%;"

*style="text-align:left;">
 Transverse Mercator Projection</br>*

*
 Coordinate System </br>GCRS87*

* *

<p style="font-size:75%;"

style="text-align:left;">

*
Map Composition: Lydia Gouta </br>*

* *

<h3>Information</h3>

* *

* *

Syros is considered to be the Master of Cyclades. As it is the capital of Cyclades, is the largest island on the island, with more than 20,000 permanent residents. Its area is up to 86 sq. Km. and consists of three large Municipalities, the Municipality of Ermoupolis, the Municipality of Ano Syros and the Municipality of Posidonia. The capital of the island is Ermoupolis, which is also the administrative center of the Cyclades. It stands out for its picturesqueness, the imposing neoclassical houses and the majestic churches that has.
*
 <img src="ermoupolis.jpg" alt="Ερμούπολη" width="180" height="150"> </br>*

* *

* *

<h3>Water Measurement Stations</h3>

* *

<p style="font-size:70%;"

style="text-align:center;"> Water resources are of vital importance to humans and especially to the islands during the summer months where over-pumping is a frequent phenomenon and can disrupt the natural cycle of water. This, combined with the brackish water quality, makes it necessary to set up water measurement stations throughout the islands. In this way, the waters of the area are monitored and controlled, underground and surface in terms of their qualitative and quantitative characteristics.
ypeka.gr

* *

<h3>Legend</h3>

* *

* <!-- Pinakas ypomnhmatos -->*

```
<table class="legend">
<tbody>
<tr>
<td class="legend_img"><img src="./img/legend/aeroplano.png"></td>
<td class="legend_title">Airport</td>
<td class="legend_img"><img src="./img/legend/monum.png"></td>
<td class="legend_title">Monument</td>
</tr>
</tbody>
</table>
<table class="legend">
<tbody>
<tr>
<td class="legend_img"><img src="./img/legend/elikodromio.png"></td>
<td class="legend_title">Heliport</td>
<td class="legend_img"><img src="./img/legend/spilia.png"></td>
<td class="legend_title">Cave</td>
</tr>
</tbody>
</table>
<table class="legend">
<tbody>
<tr>
<td class="legend_img"><img src="./img/legend/oikismos2.png"></td>
<td class="legend_title">Settlement</td>
<td class="legend_img"><img src="./img/legend/paralia.png"></td>
<td class="legend_title">Beach</td>
</tr>
</tbody>
</table>
<table class="legend">
```

```
<tbody>
<tr>
<td class="legend_img"><img src="./img/legend/moni.png"></td>
<td class="legend_title">Monastery</td>
<td class="legend_img"><img src="./img/legend/ski.png"></td>
<td class="legend_title">Sport</td>
</tr>
</tbody>
</table>
<table class="legend">
<tbody>
<tr>
<td class="legend_img"><img src="./img/legend/church.png"></td>
<td class="legend_title">Church</td>
<td class="legend_img"><img src="./img/legend/water.png"></td>
<td class="legend_title">Water Measurement Station </td>
</tr>
</tbody>
</table>
<table class="legend">
<tbody>
<tr>
<td class="legend_img"><img src="./img/legend/erimoklisi.png"></td>
<td class="legend_title">Chapel</td>
<td class="legend_img"><img src="./img/legend/triangulation.png"></td>
<td class="legend_title"> Triangulation Point</td>
</tr>
</tbody>
</table>
<table class="legend">
<tbody>
```

```
<tr>
<td class="legend_img"><img src="./img/legend/limani.png"></td>
<td class="legend_title">Port</td>
<td class="legend_img"><img src="./img/legend/asfalt2.png"></td>
<td class="legend_title">Asphalt Road</td>
</tr>
</tbody>
</table>
<table class="legend">
<tbody>
<tr>
<td class="legend_img"><img src="./img/legend/faros.png"></td>
<td class="legend_title">Lighthouse</td>
<td class="legend_img"><img src="./img/legend/mpeto2.png"></td>
<td class="legend_title">Concrete Road</td>
</tr>
</tbody>
</table>
<table class="legend">
<tbody>
<tr>
<td class="legend_img"><img src="./img/legend/hospred.png"></td>
<td class="legend_title">Hospital</td>
<td class="legend_img"><img src="./img/legend/xomat2.png"></td>
<td class="legend_title">Dirt Road</td>
</tr>
</tbody>
</table>
<table class="legend">
<tbody>
<tr>
```

```
<td class="legend_img"><img src="./img/legend/tsantiri.png"></td>
<td class="legend_title">Camping</td>
<td class="legend_img"><img src="./img/legend/xeimarr2.png"></td>
<td class="legend_title">Stream</td>
</tr>
</tbody>
</table>
<table class="legend">
<tbody>
<tr>
<td class="legend_img"><img src="./img/legend/venzini.png"></td>
<td class="legend_title">Gas Station</td>
<td class="legend_img"><img src="./img/legend/capital2.png"></td>
<td class="legend_title">Capital</td>
</tr>
</tbody>
</table>
</div>
</div>
</div>
</div>
<div id="xartis" name="map_element"></div>
</div>

</body>
</html>
```

III. CSS code

```
#main
{
    height: 100%;
  margin: 0;
  position: absolute;
  top: 0;
  width: 100%;
  overflow-y: hidden;
  overflow-x: auto;
    min-height: 530px;
}

#menu
{
position:absolute;
top:0px;
left:0px;
bottom:0px;
margin-left:0px;
margin-top:0px;
background-color: #ff99cc;
width:310px;
float:left;
display:block;
overflow:auto;
height:100%
}



#xartis
{
/*Standard */
width: calc(100% - 316px) !important;
width:82%;
/* Firefox */
width: -moz-calc(100% - 316px) !important;
/* WebKit */
width: -webkit-calc(100% - 316px) !important;
/* Opera */
width: -o-calc(100% - 316px) !important;
width:82%;
min-width: 900px;
min-height:20%;
height:auto;
```

```
position:absolute;
top:0px;
float:left;
left:310px;
bottom:0px;
margin-left:0px;
margin-top:0px;
background-color: #cce0ff;
border: 2px black;
z-index:100;
}

.legend
{
   width:100%;
   table-layout: fixed;
}

.legend td
{
   color:#162837;
   font-size:10px;
   text-align:left;
   padding-bottom:3px;
}

.legend_cb
{
   width:20px;
}

.legend_img
{
   width:20px;
}

.legend_title
{
   width:90px;
   text-align:left;
}
```

IV. Mapfile

```
#
# Start of map file
#
MAP
  NAME SYROS
  STATUS ON
  SIZE 1000 600
  EXTENT 552000.000 4130000.000 590000.000 4155000.000
  UNITS METERS
  FONTSET "C:\ms4w\apps\ms-ogc-workshop\etc\fonts\fonts.txt"
  SHAPEPATH "C:\ms4w\apps\mapserv-demo\data_Syros"
  IMAGECOLOR 204 224 255
  IMAGETYPE PNG

  # Projection definition, consult the PROJ.4 documentation for parameter discussion
   PROJECTION
    "init=epsg:2100"
   END

  #
  # Start of symbol definitions
  #
  SYMBOL
    NAME 'circle'
    TYPE ELLIPSE
    POINTS 1 1 END
    FILLED TRUE
  END
  #
  # Start of web interface definition (including WMS enabling metadata)
  #
  WEB
    HEADER templates/header.html
    TEMPLATE "set in index.html"
    FOOTER templates/footer.html
    MINSCALE 25000
    MAXSCALE 260000
      IMAGEPATH "C:/ms4w/tmp/ms_tmp/"
    IMAGEURL "/ms_tmp/"

      #WMS server settings begin

    METADATA
      "WMS_TITLE" "SYROS"
      "WMS_ABSTRACT" "Syros Island, Cyclades Greece"
      "WMS_ACCESSCONSTRAINTS" "none"

       "WMS_ONLINERESOURCE" "http://127.0.0.1/cgi-bin/mapserv?map=syros.map&"

      "WMS_FEATURE_INFO__MIME_TYPE" "text/html"
      "WMS_SRS" "EPSG:2100"
       "WMS_ENABLE_REQUEST" "*"
    END
  END

  #
  # Start of layer definitions
  #

  #polygono_ksiras
   LAYER
    NAME Coastpoly
    TYPE POLYGON
    STATUS DEFAULT
    DATA Coastpoly
    MINSCALEDENOM 25000
      MAXSCALEDENOM 55000
      PROJECTION
    "init=epsg:2100"
   END
      METADATA
      "WMS_TITLE" "Coastpoly"
      "WMS_NAME" "Coastpoly"
      "WMS_SERVER_VERSION" "1.1.1"
      "WMS_FORMAT" "IMAGE/PNG"
      "WMS_ONLINERESOURCE" "http://127.0.0.1/cgi-bin/mapserv?map=syros.map&"
```

```
  END
  CLASS
  STYLE
  COLOR '#FFFFD5'
  END
    END
 END

#polygono_ksiras75
 LAYER
  NAME Coast_poly_50
  TYPE POLYGON
  STATUS DEFAULT
  DATA coast_poly_50
  MINSCALEDENOM 56000
    MAXSCALEDENOM 110000
    PROJECTION
  "init=epsg:2100"
 END
    METADATA
    "WMS_TITLE" "Coastpoly75"
    "WMS_NAME"
"Coastpoly75"
    "WMS_SERVER_VERSION"
"1.1.1"
    "WMS_FORMAT"
"IMAGE/PNG"
    "WMS_ONLINERESOURCE"
"http://127.0.0.1/cgi-
bin/mapserv?map=syros.map&"
  END
    CLASS
  STYLE
  COLOR '#FFFFD5'
  END
    END
END

#polygono_ksiras150
 LAYER
  NAME coastpoly150
  TYPE POLYGON
  STATUS DEFAULT
  DATA coastpoly150
  MINSCALEDENOM 115000
    MAXSCALEDENOM  230000
    PROJECTION
  "init=epsg:2100"
 END
    METADATA
    "WMS_TITLE"  "Coastpoly15-
"
    "WMS_NAME"
"Coastpoly150"
    "WMS_SERVER_VERSION"
"1.1.1"
    "WMS_FORMAT"
"IMAGE/PNG"
    "WMS_ONLINERESOURCE"
"http://127.0.0.1/cgi-
bin/mapserv?map=syros.map&"
  END
    CLASS
  STYLE
  COLOR '#FFFFD5'
  END
    END
END

LAYER
 # name of layer
 NAME hillshade

 # projection: if the data is NOT
in the same projection as the
global
 # mapfile, you must explicitly
give the native projection code of
the data
MINSCALEDENOM 115000
    MAXSCALEDENOM 200000
 PROJECTION
  "init=epsg:2100"
 END
 METADATA
    "WMS_TITLE" "hillshade"
    "WMS_NAME" "hillshade"
    "WMS_SERVER_VERSION"
"1.1.1"
    "WMS_ONLINERESOURCE"
"http://127.0.0.1/cgi-
bin/mapserv?map=syros.map&"
    END
 # what type of data is this?
 TYPE RASTER

 # always returned with interface
 STATUS ON

 # actual data pointer
 DATA hillshade.tif

END

#DTM
```

```
 LAYER
  # name of layer
  NAME dtm

  # projection: if the data is NOT in the same projection as the global
  # mapfile, you must explicitly give the native projection code of the data
MINSCALEDENOM 115000
       MAXSCALEDENOM 200000
  PROJECTION
   "init=epsg:2100"
  END
  METADATA
       "WMS_TITLE" "dtm"
       "WMS_NAME" "dtm"
       "WMS_SERVER_VERSION" "1.1.1"
       "WMS_ONLINERESOURCE" "http://127.0.0.1/cgi-bin/mapserv?map=syros.map&"
      END
  # what type of data is this?
  TYPE RASTER

  # always returned with interface
  STATUS ON
  COMPOSITE
    OPACITY 50
  END # COMPOSITE
  # actual data pointer
  DATA dtm.tif
  #styl
  CLASSITEM "[pixel]"

  # class using an EXPRESSION using only [pixel].

   CLASS
    EXPRESSION ([pixel] < 64 )
    STYLE
      COLOR 153 204 153
    END
  END
    CLASS
    EXPRESSION ([pixel] >= 64 AND [pixel] < 140)
    STYLE
      COLOR 237 220 120
    END
  END
    CLASS
   EXPRESSION ([pixel] > 140 AND [pixel] < 210)
   STYLE
     COLOR 242 213 141
   END
        END
          CLASS
   EXPRESSION ([pixel] > 210 AND [pixel] < 350)
   STYLE
    COLOR 230 187 131
   END
        END
         CLASS
   EXPRESSION ([pixel] > 350 AND [pixel] < 420)
   STYLE
    COLOR 209 159 130
   END
        END
        CLASS
   EXPRESSION ([pixel] > 420 )
   STYLE
    COLOR 183 111 98
   END
        END
  END

#aktogrammi
  LAYER
    NAME Coastline
       TYPE LINE
       STATUS DEFAULT
       DATA Coastline
      MINSCALEDENOM 25000
      MAXSCALEDENOM 55000
      PROJECTION
   "init=epsg:2100"
  END
       METADATA
       "WMS_TITLE" "Coastline"
       "WMS_NAME" "Coastline"
       "WMS_SERVER_VERSION" "1.1.1"
       "WMS_ONLINERESOURCE" "http://127.0.0.1/cgi-bin/mapserv?map=syros.map&"
       END
       CLASS
       STYLE
         COLOR 0 0 225
```

```
      WIDTH 0.5
      END
      END
 END

 #coast75
 LAYER
   NAME Coast75
      TYPE LINE
      STATUS DEFAULT
      DATA coast_simpl50
      MINSCALEDENOM 56000
      MAXSCALEDENOM 110000
      PROJECTION
  "init=epsg:2100"
 END
      METADATA
      "WMS_TITLE" "Coast75"
      "WMS_NAME" "Coast75"
      "WMS_SERVER_VERSION"
"1.1.1"
      "WMS_ONLINERESOURCE"
"http://127.0.0.1/cgi-
bin/mapserv?map=syros.map&"
      END
      CLASS
      STYLE
        COLOR 0 0 225
              WIDTH 0.5
      END
      END
 END

 #coast150
 LAYER
   NAME Coast150
      TYPE LINE
      STATUS DEFAULT
      DATA coast150
      MINSCALEDENOM 115000
      MAXSCALEDENOM 250000
      PROJECTION
  "init=epsg:2100"
 END
      METADATA
      "WMS_TITLE" "Coast150"
      "WMS_NAME" "Coast150"
      "WMS_SERVER_VERSION"
"1.1.1"
      "WMS_ONLINERESOURCE"
"http://127.0.0.1/cgi-
bin/mapserv?map=syros.map&"
      END
       CLASS
      STYLE
        COLOR 0 0 225
      WIDTH 0.3
      END
      END
  END

#isoipseis
 LAYER
   NAME Contours
      TYPE LINE
      STATUS DEFAULT
      DATA Contours
      MINSCALEDENOM 25000
      MAXSCALEDENOM 55000
      PROJECTION
  "init=epsg:2100"
 END
      METADATA
      "WMS_TITLE" "Contours"
      "WMS_NAME" "Contours"
      "WMS_SERVER_VERSION"
"1.1.1"
      "WMS_ONLINERESOURCE"
"http://127.0.0.1/cgi-
bin/mapserv?map=syros.map&"
      END
      CLASS
      NAME "kyria"
      STYLE
      COLOR '#B98100'
      WIDTH 0.3
      END
      END
 END

 #kyria75
 LAYER
   NAME kyria75
      TYPE LINE
      STATUS DEFAULT
      DATA kyria_isoipsis
      MINSCALEDENOM 56000
      MAXSCALEDENOM 110000
      PROJECTION
  "init=epsg:2100"
 END
      METADATA
      "WMS_TITLE" "kyria75"
      "WMS_NAME" "kyria75"
      "WMS_SERVER_VERSION"
"1.1.1"
```

```
        "WMS_ONLINERESOURCE"
"http://127.0.0.1/cgi-
bin/mapserv?map=syros.map&"
        END
       LABELITEM "Contour"
        CLASS
        NAME "kyria75"
        STYLE
        COLOR '#B98100'
        WIDTH 0.3
        END
       LABEL
       FONT Arial
    TYPE truetype
    OFFSET 0 0
   SIZE 4
   ANGLE FOLLOW
   COLOR '#B98100'
   REPEATDISTANCE 1
   OUTLINECOLOR '#ffffd5'
  ANTIALIAS TRUE
   END #label
   END #class
   END

   #deut75
  LAYER
    NAME deut75
       TYPE LINE
       STATUS DEFAULT
       DATA
deutereouses_isoypseis
       MINSCALEDENOM 56000
       MAXSCALEDENOM 110000
       PROJECTION
    "init=epsg:2100"
   END
       METADATA
       "WMS_TITLE" "deut75"
       "WMS_NAME" "deut75"
       "WMS_SERVER_VERSION"
"1.1.1"
       "WMS_ONLINERESOURCE"
"http://127.0.0.1/cgi-
bin/mapserv?map=syros.map&"
       END
       CLASS
       NAME "deut75"
       STYLE
       COLOR '#B98100'
       WIDTH 0.2
       END
       END
   END

#rema
  LAYER
    NAME Streams
       TYPE LINE
       STATUS DEFAULT
       DATA Streams
       MINSCALEDENOM 25000
       MAXSCALEDENOM 55000
       PROJECTION
    "init=epsg:2100"
   END
       METADATA
       "WMS_TITLE" "Streams"
       "WMS_NAME" "Streams"
       "WMS_SERVER_VERSION"
"1.1.1"
       "WMS_ONLINERESOURCE"
"http://127.0.0.1/cgi-
bin/mapserv?map=syros.map&"
       END
       CLASS
       STYLE
       COLOR 0 0 225
       WIDTH 0.2
       PATTERN 5 5
       END
       END
       END
   END

 #rema75
   LAYER
     NAME rema75
       TYPE LINE
       STATUS DEFAULT
       DATA
streams100_simpl_eksomal
       MINSCALEDENOM 56000
       MAXSCALEDENOM 110000
       PROJECTION
    "init=epsg:2100"
   END
       METADATA
       "WMS_TITLE" "rema75"
       "WMS_NAME" "rema75"
       "WMS_SERVER_VERSION"
"1.1.1"
       "WMS_ONLINERESOURCE"
"http://127.0.0.1/cgi-
bin/mapserv?map=syros.map&"
       END
```

```
      CLASS
      STYLE
      COLOR 0 0 255
      WIDTH 0.3
      PATTERN 4 5
      END
      END
      END
 END

 #rema150
  LAYER
   NAME rema150
      TYPE LINE
      STATUS DEFAULT
      DATA streams150
     MINSCALEDENOM 115000
     PROJECTION
  "init=epsg:2100"
  END
      METADATA
      "WMS_TITLE" "rema75"
      "WMS_NAME" "rema75"
      "WMS_SERVER_VERSION"
"1.1.1"
      "WMS_ONLINERESOURCE"
"http://127.0.0.1/cgi-
bin/mapserv?map=syros.map&"
      END
      CLASS
      STYLE
      COLOR 0 0 255
      WIDTH 0.3
      PATTERN 4 5
      END
      END
      END
 END

#capital_ermoupoli
LAYER
 NAME Capital
 TYPE POLYGON
 STATUS DEFAULT
 DATA Capital
      MINSCALEDENOM 25000
      MAXSCALEDENOM 55000
      PROJECTION
  "init=epsg:2100"
 END
 METADATA
      "WMS_TITLE" "Capital"
      "WMS_NAME" "Capital"
      "WMS_SERVER_VERSION"
"1.1.1"
      "WMS_ONLINERESOURCE"
"http://127.0.0.1/cgi-
bin/mapserv?map=syros.map&"
      END
 CLASS
 STYLE
 COLOR "#ffccd0"
 OUTLINECOLOR "#999999"
 END
 END
 END

 #capital75
LAYER
 NAME Capital75
 TYPE POLYGON
 STATUS DEFAULT
 DATA capital_simpl
      MINSCALEDENOM 56000
      MAXSCALEDENOM 110000
      PROJECTION
  "init=epsg:2100"
 END
 METADATA
      "WMS_TITLE" "Capital75"
      "WMS_NAME" "Capital75"
      "WMS_SERVER_VERSION"
"1.1.1"
      "WMS_ONLINERESOURCE"
"http://127.0.0.1/cgi-
bin/mapserv?map=syros.map&"
      END
 CLASS
 STYLE
 COLOR "#ffccd0"
 OUTLINECOLOR "#999999"
 END
 END
 END

  #capital150
LAYER
 NAME Capital150
 TYPE POLYGON
 STATUS DEFAULT
 DATA capital150
      MINSCALEDENOM 115000
      MAXSCALEDENOM 260000
      PROJECTION
  "init=epsg:2100"
 END
```

```
  METADATA
       "WMS_TITLE" "Capital150"
       "WMS_NAME" "Capital150"
       "WMS_SERVER_VERSION"
"1.1.1"
       "WMS_ONLINERESOURCE"
"http://127.0.0.1/cgi-
bin/mapserv?map=syros.map&"
       END
  CLASS
  STYLE
  COLOR "#ffccd0"
  OUTLINECOLOR "#999999"
  END
  END
  END

  #kyklades
 LAYER
  NAME kuklades_pl
  TYPE POLYGON
  STATUS DEFAULT
  DATA kuklades_pl
  MINSCALEDENOM 115000
     PROJECTION
   "init=epsg:2100"
  END
  METADATA
       "WMS_TITLE" "kuklades"
       "WMS_NAME" "kuklades"
       "WMS_SERVER_VERSION"
"1.1.1"
       "WMS_ONLINERESOURCE"
"http://127.0.0.1/cgi-
bin/mapserv?map=syros.map&"
      END

  CLASS
  NAME "kuklades_pl"
  STYLE
  COLOR '#ffffe6'
  OUTLINECOLOR '#0039e6'
  WIDTH 0.5
  END
  END
 END

#paralia
LAYER
  NAME beach
  TYPE POINT
  STATUS DEFAULT
  DATA beach
       MINSCALEDENOM 25000
       MAXSCALEDENOM 55000
       PROJECTION
   "init=epsg:2100"
  END
  METADATA
       "WMS_TITLE" "beach"
       "WMS_NAME" "beach"
       "WMS_SERVER_VERSION"
"1.1.1"
       "WMS_ONLINERESOURCE"
"http://127.0.0.1/cgi-
bin/mapserv?map=syros.map&"
       END

  CLASS
  NAME "beach"
  STYLE
  SYMBOL
"C:\ms4w\apps\mapserv-
demo\symbols\paralia.png"
  SIZE 10
  END
  END
  END

 #paralia75
LAYER
  NAME beach
  TYPE POINT
  STATUS DEFAULT
  DATA beach75
     MINSCALEDENOM 56000
     MAXSCALEDENOM 110000
     PROJECTION
   "init=epsg:2100"
  END
  METADATA
       "WMS_TITLE" "beach"
       "WMS_NAME" "beach"
       "WMS_SERVER_VERSION"
"1.1.1"
       "WMS_ONLINERESOURCE"
"http://127.0.0.1/cgi-
bin/mapserv?map=syros.map&"
       END
    CLASS
  NAME "beach"
  STYLE
  SYMBOL
"C:\ms4w\apps\mapserv-
demo\symbols\paralia.png"
  SIZE 9
```

```
 END
 END
 END

 #trigonometrika
 LAYER
 NAME Zpoints
 TYPE POINT
 STATUS DEFAULT
 DATA Zpoints
  MINSCALEDENOM 25000
  MAXSCALEDENOM 55000
     PROJECTION
  "init=epsg:2100"
 END
 METADATA
      "WMS_TITLE" "Zpoints"
      "WMS_NAME" "Zpoints"
      "WMS_SERVER_VERSION"
"1.1.1"
      "WMS_ONLINERESOURCE"
"http://127.0.0.1/cgi-
bin/mapserv?map=syros.map&"
     END
  LABELITEM "Elevation"
  CLASS
  NAME "trigonometrika"
  STYLE
  SYMBOL
"C:\ms4w\apps\mapserv-
demo\symbols\Zpoints.png"
  SIZE 6
  END
  LABEL
  COLOR 0 0 0
  FONT "arial"
  TYPE truetype
  SIZE 5
  POSITION UC
  ANTIALIAS TRUE
  PARTIALS FALSE
  END  #label
END
 END

 #ypsometrika
 LAYER
 NAME ypsometrika
 TYPE POINT
 STATUS DEFAULT
 DATA ypsometrika
  MINSCALEDENOM 25000
  MAXSCALEDENOM 55000
     PROJECTION
  "init=epsg:2100"
 END
 METADATA
      "WMS_TITLE"
"ypsometrika"
      "WMS_NAME"
"ypsometrika"
      "WMS_SERVER_VERSION"
"1.1.1"
      "WMS_ONLINERESOURCE"
"http://127.0.0.1/cgi-
bin/mapserv?map=syros.map&"
      END

      LABELITEM "Elevation"
  CLASS
   NAME "ypsometrika"
   STYLE
   SYMBOL
"C:\ms4w\apps\mapserv-
demo\symbols\ypsom.png"
   SIZE 14
      END
  LABEL
  COLOR '#B98100'
  FONT "arial"
  TYPE truetype
  SIZE 5
  POSITION UR
  OFFSET -5 -5
  PARTIALS FALSE
  ANTIALIAS TRUE
  END  #label
  END
 END

  #Zpoint75
 LAYER
 NAME Zpoint75
 TYPE POINT
 STATUS DEFAULT
 DATA zpoi75
  MINSCALEDENOM 56000
  MAXSCALEDENOM 110000
     PROJECTION
  "init=epsg:2100"
 END
 METADATA
      "WMS_TITLE" "Zpoints75"
      "WMS_NAME" "Zpoints75"
      "WMS_SERVER_VERSION"
"1.1.1"
```

```
      "WMS_ONLINERESOURCE"
"http://127.0.0.1/cgi-
bin/mapserv?map=syros.map&"
      END
    LABELITEM "Elevation"
  CLASS
  NAME "trigonometrika"
  STYLE
  SYMBOL
"C:\ms4w\apps\mapserv-
demo\symbols\Zpoints.png"
  SIZE 6
  END
  LABEL
  COLOR 0 0 0
  FONT "arial"
  TYPE truetype
  SIZE 4
  POSITION UC
  PARTIALS FALSE
  END  #label
  END
 END

#aerodromio
LAYER
  NAME airport
  TYPE POINT
  STATUS DEFAULT
  DATA airport
     MINSCALEDENOM 25000
     MAXSCALEDENOM 55000
     PROJECTION
   "init=epsg:2100"
  END
  METADATA
      "WMS_TITLE" "airport"
      "WMS_NAME" "airport"
      "WMS_SERVER_VERSION"
"1.1.1"
      "WMS_ONLINERESOURCE"
"http://127.0.0.1/cgi-
bin/mapserv?map=syros.map&"
      END
    CLASS
  NAME "airport"
  STYLE
  SYMBOL
"C:\ms4w\apps\mapserv-
demo\symbols\aeroplano.png"
  SIZE 11
  END
  END
  END

 #aerodromio75
LAYER
  NAME airport
  TYPE POINT
  STATUS DEFAULT
  DATA airport75
     MINSCALEDENOM 56000
     MAXSCALEDENOM 110000
     PROJECTION
   "init=epsg:2100"
  END
  METADATA
      "WMS_TITLE" "airport"
      "WMS_NAME" "airport"
      "WMS_SERVER_VERSION"
"1.1.1"
      "WMS_ONLINERESOURCE"
"http://127.0.0.1/cgi-
bin/mapserv?map=syros.map&"
      END
    CLASS
  NAME "airport"
  STYLE
  SYMBOL
"C:\ms4w\apps\mapserv-
demo\symbols\aeroplano.png"
  SIZE 10
  END
  END
  END

 #elikodromio
LAYER
  NAME heliport
  TYPE POINT
  STATUS DEFAULT
  DATA heliport
     MINSCALEDENOM 25000
     MAXSCALEDENOM 55000
     PROJECTION
   "init=epsg:2100"
  END
  METADATA
      "WMS_TITLE" "heliport"
      "WMS_NAME" "heliport"
      "WMS_SERVER_VERSION"
"1.1.1"
      "WMS_ONLINERESOURCE"
"http://127.0.0.1/cgi-
bin/mapserv?map=syros.map&"
      END
```

```
  CLASS
  NAME "heliport"
  STYLE
  SYMBOL
"C:\ms4w\apps\mapserv-
demo\symbols\elikodromio.png"
  SIZE 11
  END
  END
  END

 #elikodromio75
LAYER
  NAME heliport
  TYPE POINT
  STATUS DEFAULT
  DATA heliport75
      MINSCALEDENOM 56000
      MAXSCALEDENOM 110000
      PROJECTION
   "init=epsg:2100"
  END
  METADATA
       "WMS_TITLE" "heliport75"
       "WMS_NAME" "heliport75"
       "WMS_SERVER_VERSION"
"1.1.1"
       "WMS_ONLINERESOURCE"
"http://127.0.0.1/cgi-
bin/mapserv?map=syros.map&"
       END
     CLASS
  NAME "heliport"
  STYLE
  SYMBOL
"C:\ms4w\apps\mapserv-
demo\symbols\elikodromio.png"
  SIZE 10
  END
  END
  END

 #limani
LAYER
  NAME port
  TYPE POINT
  STATUS DEFAULT
  DATA port
      MINSCALEDENOM 25000
      MAXSCALEDENOM 55000
      PROJECTION
   "init=epsg:2100"
  END
  METADATA
       "WMS_TITLE" "port"
       "WMS_NAME" "port"
       "WMS_SERVER_VERSION"
"1.1.1"
       "WMS_ONLINERESOURCE"
"http://127.0.0.1/cgi-
bin/mapserv?map=syros.map&"
       END
     CLASS
  NAME "port"
  STYLE
  SYMBOL
"C:\ms4w\apps\mapserv-
demo\symbols\limani.png"
  SIZE 20
  END
  END
  END

 #limani75
LAYER
  NAME port
  TYPE POINT
  STATUS DEFAULT
  DATA port75
      MINSCALEDENOM 56000
      MAXSCALEDENOM 110000
      PROJECTION
   "init=epsg:2100"
  END
  METADATA
       "WMS_TITLE" "port75"
       "WMS_NAME" "port75"
       "WMS_SERVER_VERSION"
"1.1.1"
       "WMS_ONLINERESOURCE"
"http://127.0.0.1/cgi-
bin/mapserv?map=syros.map&"
       END
     CLASS
  NAME "port75"
  STYLE
  SYMBOL
"C:\ms4w\apps\mapserv-
demo\symbols\limani.png"
  SIZE 16
  END
  END
  END

 #sports
```

```
LAYER
  NAME sports
  TYPE POINT
  STATUS DEFAULT
  DATA sports
     MINSCALEDENOM 25000
     MAXSCALEDENOM 55000
     PROJECTION
   "init=epsg:2100"
  END
  METADATA
       "WMS_TITLE" "sports"
       "WMS_NAME" "sports"
       "WMS_SERVER_VERSION"
"1.1.1"
       "WMS_ONLINERESOURCE"
"http://127.0.0.1/cgi-
bin/mapserv?map=syros.map&"
       END
    CLASS
  NAME "sports"
  STYLE
  SYMBOL
"C:\ms4w\apps\mapserv-
demo\symbols\athlimata.png"
  SIZE 10
  END
  END
  END

  #sports75
LAYER
  NAME sports75
  TYPE POINT
  STATUS DEFAULT
  DATA sports75
     MINSCALEDENOM 56000
     MAXSCALEDENOM 110000
     PROJECTION
   "init=epsg:2100"
  END
  METADATA
       "WMS_TITLE" "sports75"
       "WMS_NAME" "sports75"
       "WMS_SERVER_VERSION"
"1.1.1"
       "WMS_ONLINERESOURCE"
"http://127.0.0.1/cgi-
bin/mapserv?map=syros.map&"
       END
    CLASS
  NAME "sports75"
  STYLE
  SYMBOL
"C:\ms4w\apps\mapserv-
demo\symbols\athlimata.png"
  SIZE 9
  END
  END
  END

 #faroi
LAYER
  NAME sealight
  TYPE POINT
  STATUS DEFAULT
  DATA sealight
     MINSCALEDENOM 25000
     MAXSCALEDENOM 55000
     PROJECTION
   "init=epsg:2100"
  END
  METADATA
       "WMS_TITLE" "sealight"
       "WMS_NAME" "sealight"
       "WMS_SERVER_VERSION"
"1.1.1"
       "WMS_ONLINERESOURCE"
"http://127.0.0.1/cgi-
bin/mapserv?map=syros.map&"
       END
         CLASSITEM "Id"
  CLASS
  NAME "sealight"
  EXPRESSION "1"
  STYLE
  SYMBOL
"C:\ms4w\apps\mapserv-
demo\symbols\faros.png"
  SIZE 22
  END
  END
  CLASS
  NAME "anapodos"
  EXPRESSION "5"
  STYLE
  SYMBOL
"C:\ms4w\apps\mapserv-
demo\symbols\seelicht.png"
  SIZE 20
  END
  END
 END

 #faroi75
LAYER
```

```
  NAME sealight75
  TYPE POINT
  STATUS DEFAULT
  DATA faros75
     MINSCALEDENOM 56000
     MAXSCALEDENOM 110000
     PROJECTION
   "init=epsg:2100"
  END
  METADATA
       "WMS_TITLE" "sealight"
       "WMS_NAME" "sealight"
       "WMS_SERVER_VERSION"
"1.1.1"
       "WMS_ONLINERESOURCE"
"http://127.0.0.1/cgi-
bin/mapserv?map=syros.map&"
       END
                CLASSITEM "Type"
  CLASS
  NAME "sealight"
  EXPRESSION "1"
  STYLE
  SYMBOL
"C:\ms4w\apps\mapserv-
demo\symbols\faros.png"
  SIZE 22
  END
  END
  CLASS
  NAME "anapodos"
    STYLE
  SYMBOL
"C:\ms4w\apps\mapserv-
demo\symbols\seelicht.png"
  SIZE 18
  END
  END
 END

 #spilies
LAYER
  NAME cave
  TYPE POINT
  STATUS DEFAULT
  DATA cave
     MINSCALEDENOM 25000
     MAXSCALEDENOM 55000
     PROJECTION
   "init=epsg:2100"
  END
  METADATA
       "WMS_TITLE" "cave"
       "WMS_NAME" "cave"
       "WMS_SERVER_VERSION"
"1.1.1"
       "WMS_ONLINERESOURCE"
"http://127.0.0.1/cgi-
bin/mapserv?map=syros.map&"
       END
    CLASS
  NAME "cave"
  STYLE
  SYMBOL
"C:\ms4w\apps\mapserv-
demo\symbols\spilia.png"
  SIZE 15
  END
  END
  END

 #spilies75
LAYER
  NAME cave75
  TYPE POINT
  STATUS DEFAULT
  DATA cave_generalize
     MINSCALEDENOM 56000
     MAXSCALEDENOM 110000
     PROJECTION
   "init=epsg:2100"
  END
  METADATA
       "WMS_TITLE" "cave75"
       "WMS_NAME" "cave75"
       "WMS_SERVER_VERSION"
"1.1.1"
       "WMS_ONLINERESOURCE"
"http://127.0.0.1/cgi-
bin/mapserv?map=syros.map&"
       END
    CLASS
  NAME "cave75"
  STYLE
  SYMBOL
"C:\ms4w\apps\mapserv-
demo\symbols\spilia.png"
  SIZE 12
  END
  END
  END

 #camping
LAYER
  NAME camping
  TYPE POINT
```

```
  STATUS DEFAULT
  DATA camping
      MINSCALEDENOM 25000
      MAXSCALEDENOM 55000
      PROJECTION
   "init=epsg:2100"
  END
  METADATA
       "WMS_TITLE" "camping"
       "WMS_NAME" "camping"
       "WMS_SERVER_VERSION"
"1.1.1"
       "WMS_ONLINERESOURCE"
"http://127.0.0.1/cgi-
bin/mapserv?map=syros.map&"
       END
    CLASS
  NAME "camping"
  STYLE
  SYMBOL
"C:\ms4w\apps\mapserv-
demo\symbols\tsantiri.png"
  SIZE 11
  END
  END
  END

  #camping75
LAYER
  NAME camping
  TYPE POINT
  STATUS DEFAULT
  DATA camping75
      MINSCALEDENOM 56000
      MAXSCALEDENOM 110000
      PROJECTION
   "init=epsg:2100"
  END
  METADATA
       "WMS_TITLE" "camping75"
       "WMS_NAME"
"camping75"
       "WMS_SERVER_VERSION"
"1.1.1"
       "WMS_ONLINERESOURCE"
"http://127.0.0.1/cgi-
bin/mapserv?map=syros.map&"
       END
    CLASS
  NAME "camping75"
  STYLE
  SYMBOL
"C:\ms4w\apps\mapserv-
demo\symbols\tsantiri.png"
  SIZE 9
  END
  END
  END

 #mnimio
LAYER
  NAME monument
  TYPE POINT
  STATUS DEFAULT
  DATA monument
      MINSCALEDENOM 25000
      MAXSCALEDENOM 55000
      PROJECTION
   "init=epsg:2100"
  END
  METADATA
       "WMS_TITLE" "monument"
       "WMS_NAME" "monument"
       "WMS_SERVER_VERSION"
"1.1.1"
       "WMS_ONLINERESOURCE"
"http://127.0.0.1/cgi-
bin/mapserv?map=syros.map&"
  END
    CLASS
  NAME "monument"
  STYLE
  SYMBOL
"C:\ms4w\apps\mapserv-
demo\symbols\monum.png"
  SIZE 17
  END
  END
  END

 #mnimio75
LAYER
  NAME monument75
  TYPE POINT
  STATUS DEFAULT
  DATA monument_generalize
      MINSCALEDENOM 56000
      MAXSCALEDENOM 110000
      PROJECTION
   "init=epsg:2100"
  END
  METADATA
       "WMS_TITLE"
"monument75"
```

```
        "WMS_NAME"
"monument75"
        "WMS_SERVER_VERSION"
"1.1.1"
        "WMS_ONLINERESOURCE"
"http://127.0.0.1/cgi-
bin/mapserv?map=syros.map&"
   END
     CLASS
   NAME "monument75"
   STYLE
   SYMBOL
"C:\ms4w\apps\mapserv-
demo\symbols\monum.png"
   SIZE 15
   END
   END
   END

  #mones75
LAYER
   NAME mones
   TYPE POINT
   STATUS DEFAULT
   DATA mones_generalize
        MINSCALEDENOM 56000
       MAXSCALEDENOM 110000
       PROJECTION
    "init=epsg:2100"
   END
   METADATA
        "WMS_TITLE" "mones"
        "WMS_NAME" "mones"
        "WMS_SERVER_VERSION"
"1.1.1"
        "WMS_ONLINERESOURCE"
"http://127.0.0.1/cgi-
bin/mapserv?map=syros.map&"
        END
      CLASS
   NAME "mones"
   STYLE
   SYMBOL
"C:\ms4w\apps\mapserv-
demo\symbols\moni.png"
   SIZE 11
   END
   END
   END

    #ekklisies75
LAYER
   NAME ekklisies
   TYPE POINT
   STATUS DEFAULT
   DATA eklisies_general
        MINSCALEDENOM 56000
       MAXSCALEDENOM 110000
       PROJECTION
    "init=epsg:2100"
   END
   METADATA
        "WMS_TITLE" "church"
        "WMS_NAME" "church"
        "WMS_SERVER_VERSION"
"1.1.1"
        "WMS_ONLINERESOURCE"
"http://127.0.0.1/cgi-
bin/mapserv?map=syros.map&"
        END
      CLASS
   NAME "church"
   STYLE
   SYMBOL
"C:\ms4w\apps\mapserv-
demo\symbols\church.png"
   SIZE 14
   END
   END
   END

   #erimoklisia75
LAYER
   NAME erimokklisia
   TYPE POINT
   STATUS DEFAULT
   DATA erimokklisia75
        MINSCALEDENOM 56000
       MAXSCALEDENOM 110000
       PROJECTION
    "init=epsg:2100"
   END
   METADATA
        WMS_TITLE "erimokl"
        WMS_NAME "erimokl"
        WMS_SERVER_VERSION
"1.1.1"
        WMS_ONLINERESOURCE
"http://127.0.0.1/cgi-
bin/mapserv?map=syros.map&"
        END
   CLASS
   NAME "erimokl"
   STYLE
```

```
  SYMBOL
"C:\ms4w\apps\mapserv-
demo\symbols\erimok.png"
  SIZE 14
  END
  END
  END

 #dromoi
 LAYER
   NAME Roads
      TYPE LINE
      STATUS DEFAULT
      DATA Roads
      MINSCALEDENOM 25000
      MAXSCALEDENOM 55000
      PROJECTION
    "init=epsg:2100"
  END
      METADATA
       "WMS_TITLE" "Roads"
       "WMS_NAME" "Roads"
       "WMS_SERVER_VERSION"
"1.1.1"
       "WMS_ONLINERESOURCE"
"http://127.0.0.1/cgi-
bin/mapserv?map=syros.map&"
       END
       CLASSITEM "Type"
 CLASS
 NAME "asfaltos"
 EXPRESSION "1"
 STYLE
 COLOR 0 0 0
 WIDTH 1.2
 END
 STYLE
 COLOR 255 0 0
 WIDTH 1
 END
 END
  CLASS
 NAME "skyra"
 EXPRESSION "2"
      STYLE
      COLOR 0 0 0
      WIDTH 1.2
      END
      STYLE
      COLOR 255 0 0
      WIDTH 1
      END
      STYLE
         COLOR 255 255 255
         WIDTH 0.8
       PATTERN 5 5
         END
         END
   END
   CLASS
   NAME "xomat"
   EXPRESSION "3"
   STYLE
   COLOR 0 0 0
   WIDTH 1.2
   END
   STYLE
   COLOR 255 255 255
   WIDTH 1.15
   END
  END
END

 #asfaltos75
 LAYER
    NAME asfalt75
      TYPE LINE
      STATUS DEFAULT
      DATA
road_asfalt_generalize
      MINSCALEDENOM 56000
      MAXSCALEDENOM 110000
      PROJECTION
    "init=epsg:2100"
   END
      METADATA
       "WMS_TITLE" "asfalt75"
       "WMS_NAME" "asfalt75"
       "WMS_SERVER_VERSION"
"1.1.1"
       "WMS_ONLINERESOURCE"
"http://127.0.0.1/cgi-
bin/mapserv?map=syros.map&"
       END
         CLASS
 NAME "asfaltos"
 STYLE
 COLOR 0 0 0
 WIDTH 0.8
 END
 STYLE
 COLOR 255 0 0
 WIDTH 0.7
 END
 END
END
```

```
#skyra75
 LAYER
   NAME skyra75
      TYPE LINE
      STATUS DEFAULT
      DATA roads_skyr_generalize
      MINSCALEDENOM 56000
      MAXSCALEDENOM 110000
      PROJECTION
   "init=epsg:2100"
  END
      METADATA
       "WMS_TITLE" "skyr75"
       "WMS_NAME" "skyr75"
       "WMS_SERVER_VERSION"
"1.1.1"
       "WMS_ONLINERESOURCE"
"http://127.0.0.1/cgi-
bin/mapserv?map=syros.map&"
       END
       CLASS
 NAME "skyra75"
      STYLE
      COLOR 0 0 0
      WIDTH 1
      END
      STYLE
      COLOR 255 0 0
      WIDTH 0.8
      END
      STYLE
      COLOR 255 255 255
      WIDTH 0.6
   PATTERN 5 5
      END
      END
      END
END

#xoma75
 LAYER
   NAME xoma75
      TYPE LINE
      STATUS DEFAULT
      DATA
road_xomat_generalize
      MINSCALEDENOM 56000
      MAXSCALEDENOM 110000
      PROJECTION
   "init=epsg:2100"
  END
      METADATA
       "WMS_TITLE" "xoma75"
       "WMS_NAME" "xoma75"
       "WMS_SERVER_VERSION"
"1.1.1"
       "WMS_ONLINERESOURCE"
"http://127.0.0.1/cgi-
bin/mapserv?map=syros.map&"
       END
      CLASS
  NAME "xomat75"
  STYLE
  COLOR 0 0 0
  WIDTH 1
  END
  STYLE
  COLOR 255 255 255
  WIDTH 0.95
  END
 END
END


#venzinadika
LAYER
  NAME gas_station
  TYPE POINT
  STATUS DEFAULT
  DATA gas_station
   MINSCALEDENOM 25000
   MAXSCALEDENOM 55000
      PROJECTION
   "init=epsg:2100"
  END
  METADATA
       "WMS_TITLE" "gas_station"
       "WMS_NAME"
"gas_station"
       "WMS_SERVER_VERSION"
"1.1.1"
       "WMS_ONLINERESOURCE"
"http://127.0.0.1/cgi-
bin/mapserv?map=syros.map&"
       END
     CLASS
  NAME "gas_station"
  STYLE
  SYMBOL
"C:\ms4w\apps\mapserv-
demo\symbols\venzini.png"
  SIZE 11
  END
  END
  END
```

```
 #venzinadika75
LAYER
  NAME gas_station
  TYPE POINT
  STATUS DEFAULT
  DATA venzinadika_generalize
   MINSCALEDENOM 56000
   MAXSCALEDENOM 110000
      PROJECTION
   "init=epsg:2100"
  END
  METADATA
       "WMS_TITLE" "gas75"
       "WMS_NAME" "gas75"
       "WMS_SERVER_VERSION"
"1.1.1"
       "WMS_ONLINERESOURCE"
"http://127.0.0.1/cgi-
bin/mapserv?map=syros.map&"
      END
     CLASS
  NAME "gas75"
  STYLE
  SYMBOL
"C:\ms4w\apps\mapserv-
demo\symbols\venzini.png"
  SIZE 9
  END
  END
  END

  #nosokomio
LAYER
  NAME hospital
  TYPE POINT
  STATUS DEFAULT
  DATA hospital
   MINSCALEDENOM 25000
   MAXSCALEDENOM 55000
      PROJECTION
   "init=epsg:2100"
  END
  METADATA
       "WMS_TITLE" "hospital"
       "WMS_NAME" "hospital"
       "WMS_SERVER_VERSION"
"1.1.1"
       "WMS_ONLINERESOURCE"
"http://127.0.0.1/cgi-
bin/mapserv?map=syros.map&"
      END
  CLASS
  NAME "hospital"
  STYLE
  SYMBOL
"C:\ms4w\apps\mapserv-
demo\symbols\hospred.png"
  SIZE 22
  END
  END
  END

 #nosokomio75
LAYER
  NAME hospital75
  TYPE POINT
  STATUS DEFAULT
  DATA hospital75
   MINSCALEDENOM 56000
   MAXSCALEDENOM 110000
      PROJECTION
   "init=epsg:2100"
  END
  METADATA
       "WMS_TITLE" "hospital75"
       "WMS_NAME" "hospital75"
       "WMS_SERVER_VERSION"
"1.1.1"
       "WMS_ONLINERESOURCE"
"http://127.0.0.1/cgi-
bin/mapserv?map=syros.map&"
      END
     CLASS
  NAME "hospital75"
  STYLE
  SYMBOL
"C:\ms4w\apps\mapserv-
demo\symbols\hospred.png"
  SIZE 19
  END
  END
  END

      #oikismoi50
LAYER
  NAME oikismoi50
  TYPE POINT
  STATUS DEFAULT
  DATA oikismoi50
   MINSCALEDENOM 25000
   MAXSCALEDENOM 55000
      PROJECTION
   "init=epsg:2100"
  END
```

```
  METADATA
      "WMS_TITLE" "oiksmoi50"
      "WMS_NAME" "oikismoi50"
      "WMS_SERVER_VERSION"
"1.1.1"
      "WMS_ONLINERESOURCE"
"http://127.0.0.1/cgi-
bin/mapserv?map=syros.map&"
     END
  LABELITEM "Name"
CLASSITEM "Kind"
  CLASS
EXPRESSION "1"
  STYLE
  SYMBOL "circle"
  SIZE 6
  COLOR 0 0 0
  END
  STYLE
  SYMBOL "circle"
  SIZE 5
  COLOR 255 255 255
  END
 LABEL
 FONT Garamond08-Regular
   TYPE truetype
   OFFSET 15 -12
  SIZE 10
  COLOR 0 0 0
  OUTLINECOLOR '#ffffd5'
 ANTIALIAS TRUE
  END #label
  END
  CLASS #pano aristera
EXPRESSION "2"
  STYLE
  SYMBOL "circle"
  SIZE 6
  COLOR 0 0 0
  END
  STYLE
  SYMBOL "circle"
  SIZE 5
  COLOR 255 255 255
  END
 LABEL
 FONT Garamond08-Regular
   TYPE truetype
   OFFSET 0 0
  SIZE 10
  POSITION UL
  COLOR 0 0 0
  OUTLINECOLOR '#ffffd5'
 ANTIALIAS TRUE
  END #label
  END
   CLASS
EXPRESSION "3"
  STYLE
  SYMBOL "circle"
  SIZE 6
  COLOR 0 0 0
  END
  STYLE
  SYMBOL "circle"
  SIZE 5
  COLOR 255 255 255
  END
 LABEL
 FONT Garamond08-Regular
   TYPE truetype
   OFFSET -17 -4
      POSITION LL
  SIZE 10
  COLOR 0 0 0
  OUTLINECOLOR '#ffffd5'
 ANTIALIAS TRUE
  END #label
  END
  CLASS #dani
EXPRESSION "4"
  STYLE
  SYMBOL "circle"
  SIZE 6
  COLOR 0 0 0
  END
  STYLE
  SYMBOL "circle"
  SIZE 5
  COLOR 255 255 255
  END
 LABEL
 FONT Garamond08-Regular
   TYPE truetype
   OFFSET -10 1
      POSITION UL
  SIZE 10
  COLOR 0 0 0
  OUTLINECOLOR '#ffffd5'
 ANTIALIAS TRUE
  END #label
  END
  CLASS #azolimnos
EXPRESSION "5"
  STYLE
  SYMBOL "circle"
```

```
  SIZE 6
  COLOR 0 0 0
  END
  STYLE
  SYMBOL "circle"
  SIZE 5
  COLOR 255 255 255
  END
 LABEL
 FONT Garamond08-Regular
   TYPE truetype
   OFFSET 0 2
      POSITION UL
  SIZE 10
  COLOR 0 0 0
  OUTLINECOLOR '#ffffd5'
 ANTIALIAS TRUE
  END #label
  END
  CLASS #kipoi
EXPRESSION "6"
  STYLE
  SYMBOL "circle"
  SIZE 6
  COLOR 0 0 0
  END
  STYLE
  SYMBOL "circle"
  SIZE 5
  COLOR 255 255 255
  END
 LABEL
 FONT Garamond08-Regular
   TYPE truetype
   OFFSET 0 -2
      POSITION LR
  SIZE 10
  COLOR 0 0 0
  OUTLINECOLOR '#ffffd5'
 ANTIALIAS TRUE
  END #label
  END
 END

 #oikismoi75
LAYER
  NAME oikismoi75
  TYPE POINT
  STATUS DEFAULT
  DATA oikismoi75_final
   MINSCALEDENOM 56000
   MAXSCALEDENOM 110000
      PROJECTION
        "init=epsg:2100"
      END
      METADATA
          "WMS_TITLE" "oikismoi75"
          "WMS_NAME" "oikismoi75"
          "WMS_SERVER_VERSION"
"1.1.1"
          "WMS_ONLINERESOURCE"
"http://127.0.0.1/cgi-
bin/mapserv?map=syros.map&"
        END
  LABELITEM "Name"
CLASSITEM "type"
  CLASS
  EXPRESSION "1"
  STYLE
  SYMBOL "circle"
  SIZE 5
  COLOR 0 0 0
  END
  STYLE
  SYMBOL "circle"
  SIZE 4
  COLOR 255 255 255
  END
 LABEL
 FONT Garamond08-Regular
   TYPE truetype
   OFFSET 15 -9
  SIZE 8
  COLOR 0 0 0
  OUTLINECOLOR '#ffffd5'
 ANTIALIAS TRUE
  END #label
  END #class
 CLASS #komito
  EXPRESSION "2"
  STYLE
  SYMBOL "circle"
  SIZE 5
  COLOR 0 0 0
  END
  STYLE
  SYMBOL "circle"
  SIZE 4
  COLOR 255 255 255
  END
 LABEL
 FONT Garamond08-Regular
   TYPE truetype
   OFFSET 0 -2
  POSITION LR
  SIZE 8
```

```
  COLOR 0 0 0
  OUTLINECOLOR '#ffffd5'
 ANTIALIAS TRUE
  END #label
  END #class
  CLASS #lazareto
  EXPRESSION "3"
  STYLE
  SYMBOL "circle"
  SIZE 5
  COLOR 0 0 0
  END
  STYLE
  SYMBOL "circle"
  SIZE 4
  COLOR 255 255 255
  END
LABEL
FONT Garamond08-Regular
   TYPE truetype
   OFFSET -16 -2
  POSITION LL
  SIZE 8
  COLOR 0 0 0
  OUTLINECOLOR '#ffffd5'
 ANTIALIAS TRUE
  END #label
  END #class
  CLASS #ano syros
  EXPRESSION "9"
  STYLE
  SYMBOL "circle"
  SIZE 5
  COLOR 0 0 0
  END
  STYLE
  SYMBOL "circle"
  SIZE 4
  COLOR 255 255 255
  END
LABEL
FONT Garamond08-Regular
   TYPE truetype
   OFFSET -4 -6
  POSITION LL
  SIZE 8
  COLOR 0 0 0
  OUTLINECOLOR '#ffffd5'
 ANTIALIAS TRUE
  END #label
  END #class
  CLASS #azolimnos
  EXPRESSION "4"
  STYLE
  SYMBOL "circle"
  SIZE 5
  COLOR 0 0 0
  END
  STYLE
  SYMBOL "circle"
  SIZE 4
  COLOR 255 255 255
  END
 LABEL
 FONT Garamond08-Regular
   TYPE truetype
   OFFSET -2 4
  POSITION UL
  SIZE 8
  COLOR 0 0 0
  OUTLINECOLOR '#ffffd5'
 ANTIALIAS TRUE
  END #label
  END #class
  END

 #oikismoi150
LAYER
  NAME oikismoi150
  TYPE POINT
  STATUS DEFAULT
  DATA oikismoi150_new
   MINSCALEDENOM 115000
      PROJECTION
   "init=epsg:2100"
  END
  METADATA
       "WMS_TITLE" "oikismoi150"
       "WMS_NAME"
"oikismoi150"
       "WMS_SERVER_VERSION"
"1.1.1"
       "WMS_ONLINERESOURCE"
"http://127.0.0.1/cgi-
bin/mapserv?map=syros.map&"
      END
      LABELITEM "Name"
CLASSITEM "Kind"
  CLASS
  EXPRESSION "1"
  STYLE
  SYMBOL "circle"
  SIZE 4
  COLOR 0 0 0
  END
  STYLE
```

```
  SYMBOL "circle"
  SIZE 3
  COLOR 255 255 255
  END
LABEL
FONT Garamond08-Regular
  TYPE truetype
POSITION UR
OFFSET 0 2
  SIZE 7
  COLOR 0 0 0
 ANTIALIAS TRUE
  END #label
  END #class
  CLASS
  EXPRESSION "2"
  STYLE
  SYMBOL "circle"
  SIZE 4
  COLOR 0 0 0
  END
  STYLE
  SYMBOL "circle"
  SIZE 3
  COLOR 255 255 255
  END
LABEL
FONT Garamond08-Regular
  TYPE truetype
  SIZE 7
  COLOR 0 0 0
  POSITION UL
  OFFSET 2 2
 ANTIALIAS TRUE
  END #label
  END #class
  CLASS
  EXPRESSION "3"
  STYLE
  SYMBOL "circle"
  SIZE 4
  COLOR 0 0 0
  END
  STYLE
  SYMBOL "circle"
  SIZE 3
  COLOR 255 255 255
  END
LABEL
FONT Garamond08-Regular
  TYPE truetype
  OFFSET 0 2
    POSITION UR
  SIZE 7
  COLOR 0 0 0
 ANTIALIAS TRUE
  END #label
  END #class
  CLASS
  EXPRESSION "4"
  STYLE
  SYMBOL "circle"
  SIZE 4
  COLOR 0 0 0
  END
  STYLE
  SYMBOL "circle"
  SIZE 3
  COLOR 255 255 255
  END
LABEL
FONT Garamond08-Regular
  TYPE truetype
  OFFSET 2 4
    POSITION UL
  SIZE 7
  COLOR 0 0 0
 ANTIALIAS TRUE
  END #label
  END #class
  CLASS
  EXPRESSION "5"
  STYLE
  SYMBOL "circle"
  SIZE 4
  COLOR 0 0 0
  END
  STYLE
  SYMBOL "circle"
  SIZE 3
  COLOR 255 255 255
  END
LABEL
FONT Garamond08-Regular
  TYPE truetype
  OFFSET 15 -9
  SIZE 7
  COLOR 0 0 0
 ANTIALIAS TRUE
  END #label
  END #class
  CLASS
  EXPRESSION "6"
  STYLE
  SYMBOL "circle"
  SIZE 4
```

```
  COLOR 0 0 0
  END
  STYLE
  SYMBOL "circle"
  SIZE 3
  COLOR 255 255 255
  END
 LABEL
 FONT Garamond08-Regular
   TYPE truetype
   OFFSET 15 -9
  SIZE 7
  COLOR 0 0 0
 ANTIALIAS TRUE
  END #label
  END #class
  END

 #religion_mon50
 LAYER
  NAME religion_mon
  TYPE POINT
  STATUS DEFAULT
  DATA religion_mon
      MINSCALEDENOM 25000
      MAXSCALEDENOM 55000
      PROJECTION
   "init=epsg:2100"
  END
  METADATA
       "WMS_TITLE"
"religion_mon"
       "WMS_NAME"
"religion_mon"
       "WMS_SERVER_VERSION"
"1.1.1"
       "WMS_ONLINERESOURCE"
"http://127.0.0.1/cgi-
bin/mapserv?map=syros.map&"
      END
      CLASSITEM "kind"
      CLASS
    NAME "mones"
    EXPRESSION "3"
      STYLE
  SYMBOL
"C:\ms4w\apps\mapserv-
demo\symbols\moni.png"
  SIZE 11
  END
  END
  CLASS
  NAME "ekklisies"
  EXPRESSION "2"
  STYLE
  SYMBOL
"C:\ms4w\apps\mapserv-
demo\symbols\church.png"
  SIZE 17
  END
  END
 CLASS
  NAME "erimokl"
  EXPRESSION "1"
  STYLE
  SYMBOL
"C:\ms4w\apps\mapserv-
demo\symbols\erimok.png"
  SIZE 17
  END
 END
END

#annotation50
 LAYER
  NAME anno50_new
  TYPE LINE
  STATUS DEFAULT
  DATA anno50_new
   MINSCALEDENOM 25000
   MAXSCALEDENOM 55000
      PROJECTION
   "init=epsg:2100"
  END
  METADATA
       "WMS_TITLE"
"anno50_new"
       "WMS_NAME"
"anno50_new"
       "WMS_SERVER_VERSION"
"1.1.1"
       "WMS_ONLINERESOURCE"
"http://127.0.0.1/cgi-
bin/mapserv?map=syros.map&"
      END
     LABELITEM "Name"
  CLASSITEM "Type"
  CLASS
   EXPRESSION "8"
    LABEL
    FONT arialbd
    TYPE truetype
    SIZE 10
    POSITION LC
      ANGLE FOLLOW
      ANTIALIAS TRUE
```

```
    OFFSET 15 -9
  COLOR 115 77 38
  END # LABEL
END #CLASS
CLASS
EXPRESSION "7"
  LABEL
    ALIGN CENTER
  FONT arial
  TYPE truetype
  SIZE 22
  POSITION AUTO
    ANGLE FOLLOW
    ANTIALIAS TRUE
    OFFSET 0 10
    PARTIALS TRUE
  COLOR '#005ce6'
  END # LABEL
END #CLASS
CLASS
EXPRESSION "4"
  LABEL
    ALIGN CENTER
  FONT verdana
  TYPE truetype
  SIZE 7
  POSITION CC
    ANGLE FOLLOW
    OFFSET 0 0
  COLOR 0 0 0
  END # LABEL
END #CLASS
CLASS
EXPRESSION "3"
  LABEL
    ALIGN CENTER
  FONT arial
  TYPE truetype
  SIZE 9
  POSITION AUTO
    ANGLE FOLLOW
    ANTIALIAS TRUE
    OFFSET 0 0
    PARTIALS TRUE
  COLOR '#005ce6'
  END # LABEL
END #CLASS
 CLASS
EXPRESSION "6"
  LABEL
    ALIGN CENTER
  FONT Calibri
  TYPE truetype
  SIZE 9
  POSITION AUTO
    ANGLE FOLLOW
    ANTIALIAS TRUE
    OFFSET 0 0
    PARTIALS TRUE
  COLOR 0 0 0
  END # LABEL
END #CLASS
 CLASS
EXPRESSION "5"
  LABEL
    ALIGN CENTER
  FONT Calibri
  TYPE truetype
  SIZE 9
  POSITION AUTO
    ANGLE FOLLOW
    ANTIALIAS TRUE
    OFFSET 0 0
    PARTIALS TRUE
  COLOR '#005ce6'
  END # LABEL
END #CLASS
 CLASS
EXPRESSION "1"
  LABEL
    ALIGN CENTER
  FONT Calibri
  TYPE truetype
  SIZE 20
  POSITION AUTO
    ANGLE FOLLOW
    ANTIALIAS TRUE
    OFFSET 0 0
    PARTIALS TRUE
  COLOR 0 0 0
  END # LABEL
END #CLASS
CLASS
EXPRESSION "2"
  LABEL
    ALIGN CENTER
  FONT Arial
  TYPE truetype
  SIZE 9
  POSITION UL
    ANGLE AUTO
    ANTIALIAS TRUE
    OFFSET -5 0
    PARTIALS TRUE
  COLOR 0 0 0
  END # LABEL
```

```
  END #CLASS
 END

 #annotation75
LAYER
 NAME anno75_new
 TYPE LINE
 STATUS DEFAULT
 DATA anno75_new
  MINSCALEDENOM 56000
  MAXSCALEDENOM 110000
    PROJECTION
  "init=epsg:2100"
 END
 METADATA
      "WMS_TITLE"
"anno75_new"
      "WMS_NAME"
"anno75_new"
      "WMS_SERVER_VERSION"
"1.1.1"
      "WMS_ONLINERESOURCE"
"http://127.0.0.1/cgi-
bin/mapserv?map=syros.map&"
     END
 LABELITEM "Name"
 CLASSITEM "Type"
 CLASS
  EXPRESSION "8"
   LABEL
   FONT arialbd
   TYPE truetype
   SIZE 7
   POSITION LC
     ANGLE FOLLOW
     ANTIALIAS TRUE
     OFFSET 15 -9
   COLOR 115 77 38
   END # LABEL
 END #CLASS
 CLASS
 EXPRESSION "7"
   LABEL
     ALIGN CENTER
   FONT arial
   TYPE truetype
   SIZE 19
   POSITION AUTO
     ANGLE FOLLOW
     ANTIALIAS TRUE
     OFFSET 0 10
     PARTIALS TRUE
   COLOR '#005ce6'
  END # LABEL
END #CLASS
CLASS
EXPRESSION "4"
 LABEL
    ALIGN CENTER
 FONT verdana
 TYPE truetype
 SIZE 5
 POSITION CC
    ANGLE FOLLOW
    OFFSET 0 0
 COLOR 0 0 0
 END # LABEL
END #CLASS
CLASS
EXPRESSION "3"
 LABEL
    ALIGN CENTER
 FONT arial
 TYPE truetype
 SIZE 6
 POSITION AUTO
    ANGLE FOLLOW
    ANTIALIAS TRUE
    OFFSET 0 0
    PARTIALS TRUE
 COLOR '#005ce6'
 END # LABEL
END #CLASS
 CLASS
EXPRESSION "6"
 LABEL
    ALIGN CENTER
 FONT Calibri
 TYPE truetype
 SIZE 7
 POSITION AUTO
    ANGLE FOLLOW
    ANTIALIAS TRUE
    OFFSET 0 0
    PARTIALS TRUE
 COLOR 0 0 0
 END # LABEL
END #CLASS
 CLASS
EXPRESSION "5"
 LABEL
    ALIGN CENTER
 FONT Calibri
 TYPE truetype
 SIZE 7
 POSITION AUTO
```

```
      ANGLE FOLLOW
      ANTIALIAS TRUE
      OFFSET 0 0
      PARTIALS TRUE
    COLOR '#005ce6'
    END # LABEL
  END #CLASS
  CLASS
  EXPRESSION "1"
    LABEL
      ALIGN CENTER
    FONT Calibri
    TYPE truetype
    SIZE 17
    POSITION AUTO
      ANGLE FOLLOW
      ANTIALIAS TRUE
      OFFSET 0 0
      PARTIALS TRUE
    COLOR 0 0 0
    END # LABEL
  END #CLASS
  CLASS
  EXPRESSION "2"
    LABEL
      ALIGN CENTER
    FONT Arial
    TYPE truetype
    SIZE 7
    POSITION UC
      ANGLE AUTO
      ANTIALIAS TRUE
      OFFSET -5 0
      PARTIALS TRUE
    COLOR 0 0 0
    END # LABEL
  END #CLASS
 END

#annotation150
LAYER
  NAME anno150_new
  TYPE LINE
  STATUS DEFAULT
  DATA anno150_new
   MINSCALEDENOM 115000
     PROJECTION
   "init=epsg:2100"
  END
  METADATA
       "WMS_TITLE"
"anno150_new"
       "WMS_NAME"
"anno150_new"
       "WMS_SERVER_VERSION"
"1.1.1"
       "WMS_ONLINERESOURCE"
"http://127.0.0.1/cgi-
bin/mapserv?map=syros.map&"
      END
 LABELITEM "Name"
  CLASSITEM "Type"
 CLASS
  EXPRESSION "4"
    LABEL
      ALIGN CENTER
    FONT Calibri
    TYPE truetype
    SIZE 13
    POSITION AUTO
      ANGLE FOLLOW
      ANTIALIAS TRUE
      OFFSET 0 0
      PARTIALS TRUE
    COLOR 0 0 0
    END # LABEL
  END #CLASS
  CLASS
  EXPRESSION "5"
    LABEL
      ALIGN CENTER
    FONT Calibri
    TYPE truetype
    SIZE 6
    POSITION AUTO
      ANGLE FOLLOW
      ANTIALIAS TRUE
      OFFSET 0 0
      PARTIALS TRUE
    COLOR 0 0 0
    END # LABEL
  END #CLASS
  CLASS
  EXPRESSION "3"
    LABEL
      ALIGN CENTER
    FONT arial
    TYPE truetype
    SIZE 19
    POSITION CC
      ANGLE FOLLOW
      ANTIALIAS TRUE
      PARTIALS TRUE
    COLOR '#005ce6'
    END # LABEL
```

```
 END #CLASS                                                    END # Map File
  CLASS
 EXPRESSION "1"
  LABEL
     ALIGN CENTER
  FONT Calibri
  TYPE truetype
  SIZE 15
  POSITION AUTO
     ANGLE FOLLOW
     ANTIALIAS TRUE
     OFFSET 0 0
     PARTIALS TRUE
  COLOR 0 0 0
  END # LABEL
    END
    CLASS
 EXPRESSION "2"
  LABEL
     ALIGN CENTER
  FONT Arial
  TYPE truetype
  SIZE 7
  POSITION LR
     ANGLE AUTO
     ANTIALIAS TRUE
     OFFSET -5 0
     PARTIALS TRUE
  COLOR 0 0 0
  END # LABEL
 END #CLASS
 END

LAYER
 # name of layer
 NAME hillshade


 PROJECTION
  "init=epsg:2100"
 END

 # what type of data is this?
 TYPE RASTER

 # always returned with interface
 STATUS ON

 # actual data pointer
 DATA hillshade.tiff
END
```