Εθνικό Μετσόβιο Πολυτεχνείο
Σχολή Ηλεκτρολόγων Μηχανικών
και Μηχανικών Υπολογιστών
Τομέας Τεχνολογίας Πληροφορικής και
Υπολογιστών

Μεταγλώττιση σε WebAssembly ως Υπηρεσία

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

ΝΙΚΟΛΑΟΣ Π. ΓΑΔ

Επιβλέπων : Νικόλαος Σ. Παπασπύρου
Καθηγητής Ε.Μ.Π.

Αθήνα, Οκτώβριος 2020

Εθνικό Μετσόβιο Πολυτεχνείο
Σχολή Ηλεκτρολόγων Μηχανικών
και Μηχανικών Υπολογιστών
Τομέας Τεχνολογίας Πληροφορικής και
Υπολογιστών

Μεταγλώττιση σε WebAssembly ως Υπηρεσία

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

ΝΙΚΟΛΑΟΣ Π. ΓΑΔ

Επιβλέπων : Νικόλαος Σ. Παπασπύρου
Καθηγητής Ε.Μ.Π.

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή την 6η Οκτωβρίου 2020.

.....	.....	.....
Νικόλαος Σ. Παπασπύρου Καθηγητής Ε.Μ.Π.	Γεώργιος Γκούμας Επίκ. Καθηγητής Ε.Μ.Π.	Γεώργιος Στάμου Αν. Καθηγητής Ε.Μ.Π

Αθήνα, Οκτώβριος 2020

Νικόλαος Π. Γαδ

Διπλωματούχος Ηλεκτρολόγος Μηχανικός και Μηχανικός Υπολογιστών Ε.Μ.Π.

Περίληψη

Σκοπός της παρούσας διπλωματικής εργασίας είναι η σχεδίαση ενός συστήματος το οποίο δίνει τη δυνατότητα σε ένα χρήστη να μεταγλωττίσει πηγαίο κώδικα μιας γλώσσας υψηλού επιπέδου σε Webassembly, χωρίς να γνωρίζει ο ίδιος τις εξαρτήσεις που χρειάζεται ο εκάστοτε μεταγλωττιστής ώστε να λειτουργήσει.

Τις τελευταίες δεκαετίες, οι άνθρωποι χρησιμοποιούν καθημερινά ένα μεγάλο σύνολο εφαρμογών, οι οποίες επηρεάζουν διάφορους τομείς της ζωή τους. Το είδος των εφαρμογών αυτών εκτείνεται από απλά παιχνίδια έως σύνθετα νευρωνικά δίκτυα και συστήματα τεχνητής νοημοσύνης για ιατρική υποβοήθηση. Συστηματικά, γίνεται προσπάθεια ώστε οι εφαρμογές αυτές να αξιοποιηθούν και σε άλλους τομείς ή να τροποποιηθούν ώστε να ικανοποιήσουν άλλες ανάγκες. Ωστόσο, αυτό δεν είναι πάντοτε εφικτό καθώς πολλές εφαρμογές χρειάζεται να είναι γρήγορες και αποδοτικές αλλά οι τροποποιήσεις που απαιτούνται ώστε να χρησιμοποιηθούν σε άλλους τομείς είτε είναι αδύνατες, είτε είναι δύσκολες, είτε εισάγουν καθυστέρηση στη λειτουργία τους, καθιστώντας τες δύσχρηστες και πιθανώς επικίνδυνες.

Η Webassembly αποτελεί μία γρήγορη γλώσσα χαμηλού επιπέδου, η οποία μπορεί να τρέξει σε φυλλομετρητές και, συγχρόνως, να λειτουργήσει ως γλώσσα-στόχος κατά τη μεταγλώττιση γλωσσών υψηλού επιπέδου. Με αυτόν τον τρόπο, επιτρέπεται σε εφαρμογές γραμμένες σε γλώσσες υψηλού επιπέδου που παραδοσιακά δε μπορούσαν να τρέξουν σε φυλλομετρητές, να αξιοποιηθούν από όλες τις συσκευές που διαθέτουν φυλλομετρητή, δίχως να προκαλείται μεγάλη μεταβολή στην απόδοσή τους.

Δεδομένου πως υπάρχουν διαφορετικοί μεταγλωττιστές για τις διάφορες γλώσσες, ένας προγραμματιστής χρειάζεται να χρησιμοποιήσει και να μάθει ένα μεγάλο σύνολο εργαλείων προτού μπορέσει να μετατρέψει τους υπάρχοντες κώδικές τους σε Webassembly. Το σύστημα που μελετάται και σχεδιάζεται στην παρούσα διπλωματική εργασία, χρησιμοποιώντας εξυπηρετητές, βάση δεδομένων και containers, αποτελεί μία απόδειξη πως μπορεί να σχεδιαστεί ένα σύστημα το οποίο να κρύβει από τους χρήστες του την πολυπλοκότητα των μεταγλωττιστών, δίνοντάς τους τη δυνατότητα να μεταγλωττίσουν εύκολα γλώσσες υψηλού επιπέδου σε Webassembly μέσω μίας γενικευμένης "Διεπαφής Προγραμματισμού Εφαρμογής" (API) ή μίας "Διεπαφής Χρήστη" (UI), επιστρέφοντας τα αποτελέσματα της μεταγλώττισης μέσα σε ένα συμπιεσμένο ZIP αρχείο.

Λέξεις κλειδιά

Μεταγλωττιστές, Συλλογή Μεταγλωττιστών, Webassembly, Φυλλομετρητής, Γλώσσα Φυλλομετρητή, Εξυπηρετητής, Βάση Δεδομένων, Docker Containers, Backend, Frontend, Angular, Python, PostgreSQL, Κλιμακωσιμότητα, Proof of Concept, PoC

Abstract

The purpose of this diploma thesis is the design of a system that gives user the ability to compile high-level language source code to Webassembly, without demanding from him/her to know the software dependencies that are needed from each distinct compiler to work.

During the last decades, people are using a large set of applications on a daily base that influence various areas of their lives. The type of applications ranges from simple games to complex neural networks and artificial intelligence systems for medical support. Systematically, efforts are made in order for the applications to be used in other areas or to be modified in order to satisfy other needs. However, this is not always feasible because many applications need to be fast and efficient but the necessary modifications, in order to be applied in other areas, are impossible, difficult or they introduce a latency during their usage, thus making them difficult to use and possibly dangerous.

Webassembly is a fast low-level language, capable of running in browsers and, simultaneously, of being used as a target language during high-level language compilation. This way, applications written in high-level languages that couldn't traditionally run in browsers, are able to be used by every device that has a browser, without a noticeable change in their efficiency.

Given the fact that different compilers exist for the various languages, a programmer needs to use and learn a big set of tools before being able to change the existing code to Webassembly. The system that is studied and designed in this diploma thesis, using servers, database and containers, is a proof of concept that it is possible to design a system that can hide the compiler complexity from the user, by giving the user the ability to easily compile high-level languages to Webassembly via a generic Application Programming Interface (API) or a User Interface (UI), returning the compilation results inside a compressed ZIP file.

Key words

Compilers, Compiler Collection, Webassembly, Browser, Browser Language, Server, Database, Docker Containers, Backend, Frontend, Angular, Python, PostgreSQL, Scalability, Proof of Concept, PoC

Ευχαριστίες

Ευχαριστώ τον επιβλέποντα καθηγητή της παρούσας διπλωματικής εργασίας, κ. Νίκο Παπασπύρου, για την καθοδήγηση, τις υποδείξεις και τις συμβουλές του καθ' όλη τη διάρκεια εκπόνησης της εργασίας. Θέλω, επίσης, να ευχαριστήσω την οικογένειά μου, τον πατέρα μου, Παύλο, τη μητέρα μου, Όλγα, και την αδερφή μου, Ιωάννα, οι οποίοι ήταν και είναι δίπλα μου, να με στηρίζουν και να με συμβουλεύουν, όχι μόνο κατά τη διάρκεια της διπλωματικής εργασίας αλλά σε κάθε βήμα που κάνω.

Νικόλαος Π. Γαδ,

Αθήνα, 6η Οκτωβρίου 2020

Η εργασία αυτή είναι επίσης διαθέσιμη ως Τεχνική Αναφορά CSD-SW-TR-4-20, Εθνικό Μετσόβιο Πολυτεχνείο, Σχολή Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών, Τομέας Τεχνολογίας Πληροφορικής και Υπολογιστών, Εργαστήριο Τεχνολογίας Λογισμικού, Οκτώβριος 2020.

URL: `http://www.softlab.ntua.gr/techrep/`
FTP: `ftp://ftp.softlab.ntua.gr/pub/techrep/`

Περιεχόμενα

Περίληψη 5

Abstract 7

Ευχαριστίες 9

Περιεχόμενα 11

Κατάλογος πινάκων 13

Κατάλογος σχημάτων 15

1. Εισαγωγή 17

2. Τεχνολογίες Λογισμικού 19
2.1 Εισαγωγή στη Webassembly 19
2.2 Εισαγωγή στους Containers 21

3. Αρχιτεκτονική Συστήματος 25
3.1 Microservices και Containerization 25
3.1.1 Backend Service 26
3.1.2 Database Service 27
3.1.3 Frontend Service 27
3.2 Δίκτυο και επικοινωνία μεταξύ των Containers 29

4. Αρχιτεκτονική Κώδικα 31
4.1 Σχεδίαση Containers και Multistage Builds 31
4.2 Λογισμικό Backend 32
4.2.1 Packages 32
4.2.2 Σχήμα Βάσης Δεδομένων 35
4.2.3 Διαχείριση μεταγλώττισης - Compilation Handlers 35
4.3 Λογισμικό Frontend 39
4.4 Εγκατάσταση και εκτέλεση 40

5. Παράδειγμα Λειτουργία της Υπηρεσίας Μεταγλώττισης 43

6. Συμπεράσματα 49
6.1 Σύνοψη 49
6.2 Επεκτάσεις του Συστήματος και της Εφαρμογής 50
6.2.1 Επέκταση εφαρμογής με νέες γλώσσες 50
6.2.2 Επέκταση λειτουργιών 51

Βιβλιογραφία 53

Κατάλογος πινάκων

4.1 Backend API 33
4.2 Υποστηριζόμενες Εκδόσεις Φυλλομετρητών 40
4.3 Μη Υποστηριζόμενες Εκδόσεις Φυλλομετρητών 40

Κατάλογος σχημάτων

2.1 Αρχιτεκτονική Συστήματος με Εικονικές Μηχανές 22
2.2 Αρχιτεκτονική Συστήματος με Containers 22

3.1 Μη Κλιμακωμένη Αρχιτεκτονική Συστήματος 28
3.2 Κλιμακωμένη Αρχιτεκτονική Συστήματος 30

4.1 Δομή του rest_server package 34
4.2 Διάγραμμα Οντοτήτων-Συσχετίσεων Βάσης Δεδομένων 35
4.3 Σχεσιακό Μοντέλο Βάσης Δεδομένων 36
4.4 CompilationHandler Class Attributes and Abstract Methods 37
4.5 Hello World σε C 37
4.6 Δομή Συστήματος Αρχείων στο φάκελο των αποτελεσμάτων 38
4.7 Placeholder τιμές στο .env αρχείο 40

5.1 Αρχική Σελίδα - Σελίδα Μεταγλώττισης 43
5.2 Σελίδα Επιλογών Μεταγλώττισης 43
5.3 Συμπληρωμένη Σελίδα Εγγραφής 44
5.4 Συμπληρωμένη Σελίδα Σύνδεσης 44
5.5 Αρχική Σελίδα (Εγγεγραμμένος Χρήστης) 44
5.6 Περιεχόμενα αρχείου palindrome.cpp 45
5.7 Επιλογή Αρχείου προς Μεταγλώττιση (Εγγεγραμμένος Χρήστης) 46
5.8 Συμπληρωμένες Επιλογές Μεταγλώττισης σε C++ 46
5.9 Ολοκληρωμένη Μεταγλώττιση (Εγγεγραμμένος Χρήστης) 46
5.10 Περιεχόμενα του results.zip αρχείου 46
5.11 Overview όλων των ανεβασμένων αρχείων του χρήστη 46
5.12 Επιλογή αρχείων συγκεκριμένης γλώσσας 47
5.13 Overview των ανεβασμένων αρχείων σε C του χρήστη 47
5.14 Overview των ανεβασμένων αρχείων σε C++ του χρήστη 47
5.15 Overview των ανεβασμένων αρχείων σε Golang του χρήστη 47
5.16 Λεπτομέρειες μεταγλώττισης του αρχείου palindrome.cpp 47
5.17 Διαγραφή του ανεβασμένου αρχείου palindrome.cpp 47
5.18 Πηγαίος κώδικας του palindrome.cpp μέσω UI 48

6.1 Subpackage compile: κώδικας στο αρχείο __init__.py 50

Κεφάλαιο 1

Εισαγωγή

Με την εισαγωγή της WebAssembly στους φυλλομετρητές ως μίας νέας γλώσσας, η οποία μπορεί να εκτελεστεί σε αυτούς, εισάγονται νέες δυνατότητες στην ανάπτυξη Web εφαρμογών. Ταυτόχρονα, όμως, εισάγονται νέα εργαλεία και τροποποιούνται τα υπάρχοντα ώστε να καταστεί δυνατό στους χρήστες να χρησιμοποιήσουν αυτή τη νέα assembly-like γλώσσα στις εφαρμογές τους. Πολλές εφαρμογές υλοποιούνται συνδυάζοντας πολλές διαφορετικές γλώσσες προγραμματισμού, με αποτέλεσμα να απαιτούνται και περισσότερα εργαλεία ώστε αφενός να μεταγλωττιστούν σε WebAssembly και αφετέρου να ενσωματωθούν σε Web εφαρμογές. Κατ' επέκταση, χρειάζεται και πολλαπλάσιος χρόνος ώστε να αναπτυχθούν εφαρμογές σε WebAssembly.

Σε αυτό το πλαίσιο, ένα σύστημα το οποίο αναλαμβάνει να συγκεντρώσει και να διαχειριστεί τα εργαλεία τα οποία χρειάζονται ώστε να παραχθεί κώδικας WebAssembly, εξυπηρετεί στον περιορισμό του συνολικού απαιτούμενου χρόνου υλοποίησης της εφαρμογής. Συγκεκριμένα, το σύστημα θα πρέπει να περιέχει ένα σύνολο από μεταγλωττιστές καθώς και τις εξαρτήσεις σε λογισμικό που είναι απαραίτητες ώστε οι μεταγλωττιστές αυτοί να λειτουργήσουν σωστά. Ταυτόχρονα, θα πρέπει να υλοποιεί και να παρέχει ένα API και ένα WebUI, τα οποία θα λειτουργούν ως ένα επίπεδο αφαίρεσης της εσωτερικής πολυπλοκότητας του συστήματος. Με αυτόν τον τρόπο, το σύστημα επιτρέπει στον προγραμματιστή να αφοσιωθεί στην ανάπτυξη του κώδικα και της εφαρμογής, διαχωρίζοντας, ουσιαστικά, την παραγωγή και ενσωμάτωση της WebAssembly σε μία Web εφαρμογή από την υλοποίηση του κώδικα της εφαρμογής σε μία high-level γλώσσα. Κατ' επέκταση, αυτοματοποιείται και επιταχύνεται άμεσα ένα μέρος της παραγωγής.

Στην παρούσα διπλωματική εργασία, υλοποιείται ένα σύστημα το οποίο αποτελεί το Proof of Concept του συστήματος που περιγράφεται παραπάνω. Στην παρούσα υλοποίηση προτείνεται μία cloud-native αρχιτεκτονική με χρήση των Docker Containers, η οποία στηρίζεται σε ένα δίκτυο από εξυπηρετητές που επικοινωνούν μεταξύ τους και μπορούν εύκολα να δημιουργηθούν και να πολλαπλασιαστούν ώστε να καλύψουν τις ανάγκες του συστήματος.

Κεφάλαιο 2

Τεχνολογίες Λογισμικού

2.1 Εισαγωγή στη Webassembly

Η Webassembly είναι μία γλώσσα χαμηλού επιπέδου (low-level), η οποία μπορεί να τρέξει σε όλους τους σύγχρονους φυλλομετρητές, όπως οι Google Chrome και Mozilla Firefox, αλλά και σε φυλλομετρητές που εξαρτώνται από το λειτουργικό σύστημα, όπως Safari, Microsoft Edge και Samsung Internet.[Herr18]

Η Webassembly εμφανίζεται για πρώτη φορά τον Μάρτιο του 2017. Στη συνέχεια, τον Νοέμβριο του 2017, ανακοινώνεται πως υποστηρίζεται στους πλέον χρησιμοποιούμενους φυλλομετρητές. Η Webassembly είναι, πλέον, η τέταρτη γλώσσα που μπορεί να τρέξει σε φυλλομετρητές μαζί με τις HTML, CSS και Javascript. Ο κώδικάς της ορίζει ένα Αφηρημένο Συντακτικό Δέντρο (Abstract Syntax Tree, AST) και μπορεί να εκτελεστεί από μία εικονική μηχανή στοίβας (stack based virtual machine). Το Javascript Engine που εκτελεί την Javascript μπορεί να εκτελέσει και τον κώδικα της Webassembly. Μπορεί να γραφεί είτε σε μορφή κειμένου (text format) η οποία ονομάζεται WAT είτε σε δυαδική μορφή (binary format) η οποία ονομάζεται WASM. Δεδομένου πως είναι μία χαμηλού επιπέδου γλώσσα η οποία μοιάζει με γλώσσα μηχανής (assembly-like), μπορεί να χρησιμοποιηθεί ως γλώσσα-στόχος (target language) κατά τη μεταγλώττιση άλλων γλωσσών. Κατ' επέκταση, γλώσσες οι οποίες έχουν πιο σύνθετες δομές δεδομένων και παρέχουν στο χρήστη ένα χαμηλού επιπέδου μοντέλο διαχείρισης και προσπέλασης της μνήμης, όπως για παράδειγμα οι C, C++ και Rust, έχουν πλέον τη δυνατότητα να μεταγλωττιστούν σε Webassembly και να τρέξουν στους φυλλομετρητές.[Haas]

Η Webassembly προσφέρει γρήγορη εκτέλεση και μεγάλη αποδοτικότητα καθώς μπορεί να τρέξει σχεδόν όσο γρήγορα τρέχει και ένας κώδικας γραμμένος για μία συγκεκριμένη πλατφόρμα (near-native speed).[Herr18] Ταυτόχρονα, δεδομένου πως εκτελείται σε φυλλομετρητές, η Webassembly τρέχει μέσα σε ένα απομονωμένο (sandboxed) περιβάλλον, διατηρώντας τον τελικό χρήστη ασφαλή. Επιπλέον, ο σχεδιασμός της είναι τέτοιος ώστε να ταιριάζει με τις υπάρχουσες διαδικτυακές τεχνολογίες και να είναι συμβατός με παλαιότερες εκδόσεις (backwards compatible). Σε αυτό το πλαίσιο, η Webassembly δεν αντικαθιστά την Javascript ως γλώσσα στο διαδίκτυο αλλά μπορεί να χρησιμοποιηθεί στις περιπτώσεις κατά τις οποίες η χρήση της Javascript είναι δύσκολη. Η Javascript είναι μία γλώσσα υψηλού επιπέδου που παρέχει στον χρήστη ευελιξία και μεγάλη εκφραστικότητα. Παράλληλα, διαθέτει ένα μεγάλο όγκο βιβλιοθηκών και εργαλείων. Γι' αυτούς τους λόγους, είναι κατάλληλη για την ανάπτυξη των περισσότερων διαδικτυακών εφαρμογών. Ωστόσο, σε περιπτώσεις που απαιτούν υψηλή απόδοση και μεγάλη υπολογιστική ισχύ, η Javascript υστερεί σε σχέση με τη Webassembly. Χαρακτηριστικά παραδείγματα αποτελούν οι εφαρμογές εικονικής και επαυξημένης πραγματικότητας (virtual and augmented reality apps), τα 3D παιχνίδια και γενικότερα οι εφαρμογές πραγματικού χρόνου (real time apps).

Σχετικά με την υλοποίηση της Webassembly, αυτή διαθέτει τέσσερις βασικούς τύπους τιμών, οι οποίοι είναι ακέραιοι αριθμοί των 32 και 64 bits και αριθμοί κινητής υποδιαστολής των 32 και 64 bits. Οι τύποι αυτοί μπορούν αφενός να μετατραπούν από έναν τύπο σε έναν άλλο (από τους τέσσερις διαθέσιμους) και αφετέρου να χρησιμοποιηθούν για να αναπαραστήσουν και άλλους τύπους δεδομένων όπως τιμές αλήθειας (boolean) και διευθύνσεις μνήμης. Όπως ήδη αναφέρθηκε, οι εντολές της Webassembly εκτελούνται από ένα stack based virtual machine διαδοχικά.

Οι εντολές της διακρίνονται σε δύο κατηγορίες, ”Απλές”(”Simple Instructions”) και ”Ελέγχου” (”Control Instructions”). Οι ”Απλές Εντολές” μπορούν να αφαιρούν όρους από τη στοίβα, να εφαρμόζουν έναν τελεστή στους όρους αυτούς και να τοποθετούν όρους στη στοίβα. Οι ”Εντολές Ελέγχου” είναι εντολές διακλάδωσης και επανάληψης (οι οποίες μπορούν να είναι εμφωλευμένες) και δημιουργούν έναν γράφο ροής ελέγχου (control flow graph). Επιπλέον, ο κώδικας σε Webassembly περιέχει συναρτήσεις (οι οποίες μπορούν να είναι αναδρομικές), δημιουργώντας, έτσι, μία νέα στοίβα με τις κλήσεις των συναρτήσεων (διαφορετική από τη στοίβα των δεδομένων που χρησιμοποιούν οι ”Απλές Εντολές”), η οποία δεν είναι άμεσα προσπελάσιμη από τον κώδικα. Η Webassembly προσφέρει συνεχή μνήμη (linear memory) από μεταβλητά bytes στα οποία μπορεί να αποθηκεύσει και από τα οποία μπορεί να διαβάσει δεδομένα. Ακόμα, διαθέτει πίνακες (Tables) στους οποίους μπορούν να αποθηκευτούν τιμές ενός συγκεκριμένου τύπου. Στους πίνακες μπορούν να αποθηκευτούν, για παράδειγμα, αναφορές σε συναρτήσεις με αποτέλεσμα να μπορούν να γίνουν κλήσεις συναρτήσεων χρησιμοποιώντας κατάλληλη δεικτοδότηση. Όπως ένας κώδικας γραμμένος σε κάποια γλώσσα μηχανής, έτσι και ο κώδικας της Webassembly μπορεί να προκαλέσει κάποια εξαίρεση (trap). Σε αυτήν την περίπτωση, η εκτέλεση του κώδικα διακόπτεται, το περιβάλλον της Webassembly δεν μπορεί να τη χειριστεί και πλέον το εξωτερικό περιβάλλον αναλαμβάνει να την αντιμετωπίσει.

Ο binary κώδικας της Webassembly ορίζει ένα Module. Το Module μπορεί να περιέχει imports από άλλα Modules αλλά και exports. Ως imports και exports μπορούν να χρησιμοποιηθούν συναρτήσεις, πίνακες, μνήμη και καθολικές μεταβλητές (global variables). Ένα Webassembly Module χρειάζεται να ενσωματωθεί σε ένα περιβάλλον, το οποίο θα το φιλοξενήσει (host environment). Το Host Environment ονομάζεται Embedder και είναι υπεύθυνο να φορτώσει το Webassembly Module και να του παράσχει οτιδήποτε χρειάζεται ώστε αυτό να εκτελεστεί σωστά.

Ωστόσο, για να μπορέσει να εκτελεστεί ένα Webassembly Module σε έναν φυλλομετρητή πρέπει να περάσει από ορισμένα στάδια επεξεργασίας. Σε πρώτο στάδιο, γίνεται αποκωδικοποίηση (Decoding) του Module. Το Webassembly Module σε δυαδική μορφή (δηλαδή ένα wasm αρχείο) μεταγλωττίζεται σε γλώσσα μηχανής ή σε κάποια γλώσσα που μπορεί να τρέξει το host περιβάλλον. Συγκεκριμένα, για να τρέξει σε έναν φυλλομετρητή χρησιμοποιείται ένας νέος μεταγλωττιστής, ο οποίος υπάρχει μέσα στον φυλλομετρητή και μετρατρέπει το wasm σε bytecode. Το παραγόμενο bytecode μπορεί να εκτελεστεί από την υπάρχουσα μηχανή που βελτιστοποιεί και εκτελεί το bytecode της Javascript. Σε δεύτερο στάδιο, το αποκωδικοποιημένο Module ελέγχεται για την εγκυρότητά του (validation). Ο έλεγχος εγκυρότητας που εφαρμόζεται ελέγχει ως προς τη συνέπεια τη χρήση της στοίβας με βάση τους ορισμούς των συναρτήσεων. Ένα έγκυρο αποκωδικοποιημένο module μπορεί να εκτελεστεί και να δημιουργήσει ένα στιγμιότυπο (instance) του module (Instantiation). Με τη δημιουργία του στιγμιοτύπου, το module αποκτά δική του μνήμη, πίνακες και στοίβα εκτέλεσης (execution stack), διατηρώντας έτσι μία κατάσταση, ενώ, ταυτόχρονα, μπορεί να εκτελέσει την κύρια συνάρτησή του, σε περίπτωση που αυτή είναι ορισμένη. Στη συνέχεια, οτιδήποτε εξάγεται (export) από το module μπορεί να χρησιμοποιηθεί και να κληθεί από στο host περιβάλλον μέσω του συγκεκριμένου στιγμιοτύπου (invocation). Στην περίπτωση που το host περιβάλλον καλέσει ένα export του module (για παράδειγμα μία συνάρτηση) τότε το στιγμιότυπο του module εκτελεί εκ νέου Webassembly κώδικα και επιστρέφει ένα αποτέλεσμα με βάση την κατάστασή του εκείνη τη στιγμή.

Ένα Webassembly Module δε διαθέτει κατάσταση (stateless), δηλαδή δεν έχει δική του μνήμη, αλλά χρησιμεύει ώστε να ορίσει τι χρειάζεται ο κώδικας ώστε να μπορέσει να εκτελεστεί. Σε συνδυασμό με μία κατάσταση το Webassembly Module ορίζει ένα στιγμιότυπο. Το host περιβάλλον μπορεί να παράσχει στο Webassembly Module τη μνήμη που χρειάζεται, ώστε αυτό να διατηρήσει μία κατάσταση κατά την εκτέλεσή του. Στην περίπτωση ενός φυλλομετρητή, η Javascript μπορεί να προσφέρει αυτή τη μνήμη σε ένα Webassembly Module και να δημιουργήσει ένα στιγμιότυπο.

Μάλιστα, η Javascript και η Webassembly μπορούν να χρησιμοποιηθούν συνδυαστικά. Δηλαδή, συναρτήσεις ορισμένες σε Javascript μπορούν να δοθούν ως imports σε ένα Webassembly Module ώστε να χρησιμοποιηθούν από αυτό και αντίστροφα, συναρτήσεις που ορίζονται στην

Webassembly μπορούν να χρησιμοποιηθούν σε κώδικα Javascript αν ορίζονται ως exports του Webassembly Module. Με αυτόν τον τρόπο, οι δύο αυτές γλώσσες προσφέρουν αφενός τη δυνατότητα δημιουργίας γρήγορων και σύνθετων εφαρμογών στο διαδίκτυο και αφετέρου τη δυνατότητα μεταφοράς διαφόρων εφαρμογών από άλλες πλατφόρμες στους φυλλομετρητές.

2.2 Εισαγωγή στους Containers

Οι Containers είναι μία τεχνολογία λογισμικού με την οποία είναι δυνατή η συλλογή κώδικα καθώς και όλων των εξαρτήσεων που αυτός χρειάζεται, ώστε να μπορέσει να τρέξει σωστά μία εφαρμογή.

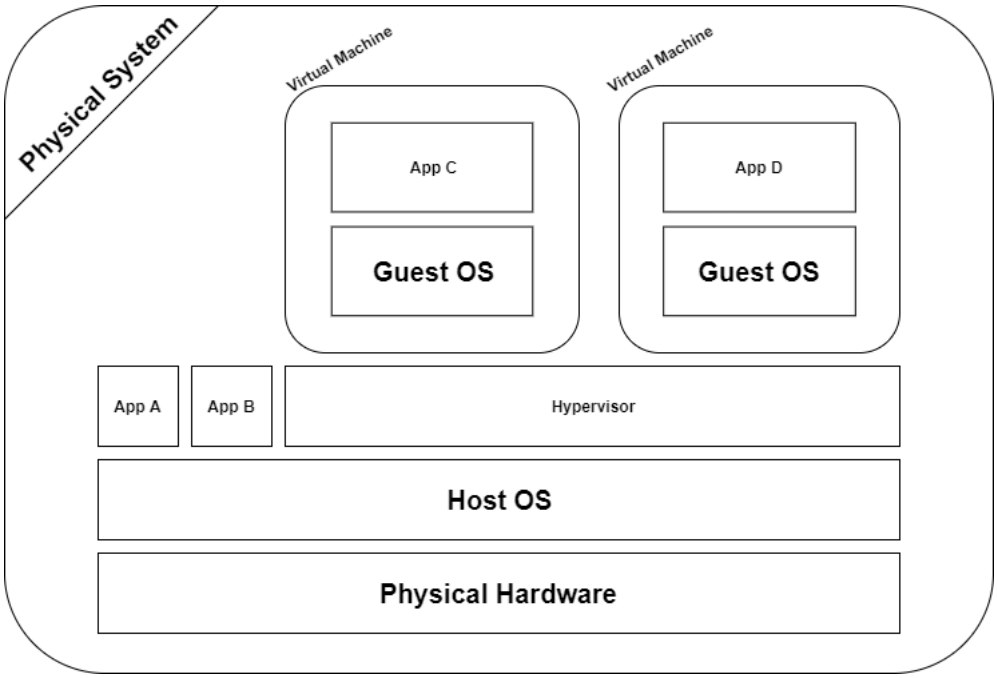
Στην πλειοψηφία τους, οι εφαρμογές, κατά την εγκατάστασή τους σε ένα σύστημα, εξαρτώνται από ένα σύνολο άλλων λογισμικών, τα οποία πρέπει να είναι προεγκατεστημένα στο σύστημα ώστε να μπορέσει να τρέξει με επιτυχία η εφαρμογή. Επιπλέον, συχνά μία εφαρμογή ενδέχεται να είναι δύσκολο να συνυπάρξει με μία άλλη εφαρμογή στο ίδιο σύστημα καθώς οι εξαρτήσεις τους μπορεί να εμφανίζουν συγκρούσεις. Για παράδειγμα μπορεί και οι δύο εφαρμογές να απαιτούν διαφορετικές εκδόσεις τους ίδιου λογισμικού. Σε αυτές τις περιπτώσεις, ο διαχειριστής του συστήματος θα πρέπει να ελέγξει αν αυτές οι συγκρούσεις μπορούν να επιλυθούν ώστε να μπορέσουν όλες οι επιθυμητές εφαρμογές να συνυπάρξουν στο σύστημα. Ωστόσο, αυτή η διαδικασία είναι αφενός χρονοβόρα και αφετέρου απαιτεί από το διαχειριστή του συστήματος να γνωρίζει πολύ καλά τις εξαρτήσεις των, μέχρι στιγμής, εγκατεστημένων λογισμικών ώστε να μπορεί να προβεί στις απαραίτητες τροποποιήσεις διατηρώντας αναλλοίωτη τη φυσιολογική λειτουργία τους. Επομένως, η παραπάνω διαδικασία ευνοεί τη δημιουργία λαθών. Το γεγονός αυτό καθιστά δύσκολη τη μετάβαση μιας εφαρμογής από το περιβάλλον στο οποίο αναπτύσσεται σε ένα άγνωστο περιβάλλον παραγωγής στο οποίο η εγκατάστασή της γίνεται από τρίτους. Κατά συνέπεια, πολύ συχνά κατά την ανάπτυξη εφαρμογών, δημιουργείται η ανάγκη να τρέξει η εφαρμογή σε ένα απομονωμένο περιβάλλον.

Μία μέθοδος είναι η χρήση Εικονικών Μηχανών (Virtual Machines, VMs) (Σχήμα 2.1). Στο φυσικό σύστημα εκτελείται το host λειτουργικό σύστημα (Host Operating System, Host OS) και ο hypervisor. Ο hypervisor, με τη σειρά του, είναι υπεύθυνος ώστε να προσφέρει ένα εικονικό hardware και να διαχειριστεί τη λειτουργία των εικονικών μηχανών. Μία εικονική μηχανή βλέπει το εικονικό hardware και πάνω σε αυτό μπορεί να τρέξει το λειτουργικό της σύστημα, δηλαδή το guest λειτουργικό σύστημα (Guest OS). Έτσι, στο guest OS μπορεί πλέον να διαχειριστεί τις δικές του διεργασίες. Επειδή, μία εικονική μηχανή μπορεί να περιέχει ένα οποιοδήποτε σύνολο απαραίτητου λογισμικού, μπορεί, ουσιαστικά, να περιλαμβάνει όλες τις εξαρτήσεις που χρειάζονται για να τρέξει μία εφαρμογή. Με αυτόν τον τρόπο, η εφαρμογή κατορθώνει να τρέξει ορθά και απομονωμένα από το υπόλοιπο σύστημα, ικανοποιώντας όλες τις απαραίτητες εξαρτήσεις σε λογισμικό, δίχως να επηρεάζεται από το λογισμικό που είναι εγκατεστημένο στο host περιβάλλον.

Έναν εναλλακτικό τρόπο ώστε να τρέξει μία εφαρμογή απομονωμένα, εξασφαλίζοντας όλες τις εξαρτήσεις λογισμικού, αποτελεί η χρήση Containers (Σχήμα 2.2). Ένα λογισμικό διαχείρισης containers είναι το Docker, το οποίο χρησιμοποιείται στην παρούσα διπλωματική εργασία και με βάση αυτό θα γίνει όλη η περιγραφή και ανάλυση της εργασίας.

Όπως και στην περίπτωση των Εικονικών Μηχανών, έτσι και σε αυτήν την περίπτωση, πρέπει στο φυσικό σύστημα να εκτελείται ένα λειτουργικό σύστημα. Στην περίπτωση, όμως, των Containers, δεν χρειάζεται να τρέχει ένας hypervisor αλλά ένας container engine. Ο container engine είναι υπεύθυνος για τη διαχείριση των containers, όπως ήταν υπεύθυνος και ο hypervisor για τη διαχείριση των εικονικών μηχανών.

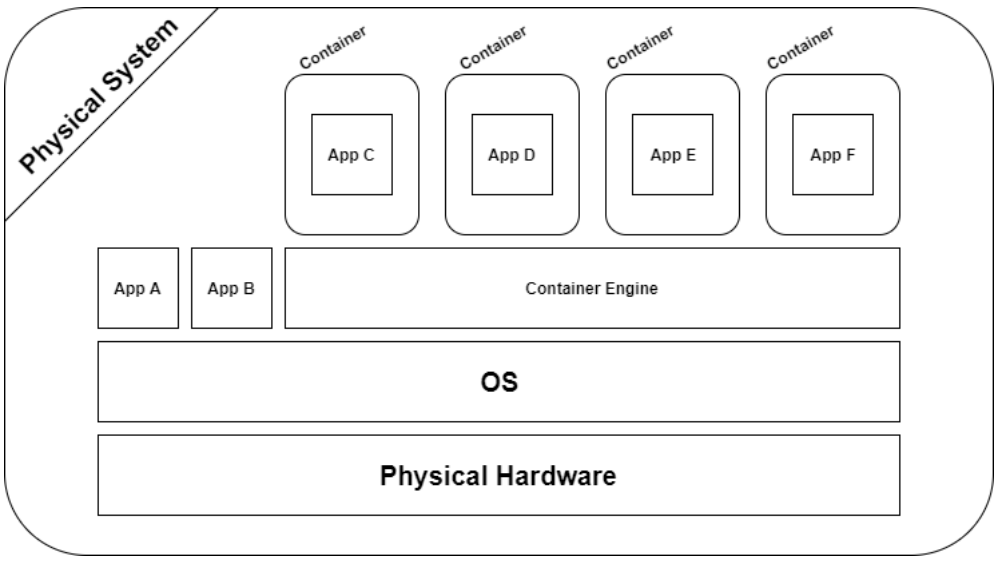
Ο container engine είναι σχεδιασμένος με βάση μία αρχιτεκτονική πελάτη-εξυπηρετητή (client-server). Ως εξυπηρετητής (server) λειτουργεί μία διεργασία που εκτελείται από το Λειτουργικό Σύστημα και τρέχει στο παρασκήνιο. Η διεργασία αυτή, η οποία ονομάζεται docker daemon και στο σύστημα εμφανίζεται με την εντολή dockerd, είναι, στην παραγματικότητα, υπεύθυνη για τη διαχείριση των containers αλλά και όλων των εργαλείων που χρειάζονται ώστε να λειτουργήσουν



Σχήμα 2.1: Αρχιτεκτονική Συστήματος με Εικονικές Μηχανές

- για παράδειγμα container images, volumes και networks. Προκειμένου ο docker daemon να είναι διαχειρίσιμος και να είναι δυνατή η αλληλεπίδραση με αυτόν, ορίζεται ένα REST API. Στον πελάτη (client), παρέχεται ένα CLI (μέσω της εντολής docker), το οποίο χρησιμοποιεί το REST API και επιτρέπει με αυτόν τον τρόπο στον χρήστη να αλληλεπιδράσει με τον docker daemon. Κατ' επέκταση, μέσω του CLI, ο χρήστης μπορεί να χειριστεί εύκολα τους containers και όλα τα εργαλεία που παρέχονται.

Ωστόσο, σε αντίθεση με τις Εικονικές Μηχανές και τον hypervisor, ο container engine επιτρέπει στους containers να τρέχουν στο χώρο χρήστη (user space) και ταυτόχρονα να επικοινωνούν με τον πυρήνα του λειτουργικού συστήματος (OS kernel) σε Linux Συστήματα. Με αυτόν τον τρόπο, ένας container δε χρειάζεται να τρέχει ένα λειτουργικό σύστημα όπως χρειαζόταν για τις Εικονικές Μηχανές.



Σχήμα 2.2: Αρχιτεκτονική Συστήματος με Containers

Μία σημαντική διαφορά μεταξύ μίας Εικονικής Μηχανής και ενός Container είναι το είδος της εικονικοποίησης που εφαρμόζει το κάθε λογισμικό. Πιο συγκεκριμένα, σε μία εικονική μηχανή, όπως ήδη αναφέρθηκε, η εικονικοποίηση γίνεται στο επίπεδο του hardware, δηλαδή ο hypervisor προσφέρει εικονικό hardware. Αντίθετα, σε έναν Container η εικονικοποίηση γίνεται στο software. Για κάθε έναν Container, δημιουργείται μία διεργασία η οποία τρέχει στον host και καταναλώνει μνήμη και επεξεργαστική ισχύ όπως κάθε άλλη διεργασία. Χάρη στον container engine, οι διεργασίες που τρέχουν μέσα στους containers είναι εικονικά απομονωμένες αλλά ταυτόχρονα, ο container engine επιτρέπει στους containers να χρησιμοποιήσουν τον kernel του host και κατ' επέκταση το hardware. Έτσι, για να τρέξει μία διεργασία μέσα σε έναν container απαιτούνται λιγότεροι πόροι από το host σύστημα, σε σχέση με μία διεργασία που πρέπει να τρέξει μέσα σε μία εικονική μηχανή. Κατ' επέκταση, ο container που τρέχει τη διεργασία καθώς και ο container engine είναι πιο γρήγορη και ελαφριά (light-weight) διαδικασία σε σχέση με την εικονική μηχανή και τον hypervisor.

Ένα Docker Image είναι ένα ελαφρύ, ανεξάρτητο, εκτελέσιμο πακέτο με λογισμικό, το οποίο περιέχει εκτελέσιμο κώδικα καθώς και όλες τις εξαρτήσεις που χρειάζεται για να εκτελεστεί. Η λογική ενός container image μπορεί να συσχετισθεί με ένα archive αρχείο. Ένα archive αρχείο περιέχει ένα ή περισσότερα αρχεία τα οποία, συνήθως, αφορούν ένα συγκεκριμένο θέμα. Ένα archive αρχείο πέρα από την οργάνωση που προσφέρει (συγκεντρώνοντας ένα σύνολο αρχείων μέσα σε ένα αρχείο) είναι χρήσιμο σε έναν χρήστη μόνο όταν αυτός θελήσει να διαβάσει τα περιεχόμενά του, οπότε και τα εξάγει. Αντίστοιχα, ένα container image περιέχει ένα ή περισσότερα εκτελέσιμα αρχεία και τμήματα κώδικα τα οποία, συνήθως, αφορούν μία συγκεκριμένη εφαρμογή. Ένα container image πέρα από την οργάνωση που προσφέρει (συγκεντρώνοντας ένα σύνολο εκτελέσιμου κώδικα μέσα σε ένα image) είναι χρήσιμο σε έναν χρήστη μόνο όταν αυτός θελήσει να τρέξει την εφαρμογή, οπότε και εκτελεί το container image. Ένας container είναι το στιγμιότυπο ενός container image που εκτελείται. Ο container καταλαμβάνει μνήμη και επεξεργαστική ισχύ στο host περιβάλλον με βάση τον εκτελέσιμο κώδικα που περιέχεται στο image. Κάθε ένας container έχει το δικό του απομονωμένο σύστημα αρχείων (filesystem). Το σύστημα αρχείων που παρέχεται σε έναν container εξαρτάται από το distribution πάνω στο οποίο στηρίζεται το container image (και για όλα τα Linux distributions είναι το ίδιο).

Κεφάλαιο 3

Αρχιτεκτονική Συστήματος

3.1 Microservices και Containerization

Σε αρκετές εφαρμογές, όλο το λογισμικό είναι συγκεντρωμένο σε μία Εικονική Μηχανή. Έτσι, όλες οι διεργασίες της εφαρμογής εκτελούνται μέσα σε αυτήν. Η αρχιτεκτονική αυτή είναι μονολιθική και κατ' επέκταση όλα τα τμήματα κώδικα είναι στενά συνδεδεμένα μεταξύ τους. Κατ' αυτόν τον τρόπο, συχνά σε τέτοιες αρχιτεκτονικές, η τροποποίηση του κώδικα καθιστάται δύσκολη εργασία καθώς μία αλλαγή ενδέχεται να προκαλέσει αλλαγές και σε τμήματα κώδικα που φαινομενικά δε σχετίζονται με το τροποποιημένο τμήμα. Και αυτό διότι η εφαρμογή δεν έχει αναπτυχθεί από ανεξάρτητα μέλη που συνεργάζονται μεταξύ τους αλλά έχει υλοποιηθεί ως μία ενιαία μεγάλη μονάδα της οποίας τα μικρότερα μέρη γνωρίζουν την ύπαρξη των υπολοίπων και έχουν αναπτυχθεί σε σχέση και με βάση τα υπόλοιπα αυτά μέρη.

Μία εναλλακτική αρχιτεκτονική είναι αυτή των Μικροϋπηρεσιών (Microservices). Σε αυτό το πλαίσιο, η εφαρμογή δεν είναι ένα μεγάλο κομμάτι κώδικα αλλά αποτελείται από μικρότερα τμήματα τα οποία είναι ρητά διαχωρισμένα ματαξύ τους από την αρχή. Τα Microservices αναπτύσσονται ξεχωριστά το ένα από το άλλο, συνήθως σε διαφορετικές γλώσσες προγραμματισμού ώστε να επιλέγονται η γλώσσα αλλά και τα εργαλεία τα οποία ταιριάζουν περισσότερο στις ανάγκες του εκάστοτε Microservice και τα οποία διευκολύνουν την ανάπτυξή του. Κατά συνέπεια, δεν είναι απαραίτητο ένα Microservice να γνωρίζει την ύπαρξη ενός άλλου Microservice. Επιπλέον, αν δύο ή περισσότερα Microservices είναι αναγκαίο να επικοινωνούν μεταξύ τους για να λειτουργεί σωστά το σύστημα, τότε αυτά επικοινωνούν μεταξύ τους χρησιμοποιώντας κάποιες διεπαφές (APIs). Έτσι, ακόμα και σε αυτήν την περίπτωση, ένα Microservice δε γνωρίζει τίποτα για την υλοποίηση ενός άλλου Microservice. Μάλιστα, ένα Microservice μπορεί να τροποποιηθεί δίχως να επηρεάσει τη λειτουργία άλλων Microservices ή τη λειτουργία του συνόλου της εφαρμογής με την προϋπόθεση πως οι διεπαφές αυτές δεν έχουν αλλάξει και είναι σωστά υλοποιημένες.

Το γεγονός πως σε μία Εικονική Μηχανή, όλος ο κώδικας είναι συγκεντρωμένος μέσα σε αυτή ως μία μονάδα καθιστά δύσκολη την κλιμακωσιμότητα της εφαρμογής. Προκειμένου η εφαρμογή να κλιμακωθεί πρέπει να προστεθεί νέος κώδικας ο οποίος, όμως, θα κάνει την εφαρμογή πιο περίπλοκη δημιουργώντας νέες εξαρτήσεις μεταξύ διαφόρων τμημάτων κώδικα. Αντιθέτως, μία εφαρμογή που αναπτύσσεται με Microservices είναι εύκολα κλιμακώσιμη. Δεδομένου πως ένα Microservice έχει αναπτυχθεί ανεξάρτητα από τα υπόλοιπα, είναι εύκολα κλιμακώσιμο καθώς μπορεί, εν γένει, να προστεθεί άλλο ένα ίδιο Microservice το οποίο θα εκτελεί την ίδια λειτουργία. Τα υπόλοιπα Microservices μπορούν να επικοινωνήσουν με αυτό χρησιμοποιώντας την διεπαφή που παρέχει.

Ταυτόχρονα, τα Microservices μπορούν να αποτελέσουν τη βάση ενός κατανεμημένου συστήματος. Τα Microservices μπορούν να τρέχουν και σε απομακρυσμένες μηχανές με την προϋπόθεση πως συνεχίζουν να επικοινωνούν μεταξύ τους χρησιμοποιώντας τη διαπαφή που προσφέρει το καθένα. Έτσι, η εφαρμογή μπορεί αφενός να κλιμακωθεί προσφέροντας απόδοση και αφετέρου να κατανεμηθεί προσφέροντας υψηλή διαθεσιμότητα (high availability) και αξιοπιστία. Αυτά τα χαρακτηριστικά οδηγούν στη δημιουργία cloud-native εφαρμογών. Πρόκειται για εφαρμογές στις οποίες δεν έχει σημασία το πού εκτελούνται αλλά κυρίως το πώς έχουν αναπτυχθεί και το πώς εκτελούνται. Είναι, δηλαδή, εφαρμογές οι οποίες μπορούν να καταναμηθούν και να εκτελε-

στούν σε διάφορες μηχανές καθώς έχουν σχεδιαστεί βασισμένες σε ξεχωριστά και ανεξάρτητα Microservices.

Η χρήση των Docker Containers βοηθά στη σχεδίαση και στην ανάπτυξη Microservices για μία εφαρμογή. Συγκεκριμένα, κάθε ένας Container αναλαμβάνει να εκτελέσει μία μικρότερη ανεξάρτητη λειτουργία μέσα στην εφαρμογή. Έτσι, ένας Container συνήθως εκτελεί μία συγκεκριμένη διεργασία. Όταν η διεργασία αυτή ολοκληρώνεται ή γενικότερα τερματίζεται για οποιονδήποτε λόγο τότε τερματίζεται και η λειτουργία του Container.

Στην παρούσα εφαρμογή, η σχεδίαση του εξυπηρετητή ο οποίος μπορεί να μεταγλωττίζει πηγαίο κώδικα μίας γλώσσας σε Webassembly, γίνεται με βάση τρεις διαφορετικούς Containers. Πιο συγκεκριμένα, ένας Container αποτελεί το backend της εφαρμογής και περιέχει όλους τους compilers ώστε να μπορεί να γίνει η μεταγλώττιση. Ο δεύτερος container αποτελεί το frontend και παρέχει μία διεπαφή χρήστη (UI) ώστε ένας χρήστης να μπορεί να χρησιμοποιήσει εύκολα την εφαρμογή. Ο τρίτος, και τελευταίος Container, θα περιέχει τη βάση δεδομένων ώστε να αποθηκεύονται μόνιμα δεδομένα τα οποία είναι απαραίτητα. Κατ' επέκταση, αναπτύσσονται τρία διαφορετικά Container Images.

3.1.1 Backend Service

Το Backend Service αποτελείται από έναν Container ο οποίος είναι υλοποιημένος σε Python και τρέχει έναν uwsgi REST εξυπηρετητή (uswgi rest server). Ο Container έχει εγκατεστημένους τους μεταγλωττιστές για τρεις γλώσσες. Πιο αναλυτικά, περιέχει το emscripten ώστε να μπορέσει να μεταγλωττίσει τις γλώσσες C και C++ καθώς επίσης έχει εγκατεστημένο τον μεταγλωττιστή για την Golang. Ο rest server παρέχει ένα REST API μέσω του οποίου ο χρήστης έχει τη δυνατότητα να μεταγλωττίσει πηγαίο κώδικα από τις τρεις προαναφερθείσες γλώσσες σε Webassembly. Ο χρήστης μπορεί να ανεβάσει ένα αρχείο με πηγαίο κώδικα στην εφαρμογή, διευκρινίζοντας τη γλώσσα και ορισμένες παραμέτρους μεταγλώττισης και στη συνέχεια να κατεβάσει τα αποτελέσματα της μεταγλώττισης έτσι όπως προέκυψαν μέσα στον Container. Ταυτόχρονα, μπορεί να εγγραφεί στην εφαρμογή (sign up) και να συνδεθεί σε αυτή (log in). Ως συνδεδεμένος χρήστης μπορεί να δει όλα τα αρχεία που έχει ανεβάσει και να κατεβάσει εκ νέου τα αποτελέσματα της μεταγλώττισης των αρχείων αυτών ή ακόμα και να τα διαγράψει από την εφαρμογή. Για να έχει τη δυνατότητα να πραγματοποιήσει τις παραπάνω ενέργειες είναι απαραίτητο να ανεβάσει τα αρχεία όντας συνδεδεμένος στην εφαρμογή αλλά και να ζητήσει να αποθηκευτούν σε δικό του λογαριασμό.

Κατά τη μεταγλώττιση των αρχείων, παράγονται άλλα αρχεία ως αποτελέσματα μέσα στο Σύστημα Αρχείων του Backend Container. Αν η λειτουργία ενός Container τερματιστεί για οποιονδήποτε λόγο και ο Container διαγραφεί, τότε όλα τα αρχεία που έχουν δημιουργηθεί μέσα στο Σύστημα Αρχείων του χάνονται. Για να αποφευχθεί αυτή η συμπεριφορά, μπορούν να χρησιμοποιηθούν Docker Volumes. Ως Volume αναφέρεται μία περιοχή αποθήκευσης σε ένα μηχάνημα (για παράδειγμα ένας Σκληρός Δίσκος, ένα μικρότερο τμήμα ενός Σκληρού Δίσκου, ακόμα και ένα Floppy Disk), στην οποία μπορούν να αποθηκευθούν δεδομένα από τον χρήστη, όπως γίνεται και με οποιοδήποτε άλλο τμήμα του Συστήματος Αρχείων του μηχανήματος (με την προϋπόθεση πως ο χρήστης έχει τα απαραίτητα δικαιώματα). Ένα Docker Volume δεσμεύει ένα τμήμα του Συστήματος Αρχείων στο Host περιβάλλον, το οποίο μπορεί να τοποθετηθεί ως Volume σε έναν ή περισσότερους Containers. Με αυτόν τον τρόπο, οποιαδήποτε αλλαγή γίνεται σε αυτό το Volume από το Host περιβάλλον αντανακλάται και στο Σύστημα Αρχείων του Container και αντιστρόφως, οποιαδήποτε αλλαγή γίνεται στο Volume από τον Container αντανακλάται και στο Σύστημα Αρχείων του Host περιβάλλοντος. Για το σκοπό αυτό, δημιουργείται ένας φάκελος στο Σύστημα Αρχείων του Host το οποίο χρησιμοποιείται ως Volume και τοποθετείται στον Backend Container. Κατά συνέπεια, όταν ένας χρήστης ανεβάζει ένα αρχείο, το αρχείο αυτό αποθηκεύεται μέσα σε αυτό το Volume. Επιπλέον, όλα τα παραγόμενα από τη μεταγλώττιση αρχεία αποθηκεύονται επίσης μέσα σε αυτό το Volume. Έτσι, ακόμα και αν διαγραφεί ένας Container όλες οι πληροφορίες και τα αρχεία που έχουν παραχθεί, διατηρούνται στο Host περιβάλλον μέσω του Volume, και ο

νέος Backend Container που θα δημιουργηθεί θα έχει πρόσβαση στα αρχεία αυτά μόλις το Docker Volume τοποθετηθεί και πάλι στον Container. Το Volume αυτό, τοποθετείται στη διαδρομή /results μέσα στο Σύστημα Αρχείων του Container.

Για λόγους ασφάλειας, η διεργασία, η οποία τρέχει στο Backend Service, δεν πρέπει να εκτελείται ως root χρήστης. Επειδή, οι containers προσφέρουν εικονικοποίηση στο software και όχι στο hardware, χρησιμοποιώντας άμεσα τον Kernel του host περιβάλλοντος, τα User IDs είναι κοινά τόσο στο host περιβάλλον όσο και μέσα στον container. Αυτό σημαίνει πως αν, για παράδειγμα, μία διεργασία, η οποία εκτελείται μέσα στον container από τον χρήστη με ID 100, τότε εκτελείται και στο host περιβάλλον από τον χρήστη με ID 100. Άρα, αν μία διεργασία εκτελείται από τον root χρήστη μέσα στον container, δηλαδή τον χρήστη με ID 0, τότε τρέχει ως root και στο host περιβάλλον. Σε περίπτωση που το λογισμικό του container έχει κενά ασφαλείας, είναι πιθανό κάποιος να προσπαθήσει να τα εκμεταλλευτεί ώστε να αποκτήσει πρόσβαση στον container. Αν καταφέρει να αποκτήσει πρόσβαση και ο χρήστης που τρέχει τις διεργασίες μέσα στον container είναι ο root, τότε ο εισβολέας αποκτά άμεσα root access και στο host περιβάλλον. Για το λόγο αυτό, οι διεργασίες μέσα στον container πρέπει να τρέχουν ως non-root χρήστης. Μάλιστα, θα πρέπει για κάθε ένα service να χρησιμοποιείται διαφορετικός non-root χρήστης ώστε αν ένας εισβολέας αποκτήσει πρόσβαση σε έναν container να έχει όσο το δυνατόν λιγότερα δικαιώματα. Στο συγκεκριμένο Service, ο server εκτελείται από τον χρήστη με UID 1000.

3.1.2 Database Service

Η βάση δεδομένων της εφαρμογής αποτελεί ένα ξεχωριστό service και για το λόγο αυτό δημιουργείται και ο αντίστοιχος Container. Ο Database Container τρέχει έναν PostgreSQL database server. Δηλαδή, τρέχει μία PostgreSQL βάση δεδομένων και έναν εξυπηρετητή, επιτρέποντας μέσω του εξυπηρετητή την επικοινωνία με τη βάση αυτή. Η PostgreSQL είναι ένα Αντικειμενοστραφές Σχεσιακό Σύστημα Διαχείρισης Βάσης Δεδομένων (κυρίως συναντόμενο ως Object-Relational Database Management System, ORDBMS), επιτρέποντας σχεδιασμό βάσεων δεδομένων τόσο με σχεσιακό όσο και με αντικειμενοστραφές μοντέλο. Μάλιστα, η PostgreSQL είναι μία από τις πιο αποδοτικές βάσεις δεδομένων. Στη συγκεκριμένη εφαρμογή, η βάση δεδομένων αποτελείται από δύο πίνακες. Ο πρώτος πίνακας είναι ο πίνακας που αφορά τους εγγεγραμμένους χρήστες και περιέχει τις πληροφορίες που σχετίζονται με κάθε έναν από αυτούς. Ο δεύτερος πίνακας είναι ο πίνακας των αρχείων πηγαίου κώδικα. Σε αυτόν τον πίνακα, αν ένας χρήστης το επιτρέπει, αποθηκεύονται πληροφορίες για τα αρχεία τα οποία έχουν ανέβει από τον συγκεκριμένο χρήστη. Με αυτόν τον τρόπο, όταν ο χρήστης το επιθυμεί, είναι δυνατός ο εντοπισμός και η ανάκτηση των αρχείων τα οποία έχει ανεβάσει αλλά και των αποτελεσμάτων της μεταγλώττισής τους, μέσα από το Σύστημα Αρχείων της εφαρμογής.

Και στο συγκεκριμένο service χρησιμοποιείται ένα Docker Volume. Συγκεκριμένα, δημιουργείται ένα φάκελος στο Host περιβάλλον στον οποίο αποθηκεύεται η βάση δεδομένων. Ο φάκελος /var/lib/postgresql/data μέσα στον Container περιέχει τη βάση δεδομένων, οπότε σε αυτή τη διαδρομή τοποθετείται και το Volume. Με αυτόν τον τρόπο, οποιαδήποτε αλλαγή συμβαίνει στη βάση δεδομένων αποθηκεύεται και στο Host περιβάλλον, ώστε ακόμα και αν διαγραφεί ο Container, τα δεδομένα της βάσης να μη διαγραφούν. Αν, δηλαδή, χρειαστεί να δημιουργηθεί ένας νέος Database Container, τότε το Volume θα τοποθετηθεί στη διαδρομή /var/lib/postgresql/data μέσα στον Container και η βάση δεδομένων θα μπορεί να χρησιμοποιηθεί και πάλι, χωρίς να έχουν χαθεί δεδομένα.

Όπως και στο Backend Service, έτσι και σε αυτό, για λόγους ασφαλείας, οι διεργασίες του container εκτελούνται από τον non-root χρήστη με UID 999.

3.1.3 Frontend Service

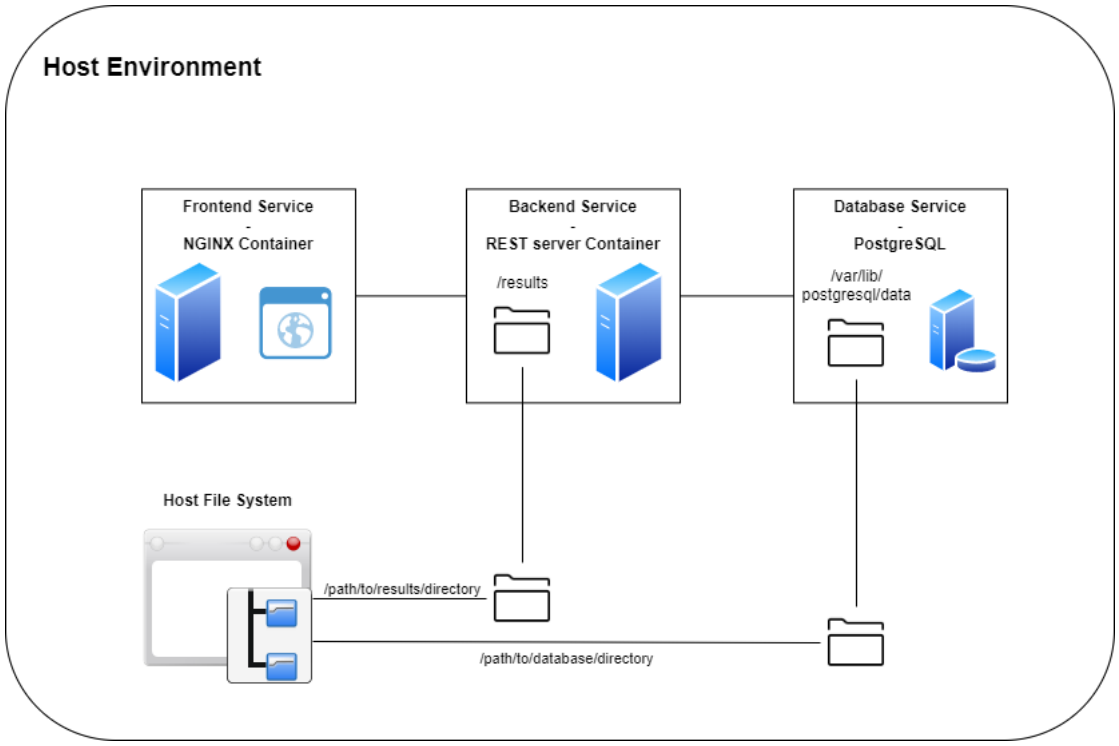
Στην παρούσα εφαρμογή υλοποιείται, επίσης, μία Διεπαφή Χρήστη (User Interface, UI). Για να μπορέσει να χρησιμοποιηθεί η διεπαφή αυτή, απαιτείται ένα επιπλέον service το οποίο υλο-

ποιείται μέσω ενός Container ο οποίος τρέχει έναν NGINX εξυπηρετητή. Η διεπαφή είναι υλοποιημένη σε Angular και έχει τοποθετηθεί μέσα στον Frontend Container. Όταν ένας χρήστης επιθυμεί να χρησιμοποιήσει τη διεπαφή (για παράδειγμα, ανοίγοντάς την σε ένα φυλλομετρητή), στέλνει ένα αίτημα (request) στον συγκεκριμένο server. Έπειτα, ανάλογα με το είδος του αιτήματος, ο NGINX απαντά με την αντίστοιχη HTML σελίδα καθώς και με την CSS και τη Javascript που τη συνοδεύουν.

Ο NGINX εξυπηρετητής είναι ένας γρήγορος εξυπηρετητής, ο οποίος μπορεί να χειριστεί πολλαπλά αιτήματα ταυτόχρονα. Για το λόγο αυτό, ο NGINX μπορεί να χρησιμοποιηθεί και ως reverse proxy εξυπηρετητής. Δηλαδή, μπορεί να χρησιμοποιηθεί ως ένας εξυπηρετητής ο οποίος λαμβάνει τα αιτήματα και στη συνέχεια τα προωθεί σε άλλους εξυπηρετητές ανάλογα με ορισμένα χαρακτηριστικά του αιτήματος που λαμβάνει, όπως για παράδειγμα το port στο οποίο δέχεται το αίτημα, το είδος του query ή μία μικρότερη συμβολοσειρά του query.

Και στο παρόν Frontend Service, λοιπόν, ο συγκεκριμένος NGINX εξυπηρετητής δεν παρέχει μόνο τη διεπαφή, δηλαδή μόνο το frontend. Αντιθέτως, είναι αυτός που δέχεται όλα τα request που φτάνουν στο σύστημα. Αν το αίτημα ζητά να λάβει τη διεπαφή, τότε ο NGINX εξυπηρετητής απαντά με την κατάλληλη HTML, CSS και Javascript. Στην περίπτωση, όμως, που το αίτημα δε ζητά να λάβει τη διεπαφή αλλά, για παράδειγμα, είναι ένα αίτημα μεταγλώττισης αρχείου ή εγγραφής στην εφαρμογή, τότε ο NGINX λειτουργεί ως reverse proxy εξυπηρετητής. Συγκεκριμένα, ο NGINX λαμβάνει το αίτημα αυτό και στη συνέχεια το προωθεί στον εξυπηρετητή που τρέχει στον Backend Container, ώστε να απαντήσει αυτός στο αίτημα. Με τη σειρά του, ο εξυπηρετητής στον Backend Container επεξεργάζεται το αίτημα και δημιουργεί μία απάντηση (response). Η απάντηση αυτή αποστέλλεται στον NGINX, ο οποίος στη συνέχεια στέλνει την απάντηση στον αρχικό πελάτη (client) από τον οποίο έλαβε το αίτημα.

Όπως και στα προηγούμενα Services, για λόγους ασφαλείας, οι διεργασίες του container εκτελούνται από έναν non-root χρήστη, ο οποίος στη συγκεκριμένη περίπτωση έχει UID 101.



Σχήμα 3.1: Μη Κλιμακωμένη Αρχιτεκτονική Συστήματος

3.2 Δίκτυο και επικοινωνία μεταξύ των Containers

Με αυτή τη σχεδίαση, κανένας Container, εκτός από τον Frontend Container ο οποίος τρέχει τον NGINX, δε χρειάζεται να είναι άμεσα ορατός σε κάποιο εξωτερικό δίκτυο, παρά μόνο μέσω του NGINX. Μόνο ο Frontend Container με τον NGINX είναι άμεσα ορατός και είναι αυτός που λαμβάνει όλα τα αιτήματα. Με αυτόν τον τρόπο, βελτιώνεται η ασφάλεια του συστήματος. Για παράδειγμα, ένας χρήστης δεν έχει τη δυνατότητα να επικοινωνήσει άμεσα με τη βάση δεδομένων και να αλληλεπιδράσει με αυτήν αφού κανένα αίτημα δε φτάνει άμεσα σε αυτή δίχως να περάσει από τον NGINX και τον uwsgi REST εξυπηρετητή.

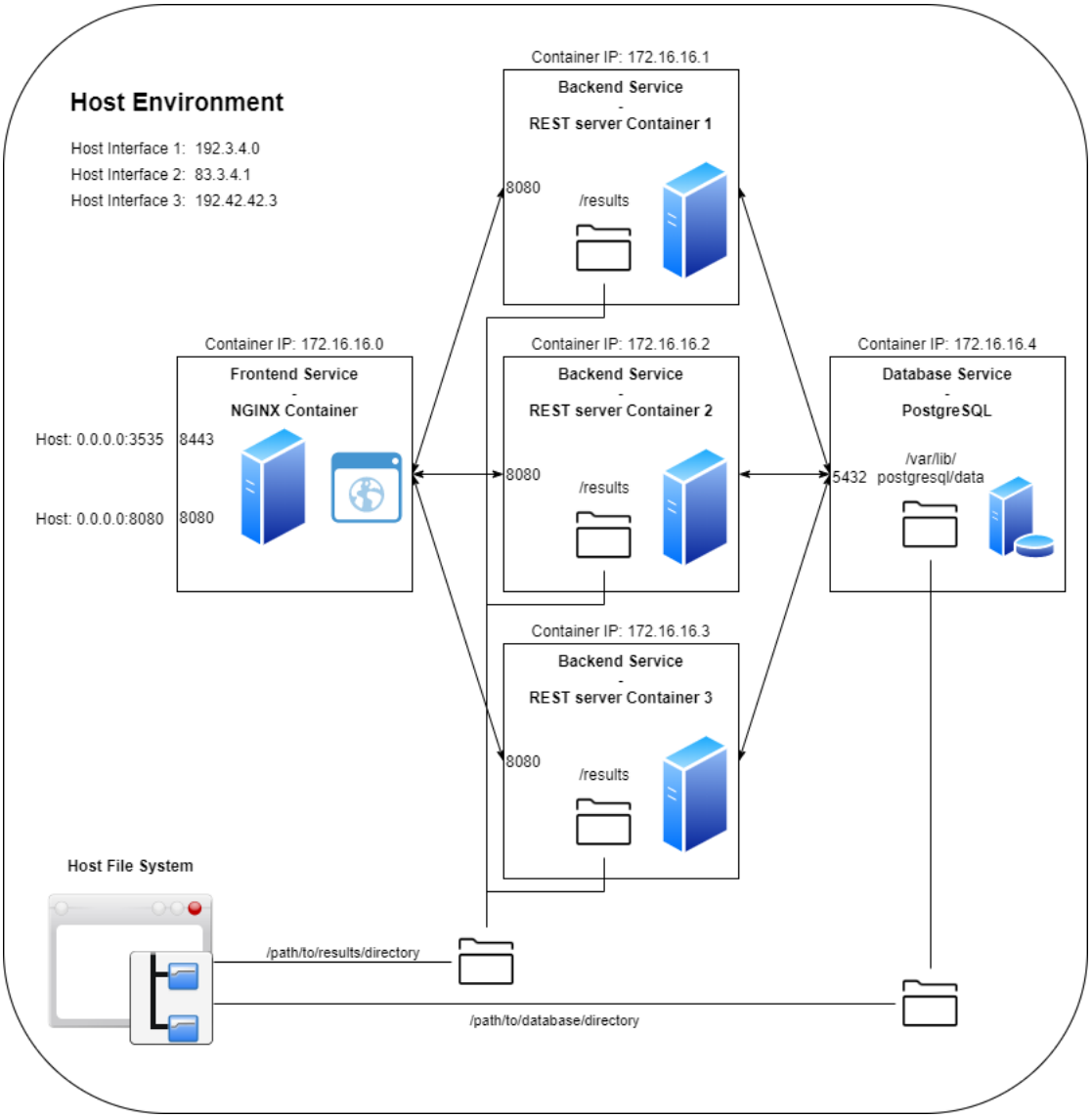
Το Docker εκτός από τη δημιουργία και τη διαχείριση των Containers, μπορεί να δημιουργεί και να διαχειρίζεται δίκτυα πάνω στα οποία συνδέονται οι Containers ώστε να επικοινωνούν μεταξύ τους. Κατά τη δημιουργία ενός νέου Docker Network, ο Docker Daemon παρέχει στο δίκτυο ένα σύνολο από διαθέσιμες IPs. Έπειτα, σε κάθε έναν Container ανατίθεται μία IP από το διαθέσιμο σύνολο. Κάθε ένας Container μπορεί να έχει παραπάνω από μία IPs αν βρίσκεται σε παραπάνω από ένα δίκτυα. Έτσι, ένας Container μπορεί να επικοινωνήσει με έναν άλλο εάν βρίσκονται τουλάχιστον σε ένα κοινό δίκτυο και εάν γνωρίζει την IP του δεύτερου. Επιπλέον, κάθε ένας Container έχει και ένα Hostname.

Δεδομένου πως η ανάθεση των IPs στους Containers γίνεται με τυχαίο τρόπο και φυσικά γίνεται μετά την υλοποίηση του κώδικα, είναι δύσκολο ένας Container να γνωρίζει εκ των προτέρων την IP ενός άλλου Container ώστε να τη χρησιμοποιήσει για να επικοινωνήσουν. Αντιθέτως, το hostname ενός Container μπορεί να είναι τυχαίο αλλά αν χρειάζεται μπορεί να είναι σαφώς ορισμένο. Στην περίπτωση που το hostname είναι ορισμένο τότε, κατά την υλοποίηση του κώδικα, ένας Container μπορεί να χρησιμοποιήσει το ρητά ορισμένο hostname ενός άλλου Container για να επικοινωνήσει με αυτόν. Αυτό είναι δυνατό, διότι σε ένα Docker Network (όχι, όμως, σε ένα από τα προϋπάρχοντα Docker Networks που δημιουργεί αυτόματα το Docker) γίνεται αυτόματο service discovery. Δηλαδή, γίνεται ανάλυση των hostnames των Containers σε IP διευθύνσεις για όλους τους containers εντός του ίδιου δικτύου (hostname resolution to IP addresses), οπότε οι Containers μπορούν να χρησιμοποιούν τα hostnames των άλλων Containers του δικτύου για να επικοινωνούν μεταξύ τους.

Ταυτόχρονα, κάθε ένα service εκθέτει (expose), αν χρειάζεται, κάποια ports στα οποία ακούει και μπορεί να λάβει αιτήματα. Μόνο οι Containers εντός του ίδιου δικτύου με τον Container που τρέχει αυτό το service μπορούν να συνδεθούν στα συγκεκριμένα ports. Αυτή η συμπεριφορά προσφέρει ασφάλεια στο δίκτυο καθώς κανένας χρήστης που δεν έχει πρόσβαση στο δίκτυο δεν μπορεί να συνδεθεί σε αυτούς τους Containers. Ωστόσο, υπάρχουν περιπτώσεις, κατά τις οποίες είναι επιθυμητό ένας χρήστης να μπορεί να συνδεθεί στα ports ενός Container. Για να επιτευχθεί αυτή η συμπεριφορά, ένα service μπορεί να δημοσιοποιήσει (publish) κάποια από τα ports του. Συγκεκριμένα, μπορεί να αντιστοιχίσει (map) τα δικά του ports με ports του host περιβάλλοντος. Η αντιστοίχιση αυτή μπορεί να γίνει είτε με κάποιο τυχαίο port μέσα σε ένα εύρος διαθέσιμων ports του host είτε μπορεί να οριστεί ρητά ένα προς ένα αντιστοίχιση μεταξύ ενός port του service με ένα port του host. Όταν ένα service εκθέτει απλά ένα port, τότε μέσα στο ίδιο δίκτυο μπορούν να συνυπάρχουν πολύ Containers οι οποίοι τρέχουν το ίδιο service (δηλαδή προέρχονται από το ίδιο Docker Image), με αποτέλεσμα το service να είναι εύκολα κλιμακώσιμο. Ωστόσο, σε ένα service στο οποίο υπάρχει ρητή ένα προς ένα αντιστοίχιση με ένα port του host, δεν είναι δυνατή η κλιμάκωσή του καθώς δύο Containers θα επιχειρήσουν να ταιριάξουν στο ίδιο port του host. Επιπλέον, ένα service μπορεί να αντιστοιχίσει ένα port του με ένα port του host σε συγκεκριμένο interface. Αν δεν προσδιορισθεί το interface στον host τότε οποιοδήποτε αίτημα έρχεται σε κάποιο interface του host στο συγκεκριμένο port θα προωθείται στον Container.

Συγκεκριμένα, το Database Service που τρέχει τη βάση δεδομένων, εκθέτει το port 5432, το οποίο είναι το προκαθορισμένο port στο οποίο ακούει η PostgreSQL. Το Backend Service έχει οριστεί να ακούει στο port 8080, το οποίο και εκθέτει. Τέλος, το Frontend Service με τον NGINX δημοσιοποιεί δύο ports, το 8443 και το 8080 τα οποία αντιστοιχίζονται στα ports 3535 και 8080 του host για οποιοδήποτε interface (δηλαδή στην IP 0.0.0.0, η οποία βλέπει όλα τα interfaces του host). Τα

αιτήματα που λαμβάνει ο NGINX στο port 8443, τα αντιμετωπίζει ως αιτήματα που αφορούν το WebUI και κατ' επέκταση απαντά με κάποια σελίδα του Angular Frontend. Τα αιτήματα τα οποία λαμβάνει στο port 8080, τα αντιμετωπίζει ως αιτήματα που αφορούν το backend και κατ' επέκταση τα προωθεί στο Backend Service. Όταν το σύστημα κλιμακώνεται και επομένως υπάρχουν περισσότεροι Backend Containers που τρέχουν έναν uwsgi REST server, τότε ο NGINX προωθεί σε αυτούς τα αιτήματα κυκλικά με Round Robin επιλογή. Μία κλιμακωμένη αρχιτεκτονική (με τρεις uwsgi rest servers) μαζί με το δίκτυο εικονίζεται στο Σχήμα 3.2.



Σχήμα 3.2: Κλιμακωμένη Αρχιτεκτονική Συστήματος

Κεφάλαιο 4

Αρχιτεκτονική Κώδικα

4.1 Σχεδίαση Containers και Multistage Builds

Κάθε ένα Docker Image για ένα service της εφαρμογής χτίζεται πάνω σε κάποιο προϋπάρχον ευρέως διαθέσιμο Docker Image. Πιο αναλυτικά, για τη βάση δεδομένων (Database Service) χρησιμοποιείται το *postgres:12.2* του Docker Hub, οπότε το μέγεθος του Image είναι 314MB. Για τα άλλα δύο Docker Images, δηλαδή του Frontend Service και του Backend Service, χρησιμοποιούνται Multistage build.

Πιο συγκεκριμένα, η κλασική μέθοδος δημιουργίας ενός Docker Image είναι να χρησιμοποιηθεί ως βάση (αρχικό επίπεδο) ένα προϋπάρχον Docker Image και στη συνέχεια να προστίθενται ως νέα επίπεδα (layers) σε αυτό άλλα πακέτα λογισμικού. Συχνά, για να παραχθούν κάποια εκτελέσιμα αρχεία (ή γενικότερα κάποια χρήσιμα αρχεία) απαιτείται να υπάρχει προεγκατεστημένο στο σύστημα κάποιο λογισμικό, το οποίο θα βοηθήσει στην παραγωγή των νέων αρχείων. Ωστόσο, από τη στιγμή που θα παραχθούν τα αρχεία αυτά δε χρειάζονται το προεγκατεστημένο αυτό λογισμικό για να λειτουργήσουν ή να χρησιμοποιηθούν από άλλα εργαλεία. Δηλαδή, το λογισμικό αυτό θα μπορούσε να αφαιρεθεί και το Docker Image θα μπορούσε να λειτουργήσει χωρίς αυτό. Το multistage build αποτελεί μία εναλλακτική μέθοδο δημιουργίας ενός Docker Image η οποία κινείται προς αυτήν την κατεύθυνση. Συγκεκριμένα, χρησιμοποιώντας την κλασική μέθοδο μπορεί να κατασκευάσει διάφορα Images, τα οποία μπορούν βασίζονται σε διαφορετικά προϋπάρχοντα Images, και τα οποία λειτουργούν ως ενδιάμεσα βήματα. Το τελικό Image μπορεί να χτίζεται πάνω σε κάποιο άλλο Image και τα νέα επίπεδα που προστίθενται σε αυτό μπορούν να αναφέρονται στα ενδιάμεσα Images και να παίρνουν αρχεία από αυτά. Με αυτόν τον τρόπο, το τελικό Image θα περιέχει μόνο τον κώδικα ο οποίος του είναι πραγματικά απαραίτητος ώστε να λειτουργήσει. Κατ' επέκταση, το μέγεθος του Image είναι μικρότερο σε σχέση με το μέγεθος που θα είχε αν περιείχε όλες τις εξαρτήσεις λογισμικού που χρειάστηκαν στα ενδιάμεσα βήματα και ταυτόχρονα το σύστημα, μέσα στο οποίο θα χρησιμοποιηθεί το Image, καθίσταται πιο ασφαλές αφού περιέχει λιγότερα εργαλεία τα οποία θα μπορούσε να εκμεταλλευτεί κάποιος για να παρέμβει κακόβουλα στο σύστημα.

Στην περίπτωση του Backend Service, δηλαδή του REST server, το τελικό Image χτίζεται πάνω στο *python:3.8-slim-buster* Image. Το Image αυτό είναι η μικρού μεγέθους έκδοση της Python 3.8, καθώς περιέχει μόνο τα εργαλεία που χρειάζονται για να μπορεί να εκτελεστεί η Python, και δεν περιέχει λογισμικό το οποίο είναι ευρέως χρησιμοποιούμενο σε διάφορες εφαρμογές αλλά είναι αχρείαστο στην παρούσα εφαρμογή. Ως ενδιάμεσα στάδιο, χρησιμοποιείται το *trzeci/emscripten-slim:sdk-tag-1.39.4-64bit*, από το οποίο χρησιμοποιούνται μόνο τα απαραίτητα εργαλεία που χρειάζονται για να τρέξει το emscripten, δηλαδή ο compiler για τις C και C++. Στη συνέχεια, εγκαθιστάται η Golang 1.13.7. Έπειτα, εγκαθιστώνται όλες οι εξαρτήσεις και οι python βιβλιοθήκες που χρειάζονται για να τρέξει ο rest server και δημιουργείται ένας non-root χρήστης. Ο non-root χρήστης ονομάζεται server, έχει UID 1000 και GID 1000. Το μέγεθος του τελικού Image είναι 915MB.

Αντίστοιχα, για το Frontend Service, δηλαδή τον NGINX, χρησιμοποιείται επίσης multistage build. Ως βήμα, χρησιμοποιεί το Image *node:13.6.0*. Πάνω σε αυτό εγκαθίσταται η Angular 8, η οποία είναι το framework υλοποίησης του frontend, καθώς και όλα τα πακέτα που χρειάζονται για την υλοποίηση του frontend. Στη συνέχεια, μεταγλωττίζεται και χτίζεται το Angular frontend. Το

στάδιο αυτό δημιουργεί ένα πολύ μεγάλο Image (1.32GB) καθώς περιέχει πολλά node πακέτα, τα οποία μετά τη δημιουργία του distribution του frontend δε χρειάζονται. Το τελικό Image χτίζεται πάνω στο *nginx:1.17.7*. Περνάει το κατάλληλο configuration για τον NGINX και αντιγράφει στο παρόν Image μόνο το frontend distribution που παρήχθει στο προηγούμενο στάδιο, το οποίο είναι μικρό σε σχέση με το συνολικό μέγεθος του ενδιάμεσου Image. Έτσι, το τελικό Image εχει μέγεθος μόλις 128MB.

4.2 Λογισμικό Backend

Το Backend είναι υλοποιημένο σε Python και χρησιμοποιεί το Flask Microframework. Στην παρούσα διπλωματική εργασία, επιλέγεται διότι επιτρέπει αντικειμενοστραφή προγραμματισμό και διευκολύνει την ανάπτυξη οργανωμένου κώδικα. Ως framework της Python, είναι εύκολο να δημιουργηθεί ένα API γρήγορα για να υλοποιηθεί το Backend. Επιπλέον, το γεγονός πως το Backend είναι υλοποιημένο με Python και το Frontend είναι υλοποιημένο με ένα Javascript framework (Angular 8), τονίζει μία δυνατότητα που παρέχει η ανάπτυξη ενός συστήματος με Containers, δηλαδή την ανάπτυξη των Services σε διαφορετικές γλώσσες. Παράλληλα, η Python διαθέτει βιβλιοθήκες όπως η SQLAlchemy. Η SQLAlchemy είναι ένας Object Relational Mapper (ORM) δηλαδή είναι μία βιβλιοθήκη υπεύθυνη να καλύψει το χάσμα που υπάρχει μεταξύ του Backend και της βάσης δεδομένων ως προς τον τρόπο επικοινωνίας και τους τύπους δεδομένων. Προσφέρει μία υψηλού επιπέδου διεπαφή ώστε κατά την ανάπτυξη του Backend να χρησιμοποιείται Python για την υλοποίηση των Queries στη Βάση δεδομένων αντί να χρειάζεται να γραφεί αγνή SQL. Ο ORM πραγματοποιεί τις κατάλληλες μετατροπές στα Python αντικείμενα ώστε να σχηματίσει αυτόματα ένα αγνό SQL Query, το οποίο είναι κατανοητό από τη Βάση Δεδομένων και το οποίο, τελικά, στέλνεται στη βάση δεδομένων.

Το Flask προτιμάται, κυρίως, σε εφαρμογές οι οποίες χρησιμοποιούν έντονα τη CPU (CPU intensive) και χρειάζεται να έχουν καλή απόδοση, και όχι σε εφαρμογές οι οποίες κάνουν CRUD (Create, Read, Update, Delete) δουλειά και πραγματοποιούν πολλά I/O γεγονότα (I/O events). Παρά το γεγονός πως η Python δεν πραγματοποιεί άμεσα η ίδια το Compilation των αρχείων αλλά χρησιμοποιεί ξένους Compilers μέσα στο σύστημα, το Compilation είναι μία εργασία η οποία χρησιμοποιεί πολύ τη CPU (και γενικότερα τους πόρους του συστήματος) και κατ' επέκταση δεν ανήκει στις CRUD εργασίες. Επομένως, το Flask επιλέγεται ως πιο ταιριαστό framework για τη συγκεκριμένη εφαρμογή.

Ωστόσο, το ίδιο το Flask δεν παρέχει έναν production ready server παρά το γεγονός πως παρέχει δυνατότητες για ταυτόχρονη εξυπηρέτηση αιτημάτων με multithreading. Ένας server, για να μπορεί να εκτελεί Python modules, πρέπει να υλοποιεί ένα Web Server Gateway Interface (WSGI). Ο WSGI server δεν επικοινωνεί ποτέ άμεσα με τον πελάτη (client) που στέλνει το request αλλά έχει πάντα μπροστά του έναν άλλο Web Server. Ο Web Server πρέπει να έχει κατάλληλο configuration ώστε να μπορεί να μιλήσει με τον WSGI Server μέσω του WSGI πρωτοκόλλου. Αν διαθέτει τέτοιο configuration τότε το request του client προωθείται στον WSGI Server, ο οποίος επεξεργάζεται το request και επιστρέφει ένα response πίσω στον Web Server. Ο Web Server στέλνει το response που λαμβάνει πίσω στον client.[PEP3] Στην συγκεκριμένη αρχιτεκτονική, ο NGINX που τρέχει στον Frontend Container επιτελεί αυτόν τον σκοπό, επικοινωνώντας με τον WSGI REST Server μέσω του WSGI πρωτοκόλλου. Ορισμένοι production ready WSGI servers είναι οι Gunicorn (Green Unicorn), CherryPy, uWSGI και Apache με mod_wsgi module. Στην παρούσα εργασία, επιλέγεται ο uWSGI, ο οποίος είναι υλοποιημένος σε C και σε συνδυασμό με τον NGINX ως Web Server δομούν ένα αποδοτικό σύστημα.

4.2.1 Packages

Ο κώδικας του REST Server είναι δομημένος ως ένα Regular Python Package. Το Package ονομάζεται *rest_server* και αποτελείται από εξειδικευμένα modules και subpackages. Πιο αναλυτικά,

περιλαμβάνει το module *authentication.py*, το module *common.py* και τα subpackages *views*, *models* και *compile*.

Το *authentication.py* module ορίζει έναν python decorator, ο οποίος χρησιμοποιείται σε όλα τα requests τα οποία πρέπει να προέρχονται από authenticated χρήστες. Όταν ένας εγγεγραμμένος χρήστης συνδέεται στην εφαρμογή με τα στοιχεία του, τότε λαμβάνει ως απάντηση ένα JSON Web Token (JWT) (και συγκεκριμένα ένα JSON Web Signature - JWS), το οποίο είναι έγκυρο για ένα ορισμένο χρονικό διάστημα και περιέχει το μοναδικό αναγνωριστικό του χρήστη. Χρησιμοποιώντας αυτό το JWT, ο χρήστης αυτός έχει τη δικαιοδοσία να χρησιμοποιήσει επιπλέον routes του API καθώς και να ζητήσει πόρους του συστήματος στους οποίους δε θα είχε διαφορετικά πρόσβαση. Για να το πετύχει αυτό, θα πρέπει στα requests να περιλαμβάνεται ο Authorization header. Ο python decorator ελέγχει αν περιλαμβάνεται στο request ο Authorization header καθώς επίσης αν αυτός έχει σωστή δομή, δηλαδή αν περιλαμβάνει ένα έγκυρο JWT. Συγκεκριμένα, ο Authorization header έχει τη μορφή: 'Authorization: Bearer <valid JWT>'.[RFC7, RFC6]

Το *common.py* module περιλαμβάνει ορισμούς συναρτήσεων οι οποίες είναι κοινές και χρησιμοποιούνται σε διάφορα modules του package.

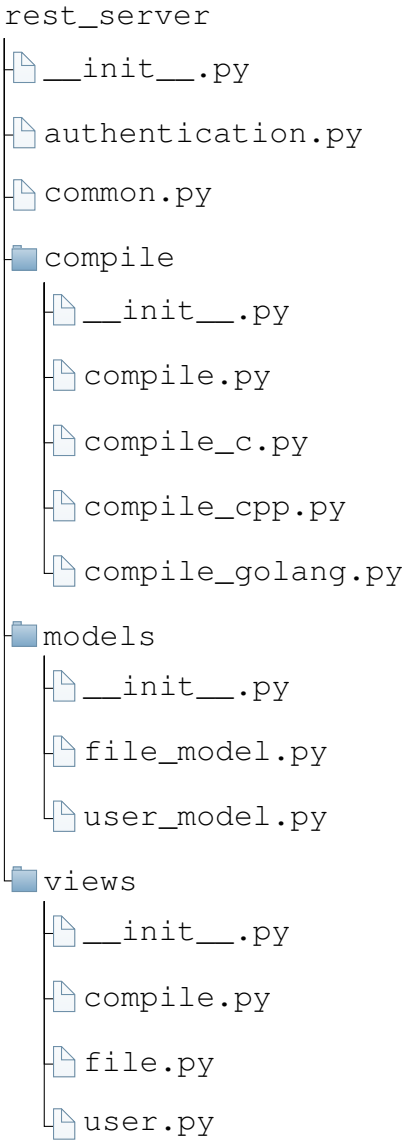
Το subpackage *views* περιέχει τα modules *compile.py*, *file.py* και *user.py*. Τα τρία παραπάνω modules ορίζουν τα views, δηλαδή το API της εφαρμογής. Έτσι, προκύπτουν τα ακόλουθα routes για το API:

Methods	**URI**	**Specifications**
POST	/api/compile/<language>	<language>=C | Cpp | Golang Body: multipart/form-data Fields: • code: File • compilation_options: JSON
POST	/api/compile/<language>/store	<language>=C | Cpp | Golang Body: multipart/form-data Fields: • code: File • compilation_options: JSON
GET	/api/files/personal_file_content	Query Parameters: • language • directory • name
GET	/api/files/all_personal	None
DELETE	/api/files/personal_file/<file_id>	<file_id> $\in \mathbb{N}$
POST	/api/signup	Body: multipart/form-data Fields: • username • password • email
POST	/api/login	Body: multipart/form-data Fields: • username • password

Πίνακας 4.1: Backend API

Το subpackage *models* περιλαμβάνει κώδικα που αφορά την αλληλεπίδραση με τη βάση δεδομένων και περιέχει τα modules *file_model.py* και *user_model.py*. Το subpackage *compile* περιλαμβάνει κώδικα που διαχειρίζεται τη μεταγλώττιση των αρχείων και περιλαμβάνει τα modules *compile.py*, *compile_c.py*, *compile_cpp.py* και *compile_golang.py*. Τόσο το subpackage *models* όσο και

το *compile* περιγράφονται αναλυτικότερα στις ενότητες που ακολουθούν. Έτσι, προκύπτει ένα package με την δομή που εικονίζεται στο Σχήμα 4.1.



Σχήμα 4.1: Δομή του rest_server package

Το παραπάνω package γίνεται import στο module wsgi.py, στο οποίο τρέχει το Flask app και δέχεται requests από όλα το interfaces (IP 0.0.0.0) στο port 8080. Επιπλέον, το αρχείο uwsgi.ini ορίζει το configuration με το οποίο τρέχει ο uWSGI production ready server.

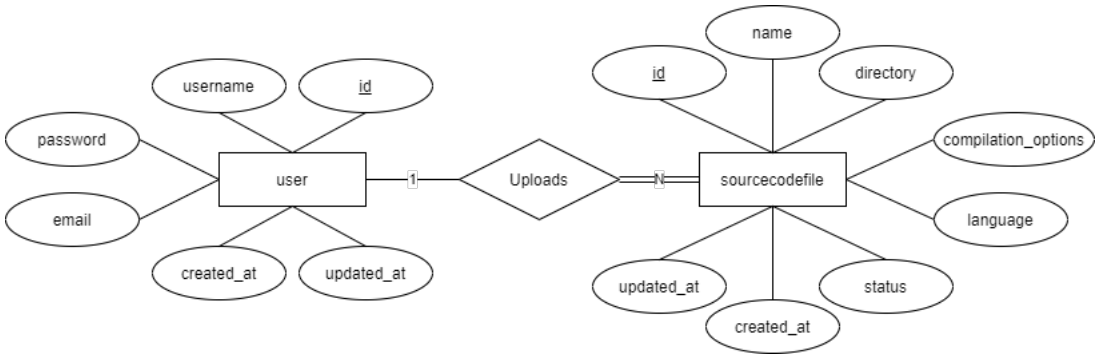
Ακόμη, στο ίδιο επίπεδο με το *rest_server* package, υπάρχει ένα python package το οποίο ονομάζεται *tests* και περιλαμβάνει τα Unit Tests για το πακέτο *rest_server*. Για το σκοπό αυτό, το package *tests* μιμείται τη δομή του *rest_server* package, ενώ, επιπλέον, περιλαμβάνει κάποια αρχεία με απλό κώδικα σε C, τα οποία χρησιμοποιούνται και μεταγλωττίζονται για να ελεγχθούν οι συναρτήσεις που είναι υπεύθυνες για τη μεταγλώττιση. Τα Unit Tests μπορούν να εκτελεστούν χρησιμοποιώντας το run_backend_tests.py script.

Το τελευταίο αρχείο που αφορά το Backend είναι το database_accepts_connections.py, το οποίο χρησιμοποιείται για να διαπιστωθεί πως η βάση δεδομένων ακούει και είναι έτοιμη να δεχθεί αιτήματα, πριν ο WSGI REST server συνδεθεί σε αυτήν και αρχίσει να επικοινωνεί με αυτήν.

4.2.2 Σχήμα Βάσης Δεδομένων

Ως βάση δεδομένων στο παρόν σύστημα επιλέγεται η PostgreSQL. Η PostgreSQL, όπως έχει ήδη αναφερθεί, είναι μία SQL βάση δεδομένων, και συγκεκριμένα Object-Relational βάση δεδομένων. Επειδή επιτρέπει την ανάπτυξη Σχεσιακών Βάσεων Δεδομένων, επιλέγεται σε σχέση με άλλες βάσεις δεδομένων οι οποίες δεν είναι σχεσιακές. Επιπλέον, είναι πολύ αποδοτική καθώς μπορεί να διαχειριστεί πολλά queries ταυτόχρονα, ενώ υποστηρίζει τύπους δεδομένων όπως λίστες και JSON, τους οποίους άλλες SQL βάσεις δεδομένων δεν υποστηρίζουν.

Στη συγκεκριμένη εφαρμογή, αφενός αποθηκεύονται πληροφορίες οι οποίες αφορούν τους χρήστες και επιτρέπουν τη σύνδεσή τους σε αυτή και αφετέρου αποθηκεύονται πληροφορίες σχετικές με τα ανεβασμένα αρχεία ώστε να είναι δυνατή η ανάκτησή τους. Επομένως, χρειάζονται δύο πίνακες στη βάση δεδομένων. Πιο συγκεκριμένα, ο πρώτος πίνακας είναι ο ”users” πίνακας και ο δεύτερος είναι ο ”sourcecodefiles” πίνακας. Το διάγραμμα Οντοτήτων-Συσχετίσεων (Entity-Relational Diagram) καθώς και το Σχεσιακό Μοντέλο της Βάσης Δεδομένων εικονίζονται στα Σχήματα 4.2 και 4.3, αντίστοιχα. Τα πεδία των πινάκων ”users” και ”sourcecodefiles” ορίζονται στα modules *user_model.py* και *file_model.py*, μέσα στις κλάσεις ”User” και ”SourceCodeFile”, αντίστοιχα. Στα αρχεία αυτά ορίζονται και κλάσεις, οι οποίες βοηθούν τόσο στο serialization όσο και στο deserialization των αντίστοιχων δομών δεδομένων που επιστρέφονται από τη βάση δεδομένων. Δηλαδή, ένα dictionary μπορεί να γίνει deserialized σε User object και αντίστροφα, ένα User object μπορεί να γίνει serialized σε dictionary ή σε JSON string. Με αυτόν τον τρόπο, είναι δυνατό να ελεγχθεί αν, για παράδειγμα, ένα dictionary περιλαμβάνει όλα τα απαραίτητα πεδία για να οριστεί ένα User object ή ένα SourceCodeFile object, καθώς η μετρατροπή επιστρέφει σφάλμα αν τα πεδία δεν είναι σωστά.

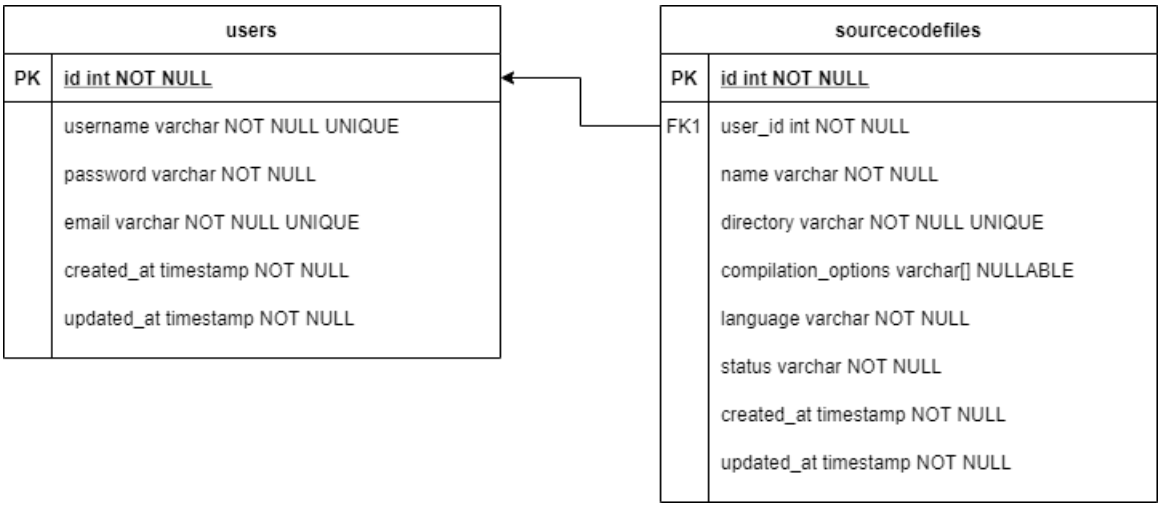


Σχήμα 4.2: Διάγραμμα Οντοτήτων-Συσχετίσεων Βάσης Δεδομένων

4.2.3 Διαχείριση μεταγλώττισης - Compilation Handlers

Το subpackage *compile* είναι υπεύθυνο για τη μεταγλώττιση των αρχείων. Συγκεκριμένα, ορίζεται μία γενική κλάση CompilationHandler, δηλαδή σχεδιάζεται ένας Χειριστής Μεταγλώττισης, ο οποίος δε σχετίζεται άμεσα με μία γλώσσα αλλά αποτελεί το πρότυπο το οποίο θα κληρονομήσουν οι Compilation Handlers για κάθε μία γλώσσα. Δηλαδή, ορίζει τα γνωρίσματα που χρειάζονται σε όλους τους Compilation Handlers ανεξαρτήτως της γλώσσας που μεταγλωττίζουν καθώς και τις μεθόδους οι οποίες είναι ανεξάρτητες της γλώσσας. Για κάθε μία γλώσσα του συστήματος, ορίζεται μία νέα κλάση με όνομα <language>CompilationHandler, η οποία κληρονομεί την κλάση CompilationHandler.

Αναλυτικότερα, στο module *compile.py*, ορίζεται η κλάση CompilationHandler η οποία ορίζει τα γνωρίσματα ”language” και ”root_upload_path”. Το ”root_upload_path” είναι language specific και προορίζεται ώστε να περιέχει το σταθερό μέρος της διαδρομής στην οποία θα αποθηκεύεται ένα ανεβασμένο αρχείο.



Σχήμα 4.3: Σχεσιακό Μοντέλο Βάσης Δεδομένων

Σχετικά με τις μεθόδους της κλάσης, δηλώνονται τρεις αφηρημένες μέθοδοι, οι οποίες ονομάζονται *compilation_options_parser*, *compilation_command_generator* και *results_zip_appender*. Για κάθε μία γλώσσα του συστήματος, ο ειδικός CompilationHandler για τη συγκεκριμένη γλώσσα υλοποιεί εκ νέου τις αφηρημένες αυτές μεθόδους. Η μέθοδος *compilation_options_parser* είναι υπεύθυνη ώστε να ελέγξει τις επιλογές μεταγλώττισης που λαμβάνονται (και πιθανώς να απορρίψει κάποιες από αυτές) αλλά και να τις μετατρέψει σε μορφή που να μπορεί να τις αναγνωρίσει ο compiler του συστήματος. Η μέθοδος *compilation_command_generator* δημιουργεί την εντολή μεταγλώττισης, η οποία θα εκτελεστεί στο σύστημα, συνδυάζοντας κατάλληλα τον μεταγλωττιστή, τις επιλογές μεταγλώττισης που έχουν δημιουργηθεί νωρίτερα από την *compilation_options_parser* καθώς και τη διαδρομή στην οποία θα πραγματοποιηθεί η μεταγλώττιση και στην οποία θα δημιουργηθούν τα αποτελέσματά της. Τέλος, η συνάρτηση *results_zip_appender*, δεδομένης μίας διαδρομής στην οποία βρίσκονται τα αποτελέσματα της μεταγλώττισης, επεκτείνει ένα .zip αρχείο (το οποίο περιέχει το standard output και το standard error που ενδεχομένως δημιούργησε η εντολή μεταγλώττισης), με ένα .wasm αρχείο, ένα .js αρχείο και ένα .html αρχείο. Το .wasm αρχείο περιέχει το μεταγλωττισμένο κώδικα σε WebAssembly ενώ τα αρχεία .js και .html περιέχουν βοηθητικό κώδικα ώστε να μπορεί να εκτελεστεί εύκολα η WebAssembly από τον τελικό χρήστη.

Επιπλέον, υπάρχουν και μέθοδοι οι οποίες είναι ήδη ορισμένες και είναι κοινές σε όλους τους Compilation Handlers. Συγκεκριμένα, υπάρχουν δύο μέθοδοι, οι οποίες αφορούν τη δημιουργία της διαδρομής στην οποία θα μεταγλωττιστεί ένα αρχείο και ονομάζονται *_generate_file_subpath* και *_format_full_file_path*. Η μέθοδος *_generate_file_subpath* υπολογίζει το SHA256 του ανεβασμένου αρχείο και κατασκευάζει το όνομα ενός φακέλου, συνδυάζοντας τη χρονική στιγμή του υπολογισμού και το SHA256, στην ακόλουθη μορφή: %Y%m%d%H%M%S%f_<SHA256>, όπου το πρώτο μέρος είναι χρονική στιγμή με σύμβολα της datetime βιβλιοθήκης της Python και αντιστοιχεί σε ΈτοςΜήναςΗμέραΏραΛεπτόΔευτερόλεπτοΜικροδευτερόλεπτο. Για παράδειγμα, το όνομα του φακέλου που προκύπτει για τον ακόλουθο C κώδικα (Εικόνα 4.5), στις 13/05/2020 την ώρα 18:21:38.912648 είναι:
20200513182138912648_dcf07990b1dc8dffcf40adfa88c97ca00fe73fe777f555db17f2423045352e17

Με αυτόν τον τρόπο, για να υπάρξει σύγκρουση μεταξύ των ονομάτων δύο φακέλων πρέπει να ανέβουν ταυτόχρονα ακριβώς τα ίδια αρχεία στον server, γεγονός το οποίο είναι πολύ δύσκολο να συμβεί. Το όνομα που προκύπτει από αυτή τη μέθοδο αποτελεί στο μεταβλητό τμήμα των διαδρομών αφού αλλάζει ανάλογα με το περιεχόμενο του αρχείου και τη χρονική στιγμή κατά την οποία ανέβηκε στο σύστημα. Αυτό το subpath είναι πολύ σημαντικό για την κατασκευή της πλήρους διαδρομής.

Έτσι, δεδομένων ενός μοναδικού αναγνωριστικού χρήστη και ενός subpath, η δεύτερη μέθο-

```python
class CompilationHandler(metaclass=abc.ABCMeta):
    def __init__(self, language, root_upload_path):
        self.language = language
        self.root_upload_path = root_upload_path

    @abc.abstractmethod
    def compilation_options_parser(self, *args, **kwargs):
        pass

    @abc.abstractmethod
    def compilation_command_generator(self, *args, **kwargs):
        pass

    @abc.abstractmethod
    def results_zip_appender(self, *args, **kwargs):
        pass
```

Σχήμα 4.4: CompilationHandler Class Attributes and Abstract Methods

```c
#include <stdio.h>
int main(int argc, char ** argv) {
  printf("Hello world!\n");
}
```

Σχήμα 4.5: Hello World σε C

δος *_format_full_file_path* αναλαμβάνει να ανακατασκευάσει την πλήρη διαδρομή (full path). Το path αυτό, αποτελείται από πολλά επίπεδα φακέλων. Πιο αναλυτικά, σε πρώτο επίπεδο, ανάλογα τη γλώσσα, επιλέγει το σταθερό μέρος της διαδρομής, δηλαδή το "root_upload_path". Σε δεύτερο επίπεδο, επιλέγει τον χρήστη και επεκτείνει τη διαδρομή με το μοναδικό αναγνωριστικό του. Σε τρίτο επίπεδο, προχωρά σε ένα υποσύνολο των αρχείων του χρήστη, επεκτείνοντας της διαδρομή με τους τρεις τελευταίους χαρακτήρες του SHA256 του αρχείου (δηλαδή με τους τρεις τελευταίους χαρακτήρες του subpath). Σε τέταρτο επίπεδο, επεκτείνει το path με το subpath και με αυτόν τον τρόπο επιλέγει τον φάκελο του αρχείου.

Η παραπάνω δομή του συστήματος αρχείων, με τη χρήση τεσσάρων επιπέδων, ταξινομεί τα αρχεία με τέτοιο τρόπο ώστε αφενός η αναζήτηση ενός αρχείου να είναι γρήγορη και να μην δυσχεραίνεται όσο αυξάνεται το πλήθος των αρχείων και αφετέρου να είναι γρήγορη η ανακάλυψη του πλήρους path, δεδομένων του subpath και του αναγνωριστικού του χρήστη. Για κάθε μεταγλωττιστή, υπάρχει ένα διαφορετικό "root_upload_path", αλλά για τις C και C++ υπάρχει ένα κοινό "root_upload_path" αφού χρησιμοποιούν και οι δύο το emscripten. Όταν δεν είναι γνωστό το μοναδικό ανανγωριστικό του χρήστη, τότε στη θέση του χρησιμοποιείται ένας φάκελος που ονομάζεται "unknown". Επομένως, η δομή του συστήματος αρχείων που δημιουργείται εικονίζεται στο Σχήμα 4.6. Ο owner όλων αυτών των φακέλων είναι ο χρήστης που τρέχει τον container, δηλαδή ο χρήστης με UID 1000.

Η τελευταία κοινή μέθοδος σε όλους τους Compilation Handlers είναι η *compile*. Η μέθοδος αυτή δέχεται ως όρισμα ένα αρχείο προς μεταγλώττιση, ένα JSON string με επιλογές μεταγλώττισης καθώς και ένα boolean όρισμα το οποίο δηλώνει αν θα αποθηκευτούν πληροφορίες του αρχείου στη βάση δεδομένων για να είναι δυνατή η ανάκτησή του. Η παρούσα μέθοδος χρησιμοποιεί τις προαναφερθείσες μεθόδους, *_generate_file_subpath* και *_format_full_file_path*, ώστε να παράξει το μοναδικό subpath για το αρχείο καθώς και το πλήρες path στο οποίο θα πραγματοποιηθεί η μεταγλώττιση. Στη συνέχεια, χρησιμοποιεί την αφηρημένη μέθοδο *compilation_options_parser* για

Σχήμα 4.6: Δομή Συστήματος Αρχείων στο φάκελο των αποτελεσμάτων

να ελέγξει και να διαμορφώσει τις επιλογές μεταγλώττισης, τις οποίες, έπειτα, τροφοδοτεί στην αφηρημένη μέθοδο *compilation_command_generator* για να διαμορφώσει την πλήρη εντολή μεταγλώττισης. Ακολουθεί η δημιουργία του path στο Σύστημα Αρχείων και η αποθήκευση του ανεβασμένου αρχείου σε αυτό το σημείο ώστε να μπορεί να εντοπιστεί από τον μεταγλωττιστή. Ύστερα, εκτελείται η εντολή μεταγλώττισης η οποία παράγει τα αποτελέσματά της στο συγκεκριμένο path. Στο ίδιο σημείο του Συστήματος Αρχείων δημιουργείται ένα αρχείο ZIP, το οποίο περιέχει, όπως ήδη αναφέρθηκε, το standard output και το standard error τα οποία ενδεχομένως δημιούργησε η εντολή μεταγλώττισης. Σε περίπτωση επιτυχούς μεταγλώττισης, το αρχείο ZIP επεκτείνεται χρησιμοποιώντας την αφηρημένη μέθοδο *results_zip_appender*. Επιπλέον, αν χρειάζεται να αποθηκευθούν πληροφορίες σχετικά με το αρχείο και τη μεταγλώττισή του στη βάση δεδομένων, τότε δημιουργείται ένα dictionary με τις απαραίτητες πληροφορίες, δηλαδή το μοναδικό αναγνωριστικό του χρήστη, το όνομα του αρχείου, το μοναδικό subpath του αρχείου, τις επιλογές μεταγλώττισης, τη γλώσσα του κώδικα καθώς και την κατάσταση της μεταγλώττισης και χρησιμοποιώντας μεθόδους από το *file_model.py* module, αποθηκεύονται οι πληροφορίες στη βάση δεδομένων. Η μέθοδος *compile*, ουσιαστικά, λειτουργεί ως ένα pipeline, το οποίο οργανώνει και χρησιμοποιεί τις υπόλοιπες μεθόδους. Κάθε εξειδικευμένος Compilation Handler χρησιμοποιεί τις δικές του, ειδι-

κές, μεθόδους στη θέση των αφηρημένων μεθόδων, δημιουργώντας το επιθυμητό αποτέλεσμα. Κατά την εισαγωγή του *compile* subpackage δημιουργείται ένα dictionary το οποίο ως κλειδιά έχει τις υποστηριζόμενες γλώσσες και ως τιμές έχει στιγμιότυπα των Compilation Handlers. Σε κάθε εισερχόμενο request ανάλογα με τη γλώσσα, επιλέγεται ο κατάλληλος Compilation Handler και εκτελεί τη δική του *compile* μέθοδο με κατάλληλες παραμέτρους.

4.3 Λογισμικό Frontend

Το Frontend είναι υλοποιημένο σε Angular 8. Ως Javascript framework, διαθέτει πολλές βιβλιοθήκες οι οποίες είναι εύκολα διαχειρίσιμες χρησιμοποιώντας το NPM. Το frontend είναι δομημένο σε components, δηλαδή σύνολα τα οποία περιέχουν ένα html, ένα css και ένα typescript αρχείο το καθένα. Κάθε λειτουργία η οποία παρέχεται από το frontend, έχει τουλάχιστον ένα component αφιερωμένο σε αυτήν. Για παράδειγμα, τόσο η log in όσο και η sign up σελίδες έχουν ένα αντίστοιχο component που τις αφορά.

Ορισμένες σελίδες, ωστόσο, διαμορφώνονται από περισσότερα του ενός components. Πιο συγκεκριμένα, η σελίδα, η οποία επιτρέπει το ανέβασμα ενός αρχείου προς μεταγλώττιση, αποτελείται από ένα βασικό component και τρία βοηθητικά components. Το βασικό component ονομάζεται *file-upload* και παρέχει τις γενικές δυνατότητες όπως την επιλογή γλώσσας, την αποθήκευση των αποτελεσμάτων στη βάση δεδομένων και το κατέβασμα των αποτελεσμάτων της μεταγλώττισης. Τα βοηθητικά components της σελίδας αυτής αφορούν τις επιλογές μεταγλώττισης για κάθε μία γλώσσα. Ανάλογα με τη γλώσσα που επιλέγεται στο βασικό component, εμφανίζεται το αντίστοιχο βοηθητικό component, το οποίο διαθέτει τις εξειδικευμένες επιλογές μεταγλώττισης για τον επιλεγμένο μεταγλωττιστή. Συγκεκριμένα, εμφανίζεται ένα από τα components *c-upload-form*, *cpp-upload-form* και *golang-upload-form*.

Αντίστοιχα, η σελίδα με τα αρχεία ενός συγκεκριμένου χρήστη αποτελείται από δύο ξεχωριστά components. Το πρώτο component ονομάζεται *personal-files* και παρέχει στο χρήστη μία συνολική εικόνα για τα αρχεία τα οποία έχει ανεβάσει στον server. Το δεύτερο component ονομάζεται *personal-file-details* και παρέχει λεπτομέρειες σχετικά με τη μεταγλώττιση και την κατάσταση ενός συγκεκριμένου αρχείου που ο χρήστης επιλέγει. Επιπλέον, παρέχει στον χρήστη τη δυνατότητα να κατεβάσει εκ νέου τα αποτελέσματα της μεταγλώττισης του αρχείου αυτού, να δει τον κώδικα που περιέχει ή ακόμα και να το διαγράψει.

Ακόμα, ο κώδικας του frontend περιέχει services, δηλαδή αρχεία τα οποία περιέχουν κώδικα σε Typescript και ορίζουν συναρτήσεις οι οποίες χρησιμοποιούνται από τα components (και γενικότερα από τον υπόλοιπο κώδικα) ως διεπαφή για να εκτελέσουν σύνθετες λειτουργίες. Συγκεκριμένα, για την επικοινωνία με το backend, τα components δε χρειάζεται να γνωρίζουν τον τρόπο επικοινωνίας και την προέλευση των πληροφοριών που παρουσιάζουν. Γι' αυτό το λόγο, δημιουργούνται κατάλληλα services ώστε να κρύψουν από τα components τις τεχνικές λεπτομέρειες της επικοινωνίας με το backend. Τα services δημιουργούν απλές συναρτήσεις οι οποίες αναλαμβάνουν να δημιουργήσουν ένα request, να το διαμορφώσουν με τα απαραίτητα query parameters, να κατασκευάσουν το request body και να το στείλουν στο backend.

Επιπλέον, κλάσεις που ονομάζονται Interceptors παρεμβάλλονται μεταξύ του frontend και του backend ώστε να τροποποιήσουν τόσο το request κατά την αποστολή του στο backend όσο και το response κατά τη λήψη του από το frontend. Έτσι, αναλαμβάνουν να προσθέσουν τους απαραίτητους request headers πριν το request φτάσει στο backend αλλά και να χειριστούν μηνύματα σφάλματος που επιτρέφονται από το backend. Τα components χρησιμοποιούν τις συναρτήσεις οι οποίες παρέχονται από τα services και είναι υπεύθυνα να χειριστούν ασύγχρονα την επιτυχή απάντηση που λαμβάνουν.

Επειδή όλο το Angular frontend μετατρέπεται σε ένα σύνολο από html και javascript τα οποία αποστέλλονται στον χρήστη, δεν μπορούν να αναφέρονται σε μεταβλητές περιβάλλοντος που βρίσκονται στον container, όπως για παράδειγμα το API URI του backend. Για να αντιμετωπιστεί αυτό το πρόβλημα, υπάρχει ένα javascript αρχείο το οποίο ονομάζεται env.template.js και περιέ-

χει placeholders με γενικές μεταβλητές που χρειάζονται για να λειτουργήσει σωστά η εφαρμογή. Κατά το deployment του συστήματος, όλες οι πληροφορίες που είναι απαραίτητες (όπως για παράδειγμα η IP του συστήματος και κατ' επέκταση το API URI του backend) αντικαθιστούν τα placeholders και δημιουργούν ένα env.js αρχείο. Το αρχείο αυτό αποστέλλεται στο χρήστη (μαζί με όλα τα υπόλοιπα αρχεία που χρειάζονται) και οι χρήσιμες μεταβλητές που περιέχει αποθηκεύονται στο τρέχον παράθυρό του ώστε να μπορεί να έχει πρόσβαση σε αυτές όλος ο υπόλοιπος javascript κώδικας.

Όταν το σύστημα είναι deployed, το frontend έχει ελεγχθεί πως παρουσιάζεται και λειτουργεί ορθά τουλάχιστον στις ακόλουθες εκδόσεις των φυλλομετρητών:

Browser	Version
Chrome	81.0.4044.138 (64-bit)
Firefox	78.0.1 (64-bit)
Microsoft Edge	84.0.522.40 (64-bit)

Πίνακας 4.2: Υποστηριζόμενες Εκδόσεις Φυλλομετρητών

ενώ στις παρακάτω εκδόσεις δεν υποστηρίζονται όλα τα Angular features:

Browser	Version
Chrome	57.0.2987.110 (64-bit)

Πίνακας 4.3: Μη Υποστηριζόμενες Εκδόσεις Φυλλομετρητών

4.4 Εγκατάσταση και εκτέλεση

Για να είναι δυνατή η εγκατάσταση του συστήματος και η διαχείρισή του είναι απαραίτητο στο host περιβάλλον να υπάρχει μία έκδοση του Docker καθώς και μία release του Docker Compose που να υποστηρίζει αρχεία compose version 3. Εγκαθιστώντας τις νεότερες εκδόσεις εξασφαλίζονται οι παραπάνω απαιτήσεις καθώς οι εκδόσεις είναι backwards compatible.

Στη συνέχεια, χρειάζεται να γίνει clone το repository με τον κώδικα, τροποποιώντας το όνομα του φακέλου σε *WebassemblyCompilerCollection*. Μέσα σε αυτόν το φάκελο υπάρχει ένα αρχείο με όνομα *".env"*. Οι πρώτες οκτώ γραμμές περιέχουν μεταβλητές με dummy τιμές, οι οποίες θα πρέπει να αλλάξουν και να συμπληρωθούν με νέες τιμές, όπως φαίνεται και στην Εικόνα 4.7. Οι υπόλοιπες γραμμές δε χρειάζεται να τροποποιηθούν.

```
1  # Fill the following variables
2  SYSTEM_IP=localhost
3  FRONTEND_HOST_PORT=3535
4  BACKEND_HOST_PORT=8080
5  HOST_RESULTS_DIR_PREFIX=/home/nikos
6  POSTGRES_PASSWORD=example
7  POSTGRES_USER=database
8  JWT_SECRET_KEY=jwt_secret_key
9
10 # DO NOT CHANGE ANYTHING BELOW THIS POINT
11 # Development
12 ...
```

Σχήμα 4.7: Placeholder τιμές στο .env αρχείο

Συγκεκριμένα, η μεταβλητή SYSTEM_IP πρέπει να συμπληρωθεί με την Public Floating IP του συστήματος. Οι FRONTEND_HOST_PORT και BACKEND_HOST_PORT δεν είναι απαραίτητο να τροποποιηθούν αλλά είναι δυνατό να αντικατασταθούν με δύο διαθέσιμα ports του συστήματος αν τα 3535 και 8080 δεν είναι διαθέσιμα ή αν κρίνεται πως πρέπει να χρησιμοποιηθούν διαφορετικά ports. Η HOST_RESULTS_DIR_PREFIX πρέπει να συμπληρωθεί με ένα path το οποίο υπάρχει ήδη και μέσα στο οποίο θα τοποθετηθούν τα volumes των containers (οπότε είναι καλό το path αυτό να οδηγεί σε έναν υπάρχοντα και κενό φάκελο). Η POSTGRES_PASSWORD πρέπει να συμπληρωθεί, για λόγους ασφαλείας, με ένα διαφορετικό κωδικό με το επίπεδο ασφάλειας του κωδικού να επαφίεται στην κρίση του χρήστη. Η POSTGRES_USER δεν είναι απαραίτητο να τροποποιηθεί αλλά μπορεί να τροποποιηθεί αν κρίνεται απαραίτητο πριν το πρώτο deployment. Τέλος, η JWT_SECRET_KEY πρέπει να τροποποιηθεί, για λόγους ασφαλείας, με το επίπεδο ασφαλείας του να επαφίεται και αυτό στην κρίση του χρήστη.

Στη συνέχεια το bash script deploy.sh παρέχει κάποιες απλές δυνατότητες δημιουργίας και εγκατάστασης του συστήματος. Συγκεκριμένα, τρέχοντας ”sudo bash deploy.sh –build” χτίζονται όλα τα απαραίτητα Images. Δεδομένου πως όλα τα παραπάνω έχουν γίνει με επιτυχία, αρκεί να εκτελεστεί ”sudo bash deploy.sh –deploy” ώστε να κάνει deploy το σύστημα. Η παραπάνω εντολή δημιουργεί τα volumes αν δεν υπάρχουν και ζητάει άδεια για να ρυθμίσει τα permissions των φακέλων έτσι ώστε κάποια από αυτά να μπορεί να τα χειριστεί ο χρήστης του rest_server container, δηλαδή ο χρήστης με UID 1000, και κάποια άλλα να μπορεί να τα χειριστεί ο χρήστης του ucrm_db (User Compilation Results Management Database) container, δηλαδή ο χρήστης με UID 999. Για να μπορέσει να πραγματοποιηθεί η αλλαγή στα permissions, χρειάζεται η εντολή να τρέξει με sudo. Έπειτα, δημιουργείται ένας NGINX server και frontend container, τρεις wsgi REST server containers και ένας PostgreSQL server container. Είναι δυνατό να γίνουν deploy περισσότεροι ή λιγότεροι containers κάνοντας χρήση του docker-compose API στη θέση του deploy.sh script.

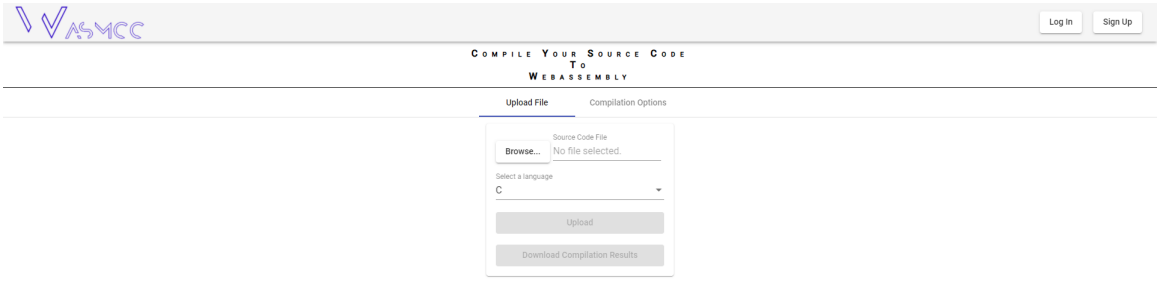
Σε αυτό το σημείο, η εφαρμογή είναι διαθέσιμη στον υπόλοιπο κόσμο. Αν χρειαστεί να σταματήσει η εφαρμογή, τότε μπορεί να χρησιμοποιηθεί το API του docker-compose ή να εκτελεστεί η εντολή ”sudo bash deploy.sh –deploy-down”.

Ο πλήρης κώδικας, τόσο για το frontend όσο και για το backend, βρίσκεται στο repository: https://github.com/NikosGad/Webassembly_Compiler_Collection.

Κεφάλαιο 5

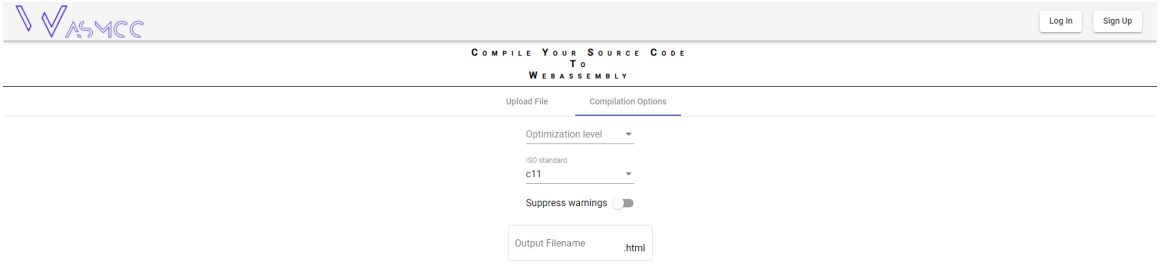
Παράδειγμα Λειτουργία της Υπηρεσίας Μεταγλώττισης

Όταν το σύστημα εκτελείται με επιτυχία, τότε, αυτομάτως, είναι διαθέσιμο για χρήση το frontend της υπηρεσίας. Οποιοσδήποτε χρήστης (ανεξαρτήτως της εγγραφής του στο σύστημα) μπορεί να επιλέξει το αρχείο προς μεταγλώττιση και τη γλώσσα που επιθυμεί να μεταγλωττίσει (Εικόνα 5.1).



Σχήμα 5.1: Αρχική Σελίδα - Σελίδα Μεταγλώττισης

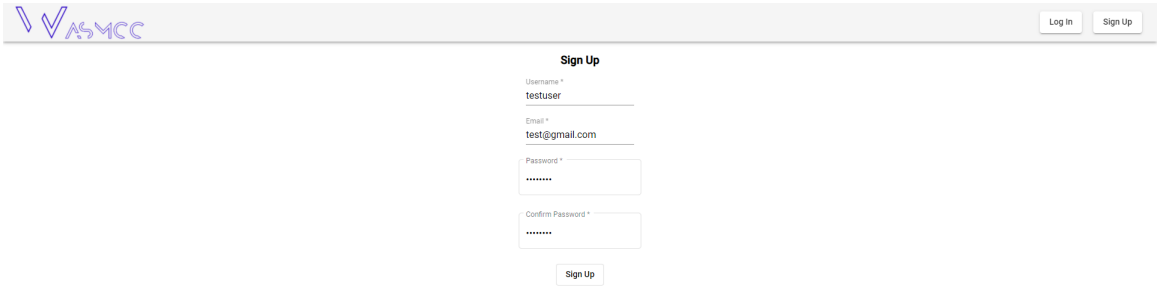
Μάλιστα, ανάλογα με την επιλεγμένη γλώσσα είναι διαθέσιμες και οι εξειδικευμένες επιλογές μεταγλώττισης (Εικόνα 5.2).



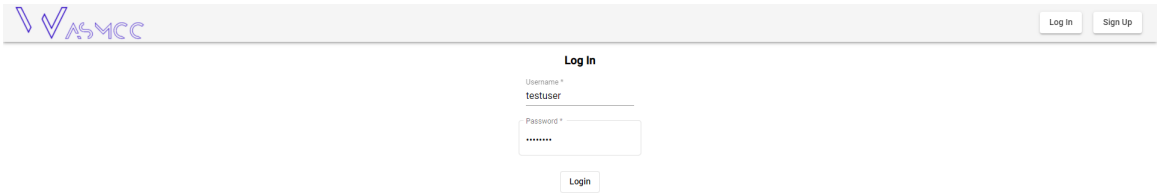
Σχήμα 5.2: Σελίδα Επιλογών Μεταγλώττισης

Αξίζει να μελετηθεί το σενάριο κατά το οποίο ένας νέος χρήστης εγγράφεται, συνδέεται και μεταγλωττίζει ένα αρχείο και στη συνέχεια ελέγχει το αρχείο από την προσωπική σελίδα με όλα τα αρχεία του. Πιο αναλυτικά, ένας μη εγγεγραμμένος χρήστης, ο οποίος επιθυμεί να εγγραφεί, επισκέπτεται τη ”Sign Up” σελίδα και συμπληρώνει τα πεδία (Εικόνα 5.3). Αφού πατήσει το κουμπί εγγραφής, ένα μήνυμα εμφανίζεται και τον ειδοποιεί πως η εγγραφή του ήταν επιτυχής. Στη συνέχεια, οδηγείται στην ”Log In” σελίδα, στην οποία, ομοίως, συμπληρώνει τα πεδία με τα στοιχεία του και εισέρχεται πατώντας το κουμπί σύνδεσης (Εικόνα 5.4) σύνδεσης. Αυτομάτως, ο χρήστης οδηγείται στην αρχική σελίδα, στην οποία πλέον βρίσκεται διαθέσιμη και η επιλογή αποθήκευσης στη βάση δεδομένων (Εικόνα 5.5).

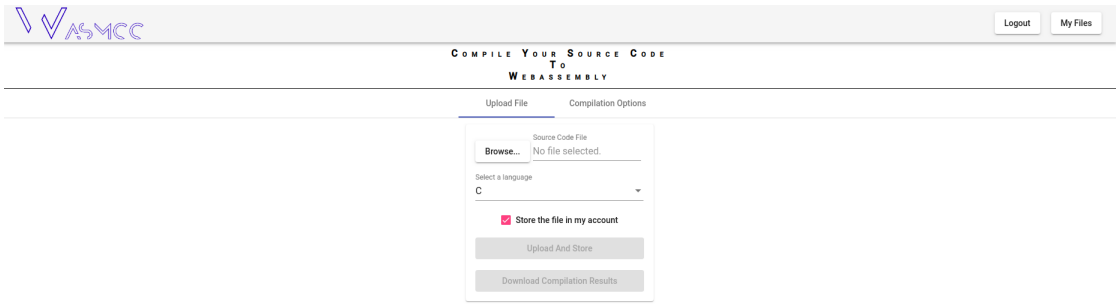
Ο χρήστης επιλέγει ένα αρχείο με κώδικα C++ (”palindrome.cpp”, Εικόνα 5.6), όπως φαίνεται στην Εικόνα 5.7. Μόλις επιλεχθεί το αρχείο, ενεργοποιείται και το κουμπί ”Upload And Store” ώστε να ξεκινήσει η μεταγλώττιση του αρχείου (το οποίο θα αποθηκευθεί στο σύστημα



Σχήμα 5.3: Συμπληρωμένη Σελίδα Εγγραφής



Σχήμα 5.4: Συμπληρωμένη Σελίδα Σύνδεσης



Σχήμα 5.5: Αρχική Σελίδα (Εγγεγραμμένος Χρήστης)

ώστε να μπορεί να ανακτηθεί). Επιπλέον, στην καρτέλα ”Compilation Options”, ορίζει πρόσθετες παραμέτρους μεταγλώττισης του C++ αρχείου, όπως φαίνεται στην Εικόνα 5.8. Έπειτα, ανεβάζει το αρχείο και ξεκινά τη μεταγλώττιση πατώντας το ενεργοποιημένο κουμπί ”Upload And Store”. Όταν η μεταγλώττιση ολοκληρωθεί τότε και μόνο τότε ενεργοποιείται το κουμπί ”Download Compilation Results” ώστε να γίνει η λήψη του ZIP αρχείου με τα αποτελέσματα (Εικόνα 5.9). Με αυτόν τον τρόπο, ο χρήστης κατεβάζει το αρχείο ”results.zip”, το οποίο περιέχει τα τρία αρχεία, palindrome.html, palindrome.js και palindrome.wasm (Εικόνα 5.10).

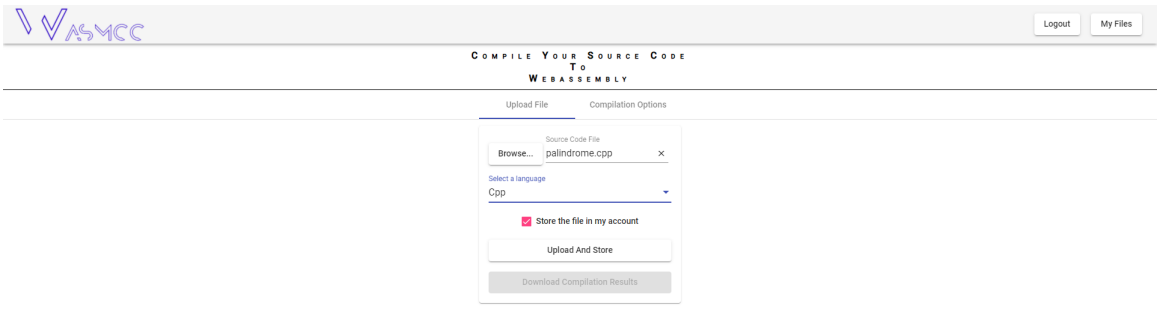
Έστω ότι ο χρήστης ανεβάζει και μεταγλωττίζει, με παρόμοιο τρόπο, ένα Go αρχείο με όνομα ”hello_world.go” και δεν ανεβάζει κανένα C αρχείο. Σε περίπτωση που θελήσει να δει τα αρχεία που έχει ήδη ανεβάσει, επιλέγει το ”My Files” (Εικόνα 5.11). Επιπλέον, έχει τη δυνατότητα να οργανώσει τα αρχεία του με βάση τη γλώσσα, όπως φαίνεται στις Εικόνες 5.12, 5.13, 5.14, 5.15. Σε αυτό το σημείο, ο χρήστης έχει τη δυνατότητα να επιλέξει ένα συγκεκριμένο αρχείο και να δει πληροφορίες γι’ αυτό. Για παράδειγμα, ο χρήστης επιλέγει το αρχείο ”palindrome.cpp” που ανέβασε (Εικόνα 5.16). Εκεί, μπορεί να δει και να θυμηθεί πληροφορίες όπως την κατάσταση της μεταγλώττισης (Επιτυχής ή Εσφαλμένη) και τις επιλογές μεταγλώττισης που χρησιμοποίησε. Ταυτόχρονα, μπορεί να κατεβάσει εκ νέου το results.zip αρχείο με τα αποτελέσματα της μεταγλώττισης, να διαγράψει το αρχείο αν το επιθυμεί (Εικόνα 5.17) αλλά και να δει τον κώδικα που περιέχει το αρχείο που ανέβασε (Εικόνα 5.18). Έπειτα, κάνοντας ”Logout”, ο χρήστης οδηγείται στην ”Log In” σελίδα, ενώ, πατώντας πάνω στο ”Wasmcc” Logo, επιστρέφει στην αρχική σελίδα.

Αξίζει να σημειωθεί πως για να μπορέσει να τρέξει η HTML μαζί με τη Javascript και τη

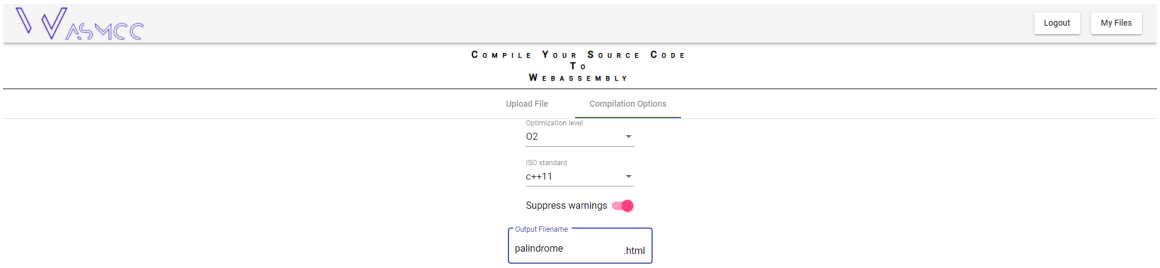
```cpp
#include <iostream>

int main(int argc, char **argv) {
    std::string word;
    int word_length;

    std::getline(std::cin, word);
    word_length = word.length();

    int DP[2][word_length];
    int *current_col_ptr;
    int *next_col_ptr;

    current_col_ptr = DP[0];
    next_col_ptr = DP[1];

    current_col_ptr[0] = 1;

    for (int j = 0; j < word_length - 1; j++) {
        current_col_ptr[j+1] = 0;
        next_col_ptr[j+1] = 1;
        for (int i = j; i >= 0; i--) {
            if (word[j+1] == word[i]) {
                next_col_ptr[i] = (next_col_ptr[i+1]
                                   + current_col_ptr[i]
                                   + 1)
                                   % 20130401;
            }
            else {
                next_col_ptr[i] = (next_col_ptr[i+1]
                                   + current_col_ptr[i]
                                   - current_col_ptr[i+1])
                                   % 20130401;
            }
        }

        int *tmp_ptr = current_col_ptr;
        current_col_ptr = next_col_ptr;
        next_col_ptr = tmp_ptr;
    }

    std::cout << "Palindromes: " << current_col_ptr[0] << std::endl;

    return 0;
}
```

Σχήμα 5.6: Περιεχόμενα αρχείου palindrome.cpp

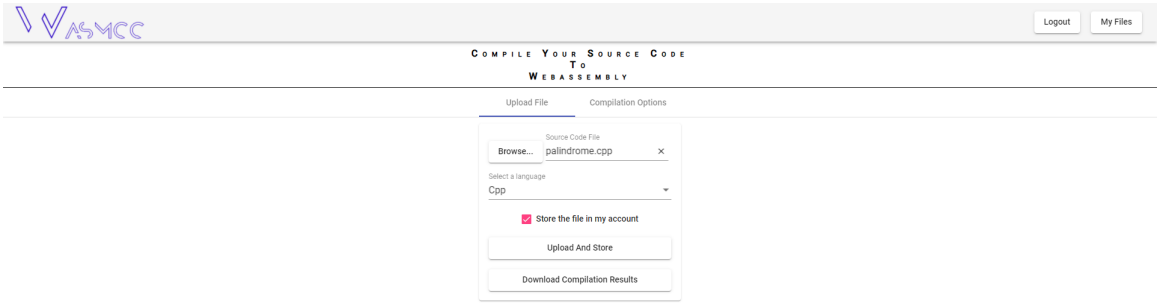
WebAssembly, δεν αρκεί να ανοίξει το HTML αρχείο ως "file://" στον browser αλλά χρειάζεται να τρέξει ένας local web server, ο οποίος βλέπει ολοκληρο το φάκελο με τα τρία αρχεία.



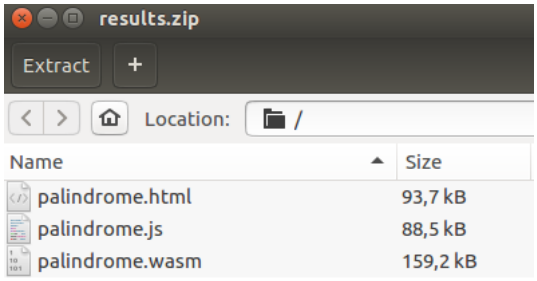
Σχήμα 5.7: Επιλογή Αρχείου προς Μεταγλώττιση (Εγγεγραμμένος Χρήστης)



Σχήμα 5.8: Συμπληρωμένες Επιλογές Μεταγλώττισης σε C++

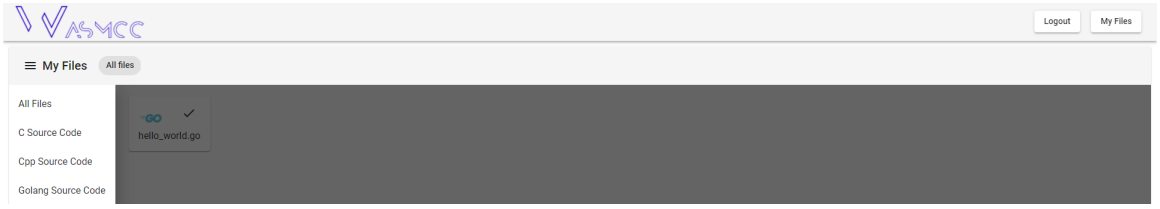


Σχήμα 5.9: Ολοκληρωμένη Μεταγλώττιση (Εγγεγραμμένος Χρήστης)

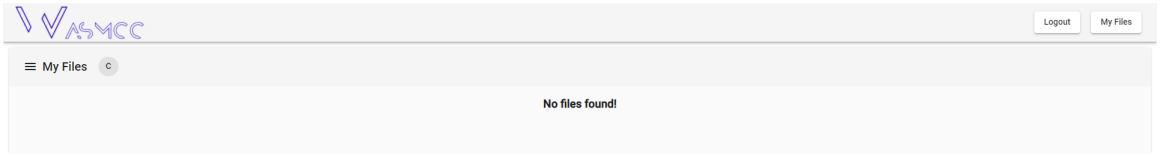


Σχήμα 5.10: Περιεχόμενα του results.zip αρχείου

Σχήμα 5.11: Overview όλων των ανεβασμένων αρχείων του χρήστη



Σχήμα 5.12: Επιλογή αρχείων συγκεκριμένης γλώσσας



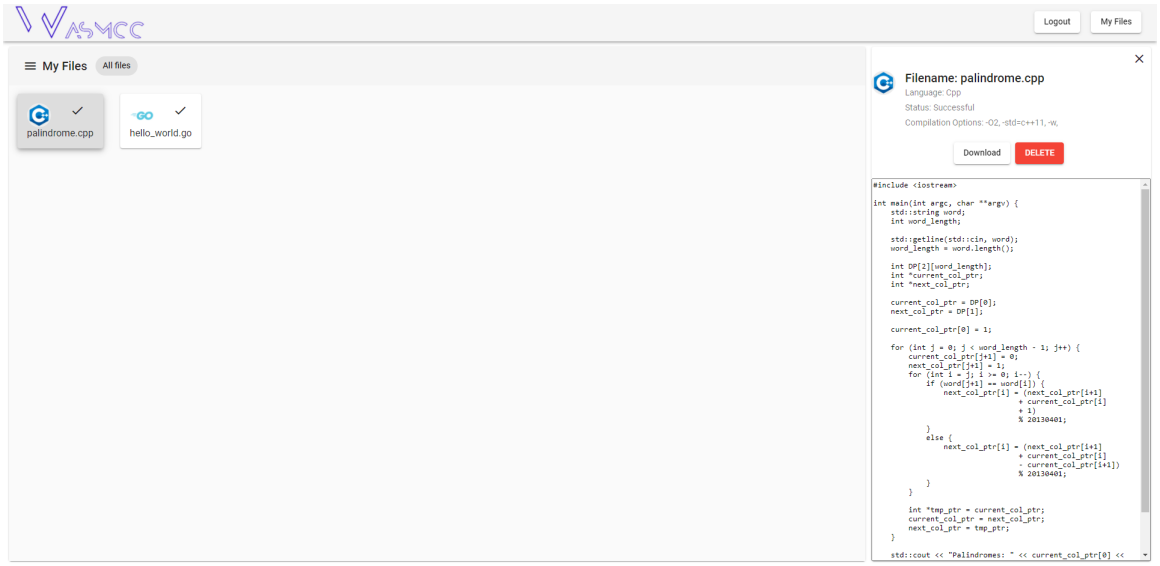
Σχήμα 5.13: Overview των ανεβασμένων αρχείων σε C του χρήστη

Σχήμα 5.14: Overview των ανεβασμένων αρχείων σε C++ του χρήστη

Σχήμα 5.15: Overview των ανεβασμένων αρχείων σε Golang του χρήστη

Σχήμα 5.16: Λεπτομέρειες μεταγλώττισης του αρχείου palindrome.cpp

Σχήμα 5.17: Διαγραφή του ανεβασμένου αρχείου palindrome.cpp



Σχήμα 5.18: Πηγαίος κώδικας του palindrome.cpp μέσω UI

Κεφάλαιο 6

Συμπεράσματα

6.1 Σύνοψη

Η εισαγωγή της WebAssembly ως μίας νέας γλώσσας στους browsers, μαζί με τις τρεις προϋπάρχουσες γλώσσες (HTML, CSS και JavaScript), δίνει νέες δυνατότητες στον τρόπο ανάπτυξης Web εφαρμογών. Το γεγονός πως είναι μία χαμηλού επιπέδου, assembly-like, γλώσσα, την καθιστά κατάλληλη ώστε να λειτουργήσει ως γλώσσα στόχος κατά τη μεταγλώττιση προγραμμάτων γραμμένων σε γλώσσες υψηλού επιπέδου. Πριν την εμφάνιση της WebAssembly, η μεταγλώττιση των προγραμμάτων γραμμένων σε γλώσσες υψηλού επιπέδου μπορούσε να παράξει assembly για συγκεκριμένες αρχιτεκτονικές υπολογιστών. Αυτό είχε ως αποτέλεσμα, ορισμένες εφαρμογές να περιορίζονται σε συγκεκριμένες συσκευές είτε διότι είχαν κάποια ειδική αρχιτεκτονική είτε διότι η μεταγλώττισή τους για περισσότερες από μία πλατφόρμες ήταν δύσκολη και κοστοβόρα.

Αντίθετα, η WebAssembly μπορεί να τρέξει σε όλους τους σύγχρονους browsers ανεξαρτήτως της αρχιτεκτονικής του επεξεργαστή. Αυτό έχει ως άμεση συνέπεια, ένα πρόγραμμα το οποίο μεταγλωττίζεται σε WebAssembly να μπορεί να εκτελεστεί σε όλες τις συσκευές οι οποίες μπορούν να υποστηρίξουν έναν browser, ο οποίος αναγνωρίζει WebAssembly. Με αυτόν τον τρόπο, εφαρμογές οι οποίες ήταν δεσμευμένες με συγκεκριμένους επεξεργαστές και συσκευές, μπορούν να μεταφερθούν στους browsers (και κατ' επέκταση σε πολλές συσκευές), χάρη στη WebAssembly. Η ταχύτητα και η αποδοτικότητα της WebAssembly βοηθά ώστε κρίσιμες εφαρμογές, οι οποίες απαιτούν άμεση απόκριση και γρήγορους υπολογισμούς (όπως Virtual reality, Augmented reality, 3D gaming και real-time εφαρμογές), να διατηρούν την ταχύτητά τους σχεδόν αναλλοίωτη όταν μεταφέρονται στους browsers και εκτελούνται σε WebAssembly. Αυτή η συμπεριφορά είναι σημαντική διότι η JavaScript, παρά το γεγονός πως διαθέτει ένα πολύ μεγάλο σύνολο βιβλιοθηκών που θα μπορούσαν να βοηθήσουν ώστε να γραφτούν οι εφαρμγές αυτές εξαρχής σε JavaScript, δεν είναι πάντοτε αποδοτική με αποτέλεσμα να μην μπορεί να εξασφαλιστεί η ταχύτητα που χρειάζονται ορισμένες εφαρμογές ώστε να λειτουργήσουν με επιτυχία.

Ωστόσο, η εμφάνιση της WebAssembly δεν έρχεται χωρίς κόστος καθώς οι ομάδες ανάπτυξης λογισμικού χρειάζεται να αφιερώσουν χρόνο ώστε να κατανοήσουν τη νέα αυτή γλώσσα και να είναι σε θέση να χρησιμοποιήσουν επιτυχώς αυτή και τα εργαλεία της. Το γεγονός αυτό εισάγει μία επιπλέον καθυστέρηση στην παραγωγή και ανάπτυξη Web εφαρμογών. Κατ' επέκταση, ένα σύστημα το οποίο θα παρέχει τη μεταγλώττιση της WebAssembly ως Υπηρεσία βοηθά στον περιορισμό του προβλήματος αυτού.

Ένα τέτοιο σύστημα είναι δυνατό να δημιουργηθεί. Πιο συγκεκριμένα, συλλέγει ένα σύνολο μεταγλωττιστών για γλώσσες που υποστηρίζουν μεταγλώττιση σε WebAssembly και είναι σε θέση να παράξει, αυτόματα, τη WebAssembly (μαζί με βοηθητική JavaScript και HTML) αν του δοθεί πηγαίος κώδικας σε μία από τις γλώσσες υψηλού επιπέδου που υποστηρίζει. Η υπηρεσία, κρύβει από το χρήστη τον τρόπο λειτουργίας των εργαλείων που χρησιμοποιεί καθώς και τον τρόπο παραγωγής του τελικού εκτελέσιμου, παρέχοντας ταυτόχρονα βοηθητικό κώδικα σε HTML και JavaScript ώστε να διευκολύνουν το χρήστη με την ενσωμάτωση της WebAssembly στη Web εφαρμογή του. Ο χρήστης βλέπει και χρησιμοποιεί ένα WebUI, το οποίο παρέχει ορισμένες επιλογές μεταγλώττισης για κάθε γλώσσα ώστε να προσαρμοστεί η μεταγλώττιση του κώδικα στις ανάγκες του. Επιπλέον, η υπηρεσία παρέχει περισσότερες δυνατότητες σε εγγεγραμμένους χρήστες,

όπως η δυνατότητα επίβλεψης όλων των ανεβασμένων αρχείων κώδικα του χρήστη καθώς και η δυνατότητα επανεύρεσης των αποτελεσμάτων μίας προγενέστερης μεταγλώττισης.

Το σύστημα έχει cloud-native αρχιτεκτονική. Συγκεκριμένα, χρησιμοποιώντας Containers, οργανώνει τις επιμέρους λειτουργίες του σε ανεξάρτητα microservices. Με αυτόν τον τρόπο, η υπηρεσία μπορεί να εκτελεστεί σε οποιοδήποτε περιβάλλον υποστηρίζει και μπορεί να οργανώσει Docker Containers ενώ, ταυτόχρονα, χάρη στα microservices, είναι εύκολα scalable. Επιπρόσθετα, λόγω του δικτύου μέσα στο οποίο βρίσκονται οι Containers και της δυνατότητας επικοινωνίας τους μέσω ορισμένων API, είναι δυνατό containers που σχετίζονται με το ίδιο microservice, να κατανεμηθούν σε διαφορετικά φυσικά μηχανήματα. Όλα τα παραπάνω, έχουν ως συνέπεια την ανάπτυξη ενός συστήματος που εξασφαλίζει αποδοτικότητα και high-availability.

6.2 Επεκτάσεις του Συστήματος και της Εφαρμογής

6.2.1 Επέκταση εφαρμογής με νέες γλώσσες

Στο παρόν σύστημα είναι δυνατό να προστεθούν μεταγλωττιστές και για άλλες γλώσσες, δίχως να χρειάζονται πολλές παρεμβάσεις ή τροποποιήσεις στον κώδικα τόσο του frontend όσο και του backend. Πιο αναλυτικά, για να υποστηριχθεί μία νέα γλώσσα στο backend χρειάζεται να γίνουν οι ακόλουθες προσθήκες. Αρχικά, θα πρέπει να προστεθεί ο νέος μεταγλωττιστής στο Docker Image του rest_server. Επιπλέον, θα πρέπει να προσδιοριστει το ”root_upload_path” της γλώσσας ως μεταβλητή περιβάλλοντος στον container. Στη συνέχεια, χρειάζεται να προστεθεί ένα νέο module κάτω από το subpackage *compile*, στο οποίο θα υπάρχουν οι υλοποιήσεις των αφηρημένων μεθόδων της CompilationHandler κλάσης. Τέλος, στο *__init__.py* αρχείο του subpackage *compile* χρειάζεται να προστεθεί το νέο module ως import καθώς και να επεκταθεί το dictionary των Compilation Handlers με ένα στιγμιότυπο του Compilation Handler της νέας γλώσσας.

```
from .compile_c import CCompilationHandler
from .compile_cpp import CppCompilationHandler
from .compile_golang import GolangCompilationHandler

compilation_handlers_dictionary = {
    "C": CCompilationHandler(),
    "Cpp": CppCompilationHandler(),
    "Golang": GolangCompilationHandler()
}
```

Σχήμα 6.1: Subpackage compile: κώδικας στο αρχείο __init__.py

Ο φάκελος στο νέο και μοναδικό ”root_upload_path” για τη νέα γλώσσα μπορεί είτε να δημιουργηθεί στο host περιβάλλον απευθείας, είτε να προστεθεί στο script που είναι υπεύθυνο για το deployment του συστήματος (όπως περιγράφεται στη συνέχεια). Σε κάθε περίπτωση, λόγω του Volume που χρησιμοποιείται μεταξύ του host και των containers, το νέο path θα είναι ορατό μέσα στους containers.

Για να υποστηριχθεί μία νέα γλώσσα στο frontend χρειάζεται να γίνουν ορισμένα, ακόμα, βήματα. Αρχικά, χρειάζεται να επεκταθεί η δομή AvailableLanguages στο αρχείο models/source-code-file.ts με το όνομα της νέας γλώσσας. Στη συνέχεια, πρέπει να προστεθεί ένα typescript αρχείο με όνομα <language>-compilation-options.ts κάτω από το φάκελο compilation-options (όπου το <language> είναι γραμμένο με πεζούς χαρακτήρες), το οποίο θα περιέχει μία νέα κλάση η οποία θα ορίζει τις επιλογές μεταγλώττισης της γλώσσας καθώς και ένα αντικείμενο το οποίο θα περιέχει τα σύνολα των επιτρεπόμενων τιμών που μπορούν να πάρουν οι επιλογές μεταγλώττισης. Έπειτα, είναι απαραίτητο να προστεθεί ένα component με όνομα <language>-upload-form (όπου το <language> είναι γραμμένο με πεζούς χαρακτήρες), το οποίο θα είναι υπεύθυνο για

τη σχεδίαση του UI με τις επιλογές μεταγλώττισης της συγκεκριμένης γλώσσας. Με την προσθήκη του νέου component θα πρέπει να ενημερωθεί το *file-upload* component ώστε να αναγνωρίζει το νέο component και τις νέες επιλογές μεταγλώττισης για τη νέα γλώσσα. Τέλος, χρειάζεται να προστεθεί μία εικόνα με το logo της γλώσσας μέσα στο φάκελο assets/images, με όνομα <language>-logo.png (όπου το <language> είναι το όνομα της γλώσσας όπως δηλώνεται στη δομή AvailableLanguages).

6.2.2 Επέκταση λειτουργιών

Οι λειτουργίες της εφαρμογής τη στιγμή της συγγραφής συγκεντρώνονται στις δυνατότητες του χρήστη να εγγράφεται στην εφαρμογή και να συνδέεται σε αυτήν, να μεταγλωττίζει αρχεία πηγαίου κώδικα από C, C++ και Golang σε WebAssembly καθώς και να διαχειρίζεται και να εποπτεύει τα ανεβασμένα αρχεία του στο σύστημα. Ωστόσο, οι λειτουργίες του συστήματος μπορούν να επεκταθούν. Πιο συγκεκριμένα, στο εφαρμογή μπορεί να ενταχθεί ο ρόλος του "Διαχειριστή Συστήματος". Έτσι, οι χρήστες οι οποίοι θα έχουν αυτό το ρόλο θα μπορούν για παράδειγμα να ελέγχουν τις επιδόσεις του συστήματος και να αλληλεπιδρούν με τους containers είτε μέσω ενός WebUI είτε μέσω CLI αλλά και να διαγράφουν άλλους χρήστες σε περίπτωση που αυτοί προσπαθήσουν να καταχραστούν ή να βλάψουν το σύστημα.

Μία επιπρόσθετη λειτουργία αποτελεί η δυνατότητα εκτέλεσης των παραγόμενων αρχείων της WebAssembly στον φυλλομετρητή του πελάτη. Στην παρούσα υλοποίηση, ο χρήστης ανεβάζει ένα αρχείο προς μεταγλώττιση και αν η μεταγλώττιση είναι επιτυχής, τότε λαμβάνει ως απάντηση ένα αρχείο ZIP με html, javascript και WebAssembly ώστε να μπορεί να τη χρησιμοποιήσει και να ενσωματώσει εύκολα στην εφαρμογή του τον κώδικα που επιθυμεί. Μία νέα, λοιπόν, λειτουργία μπορεί να αποτελέσει η εκτέλεση των αρχείων αυτών στον φυλλομετρητή του χρήστη. Για να επιτευχθεί αυτό, αρκεί ο wsgi REST server να απαντήσει με την html, τη javascript και τη wasm που ο ίδιος παρήγαγε κατά τη μεταγλώττιση. Δεδομένου πως ένας wsgi server μπορεί γενικά να απαντήσει με HTML και με άλλα αρχεία τα οποία, ενδεχομένως, χρειάζεται να τη συνοδεύσουν, καθώς και το γεγονός πως το υλοποιημένο σύστημα διαθέτει ήδη μηχανισμούς εντοπισμού ενός αρχείου, είναι χρήσιμο και εφικτό, χωρίς πολλές παρεμβάσεις στον υπάρχοντα κώδικα, να εκτελείται ο κώδικας στον φυλλομετρητή του πελάτη και κατά συνέπεια να επεκταθεί η εφαρμογή με αυτή τη λειτουργία.

Βιβλιογραφία

[Haas] Andreas Haas, Andreas Rossberg, Derek L. Schuff, Ben L. Titzer, Michael Holman, Dan Gohman, Luke Wagner, Alon Zakai and JF Bastien, "Bringing the Web up to Speed with WebAssembly".

[Herr18] David Herrera, Hanfeng Chen, Erick Lavoie and Laurie Hendren, "WebAssembly and JavaScript Challenge: Numerical program performance using modern browser technologies and devices", Techical Report McLAB-2018-02, McGill University, School of Computer Science, Sable Research Group, 2018.

[PEP3] "https://www.python.org/dev/peps/pep-3333/".

[RFC6] "https://tools.ietf.org/html/rfc6750".

[RFC7] "https://tools.ietf.org/html/rfc7519".