ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ

ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ
ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ
ΤΟΜΕΑΣ ΗΛΕΚΤΡΙΚΩΝ ΒΙΟΜΗΧΑΝΙΚΩΝ ΔΙΑΤΑΞΕΩΝ
ΚΑΙ ΣΥΣΤΗΜΑΤΩΝ ΑΠΟΦΑΣΕΩΝ

Ανάλυση των προτύπων για APIs και αναγνώριση της σημασίας αυτών στον σημασιολογικό ιστό

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Μουρίκη Μ. Ευαγγελία

Επιβλέπων : Δημήτριος Ασκούνης
Αν. Καθηγητής Ε.Μ.Π

Αθήνα, Ιούλιος 2014

ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ

ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ
ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ
ΤΟΜΕΑΣ ΗΛΕΚΤΡΙΚΩΝ ΒΙΟΜΗΧΑΝΙΚΩΝ ΔΙΑΤΑΞΕΩΝ
ΚΑΙ ΣΥΣΤΗΜΑΤΩΝ ΑΠΟΦΑΣΕΩΝ

Ανάλυση των προτύπων για APIs και αναγνώριση της σημασίας αυτών στον σημασιολογικό ιστό

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Μουρίκη Μ. Ευαγγελία

Επιβλέπων : Δημήτριος Ασκούνης
Αν. Καθηγητής Ε.Μ.Π

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή την … Ιουλίου 2014

……	……	……..
Δ. Ασκούνης	Ι. Ψαρράς	Β. Ασημακόπουλος
Αν. Καθηγητής Ε.Μ.Π	Καθηγητής Ε.Μ.Π	Καθηγητής Ε.Μ.Π

Αθήνα, Ιούλιος 2014

……..

Ευαγγελία Μ. Μουρίκη

Διπλωματούχος Ηλεκτρολόγος μηχανικός και Μηχανικός Υπολογιστών Ε.Μ.Π

ΠΕΡΙΛΗΨΗ

Τα web APIs είναι μία τεχνολογία με σύντομη ιστορία καθώς πρωτοεμφανίστηκαν γύρω στο 2000. Από τότε ο αριθμός των κλήσεων API έχει πολλαπλασιαστεί και έχουν επεκταθεί σε πολλούς τομείς των επιχειρήσεων ακολουθώντας την εξέλιξη του διαδικτύου. Η πρώτη εμφάνισή τους ήταν σχετικά με το εμπόριο και τα κοινωνικά δίκτυα και στη συνέχεια χρησιμοποιήθηκαν για marketing, χαρτογράφηση και cloud computing. Τελευταία τάση των web APIs είναι η χρήση τους στα κινητά.

Για την διευκόλυνση των προγραμματιστών σχετικά με την ανάπτυξη web APIs έχουν δημιουργηθεί πρότυπα και μοντέλα που συστηματοποιούν την δομή και την γλώσσα τους. Τα τρία πρότυπα που αναλύονται στην παρούσα εργασία είναι το Swagger, η RAML και το API-Blueprint. Το Swagger είναι ένα πλήρες πλαίσιο για την περιγραφή, δημιουργία και κατανάλωση των RESTful υπηρεσιών web. Κύριος στόχος του είναι οι clients και τα συστήματα τεκμηρίωσης να ανανεώνονται στον ίδιο ρυθμό με τον server. Η RAML είναι ένας απλός και περιεκτικός τρόπος για την περιγραφή των πρακτικά RESTful APIs. Ενθαρρύνει την επαναχρησιμοποίηση, διευκολύνει την εύρεση και την ανταλλαγή μοτίβων και έχει στόχο την επικράτηση των καλύτερων πρακτικών. Τέλος, Το API Blueprint παρέχει εξαιρετικά εργαλεία για όλο τον κύκλο ζωής του API. Κάνει τον σχολιασμό σχετικά με κάποιο API εύκολο και παράγει τεκμηρίωση αυτόματα.

Στην συνέχεια παρουσιάζεται ο ρόλος και η σημασία του σημασιολογικού ιστού, ο οποίος αποτελεί ένα νέο στάδιο στην ιστορία του διαδικτύου και έχει στόχο να κάνει ευκολότερη την σύνδεση των δεδομένων μεταξύ τους και την δημιουργία Linked Data όπως ορίστηκαν από τον Berners-Lee. Με τον τρόπο αυτό πολλές διαδικασίες που απαιτούν την παρουσία του ανθρώπου θα μπορέσουν να αυτοματοποιηθούν. Τον σημασιολογικό ιστό προσπαθεί να εκμεταλλευτεί η Hydra. Η Hydra είναι μία προσπάθεια που στόχο έχει να απλοποιήσει την ανάπτυξη των hypermedia-driven web APIs. Προσπαθεί να δημιουργήσει ένα καθολικό πρότυπο που θα συνδυάζει τα πλεονεκτήματα των επιμέρους προτύπων που αναλύθηκαν και θα χρησιμοποιείται από όλους τους προγραμματιστές ώστε να είναι ευκολότερη η συντήρηση και ανάπτυξη των APIs. Έτσι θα είναι δυνατή η δημιουργία γενικών πελατών API σε αντίθεση με την κατάσταση που επικρατεί σήμερα.

Για την καλύτερη κατανόηση της χρήσης και της λειτουργίας των APIs παρουσιάζεται ένα συγκεκριμένο παράδειγμα που αφορά τις κλήσεις API που εκτελούνται για να ολοκληρωθεί μία αγορά από το eBay μέσω του PayPal. Επιπλέον υπάρχει ενδεικτικά μία μοντελοποίηση των κλήσεων αυτών σύμφωνα με τη Hydra.

Τέλος παρουσιάζεται η σημασία των APIs στην ανάπτυξη μίας επιχείρησης καθώς μπορούν να αποτελέσουν ανταγωνιστικό στοιχείο και να συμβάλλουν καθοριστικά στην ανάπτυξή της.

ΛΕΞΕΙΣ ΚΛΕΙΔΙΑ

web API, πρότυπα API, REST, Swagger, Raml, API-Blueprint, ιστορία API, σημασιολογικός ιστός, Hydra, παράδειγμα κλήσεων API, JSON

ABSTRACT

Web APIs is a technology with a short history as they first came out in 2000. Since then the number of API calls has been increased and APIs have been expanded in many business sectors by following the evolution of web. They first emerged in commerce and social platforms and then they were used for marketing, mapping and cloud computing. The last trend for APIs is related to mobile.

In order developers to easier create web APIs, standards and models have been created that regularize their structure and language. The three standards that are analyzed in this thesis are Swagger, RAML and API-Blueprint. Swagger is a specification and complete framework implementation for describing, producing, consuming, and visualizing RESTful web services. The overarching goal of Swagger is to enable client and documentation systems to update at the same pace as the server. RAML is a simple and succinct way of describing practically-RESTful APIs. It encourages reuse, enables discovery and pattern-sharing, and aims for merit-based emergence of best practices. Finally, API Blueprint gives tools for the whole API lifecycle. It enables comments on an API and generates documentation automatically.

Then the role and meaning of semantic web is presented. Semantic web is the new stage in the history of web and aims to the connection of data and to the creation of Linked Data as they defined by Berners-Lee. In this way, many procedures that demand the presence of a man can be automated. Hydra is an effort to simplify the development of interoperable, hypermedia-driven Web APIs and tries to take advantage of semantic web. Its main goal is to create a global standard that combines the advantages of the individual standards that have already been described and to be used by the majority of the developers. This would result in the easier maintenance and creation of APIs and in the creation of general API clients.

In order to better understand the usage and operation of web APIs a particular example is described. This example is related to the API calls that are executed for the completion of a purchase from eBay by using PayPal. Moreover there is a standardization of these calls according to Hydra.

Finally the significance of the APIs in the evolution of a business is presented. APIs are a competitive element and can contribute substantially in its successful development.

KEYWORDS

web API, API standards, REST, Swagger, Raml, API-Blueprint, API history, semantic web, Hydra, API calls, JSON

Ευχαριστίες

Θα ήθελα να ευχαριστήσω θερμά τον Αναπληρωτή Καθηγητή κύριο Δημήτριο Ασκούνη ως επιβλέποντα της εργασίας αυτής για τις γνώσεις που μου προσέφερε και για την πολύτιμη βοήθειά του ώστε να συνεχίσω τις σπουδές μου στο εξωτερικό.
Επίσης, θα ήθελα να ευχαριστήσω τον υποψήφιο διδάκτορα κύριο Μιχάλη Πετυχάκη για την ευκαιρία που μου έδωσε να εργαστώ πάνω σε ένα σύγχρονο αντικείμενο καθώς και για τη συνεχή βοήθεια και καθοδήγησή του κατά την εκπόνηση της εργασίας.

Πίνακας περιεχομένων

1. Περιγραφή των APIs

Αντικείμενο της παρούσας διπλωματικής εργασίας είναι η μελέτη και η εξέλιξη των web APIs (Application Programming Interface – Διεπαφή προγραμματισμού εφαρμογών) τα οποία είναι άρρηκτα συνδεδεμένα με την εξέλιξη του διαδικτύου και κερδίζουν όλο και μεγαλύτερο χώρο στην καθημερινότητά μας.

1.1 Ορισμός API

Ένας από τους βασικούς σκοπούς μίας διεπαφής (API) είναι να ορίζει και να διατυπώνει το σύνολο των λειτουργιών-υπηρεσιών που μπορεί να παρέχει μια βιβλιοθήκη ή ένα λειτουργικό σύστημα σε άλλα προγράμματα, χωρίς να επιτρέπει πρόσβαση στον κώδικα που υλοποιεί αυτές τις υπηρεσίες. Η διεπαφή, ένα «συμβόλαιο κλήσης» μεταξύ καλούντος και καλούμενου, διαχωρίζει την προγραμματιστική υλοποίηση κάποιων υπηρεσιών από τη χρήση τους. Έτσι για παράδειγμα το λειτουργικό σύστημα Windows έχει τη δική του διεπαφή (κλήσεις συστήματος), το φορμά της οποίας διατίθεται από την κατασκευάστρια εταιρεία Microsoft, και περιγράφει τους τρόπους αξιοποίησης του συνόλου των υπηρεσιών που παρέχει το λειτουργικό από προγράμματα χρήστη. Το τμήμα του λειτουργικού συστήματος το οποίο υλοποιεί τις υπηρεσίες που περιγράφονται στη διεπαφή, συνήθως στον πυρήνα του, λέμε ότι είναι η *υλοποίηση της διεπαφής*.

1.2 Ορισμός web API

Όταν χρησιμοποιείται για την ανάπτυξη ιστοσελίδων ένα API συνήθως ορίζεται ως ένα σύνολο από HTTP ζητούμενα μηνύματα μαζί με έναν ορισμό της δομής των μηνυμάτων ανταπόκρισης ο οποίος είναι συνήθως σε μορφή επεκτάσιμης γλώσσας σήμανσης (XML) ή σημειογραφίας αντικειμένων JavaScript (JSON). Αν και ο όρος «web API» ιστορικά ήταν συνώνυμος με τον όρο υπηρεσίες web η σύγχρονη τάση απομακρύνεται από το Simple Object Access Protocol (SOAP) που βασίζεται σε υπηρεσίες web και service-oriented architecture (SOA) και κατευθύνεται σε στυλ representation state transfer (REST), ηλεκτρονικούς πόρους και resource-oriented architecture (ROA).

1.3 Σενάριο Χρήσης

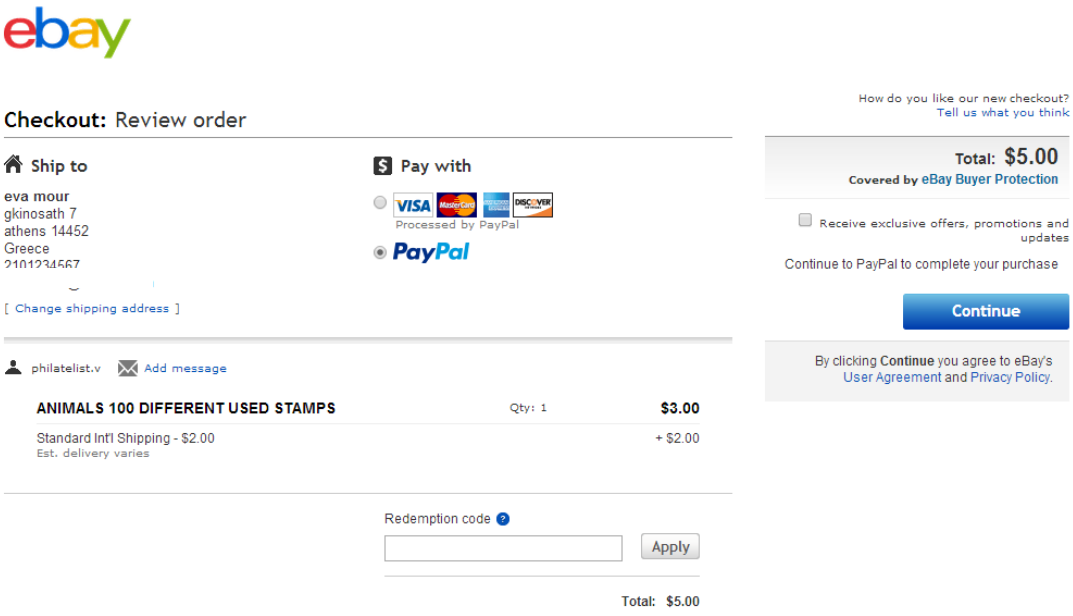
Για την κατανόηση της λειτουργίας των APIs αλλά και του τρόπου χρήσης τους, κρίθηκε σκόπιμη η περιγραφή ενός προβλήματος από την καθημερινή ζωή που έχει να κάνει με κλήσεις API. Συγκεκριμένα θα μελετήσουμε την διαδικασία που απαιτείται από το ξεκίνημα μέχρι την ολοκλήρωση μίας αγοράς από το eBay χρησιμοποιώντας το PayPal.

Το PayPal είναι θυγατρική επιχείρηση του eBay παρόλα αυτά μεταξύ τους παραμένουν ανεξάρτητες. Και ενώ ο καθένας μπορεί να ρυθμίσει τους λογαριασμούς του στα δύο αυτά

site ώστε να συνεργάζονται, ταυτόχρονα μπορεί να τους διαχειρίζεται ξεχωριστά. Στο παράδειγμα που ακολουθεί θα θεωρήσουμε ότι οι λογαριασμοί δεν έχουν συνδεθεί, έτσι είναι ακόμα πιο φανερή η λειτουργία των APIs.

Τα βήματα που ακολουθούνται αλλά και τα αντίστοιχα API calls για την πραγματοποίηση μιας αγοράς είναι τα εξής:

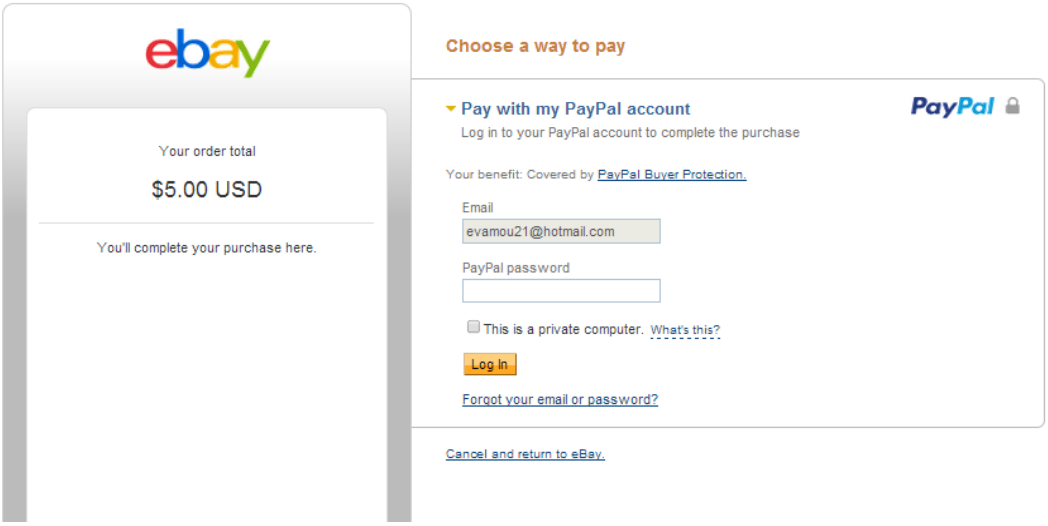
Αρχικά επιλέγεται το προϊόν που πρόκειται να αγοραστεί και η πληρωμή μέσω PayPal όπως φαίνεται στην παρακάτω εικόνα:



Για να ολοκληρωθεί η αγορά μέσω του PayPal απαιτούνται τρία βήματα:

1. Προσδιορισμός των πληροφοριών που σχετίζονται με την πληρωμή (π.χ τιμή, στοιχεία αποστολής)
2. Έγκριση πληρωμής
3. Εκτέλεση της πληρωμής στον λογαριασμό PayPal του χρήστη

Βέβαια για να φτάσουμε σε αυτό το σημείο ο χρήστης θα πρέπει να έχει συνδεθεί στον λογαριασμό του στο PayPal δίνοντας τα κατάλληλα username και password όπως φαίνεται στην οθόνη που ακολουθεί:



Όταν ο χρήστης συμπληρώσει τα στοιχεία αυτά και πατήσει Log in γίνεται ένα GET request στον server του PayPal για να επιβεβαιώσει αν τα στοιχεία που δόθηκαν είναι σωστά και αντιστοιχούν σε κάποιον υπάρχοντα λογαριασμό. Σε περίπτωση που είναι σωστά στέλνεται το αντίστοιχο response και ο χρήστης συνδέεται στον λογαριασμό του για να ολοκληρώσει την αγορά. Αν δεν είναι σωστά όμως τα στοιχεία ο server δεν επιτρέπει την πρόσβαση και εμφανίζει μήνυμα λάθους.

Για την συγκέντρωση των πληροφοριών της πληρωμής γίνεται κλήση `/payment` στον server του PayPal από το eBay. Το `intent` τίθεται `sale`, και το `payment_method` τίθεται `paypal`. Επιπλέον εισάγονται διευθύνσεις ανακατεύθυνσης οι οποίες χρησιμοποιούνται σε περίπτωση αποδοχής ή ακύρωσης της πληρωμής.

Μία τέτοια κλήση θα μπορούσε να είναι της μορφής:

```
curl -v https://api.sandbox.paypal.com/v1/payments/payment \
-H 'Content-Type: application/json' \
-H 'Authorization: Bearer {accessToken}' \
-d '{
  "intent":"sale",
  "redirect_urls":{
    "return_url":"http://<return URL here>",
    "cancel_url":"http://<cancel URL here>"
  },
  "payer":{
    "payment_method":"paypal"
  },
  "transactions":[
    {
      "amount":{
        "total":"7.47",
```

```
        "currency":"USD"
      },
      "description":"This is the payment transaction description."
    }
  ]
}'
```

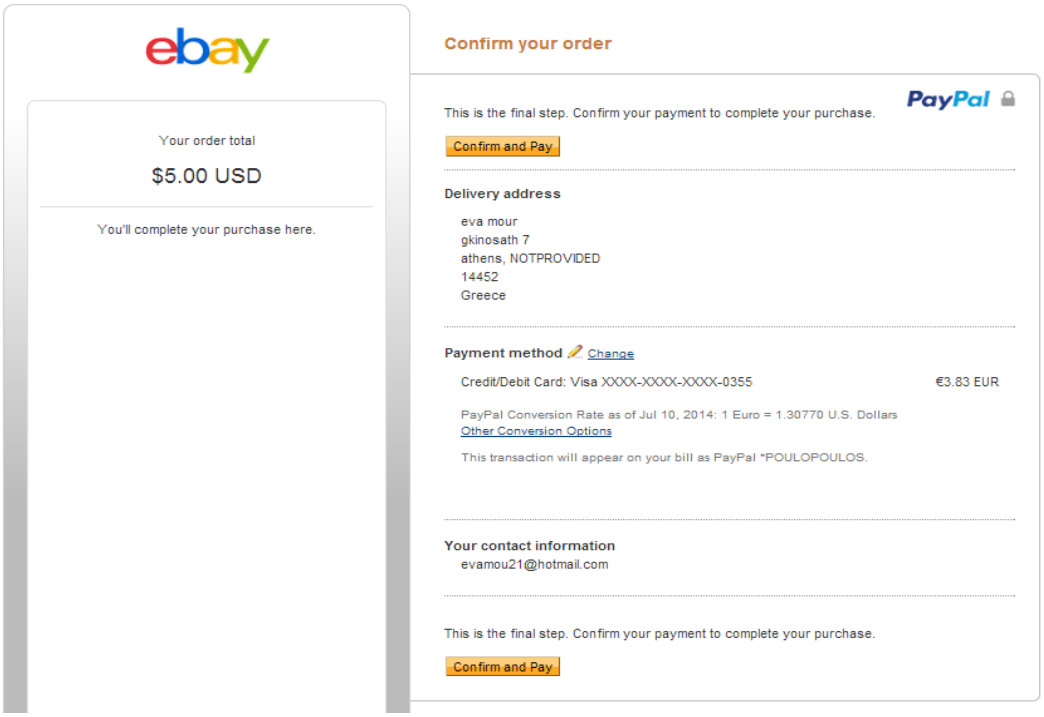
Εάν η κλήση είναι επιτυχημένη τότε το PayPal στέλνει μία βεβαίωση της συναλλαγής με το πεδίο `state` να είναι `created`.

Μία τέτοια βεβαίωση έχει την εξής μορφή:

```
{
  "id": "PAY-6RV70583SB702805EKEYSZ6Y",
  "create_time": "2013-03-01T22:34:35Z",
  "update_time": "2013-03-01T22:34:36Z",
  "state": "created",
  "intent": "sale",
  "payer": {
    "payment_method": "paypal"
  },
  "transactions": [
    {
      "amount": {
        "total": "7.47",
        "currency": "USD",
        "details": {
          "subtotal": "7.47"
        }
      },
      "description": "This is the payment transaction description."
    }
  ],
  "links": [
    {
      "href": "https://api.sandbox.paypal.com/v1/payments/payment/PAY-
6RV70583SB702805EKEYSZ6Y",
      "rel": "self",
      "method": "GET"
    },
    {
```

```
      "href": "https://www.sandbox.paypal.com/webscr?cmd=_express-
checkout&token=EC-60U79048BN7719609",
      "rel": "approval_url",
      "method": "REDIRECT"
    },
    {
      "href": "https://api.sandbox.paypal.com/v1/payments/payment/PAY-
6RV70583SB702805EKEYSZ6Y/execute",
      "rel": "execute",
      "method": "POST"
    }
  ]
}
```

Προκειμένου ο χρήστης να εγκρίνει την πληρωμή πρέπει να γίνει μία ανακατεύθυνση στο `approval_url` του site του PayPal. Η εκτέλεση της πληρωμής και η ολοκλήρωση της αγοράς δεν μπορεί να πραγματοποιηθεί παρακάμπτοντας το βήμα της έγκρισης.



Τέλος, όταν ο χρήστης εγκρίνει την πληρωμή, το PayPal ανακατευθύνει τον χρήστη στο `return_url` που ορίστηκε κατά την δημιουργία της πληρωμής. Το ID του αγοραστή επισυνάπτεται στο URL της επιστροφής ως `PayerID`:

`http://<return_url>?token=EC-60U79048BN7719609&PayerID=7E7MGXCWTTKK2`

Η τιμή token που δίνεται σε αυτό το URL δεν χρειάζεται κατά την εκτέλεση της πληρωμής. Μετά την έγκριση του χρήστη, γίνεται μία κλήση a `/payment/execute/` .

Στο σώμα αυτής της αίτησης χρησιμοποιείται η τιμή `payer_id` από το URL της επιστροφής. Στην επικεφαλίδα χρησιμοποιείται το access token που παράχθηκε κατά την δημιουργία της πληρωμής.

```
curl -v https://api.sandbox.paypal.com/v1/payments/payment/PAY-
6RV70583SB702805EKEYSZ6Y/execute/ \
-H 'Content-Type: application/json' \
-H 'Authorization: Bearer {accessToken}' \
-d '{ "payer_id" : "7E7MGXCWTTKK2" }'
```

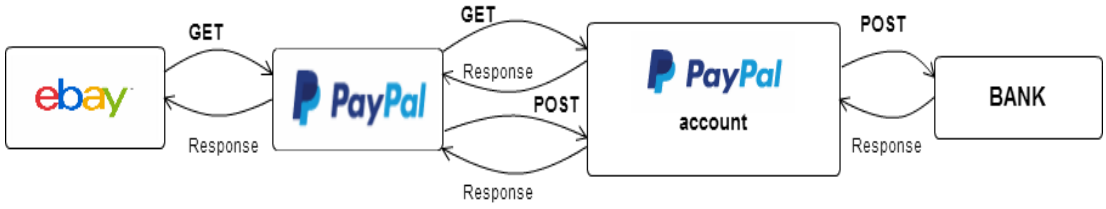
Από την στιγμή που η πληρωμή ολοκληρωθεί η συγκεκριμένη συναλλαγή θα αναφέρεται ως πώληση.

Η συνέχεια της πληρωμής λαμβάνει χώρα μεταξύ του PayPal και της τράπεζας στην οποία βρίσκεται ο λογαριασμός ή η πιστωτική κάρτα που θα χρεωθεί μέσω API calls. Το PayPal εκτελεί ένα POST request με το οποίο ζητά να χρεωθεί η συγκεκριμένη πιστωτική κάρτα ή λογαριασμός κατά το ποσό που εμφανίζεται στην οθόνη του χρήστη. Στη συγκεκριμένη περίπτωση δηλαδή 3,83€. Μαζί με αυτό βέβαια στέλνει και τα στοιχεία της κάρτας όπως ο αριθμός της, η ημερομηνία λήξης της και το όνομα του κατόχου.

Αφού γίνει η ταυτοποίηση των στοιχείων αυτών μέσα στον server της τράπεζας και βρεθεί ότι τα στοιχεία αυτά ανταποκρίνονται σε πραγματική κάρτα ολοκληρώνεται η συναλλαγή μέσω ενός response με το κατάλληλο μήνυμα. Τότε ο server του PayPal ενημερώνει τον λογαριασμό του χρήστη του με POST request προσθέτοντας την συναλλαγή αυτή στη λίστα με τις συναλλαγές του.

Στο σημείο αυτό έχει ολοκληρωθεί η διαδικασία αγοράς του επιθυμητού προϊόντος.

Στην εικόνα που ακολουθεί παρουσιάζεται το διάγραμμα με τα API calls τα οποία έγιναν για αυτή τη συναλλαγή



Παρατηρούμε ότι με την βοήθεια των API calls η διαδικασία αυτοματοποιείται σε μεγάλο βαθμό. Χαρακτηριστικό είναι ότι γίνονται αυτόματα ανακατευθύνσεις σε άλλα site όπως έγινε με το να ανοίξει το PayPal μέσω του eBay. Επιπλέον παρατηρούμε ότι δεν υπάρχει καμία άμεση επικοινωνία του αγοραστή με την τράπεζα. Το μόνο που έκανε ο αγοραστής ήταν να συνδεθεί με τον λογαριασμό του στο PayPal. Γίνεται λοιπόν φανερό ότι ο καθένας χρησιμοποιεί APIs χωρίς όμως να το αντιλαμβάνεται.

Στο σημείο αυτό πρέπει να αναφερθεί ότι επειδή το συγκεκριμένο παράδειγμα που επιλέχθηκε αφορά τράπεζες και χρήματα γενικότερα οι ταυτοποιήσεις που χρειάζονται ανάμεσα στις κλήσεις των APIs είναι πολυσύνθετες. Αυτό γίνεται με σκοπό να υπάρχει μεγαλύτερη ασφάλεια στις συναλλαγές και να αποφεύγεται ο κίνδυνος για χρέωση λάθος καρτών και λογαριασμών.

Σχετικά με τα API που διαθέτει κάθε μέρος της συναλλαγής, το eBay, το PayPal και η τράπεζα πρέπει να αναφερθεί ότι υπάρχουν αρκετές διαφορές. Οι διαφορές αυτές πηγάζουν από τον διαφορετικό ρυθμό ανανέωσης των APIs αυτών των φορέων. Το eBay ως μια επιχείρηση που δραστηριοποιείται στο διαδίκτυο υιοθετεί τις νέες τάσεις για το API του. Είναι μάλιστα μία πρωτοπόρα επιχείρηση στον τομέα των APIs και μία επιχείρηση που αποτελεί σημείο αναφοράς για αυτά. Το PayPal προσπαθεί και αυτό να ανανεώνει τις δομές του αλλά με μικρότερο ρυθμό καθώς έχει να κάνει με οικονομικές συναλλαγές και πρέπει να τηρούνται αυστηρά μέτρα ασφαλείας και ελέγχου των νέων εκδόσεων των APIs. Τέλος με τον πιο αργό ρυθμό εξέλιξης των APIs τους προχωρούν οι τράπεζες. Αυτό συμβαίνει γιατί οι τράπεζες διαθέτουν σύνθετες δομές, κάτι που δεν επιτρέπει την εύκολη και με μικρό κόστος αναβάθμιση. Επιπλέον καθώς οι τράπεζες δεν έχουν ως κύριο τομέα εργασιών τους τους υπολογιστές, οι νέες τάσεις αργούν να αφομοιωθούν. Σύμφωνα με τα παραπάνω, τα APIs μιας τράπεζας είναι ελάχιστα ευέλικτα και επομένως τα APIs των συνεργατών της όπως το PayPal πρέπει να παραμένουν συμβατά μαζί τους κάτι που επιφέρει συνέπειες στην ταχεία εξέλιξή τους.

Στα επόμενα κεφάλαια παρουσιάζονται ορισμένα πρότυπα για την ανάπτυξη APIs προκειμένου να είναι δομημένα, επαναχρησιμοποιήσιμα και εύκολα στην κατανόηση από μεγάλο αριθμό προγραμματιστών. Επίσης παρουσιάζεται μια προσπάθεια για την ανάπτυξη μιας κοινής πρακτικής σε σχέση με τα APIs που ονομάζεται Hydra και βασίζεται στον σημασιολογικό ιστό.

2. Ιστορία των APIs

Πριν μελετήσουμε την ιστορική εξέλιξη των APIs γίνεται μία σύντομη αναδρομή στην ίδρυση του Internet και του World Wide Web που αποτέλεσαν τη βάση για αυτά. Τα APIs αρχικά ακολουθούσαν την εξέλιξη και τις τάσεις του διαδικτύου και πλέον μπορεί κανείς να πει ότι βρισκόμαστε στην φάση που τα APIs καθορίζουν την πορεία του.

2.1 Internet

Το Internet είναι μία προέκταση της τεχνολογίας των δικτύων των υπολογιστών. Οι πρώτοι υπολογιστές λειτουργούσαν ανεξάρτητα. Τις δεκαετίες του 1960 και 1970 οι υπολογιστές που λειτουργούσαν σε έναν οργανισμό (π.χ. πανεπιστήμιο, κυβέρνηση, επιχείρηση) άρχισαν να συνδέονται μεταξύ τους σε ένα δίκτυο. Ταυτόχρονα γίνονταν πειράματα για την σύνδεση ολόκληρων δικτύων μεταξύ τους συμπεριλαμβανομένου του ARPANET στις Ηνωμένες Πολιτείες. Στις αρχές τις δεκαετίας του ’80 πρωτυποποιήθηκε το Internet Protocol Suite (TCP/IP) για το ARPANET, το οποίο παρείχε τις βάσεις για το δίκτυο των δικτύων που θα έφτανε σε όλο τον κόσμο. Το Internet εξαπλώθηκε κυρίως στην Ευρώπη και την Αυστραλία την δεκαετία του ’80 και στον υπόλοιπο κόσμο την δεκαετία του ’90.

Η τεχνολογία που υποστηρίζει το Internet περιλαμβάνει το IP (Internet Protocol) σύστημα για την διευθυνσιοδότηση των υπολογιστών, ώστε τα μηνύματα να μπορούν να σταλούν από τον ένα υπολογιστή στον άλλο. Κάθε υπολογιστής στο Internet έχει έναν αριθμό IP που μπορεί να γραφεί ως τέσσερις ακέραιοι από 0-255 χωρισμένοι με τελείες, π.χ. 185.56.200.4 (αυτή η μορφή ίσχυε μέχρι την τέταρτη έκδοση του IP και δεν ισχύει πια). Η δομή των μηνυμάτων καθορίζεται από πρωτόκολλα εφαρμογών που ποικίλουν ανάλογα με την υπηρεσία που απαιτείται (π.χ. email, τηλεφωνία, μεταφορά αρχείων, hypertext). Παραδείγματα τέτοιων πρωτοκόλλων είναι το FTP (μεταφορά αρχείων), USENET και HTTP (μεταφορά HyperText).

2.2 Hypertext

Η ιδέα του hypertext πηγάζει από το άρθρο του Bush και Wang το 1945 με τίτλο “as we may think”, το οποίο πρότεινε την οργάνωση εξωτερικών καταχωρήσεων (βιβλία, άρθρα, φωτογραφίες) σύμφωνα με τον τρόπο που συσχετίζονται οι ιδέες στην ανθρώπινη μνήμη. Μέχρι το 1960 η ιδέα αυτή εφαρμοζόταν μόνο από καινοτόμους όπως ο Douglas Engelbart και ο Ted Nelson σε προγράμματα που έδιναν τη δυνατότητα με κάποιες γέφυρες που σημειώνονταν ως Hyperlinks, ο αναγνώστης να μπορεί να μεταβεί σε ένα άλλο έγγραφο.

2.3 World Wide Web

Ανεπίσημα πολλές φορές οι άνθρωποι χρησιμοποιούν τους όρους “Internet” και “World Wide Web” (WWW) ως συνώνυμους αλλά αυτό είναι ανακριβές: το WWW είναι στην πραγματικότητα μία από τις πολλές υπηρεσίες που εκτελούνται στο Internet. Το χαρακτηριστικό γνώρισμα του WWW είναι ότι είναι μία hypertext εφαρμογή, που αξιοποιεί το Internet για να επιτρέπει συνδέσεις μεταξύ των εγγράφων σε όλο τον κόσμο.

Ο Tim Berners-Lee επινόησε τον παγκόσμιο ιστό (World Wide Web) το 1989, περίπου 20 χρόνια μετά την πρώτη σύνδεση μέσω διαδικτύου (Internet). Εκείνη την περίοδο εργαζόταν ως μηχανικός λογισμικού στο CERN και αιτία για την ανακάλυψη αυτή ήταν η δυσκολία που αντιμετώπιζαν οι επιστήμονες για την ανταλλαγή δεδομένων και αποτελεσμάτων. Ο Berners-Lee επομένως κατανόησε την ανάγκη αυτή αλλά και συνειδητοποίησε τις δυνατότητες που θα προέκυπταν αν εκατομμύρια υπολογιστές ήταν συνδεδεμένοι μεταξύ τους μέσω διαδικτύου.

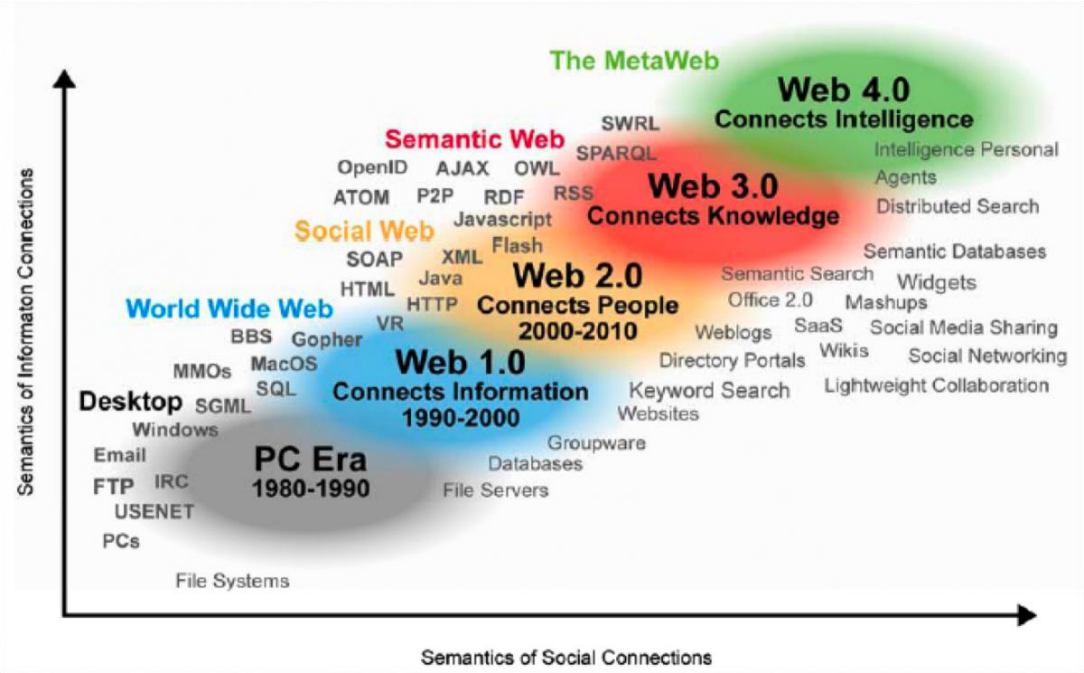
Ο Berners-Lee κατέθεσε την πρότασή του για τον παγκόσμιο ιστό στο CERN. Η πρότασή του αρχικά απορρίφθηκε. Παρόλα αυτά, ο Berners-Lee επέμεινε και τον Οκτώβριο του 1990 όρισε λεπτομερώς τις τρείς θεμελιώδεις τεχνολογίες οι οποίες και έγιναν η βάση του σημερινού Ιστού:

- HTML: HyperText Markup Language (Γλώσσα σήμανσης Υπερκειμένου). Η μορφή στην οποία δημοσιεύεται ο Ιστός συμπεριλαμβανομένης της δυνατότητας επεξεργασίας εγγράφων και της σύνδεσης με άλλα έγγραφα και πόρους.
- URI: Uniform Resource Identifier. Ένα είδος «διεύθυνσης» που είναι μοναδική για κάθε πόρο στον Ιστό
- HTTP: Hypertext Transfer Protocol (Πρωτόκολλο Μεταφοράς Υπερκειμένου). Επιτρέπει την ανάκτηση συνδεδεμένων πόρων από τον παγκόσμιο Ιστό.

Η βασική ιδέα ήταν ότι μία εφαρμογή του πελάτη που ονομάζεται web browser (φυλλομετρητής) παρέχει πρόσβαση σε ένα έγγραφο που είναι αποθηκευμένο σε έναν άλλο υπολογιστή. Αυτό γίνεται στέλνοντας ένα μήνυμα μέσω του Internet στην εφαρμογή ενός web server ο οποίος στέλνει πίσω τον κώδικα του εγγράφου. Έγγραφα ή ιστοσελίδες γράφονται σε Hypertext Markup Language (HTML), που επιτρέπει στις γέφυρες να χαρακτηρίζονται hyperlinks για ένα έγγραφο σε μία συγκεκριμένη τοποθεσία στο web, που παίρνει το όνομά της χρησιμοποιώντας ένα Universal Resource Locator (URL). Όταν ο χρήστης κάνει κλικ σε ένα hyperlink, ο browser βρίσκει την διεύθυνση IP που σχετίζεται με το URL και στέλνει ένα μήνυμα σε αυτή τη διεύθυνση IP ζητώντας το HTML αρχείο από την συγκεκριμένη τοποθεσία σύμφωνα με το σύστημα αποθήκευσης του server. Μόλις ληφθεί, το αρχείο αυτό παρουσιάζεται στον browser.

Έτσι το διαδίκτυο άλλαξε τον κόσμο και έγινε το πιο ισχυρό μέσο επικοινωνίας. Παρόλο που σήμερα χρησιμοποιείται μόνο από το 39% των ανθρώπων, έχει αλλάξει τον τρόπο που διδάσκουμε και μαθαίνουμε, αγοράζουμε και πουλάμε, ενημερώνουμε και ενημερωνόμαστε, συμφωνούμε και διαφωνούμε, συνεργαζόμαστε και λύνουμε ένα μεγάλο εύρος προβλημάτων.

2.4 Στάδια ανάπτυξης του WWW



Web 1.0 (στατικό)

Το 1993 ήταν μία καταλυτική χρονιά για το WWW με την εισαγωγή του Mosaic web browser, ο οποίος μπορούσε να παρουσιάσει όχι μόνο κείμενο αλλά και γραφικά. Από εκείνο το σημείο η χρήση του web αυξήθηκε ραγδαία, παρόλο που οι περισσότεροι χρήστες λειτουργούσαν μόνο σαν καταναλωτές περιεχομένου και όχι σαν παραγωγοί. Σε αυτή την πρώιμη περίοδο ανάπτυξης του web οι ιστοσελίδες ήταν κυρίως στατικά έγγραφα που διαβάζονταν από τον server και παρουσιάζονταν στους clients χωρίς οι χρήστες να έχουν τη δυνατότητα να συμβάλλουν στο περιεχόμενο και χωρίς το περιεχόμενο να μπορεί να προσαρμοστεί στις συγκεκριμένες ανάγκες του χρήστη.

Web 2.0 (δυναμικό)

Γύρω στο 2000 άρχισε η δεύτερη φάση ανάπτυξης του web με την αυξανόμενη χρήση τεχνολογιών που επέτρεπαν στον χρήστη του browser να αλληλεπιδράσει με τις ιστοσελίδες και να επεξεργαστεί το περιεχόμενό τους. Υπάρχουν δύο τρόποι που μπορεί να γίνει κάτι τέτοιο, client-side scripting, και server-side scripting.

Το client-side scripting κατορθώνεται μέσω κώδικα προγραμματισμού που είναι ενσωματωμένος στην HTML και συνήθως γράφεται σε JavaScript. Αυτός ο κώδικας, τρέχει στον υπολογιστή του χρήστη χωρίς να υπάρχει ανάγκη να σταλούν επιπλέον μηνύματα στον server εξού και “client- side”.

Το server-side scripting κατορθώνεται μέσα από μηνύματα προς τον server που παραπέμπουν σε εφαρμογές ικανές να δημιουργήσουν το HTML δυναμικά: το έγγραφο που τελικά παρουσιάζεται στον χρήστη είναι προσαρμοσμένο στα αιτήματά του και όχι ένα έγγραφο που ανακτήθηκε από ένα αποθηκευμένο αρχείο.

Social web

Οι τεχνολογίες του web 2.0 βοήθησαν στην δημιουργία ενός μεγάλου εύρους κοινωνικών ιστοσελίδων που είναι πλέον γνωστές σε όλους συμπεριλαμβανομένων των chat rooms, blogs, wikis, σχολιασμούς προϊόντων και e-markets. Τώρα πια ο χρήστης έχει τη δυνατότητα να προσθέσει πληροφορίες σε μία ιστοσελίδα και με αυτόν τον τρόπο να επικοινωνεί όχι μόνο με τον server αλλά μέσω του server και με άλλους χρήστες.

Web 3.0 (σημασιολογικό)

Κατά την δεκαετία του 1990 ο Berners- Lee και οι συνεργάτες του ανέπτυξαν προτάσεις για ένα ακόμη στάδιο της ανάπτυξης του web που είναι γνωστό ως Semantic web (σημασιολογικός ιστός). Αυτή η μακρινή ιδέα εκδόθηκε για πρώτη φορά σε άρθρο το 2001 στο Scientific American και σε έναν βαθμό εφαρμόζεται στο σημερινό στάδιο της ανάπτυξης του web που ονομάζεται αλλιώς web 3.0. Προς το παρόν δεν μπορούμε να δούμε ξεκάθαρα τι βρίσκεται πέρα από το web 3.0 υπάρχει όμως ένας πολύ χαλαρός ορισμός του επόμενου σταδίου, web 4.0, που φαίνεται στην εικόνα που προηγήθηκε.

Στο άρθρο τους το 2001 ο Berners- Lee και οι υπόλοιποι συγγραφείς, παρατήρησαν ότι στο υπάρχον web το περιεχόμενο μπορεί να χρησιμοποιηθεί από τους ανθρώπους αλλά όχι από τις εφαρμογές υπολογιστών. Υπάρχουν πολλές εφαρμογές υπολογιστών για εργασίες όπως ο σχεδιασμός, ο προγραμματισμός ή η ανάλυση αλλά δουλεύουν μόνο για δεδομένα που βρίσκονται σε μία συγκεκριμένη λογική μορφή και όχι για πληροφορίες που παρουσιάζονται σε κείμενο φυσικής γλώσσας. Ένας άνθρωπος μπορεί να σχεδιάσει ένα δρομολόγιο κοιτάζοντας τις ιστοσελίδες που έχουν τις ώρες των πτήσεων και την τοποθεσία των ξενοδοχείων αλλά τα προγράμματα δεν ήταν δυνατό να εξάγουν αξιόπιστα αυτές τις πληροφορίες από ιστοσελίδες που βασίζονταν σε κείμενο. Ο αρχικός σκοπός του σημασιολογικού ιστού είναι να παρέχει πρότυπα μέσω των οποίων οι άνθρωποι θα μπορούν να δημοσιεύουν έγγραφα που περιέχουν δεδομένα επιτρέποντας στα προγράμματα να συνδυάζουν δεδομένα από πολλές βάσεις δεδομένων, όπως ακριβώς ένας άνθρωπος μπορεί να συνδυάσει πληροφορίες από πολλά κείμενα προκειμένου να λύσει ένα πρόβλημα ή να εκτελέσει μία λειτουργία.

2.5 Ιστορία web API

Από τεχνολογική σκοπιά, τα APIs είναι ένα μέρος από αυτό που οι επιχειρήσεις θα ανέφεραν ως Service Oriented Architecture (SOA). Τα APIs δεν είναι κάτι καινούριο. Αλλά γύρω στο 2000 ένα μέρος των πειραματισμών πάνω στα SOP άφησαν τις επιχειρήσεις και βρήκαν πιο γόνιμο έδαφος στον κόσμο των start-ups.

Σήμερα, το 2014 υπάρχουν εμφανείς τεχνικοί λόγοι για τους οποίους τα web APIs βρίσκουν επιτυχία σε επιχειρήσεις όλων των ειδών και μεγεθών ακόμα και μέσα στην κυβέρνηση. Αλλά δεν είναι όλοι οι λόγοι για την επιτυχία τους τεχνικοί. Υπάρχουν πολλές άλλες, λιγότερο προφανείς, πτυχές των web APIs που έχουν συμβάλλει καθοριστικά στην επιτυχία τους. Τους λόγους αυτούς μπορούμε να μάθουμε μελετώντας προσεκτικά το παρελθόν και κατανοώντας γιατί μερικές καινοτομίες πάνω στα web APIs ήταν επιτυχημένες.

Το 2014, πρέπει να είμαστε σίγουροι ότι χρησιμοποιούμε τις καλύτερες πρακτικές από αυτές που έχουν εμφανιστεί τα τελευταία 14 χρόνια αλλά και να μην αγνοήσουμε μερικές κρίσιμες παραμέτρους που συνέβαλλαν στην επιτυχία ορισμένων εκ των πρώτων παρόχων APIs όπως η Amazon, η Salesforce, το Ebay και το Twitter – πολλά από τα οποία λειτουργούν ακόμη.

Ανατρέχοντας στο παρελθόν μπορούμε να αρχίσουμε να παρατηρούμε πρότυπα που καθόρισαν την βιομηχανία. Πρότυπα που πρέπει να χρησιμοποιηθούν αλλά και αυτά που πρέπει να αποφευχθούν.

2.5.1 Εμπόριο

Μόλις οι πρώτες σελίδες .COM εμφανίστηκαν, οι πλατφόρμες αναζητούσαν νέους τρόπους να συνδέουν τα προϊόντα με τις ιστοσελίδες ηλεκτρονικού εμπορίου και τα web APIs τα οποία βασίστηκαν πάνω στην ήδη υπάρχουσα υποδομή HTTP αποδείχθηκαν το σωστό εργαλείο για αυτό τον σκοπό.

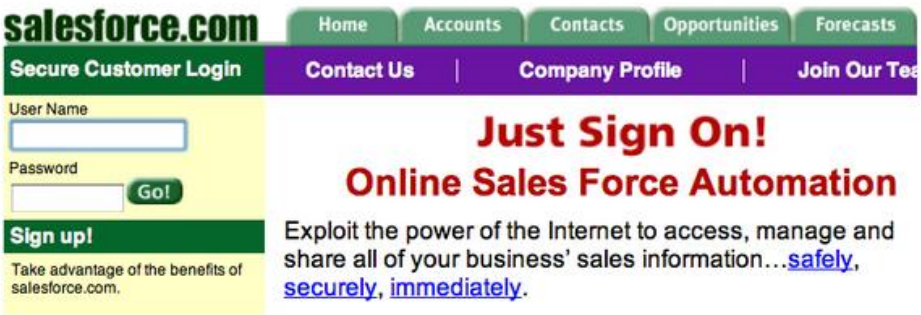
Έχοντας αυτό στο μυαλό, ένας αριθμός καινοτόμων τεχνικών καθόρισε τις πρώτες χρήσεις των APIs για πωλήσεις και διαχείριση εμπορίου ξεκινώντας μία δεκαετή εξέλιξη που τώρα καλούμε ως πρώιμη ιστορία των web APIs.

Τα APIs παρά την αρχική τους επιτυχία, δεν θα έφταναν στην ωριμότητα χωρίς επιπλέον κρίσιμα στοιχεία που θα αποδεικνύονταν εξίσου σημαντικά με στοιχεία του εμπορίου όπως τα προϊόντα, οι πληρωμές και οι πωλήσεις.

SalesForce

Στις 7 Φεβρουαρίου 2000 το Salesforce.com εμφανίστηκε επισήμως στο συνέδριο IDG Demo 2000.

Το Salesforce.com παρουσίασε τον βασισμένο στο διαδίκτυο αυτοματισμό πωλήσεων ως «διαδικτυακή υπηρεσία». Τα APIs γραμμένα σε XML ήταν μέρος του Salesforce.com από την πρώτη κιόλας μέρα. Το Salesforce.com αναγνώρισε ότι οι πελάτες επιθυμούν να μοιράζονται δεδομένα ανάμεσα σε εφαρμογές διαφορετικών επιχειρήσεων και τα APIs ήταν ο τρόπος για την υλοποίηση αυτής της ανάγκης.



Ο Marc R. Benioff, ιδρυτής και πρόεδρος του Salesforce.com ανέφερε: «Το Salesforce.com είναι η πρώτη λύση που πραγματικά αξιοποιεί το διαδίκτυο και προσφέρει την λειτουργικότητα του λογισμικού που διαθέτουν μεγάλες επιχειρήσεις με μικρότερο κόστος».

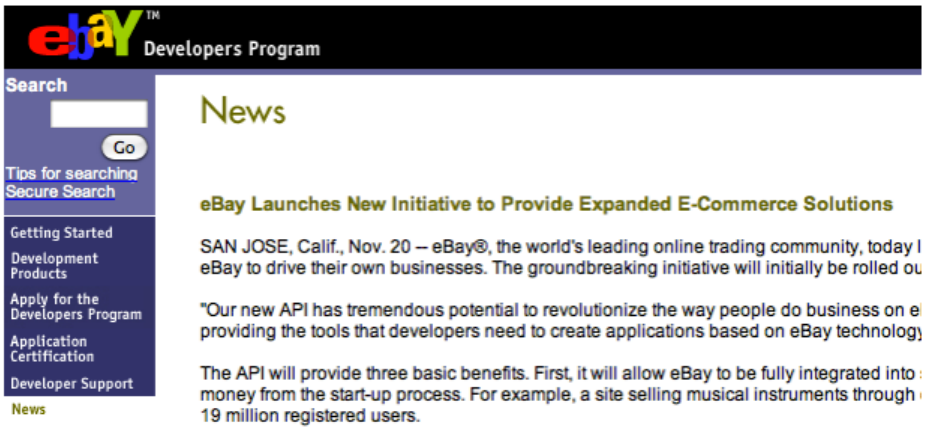
Το Salesforce.com ήταν ο πρώτος πάροχος που χρησιμοποιώντας web εφαρμογές και APIs κατάφερε να προσφέρει αυτό που σήμερα ονομάζουμε Λογισμικό ως Υπηρεσία (Software-as-a-Service).

Ebay

Στις 20 Νοεμβρίου 2000 το eBay παρουσίασε το eBay Application Program Interface (API) μαζί με το αντίστοιχο πρόγραμμα για τους προγραμματιστές (eBay Developers Program).

Το eBay API αρχικά διανεμήθηκε σε έναν περιορισμένο αριθμό προγραμματιστών και συνεταίρων του eBay.

Όπως αναφέρεται στο eBay, «το νέο μας API έχει τη δυνατότητα να φέρει επανάσταση στον τρόπο που οι άνθρωποι συναλλάσσονται στο eBay και να αυξήσει την ποσότητα των συναλλαγών στην ιστοσελίδα, κάνοντας διαθέσιμα τα εργαλεία που οι προγραμματιστές χρειάζονται για να δημιουργήσουν εφαρμογές βασισμένες στην τεχνολογία του eBay. Πιστεύουμε ότι το eBay τελικά θα ενσωματωθεί σε πολλές υπάρχουσες ιστοσελίδες όπως και σε μελλοντικές επιχειρήσεις ηλεκτρονικού εμπορίου».



Το API του eBay ήταν μία απάντηση στον αυξανόμενο αριθμό εφαρμογών που ήδη βασίζονταν στην ιστοσελίδα του νόμιμα ή παράνομα. Το API είχε στόχο να τυποποιήσει το πώς οι εφαρμογές ενοποιούνταν με το eBay και να κάνει ευκολότερο σε συνεργάτες και προγραμματιστές να δημιουργήσουν μία επιχείρηση γύρω από το eBay.

Το eBay θεωρείται ο ηγέτης στην σύγχρονη περίοδο των web APIs και των web υπηρεσιών και διαθέτει ένα από τα πιο επιτυχημένα συστήματα για προγραμματιστές.

2.5.2 Κοινωνικά

Καθώς οι εμπορικές πλατφόρμες που βασίζονταν σε APIs εξακολουθούσαν να εξελίσσονται και να κατανοούν τους καλύτερους τρόπους για να χρησιμοποιούν APIs, ένα νέο είδος πλατφόρμας εμφανίστηκε στον τομέα περιεχομένου και επικοινωνίας μέσω διαδικτύου. Η πλατφόρμα αυτή είχε στο κέντρο της τον χρήστη και έναν πολύ κοινωνικό χαρακτήρα.

Η δημοσίευση περιεχομένου που παράγεται από τους χρήστες, η ανταλλαγή web links, φωτογραφιών και άλλων στοιχείων μέσω APIs εμφανίστηκε με τη δημιουργία νέων κοινωνικών δικτύων ανάμεσα στο 2003 και το 2006. Αυτή ήταν μια νέα εποχή για τα APIs που δεν είχε να κάνει με χρήματα αλλά με συνδέσεις.

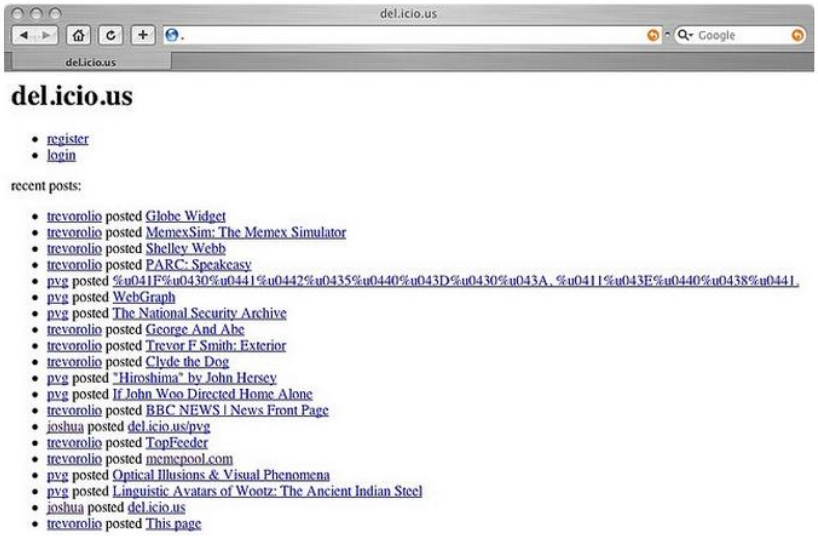
Αυτά τα νέα κοινωνικά δίκτυα που βασίζονταν σε APIs θα έφταναν την τεχνολογία σε νέα υψηλά επίπεδα και θα διαβεβαίωναν ότι οι νέες εφαρμογές από εδώ και στο εξής θα έχουν κοινωνικά χαρακτηριστικά που θα καθορίζονται από τα APIs τους.

Ο κοινωνικός χαρακτήρας ήταν ένα σημαντικό συστατικό που έλειπε από την βιομηχανία των APIs.

Del.icio.us

Το Del.icio.us είναι μία υπηρεσία για την αποθήκευση, την ανταλλαγή και την εύρεση ηλεκτρονικών σελιδοδεικτών για ιστοσελίδες που ιδρύθηκε από την Jousha Schachter το 2003.

Το del.icio.us εφάρμοζε ένα πολύ απλό σύστημα ετικετών που επέτρεπε στους χρήστες να οργανώνουν τους σελιδοδείκτες τους με έναν λογικό τρόπο, αλλά ταυτόχρονα εδραίωσε ένα είδος ομοιομορφίας σε αυτή τη διαδικασία ανάμεσα στους χρήστες της πλατφόρμας. Αυτό αποδείχθηκε ένα πολύ ισχυρό μέσο για την σύνταξη καταλόγων και την ανταλλαγή web links.



Η καινοτόμα μέθοδος δημιουργίας ετικετών επέτρεπε να ανακτήσει κανείς μία λίστα σελιδοδεικτών χρησιμοποιώντας το URL http://del.icio.us/tag/[tagname]/ . Επομένως αν

κάποιος έψαχνε για σελιδοδείκτες σχετικούς με αεροπλάνα θα μπορούσε γράφοντας http://del.icio.us/tag/airplane/ να πάρει (GET) τη λίστα με όλους τους σελιδοδείκτες με την ετικέτα αεροπλάνα. Ήταν μία απλή διαδικασία.

Σχετικά με την προγραμματιστική διεπαφή, το API ήταν ενσωματωμένο μέσα στην ιστοσελίδα δημιουργώντας μία «αόρατη» εμπειρία για τον χρήστη. Αν κάποιος ήθελε την ετικέτα airplane μέσω HTML έγραφε http://del.icio.us/tag/airplane/, αν ήθελε RSS έγραφε http://del.icio.us/rss/tag/airplane/ και αν ήθελε XML μορφή έγραφε http://del.icio.us/api/tag/airplane/. Αυτό όμως έχει αλλάξει στη νέα μορφή του Delicious API.

Το del.icio.us ήταν το πρώτο παράδειγμα του πώς το διαδίκτυο μπορεί να φέρει περιεχόμενο σε HTML ή ταυτόχρονα σε RSS και XML χρησιμοποιώντας μία δομή για URL που ήταν απλή και κατανοητή από τον άνθρωπο. Αυτή η προσέγγιση στην ανταλλαγή σελιδοδεικτών έθεσε τη βάση για μελλοντικά APIs, εύκολα στην κατανόηση για τους προγραμματιστές αλλά και για τους απλούς χρήστες. Οποιοσδήποτε λίγο εξοικειωμένος χρήστης μπορούσε εύκολα να αναλύσει το RSS ή το XML και να αναπτύξει εφαρμογές και widgets σχετικές με το περιεχόμενο του del.icio.us.

Το del.icio.us έχει πουληθεί δύο φορές λόγω της δημοτικότητάς του. Αρχικά στην Yahoo! το 2005 και στη συνέχεια στο AVOS Systems το 2011. Ήταν μία από τις βασικές πλατφόρμες που εισήλθαν στην κοινωνική εποχή των APIs και εδραίωσαν την ανταλλαγή μέσω APIs ως σημαντικό στοιχείο τους. Τέλος μας έδειξε ότι το απλό πρέπει να κυριαρχεί στον σχεδιασμό των APIs.

Flickr

Τον Φεβρουάριο του 2004 εμφανίστηκε μία νέα δημοφιλής ιστοσελίδα για ανταλλαγή φωτογραφιών, το Flickr. Έξι μήνες αργότερα παρουσιάστηκε το κακόφημο πλέον API του και άλλους έξι μήνες μετά αγοράστηκε από την Yahoo!

Το Flickr είχε σχεδιαστεί αρχικά ως online παιχνίδι αλλά γρήγορα εξελίχθηκε σε μία κοινωνική μορφή ανταλλαγής φωτογραφιών.

Η εμφάνιση των RESTful APIs βοήθησε το Flickr στο να γίνει γρήγορα η πλατφόρμα από την οποία επέλεγαν εικόνες τα πρώτα blogs και τα κοινωνικά δίκτυα, επιτρέποντας στους χρήστες να τις ενσωματώνουν εύκολα.

Το Flickr API είναι η βάση πίσω από το BizDev 2.0, έναν όρο που επινοήθηκε από την Caterina Fake, συνιδρυτή του Flickr. Το Flickr δεν μπορούσε να ανταπεξέλθει στη ζήτηση των υπηρεσιών του και έτσι δημιούργησε το API σαν ένα μέσο self-service για να ανταποκριθεί στην επιχειρηματική ανάπτυξη.

Οι βασικές έννοιες που εδραιώθηκαν από το Flickr μέσω του API του προώθησαν την εταιρία και οδήγησαν στην εξαγορά της από την Yahoo!. Η επιχειρηματική ανάπτυξη χρησιμοποιώντας API έχει ενσωματωθεί στην φιλοσοφία των επιχειρήσεων μετατρέποντάς τα σε κάτι όχι μόνο τεχνικό. Τα APIs έγιναν το μέσο που κάθε εταιρία θα μπορούσε να χρησιμοποιήσει ώστε να εδραιώσει συναλλαγές με τους συνεργάτες της και το κοινό. Υπήρχε όμως ακόμα μέλλον μέχρι την ωρίμανση τους.

Facebook

Στις 15 Αυγούστου 2006 το Facebook παρουσίασε την πολυαναμενόμενη πλατφόρμα για προγραμματιστές και το API του. Η έκδοση 1.0 του Facebook Development Platform έδωσε την δυνατότητα στους προγραμματιστές να έχουν πρόσβαση σε φίλους, φωτογραφίες, εκδηλώσεις και πληροφορίες προφίλ από το Facebook.

Το API χρησιμοποιούσε REST και ο κώδικας ήταν διαθέσιμος σε μορφή XML ακολουθώντας κοινές προσεγγίσεις με άλλους παρόχους κοινωνικών API της εποχής.

Σχεδόν αμέσως οι προγραμματιστές άρχισαν να δημιουργούν κοινωνικές εφαρμογές και παιχνίδια με τα νέα εργαλεία που είχαν στη διάθεσή τους.

Το Facebook Development Platform έδωσε στο Facebook πλεονέκτημα απέναντι στον δημοφιλή ανταγωνιστή του, το MySpace, και το εδραίωσε ως την κορυφαία πλατφόρμα κοινωνικών παιχνιδιών με παιχνίδια όπως το Farmville.

Αν και το API όπως και η πλατφόρμα του Facebook θεωρούνται από πολλούς προγραμματιστές ασταθή, συνεχίζουν να διαδραματίζουν έναν σημαντικό ρόλο στην εξέλιξη της πλατφόρμας με εφαρμογές και συνεργασίες που εισάγουν νέα χαρακτηριστικά και προσφέρουν νέες εμπειρίες στο Facebook.

Twitter

Στις 20 Σεπτεμβρίου του 2006 το Twitter μας σύστησε το Twitter API.

Όμοια με τη δημιουργία του eBay API, το Twitter API δημιουργήθηκε για να ανταποκριθεί στην αυξανόμενη χρήση του Twitter από αυτούς που εκμεταλλεύονταν την ιστοσελίδα και δημιουργούσαν πλαστά APIs. Το Twitter έδωσε το Twitter API μέσα από μία REST διεπαφή

χρησιμοποιώντας JSON και XML.

Αρχικά, το Twitter χρησιμοποιούσε Basic Auth για την πιστοποίηση του API του κάτι που μετά από τέσσερα περίπου χρόνια κατέληξε στο κακόφημο πια Twitter OAuth Apocalypse. Τότε το Twitter ανάγκασε όλους όσους χρησιμοποιούσαν το API να στραφούν σε OAuth.

Σε τέσσερα χρόνια το API του Twitter είχε γίνει το κέντρο αμέτρητων πελατών desktop, εφαρμογών για κινητά, web εφαρμογών και επιχειρήσεων – ακόμα και από το ίδιο το Twitter στο iPhone, iPad, και στις Android εφαρμογές μέσω της ιστοσελίδας του.

Το Twitter είναι μία από τις πιο σημαντικές πλατφόρμες API που είναι διαθέσιμες, δείχνοντας τι είναι δυνατό να συμβεί όταν μία υπερβολικά απλή πλατφόρμα κάνει ένα πράγμα καλά, στη συνέχεια επιτρέψει την πρόσβαση σε αυτή μέσω API και αφήσει ένα σύστημα από APIs να αναπτυχθεί γύρω της.

2.5.3 Επιχειρήσεις και marketing

Καθώς τα APIs εξελίχθηκαν από εμπορικά σε κοινωνικά ήταν ξεκάθαρο ότι η βιομηχανία θα χρειαζόταν κάποια προτυποποίηση, εισάγοντας ορισμένες κοινές επιχειρηματικές πρακτικές. Η βιομηχανία χρειαζόταν να προτυποποιήσει πως τα APIs αναπτύσσονται όπως και να παράσχει μία προώθηση προϊόντων ώστε να κάνει γνωστές τις δυνατότητες των APIs και των κοινών επιχειρηματικών πρακτικών.

Η εδραίωση των κοινών επιχειρηματικών και προωθητικών τεχνικών χρειάστηκε μεγάλη προσπάθεια εκ μέρους των APIs. Υπάρχουν δύο διαφορετικά καινοτόμα APIs ανάμεσα στο 2005 και το 2012 που ξεχώρισαν και όρισαν την βιομηχανία των API που γνωρίζουμε σήμερα.

ProgrammableWeb

Τον Ιούλιο του 2005 ο John Musser ίδρυσε το ProgrammableWeb, ίσως τον σημαντικότερο παράγοντα της ιστορίας των APIs . Σύμφωνα με την περιγραφή που δινόταν στην ιστοσελίδα: «το ProgrammableWeb είναι μία ιστοσελίδα -πλατφόρμα αναφοράς και blog, το οποίο παρέχει νέα, πληροφορίες και πόρους για την ανάπτυξη εφαρμογών χρησιμοποιώντας το Web 2.0 APIs. Ξεκίνησα αυτή την ιστοσελίδα επειδή δεν μπορούσα να βρω αυτό που έψαχνα. Ένα σημείο εκκίνησης βασισμένο στην τεχνολογία για την ανάπτυξη ηλεκτρονικών πλατφόρμων. Αν και δεν υπάρχουν εγγυήσεις επιτυχίας, την τελευταία φορά που ξεκίνησα μία ιστοσελίδα αναφοράς έγινε η πιο δημοφιλής του Google στον συγκεκριμένο τομέα. Δεδομένου ότι η νέα ιστοσελίδα θα είναι μία συλλογική προσπάθεια, με περιεχόμενο και από το κοινό, ίσως αυτό να την κάνει δημοφιλή.» John Musser

Το αρχικό post του Musser σχετικά με το γιατί ξεκίνησε το ProgrammableWeb είναι χαρακτηριστικό: “Γιατί; Επειδή το να πάμε από μία ιστοσελίδα σε μία ηλεκτρονική πλατφόρμα είναι σημαντικό ζήτημα.”

Τα web APIs είναι σημαντικό ζήτημα. Είτε έχει να κάνει με κοινωνικά δίκτυα, με διακυβέρνηση, με το σύστημα υγείας είτε με την εκπαίδευση, το να υπάρχει μια πλατφόρμα που να διαθέτει πόρους και δεδομένα θα είναι καθοριστικό στο πως το εμπόριο και η κοινωνία θα λειτουργούν από εδώ και στο εξής.

Ο Musser πήρε μία απόφαση να δείξει RESTful προσεγγίσεις στην δημιουργία APIs σε αντίθεση με τις προσπάθειες που γίνονταν για Service Oriented Architecture (SOA) και web υπηρεσίες πολύ πριν το REST φτάσει ως την Silicon Valley.

Καθώς διασχίζουμε το 2014, μία χρονιά στην οποία τα APIs έχουν γίνει αρκετά συνηθισμένα, οφείλουμε πολλά στο ProgrammableWeb καθώς τα όσα ανέφεραν οι ιδρυτές του ήταν καίρια στο να ποσοτικοποιήσουμε και να ορίσουμε την βιομηχανία των APIs.

Mashery

Τον Νοέμβριο του 2006 το Mashery, ο πρώτος πάροχος υπηρεσιών σχετικών με API έκανε την εμφάνισή του και παρείχε υποστήριξη μέσω εγγράφων και έλεγχο πρόσβασης για επιχειρήσεις που ήθελαν να προσφέρουν ανοιχτά ή ιδιωτικά APIs.

Αυτή τη χρονική στιγμή, το 2006, τα APIs μεταφέρονται από τα κοινωνικά δίκτυα σε προγραμματισμό σε cloud με την δημιουργία του Amazon Web Services. Ήταν ξεκάθαρο ότι ο κόσμος των web APIs γινόταν πραγματικός και δίνονταν ευκαιρίες σε επιχειρήσεις να προσφέρουν την διαχείριση APIs ως υπηρεσία.

Ενώ υπήρχαν εργαλεία για την ανάπτυξη APIs, δεν υπήρχε σταθερή προσέγγιση για την διαχείριση της ανάπτυξής τους. Το Mashery ήταν το πρώτο που παρείχε ένα συγκεκριμένο σύνολο υπηρεσιών στους παρόχους των APIs, που θα έθετε τη βάση για την περαιτέρω ανάπτυξη της βιομηχανίας των APIs.

Χρειάστηκαν σχεδόν έξι ακόμη χρόνια μέχρι η βιομηχανία των APIs να είναι έτοιμη να δεχθεί αυτό το οποίο το Mashery μπορούσε να της προσφέρει.

Η κατάσταση των APIs που επικρατεί σήμερα οφείλεται σε καινοτομίες πάνω στο εμπόριο όπως το SalesForce και το Amazon, σε κοινωνικές καινοτομίες όπως το Flickr και το delicious αλλά και στο Mashery το οποίο όρισε αυτό που σήμερα ονομάζουμε επιχειρήσεις των APIs.

Το 2013 το Mashery αγοράστηκε από την Intel.

2.5.4 Χαρτογράφηση

Ένας άλλος καινοτόμος των APIs είδε την ανάγκη να παρέχει στους προγραμματιστές web έναν απλό χάρτη σε JavaScript που θα μπορούσε να βοηθήσει στην online πλοήγηση, στην τοποθέτηση περιεχομένου αλλά και στην πλοήγηση στον πραγματικό κόσμο.

Οι προγραμματιστές είδαν αμέσως την δυνατότητα που θα παρείχαν οι ενσωματωμένοι χάρτες και βρήκαν τρόπους ώστε να χρησιμοποιήσουν τους διαθέσιμους πόρους για να δημιουργήσουν επιθυμητά χαρακτηριστικά για τους χρήστες, επικεντρωμένα στην επίλυση προβλημάτων που αντιμετωπίζουμε καθημερινά.

Αυτή η πρώιμη χρήση των APIs στο να παρέχουν εργαλεία χαρτογράφησης αλλά και υπηρεσίες σε προγραμματιστές έθεσαν τη βάση για την ερχόμενη περίοδο των APIs στα κινητά.

Google Maps

Στις 29 Ιουνίου 2006 η Google παρουσίασε το Google Maps API που επέτρεπε στους προγραμματιστές να εισάγουν τους χάρτες της Google στην ιστοσελίδα τους χρησιμοποιώντας JavaScript.

Το API εμφανίστηκε δειλά έξι μήνες μετά από την κυκλοφορία του Google Maps ως εφαρμογή και ήταν μία απάντηση στις εφαρμογές που αναπτύσσονταν και «έκλεβαν» την αρχική εφαρμογή. Το Google Maps έγινε αμέσως τόσο δημοφιλές που προγραμματιστές έσπασαν το JavaScript και δημιούργησαν εφαρμογές όπως το housingmaps.com και το chicagocime.org. Η ζήτηση για πληροφορίες σχετικά με το «σπάσιμο» του Google Maps ήταν τόσο μεγάλη που εκδόθηκαν βιβλία όπως το Mapping Hacks και το Google Hacks.

Το API του Google Maps ξεκίνησε μία προσπάθεια για την ένωση διαφορετικών APIs μέσω των δεδομένων τοποθεσίας που κατείχε και ως σήμερα έχουν δημιουργηθεί πάνω από 2000 τέτοιες ενώσεις.

Το API παρουσιάζει την μεγάλη αξία των γεωγραφικών δεδομένων αλλά και των APIs που σχετίζονται με χαρτογράφηση καθώς και τη δύναμη των χρηστών στο να επηρεάσουν τον τρόπο χρήσης ενός API. Ο Lars Rasmussen, ο αρχικός προγραμματιστής του Google Maps σχολίασε πόσα έμαθε από τους υπόλοιπους προγραμματιστές βλέποντας πώς «έσπαγαν» την εφαρμογή σε πραγματικό χρόνο και πως χρησιμοποιούσαν όσα μάθαιναν και τα εφάρμοζαν στα APIs που γνωρίζουμε σήμερα.

Λίγες εταιρίες έχουν τους πόρους να αντιμετωπίσουν ένα πρόβλημα όπως η χαρτογράφηση των πόρων της γης και να δημιουργήσουν ένα επαναχρησιμοποιήσιμο API όπως έκανε η Google. Η Google έχει διαδραματίσει πολλούς ρόλους στο να προχωρήσει ο χώρος των APIs όμως το Google Maps έπαιξε έναν καθοριστικό ρόλο σε ολόκληρη την ιστορία των APIs.

2.5.5 Cloud computing

Καθώς τα APIs διέδιδαν τον κοινωνικό τους χαρακτήρα η Amazon είδε την δυναμική μιας RESTful προσέγγισης στο business και εσωτερίκευσε και είδε τα APIs με έναν τρόπο που κανείς μέχρι τότε δεν είχε δει. Δημιούργησε μια προσέγγιση στην χρήση των APIs που ήταν πολύ πάνω από το ηλεκτρονικό εμπόριο και άλλαξε τον τρόπο με τον οποίο προγραμματίζουμε.

Η Amazon εξέλιξε τον τρόπο με τον οποίο δημιουργούμε εφαρμογές στο διαδίκτυο με το να θέσει τα APIs σε λειτουργία. Αυτό που τώρα ονομάζουμε cloud computing άλλαξε τα πάντα και έκανε δυνατή την επικράτηση των APIs σε κινητά, tablets και αισθητήρες.

Το cloud computing ήταν ένα βασικό συστατικό που έλειπε από την βιομηχανία των APIs.

Amazon S3

Τον Μάρτιο του 2006 η Amazon παρουσίασε μία νέα υπηρεσία που ήταν εντελώς διαφορετική από την ιστοσελίδα του βιβλιοπωλείου της Amazon που γνωρίζαμε. Ήταν μία νέα προσπάθεια της Amazon: μία υπηρεσία που παρείχε αποθηκευτικό χώρο στο διαδίκτυο και ονομαζόταν Amazon S3.

Το Amazon S3 παρείχε μία απλή διεπαφή που μπορεί να χρησιμοποιηθεί για την αποθήκευση και την ανάκτηση οποιασδήποτε ποσότητας δεδομένων, οποτεδήποτε και από οπουδήποτε στο διαδίκτυο. Έδινε στους προγραμματιστές πρόσβαση στην ίδια αξιόπιστη, γρήγορη και φθηνή υποδομή αποθήκευσης δεδομένων που η ίδια η Amazon χρησιμοποιούσε για να τρέχει τις διάφορες ιστοσελίδες της.

Το Amazon S3 ή Simple Storage Service αρχικά ήταν μόνο ένα API. Δεν διέθετε διεπαφή στο web ούτε εφαρμογή σε κινητό. Ήταν ένα απλό RESTful API που επέτρεπε εντολές PUT και GET με αντικείμενα ή αρχεία.

Οι προγραμματιστές που χρησιμοποιούσαν το Amazon S3 API χρεώνονταν 0,15$ ανά Gigabyte ανά μήνα για να αποθηκεύουν δεδομένα στο σύννεφο.

Με αυτό τον νέο τύπο API και μοντέλου πληρωμών η Amazon εισήγαγε έναν νέο τύπο προγραμματισμού που τώρα είναι γνωστός ως cloud computing. Αυτό το γεγονός σήμαινε ότι τα APIs δεν ήταν πια μόνο για δεδομένα ή απλές λειτουργίες. Τώρα μπορούν να χρησιμοποιηθούν για να παράσχουν υπολογιστική υποδομή.

Amazon EC2

Τον Αύγουστο του 2006 λίγο μετά από τότε που η Amazon είχε παρουσιάσει την νέα της υπηρεσία Amazon S3, παρουσίασε μία ακόμα υπηρεσία σχετική με cloud computing που ονομάζεται Amazon EC2 ή Elastic Compute Cloud.

Το Amazon EC2 παρέχει υπολογιστική χωρητικότητα με μεταβαλλόμενο μέγεθος στο σύννεφο, επιτρέποντας έτσι στους προγραμματιστές να χρησιμοποιούν διαφορετικά μεγέθη από εικονικούς servers μέσα στα κέντρα δεδομένων της Amazon.

Όπως ο προκάτοχός του Amazon S3, έτσι και το Amazon EC2 ήταν απλά ένα RESTful API. Η Amazon δεν παρείχε διεπαφή στο web για τα επόμενα τρία χρόνια.

Χρησιμοποιώντας το API του Amazon EC2 οι προγραμματιστές μπορούσαν να χρησιμοποιήσουν μικρούς, μεγάλους και πολύ μεγάλους servers και πλήρωναν για κάθε ώρα που ο server έτρεχε.

Το Amazon EC2 συνδυασμένο με το Amazon S3 παρείχαν την πλατφόρμα για την επόμενη γενιά προγραμματισμού έχοντας στον πυρήνα τους APIs.

2.5.6 Κινητά

Με την εισαγωγή του iPhone και των Android smart phones, τα APIs εξελίχθηκαν από το ισχυρό ηλεκτρονικό εμπόριο, τα κοινωνικά δίκτυα και τον προγραμματισμό σε σύννεφο ώστε να παρέχουν σημαντικούς πόρους στα κινητά μας τηλέφωνα, που γρήγορα έγιναν κοινότοπα σε όλο τον κόσμο.

Τα APIs κάνουν τους πόρους τροποποιήσιμους, φορητούς και κατανεμημένους, μετατρέποντάς τους σε ένα μέσο για την ανάπτυξη εφαρμογών για κινητά και tablets όλων των μεγεθών.

Μία μικρή ομάδα από πλατφόρμες βασισμένες σε APIs έχουν κερδίσει την προτίμηση των προγραμματιστών και των τελικών καταναλωτών μέσα από τις εφαρμογές που αναπτύσσουν.

Foursquare

Τον Μάρτιο του 2009 το Foursquare παρουσιάστηκε στο SXSW διαδραστικό φεστιβάλ στο Austin TX.
Το Foursquare είναι μία πλατφόρμα για κινητά η οποία βασίζεται σε τοποθεσίες κάνοντας έτσι την εξερεύνηση μιας πόλης πιο ενδιαφέρουσα. Οι χρήστες μοιράζονται το που βρίσκονται με τους φίλους τους κάνοντας check in μέσω εφαρμογής σε smart phone ή μέσω SMS και έτσι συγκεντρώνουν πόντους και κερδίζουν εικονικά εμβλήματα.

Τον Νοέμβριο του 2009 μετά από τις πρώτες επενδύσεις από επενδυτές όπως οι Union Square Ventures και O'Reilly Alpha Tech Ventures το Foursquare παρουσίασε το API του.

Την περίοδο που εμφανίστηκε το API, το Foursquare είχε ένα εντυπωσιακό σύνολο από εφαρμογές που είχαν αναπτυχθεί από συνεργάτες του συμπεριλαμβανομένης μιας εφαρμογής Android και μίας εικονικής πραγματικότητας με την Layar.

Παρά τον αυξανόμενο ανταγωνισμό από το καινοτόμο Gowalla και από σημαντικές εταιρίες όπως το Facebook και η Google, το Foursquare εμφανίστηκε ως η κύρια πλατφόρμα από κινητά για να μοιράζεται κανείς το μέρος που βρίσκεται και να κάνει check in.

Instagram

Στις 6 Οκτωβρίου 2010 το Instagram παρουσίασε την εφαρμογή του για ανταλλαγή φωτογραφιών σε iPhone. Λιγότερο από τρείς μήνες μετά είχε ένα εκατομμύριο χρήστες.

Ο Kevin Systrom, ιδρυτής του Instagram επικεντρώθηκε στη δημιουργία μιας ισχυρής αλλά απλής εφαρμογής για iPhone η οποία θα έλυνε κοινά προβλήματα που είχαν να κάνουν με την ποιότητα των φωτογραφιών που προέρχονται από κινητά αλλά και με την δυσκολία των χρηστών στην ανταλλαγή τους.

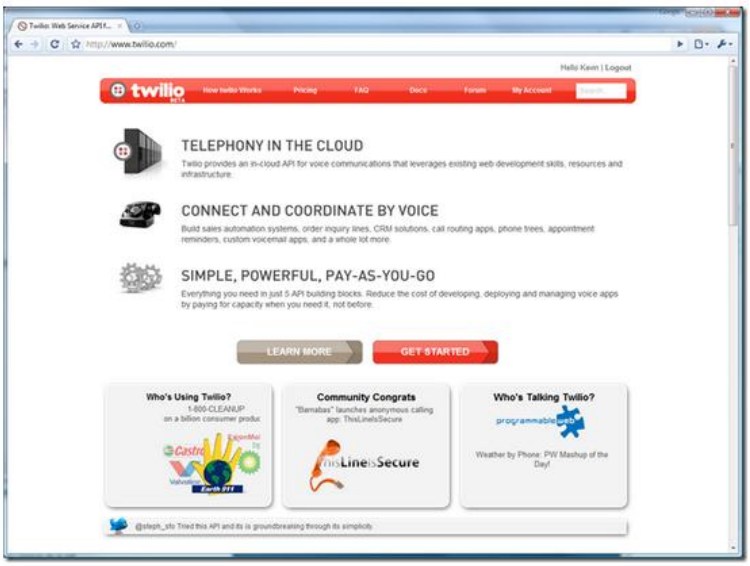
Αμέσως πολλοί χρήστες διαμαρτυρήθηκαν για την έλλειψη μίας κεντρικής ιστοσελίδας του Instagram και ενός API, με το Instagram όμως να μένει αφοσιωμένο στις ενέργειές του για την εφαρμογή του στο iPhone.

Τον Δεκέμβριο ένας προγραμματιστής με το όνομα Mislav Marohni αποφάσισε να καταλάβει πως λειτουργεί η εφαρμογή για iPhone και να φτιάξει ένα ανεπίσημο API για το Instagram. Τον Ιανουάριο, το Instagram έκλεισε αυτό το API και ανακοίνωσε ότι θα δημιουργήσει ένα δικό του το οποίο και είχε έτοιμο τον Φεβρουάριο του 2011. Μέσα σε λίγες μέρες πολλές εφαρμογές για φωτογραφίες και ιστοσελίδες ανταλλαγής φωτογραφιών άρχισαν να δημιουργούνται γύρω από το API του Instagram.

Το Instagram έγινε γρήγορα δημοφιλές ανάμεσα στα iPhone αλλά ένα API ήταν απαραίτητο για να φτάσει στο μέγιστο των δυνατοτήτων του, έτσι επέβαλλε τις πλατφόρμες ως έναν καθοριστικό παράγοντα στην «κινητή» περίοδο των APIs.

Twilio

Το 2007 μία νέα πλατφόρμα API ονομαζόμενη Twilio έκανε την εμφάνισή της και εισήγαγε φωνητικά API επιτρέποντας έτσι στους προγραμματιστές να εκτελούν και να λαμβάνουν κλήσεις μέσω οποιασδήποτε εφαρμογής cloud. Τα τελευταία τρία χρόνια το Twilio έχει δώσει και κώδικα API για ανταλλαγή SMS. Με τον τρόπο αυτό έχει γίνει ένας σημαντικός παράγοντας στην τηλεφωνία ανάμεσα στα εργαλεία των προγραμματιστών.



Το Twilio έχει βοηθήσει στο να ορίσουμε ποια τεχνικά αλλά και επιχειρηματικά στοιχεία είναι απαραίτητα για την δημιουργία μιας υγιούς πλατφόρμας API και δούλεψε σκληρά για να παρουσιάσει, υποστηρίξει και επενδύσει στους προγραμματιστές του.

Μαζί με το Foursquare και το Instagram, το Twilio καθόρισε την εξέλιξη των εφαρμογών για κινητά βοηθώντας στο να γίνουν τα API κάτι απλό και διαδεδομένο.

Μέχρι το 2011 ο τρόπος που εκτελούνταν τα APIs μέσω HTTP είχε καθοριστεί από πρώιμους καινοτόμους όπως η SalesForce και η Amazon. Όμως το Twilio έδειξε πως ωρίμασε η βιομηχανία των APIs στην περίοδο των κινητών. Βέβαια η ανάπτυξη των κινητών μέσω APIs έχει τις ρίζες της στις βάσεις που τέθηκαν από τις καινοτομίες σχετικά με το εμπόριο, τα κοινωνικά δίκτυα και το cloud.

Συνοπτικά

Μιλώντας για την ιστορία των APIs στην πραγματικότητα μιλάμε για μία περίοδο 13 ετών. Στο σύντομο αυτό χρονικό διάστημα, πολλά μπορούμε να μάθουμε από τις διάφορες καινοτόμες προσεγγίσεις των APIs. Αν δεν υπήρχαν καινοτόμες επιχειρήσεις όπως η Amazon, το Twitter και το Twilio τα APIs θα είχαν φτάσει στην ακμή τους την περίοδο του εμπορίου. Τώρα όμως έγινε κατανοητό ότι τα APIs χρειάζονται πολλά και διαφορετικά στοιχεία για να πετύχουν: εμπορικά, κοινωνικά, cloud και κινητά.

Βέβαια καθώς είναι στοιχεία επιχειρήσεων στόχος τους είναι το κέρδος. Πέρα από αυτό όμως τα APIs πρέπει να μπορούν να εφαρμοστούν σε μικρές και μεγάλες κλίμακες, να παρέχουν χρήσιμα εργαλεία, υπηρεσίες και πόρους που είναι σημαντικοί για τους τελικούς χρήστες καθώς σε αντίθετη περίπτωση δεν μπορούν να λειτουργήσουν. Τώρα βρισκόμαστε στην περίοδο που τα API ακμάζουν σε κινητές συσκευές και πρέπει να κατανοήσουμε την ιστορία τους προκειμένου να πάρουμε τις κατάλληλες αποφάσεις για το μέλλον.

3. Πρότυπα για APIs

Στο κεφάλαιο που ακολουθεί θα περιγραφούν οι τρείς κυρίαρχες προσεγγίσεις για τον σχεδιασμό API: Swagger, RAML και API Blueprint. Τα πρότυπα αυτά χρησιμοποιούνται για να ορίσουν τα APIs τα οποία είτε χρησιμοποιούν JSON είτε κάποια άλλη markdown γλώσσα. Επιτρέπουν την ποσοτικοποίηση και την περιγραφή των διεπαφών του API και παρουσιάζουν μοντέλα δεδομένων με έναν τρόπο ο οποίος διευκολύνει την επικοινωνία με άλλους προγραμματιστές, την παραγωγή APIs, την δημιουργία τεκμηρίωσης και υποδειγμάτων κώδικα. Επιπλέον παρέχουν εργαλεία τα οποία συμβάλλουν στον σχεδιασμό και την ανάπτυξη ενός API.

- **Swagger** – το Swagger είναι ένα πλήρες πλαίσιο για την περιγραφή, δημιουργία, κατανάλωση και απεικόνιση των RESTful υπηρεσιών web. Κύριος στόχος του είναι οι clients και τα συστήματα τεκμηρίωσης να ανανεώνονται στον ίδιο ρυθμό με τον server.
- **RAML** – Η RAML είναι ένας απλός και περιεκτικός τρόπος για την περιγραφή των πρακτικά RESTful APIs. Ενθαρρύνει την επαναχρησιμοποίηση, διευκολύνει την εύρεση και την ανταλλαγή μοτίβων και έχει στόχο την επικράτηση των καλύτερων πρακτικών.
- **API Blueprint** – Το API Blueprint παρέχει εξαιρετικά εργαλεία για όλο τον κύκλο ζωής του API. Κάνει τη συζήτηση σχετικά με κάποιο API εύκολη και παράγει τεκμηρίωση αυτόματα.

Δεν υπάρχει σωστό ή λάθος πρότυπο για να χρησιμοποιήσει κανείς για το API του. Στόχος αυτού του κεφαλαίου είναι να γίνουν κατανοητές οι λεπτές διαφορές που έχουν μεταξύ τους και πως το καθένα βοηθά στον ορισμό ενός API. Τα APIs είναι ιδιαίτερα ασαφή και για τον λόγο αυτό οποιαδήποτε προσπάθεια που κάνει την περιγραφή τους ευκολότερη είναι πολύ σημαντική.

3.1 SWAGGER

Το Swagger αναπτύχθηκε από την Wordnik με σκοπό την ιδιωτική του χρήση για τη δημιουργία του developer.wordnik.com. Η υλοποίηση του Swagger ξεκίνησε στις αρχές του 2010 και το πλαίσιο το οποίο κυκλοφόρησε αρχικά είναι αυτό που χρησιμοποιείται μέχρι σήμερα από τα APIs της Wordnik και προσέλκυσε εσωτερικούς και εξωτερικούς API clients.

Το Swagger είναι ένα σύνολο κανονισμών και ένα ολοκληρωμένο πλαίσιο εφαρμογής για την περιγραφή, την παραγωγή, την κατανάλωση και την απεικόνιση RESTful υπηρεσιών web. Ο πρωταρχικός στόχος του Swagger είναι να δώσει την δυνατότητα στους clients και στο σύστημα τεκμηρίωσης να ανανεώνεται με τον ίδιο ρυθμό με αυτόν του server. Η τεκμηρίωση των μεθόδων, των παραμέτρων και των μοντέλων είναι ενσωματωμένη στον κώδικα του server, επιτρέποντας στα APIs να είναι πάντα συγχρονισμένα. Επιθυμία του

Swagger είναι να κάνει την ανάπτυξη, την διαχείριση και την χρήση των APIs ιδιαίτερα εύκολη.

Στόχος του Swagger είναι να ορίσει ένα πρότυπο διεπαφής ανεξάρτητο από την γλώσσα προγραμματισμού για τα REST APIs. Αυτό θα επιτρέψει στους ανθρώπους και τους υπολογιστές να ανακαλύψουν και να κατανοήσουν τις δυνατότητες της υπηρεσίας χωρίς να έχουν πρόσβαση στον πηγαίο κώδικα (source code), την επεξήγησή του και χωρίς να αναλύουν τι γίνεται στο δίκτυο. Όταν ορίζεται σωστά ένα API μέσω Swagger, ένας καταναλωτής μπορεί να καταλάβει και να αλληλεπιδράσει με την απομακρυσμένη υπηρεσία που προσφέρει με ελάχιστη προσπάθεια. Όπως συμβαίνει με τις διεπαφές για προγραμματισμό χαμηλότερου επιπέδου έτσι και το Swagger απαλείφει την «φαντασία» για το τι θα συμβεί καλώντας την υπηρεσία.

Η χρήση διεπαφών API σε γλώσσα μηχανής περιλαμβάνει την διαδραστική τεκμηρίωση, την δημιουργία κώδικα για τεκμηρίωση, πελάτες (clients), severs, όπως και αυτοματοποιημένες δοκιμαστικές εφαρμογές. Τα APIs που βασίζονται σε Swagger δίνουν αρχεία JSON που ακολουθούν πιστά τους ορισμούς του Swagger, όπως θα αναλυθούν στη συνέχεια. Αυτά τα αρχεία μπορούν είτε να δημιουργηθούν και να δίνονται στατικά, είτε να παράγονται δυναμικά από κάθε εφαρμογή.

Χωρίς να αναλύουμε την ιστορία των διεπαφών των υπηρεσιών web το Swagger δεν είναι το πρώτο που προσπάθησε να αναπτύξει ένα πρώτυπο. Μπορούμε να πάρουμε πληροφορίες από το CORBA, το WSDL και το WADL. Αυτές οι προσπάθειες είχαν καλές προοπτικές αλλά περιορίστηκαν σε πολύ συγκεκριμένες εφαρμογές, συνδέθηκαν με μία συγκεκριμένη γλώσσα προγραμματισμού και οι στόχοι τους δεν εκπληρώθηκαν. Για τους λόγους αυτούς απέτυχαν στο να αποκτήσουν μεγάλο κοινό και να τραβήξουν την προσοχή των προγραμματιστών.

Το Swagger δεν απαιτεί το να ξαναγράψει κανείς το ήδη υπάρχον API του. Ούτε απαιτεί να συνδεθεί κάποιο πρόγραμμα στην υπηρεσία του. Απαιτεί όμως οι δυνατότητες της υπηρεσίας να μπορούν να περιγραφούν σύμφωνα με την δομή των κανονισμών Swagger (Swagger Specification). Βεβαίως δεν μπορούν όλες οι υπηρεσίες να περιγραφούν σύμφωνα με το Swagger άλλωστε αυτοί οι κανονισμοί δεν έχουν σκοπό να καλύψουν όλες τις πιθανές εφαρμογές ενός RESTful API. Το Swagger δεν ορίζει μια συγκεκριμένη διαδικασία ανάπτυξης εφαρμογών όπως για παράδειγμα να γίνεται πρώτα ο σχεδιασμός ή πρώτα ο κώδικας. Διευκολύνει και τις δύο τεχνικές καθιερώνοντας συγκεκριμένες διαδικασίες για ένα REST API.

Το Swagger «μιλά» JSON και XML αλλά και άλλες γλώσσες βρίσκονται στο στάδιο σχεδιασμού. Η σημερινή του έκδοση είναι η 1.2 και στη συνέχει αναλύονται ορισμένα από τα βασικά του χαρακτηριστικά.

Αυτό που το Swagger δεν είναι σε θέση να κάνει προς το παρόν είναι να προτείνει το πώς θα υποστηρίζονται πολλαπλές μορφές ενός API είτε από την πλευρά του πελάτη είτε από την πλευρά του server.

Στόχοι που θέτει το Swagger για την επόμενη γενιά του είναι:

- Ονοματολογία: πώς ονομάζονται διάφορα «πράγματα» στο Swagger Specification, κάτι που αποτελεί ένα από τα σημαντικότερα προβλήματα του computer science.
- Design-first APIs: η δημιουργία ενός φιλικού ως προς τον άνθρωπο συντακτικού θα καταστήσει εύκολη τη δημιουργία ενός API από το μηδέν.
- Επέκταση των μεταδεδομένων: επέκταση της δομής ώστε να περιλαμβάνει πληροφορίες SLA, vendor extensions και άλλα χρήσιμα μεταδεδομένα.
- Ανανέωση εργαλείων σχετικά με τη δημιουργία κώδικα και το Swagger UI.

3.1.1 SPECIFICATION

3.1.1.1 Ορισμοί

- Resource (πόρος): Ένα resource σε Swagger είναι μια οντότητα που έχει ένα σύνολο εξωτερικών λειτουργιών. Η οντότητα μπορεί να αντιπροσωπεύει ένα υπαρκτό αντικείμενο (κατοικίδια, χρήστες…) ή ένα σύνολο λογικών λειτουργιών που αρχειοθετούνται μαζί. Είναι απόφαση του χρήστη το αν sub-resources (υπό-πόροι) θα αναφέρονται ως μέρος του κύριου resource ή αν θα αποτελούν ένα ξεχωριστό recourse μόνοι τους. Για παράδειγμα έχουμε το ακόλουθο σύνολο URL:

```
- /users      - GET
                POST
- /users/{id} - GET
                PATCH
                DELETE
```

Σε αυτή την περίπτωση είτε υπάρχει ένα resource το “/users” που περιέχει λειτουργίες στο sub-resource “/users/{id}”, είτε υπάρχουν δύο διαφορετικά resources.

- URL: ένα πλήρως επεξηγηματικό URL.

3.1.1.2 Μορφή

Τα αρχεία που περιγράφουν τα RESTful APIs και είναι σε συμφωνία με την τεκμηρίωση του Swagger αντιπροσωπεύονται ως αντικείμενα JSON και τηρούν τα πρότυπα του JSON

Για παράδειγμα αν κάποιο πεδίο έχει την τιμή πίνακα, θα χρησιμοποιηθεί η απεικόνιση του πίνακα σε JSON:

```
{
   "field" : [...]
}
```

Εδώ πρέπει να σημειωθεί ότι αν και το API περιγράφεται χρησιμοποιώντας JSON, αυτό δεν σημαίνει ότι η είσοδος ή η έξοδός του δεν μπορεί να είναι σε XML, YAML, απλό κείμενο ή οποιαδήποτε άλλη μορφή έχει επιλέξει κανείς να χρησιμοποιεί στο API του.

3.1.1.3 Δομή αρχείων

Η Swagger απεικόνιση των APIs αποτελείται από δύο τύπους αρχείων:

1. **The Resource Listing** – Αυτό είναι το βασικό έγγραφο το οποίο περιέχει γενικές πληροφορίες για το API και καταγράφει τα resources του. Κάθε resource έχει το δικό του URL που καθορίζει τις λειτουργίες του API πάνω του.
2. **The API Declaration** – Αυτό το έγγραφο περιγράφει ένα resource συμπεριλαμβανομένων των κλήσεων API και των μοντέλων του. Υπάρχει ένα αρχείο για κάθε resource.

3.1.1.4 Τύποι δεδομένων

Στην τεκμηρίωση του Swagger οι τύποι δεδομένων χρησιμοποιούνται σε πολλές θέσεις – λειτουργίες, παράμετροι λειτουργιών, μοντέλα, αλλά και μέσα σε άλλους τύπους δεδομένων (πίνακες).

Τα πεδία που χρησιμοποιούνται για να περιγράψουν έναν δοσμένο τύπο δεδομένων προστίθενται στο σχετικό αντικείμενο. Για παράδειγμα, αν έχουμε το αντικείμενο Foo με το πεδίο name, και πούμε ότι αντιπροσωπεύει έναν τύπο δεδομένων πρέπει να προσθέσουμε το πεδίο type οπότε το Foo γίνεται:

```
"Foo" : {
   "name" : "sample",
   "type" : "string",
   ...
}
```

Η τεκμηρίωση Swagger υποστηρίζει πέντε τύπους δεδομένων:

1. Primitive (είσοδος/ έξοδος)
2. Containers (όπως πίνακες/ σύνολα) (είσοδος/ έξοδος)
3. Complex (ως μοντέλα) (είσοδος/ έξοδος)
4. Void (έξοδος)
5. File (είσοδος)

Primitives

Διαφορετικές γλώσσες προγραμματισμού αντιπροσωπεύουν τους primitives αριθμούς με διαφορετικό τρόπο. Το specification του Swagger υποστηρίζει το όνομα των primitive τύπων που υποστηρίζονται και από το JSON-Schema Draft 4. Βέβαια για καλύτερο συντονισμό ένα επιπλέον πεδίο με όνομα format μπορεί να ακολουθεί το πεδίο type για να δώσει περισσότερες πληροφορίες για τον τύπο που χρησιμοποιείται. Αν χρησιμοποιείται το πεδίο format τότε ο αντίστοιχος client πρέπει να συμμορφωθεί με αυτόν τον τύπο.

Ορισμένα παραδείγματα:

Όνομα	type	format	Σχόλια
integer	`integer`	`int32`	signed 32 bits
long	`integer`	`int64`	signed 64 bits
float	`number`	`float`	
double	`number`	`double`	
string	`string`		
byte	`string`	`byte`	
boolean	`boolean`		
date	`string`	`date`	
dateTime	`string`	`date-time`	

Void

Αυτός ο τύπος μας δείχνει ότι μία λειτουργία δεν επιστρέφει καμία τιμή. Για τον λόγο αυτό μπορεί να χρησιμοποιηθεί μόνο σαν τύπος επιστροφής.

File

Ο τύπος file είναι ένας ειδικός τύπος που υποδηλώνει το ανέβασμα αρχείου. Όταν χρησιμοποιούμε το file, το πεδίο `consumes` πρέπει να είναι `"multipart/form-data"` και το `paramType` πρέπει να είναι `"form"`

3.1.2 SCHEMA

3.1.2.1 Resource Listing

Το resource listing αποτελεί το βασικό έγγραφο για την περιγραφή του API. Περιέχει γενικές πληροφορίες για το API και έναν κατάλογο των διαθέσιμων πόρων.
Αυτό το έγγραφο πρέπει να δίνεται στο `/api-docs` path.

Όνομα πεδίου	Τύπος	Περιγραφή
swaggerVersion	`string`	**Απαραίτητο.** Προσδιορίζει την έκδοση του Swagger Specification που χρησιμοποιείται. Μπορεί να χρησιμοποιηθεί από το Swagger UI και άλλους clients για να ερμηνεύσει τη δομή του API. Η τιμή πρέπει να είναι μία υπάρχουσα έκδοση του Swagger specification. Τώρα, οι `"1.0"`, `"1.1"`, `"1.2"` είναι επιτρεπτές τιμές. Το πεδίο είναι τύπου `string` ώστε να καλύψει πιθανές μη αριθμητικές εκδόσεις στο μέλλον όπως "1.2a".

apis	[Resource Object]	**Απαραίτητο.** Καταγράφει του πόρους που πρόκειται να περιγραφούν. Ο πίνακας μπορεί να έχει 0 ή περισσότερα στοιχεία.
apiVersion	`string`	Παρέχει την έκδοση της εφαρμογής του API.
info	Info Object	Παρέχει μεταδεδομένα για το API. Τα μεταδεδομένα μπορεί να χρησιμοποιηθούν από τους clients αν χρειάζεται και μπορούν να βρίσκονται για μεγαλύτερη ευκολία στο Swagger-UI.
authorizations	Authorizations Object	Παρέχει πληροφορίες σχετικά με τις μορφές πιστοποίησης που επιτρέπονται σε αυτό το API.

Παράδειγμα:

```
{
  "apiVersion": "1.0.0",
  "swaggerVersion": "1.2",
  "apis": [
    {
      "path": "/pet",
      "description": "Operations about pets"
    },
    {
      "path": "/user",
      "description": "Operations about user"
    },
    {
      "path": "/store",
      "description": "Operations about store"
    }
  ],
  "authorizations": {
    "oauth2": {
      "type": "oauth2",
      "scopes": [
        {
          "scope": "email",
          "description": "Access to your email address"
        },
        {
          "scope": "pets",
          "description": "Access to your pets"
        }
      ],
      "grantTypes": {
        "implicit": {
          "loginEndpoint": {
            "url": "http://petstore.swagger.wordnik.com/oauth/dialog"
          },
```

```
          "tokenName": "access_token"
        },
        "authorization_code": {
          "tokenRequestEndpoint": {
            "url":
"http://petstore.swagger.wordnik.com/oauth/requestToken",
            "clientIdName": "client_id",
            "clientSecretName": "client_secret"
          },
          "tokenEndpoint": {
            "url": "http://petstore.swagger.wordnik.com/oauth/token",
            "tokenName": "access_code"
          }
        }
      }
    }
  },
  "info": {
    "title": "Swagger Sample App",
    "description": "This is a sample server Petstore server.  You can find
out more about Swagger \n    at <a href=\"http://swagger.wordnik.com\">
http://swagger.wordnik.com</a> or on irc.freenode.net, #swagger.  For this
sample,\n    you can use the api key \"special-key\" to test the
authorization filters",
    "termsOfServiceUrl": "http://helloreverb.com/terms/",
    "contact": "apiteam@wordnik.com",
    "license": "Apache 2.0",
    "licenseUrl": "http://www.apache.org/licenses/LICENSE-2.0.html"
  }
}
```

Resource Object

Όνομα πεδίου	Τύπος	Περιγραφή
Path	`string`	**Απαραίτητο.** Ένα σχετικό μονοπάτι για το API declaration από το μονοπάτι που χρησιμοποιείται για το Resource Listing. Αυτό το `path` δεν χρειάζεται απαραίτητα να αντιστοιχεί στο URL που δίνει το συγκεκριμένο resource του API αλλά εκεί που δίνεται το resource listing. Η τιμή πρέπει να είναι μία μορφή URL μονοπατιού.
description	`string`	*Προτείνεται.* Μία σύντομη περιγραφή του resource.

Παράδειγμα:

```
{
    "path": "/pets",
    "description": "Operations about pets."
}
```

Info Object

Όνομα πεδίου	Τύπος	Περιγραφή
title	`string`	**Απαραίτητο.** Ο τίτλος της εφαρμογής.
description	`string`	**Απαραίτητο.** Μία σύντομη περιγραφή της εφαρμογής.
termsOfServiceUrl	`string`	Ένα URL στους όρους χρήσης του API.
contact	`string`	Ένα email που χρησιμοποιείται για επικοινωνία σχετική με το API.
license	`string`	Το όνομα της άδειας που χρησιμοποιείται για το API.
licenseUrl	`string`	Ένα URL για την άδεια που χρησιμοποιείται για το API.

Παράδειγμα:

```
{
  "title": "Swagger Sample App",
  "description": "This is a sample server Petstore server.",
  "termsOfServiceUrl": "http://helloreverb.com/terms/",
  "contact": "apiteam@wordnik.com",
  "license": "Apache 2.0",
  "licenseUrl": "http://www.apache.org/licenses/LICENSE-2.0.html"
}
```

Authorizations Object

Το μέρος αυτό δίνει πληροφορίες σχετικά με τους τρόπους πιστοποίησης που παρέχονται στο API. Προς το παρόν το Swagger υποστηρίζει τρείς τρόπους πιστοποίησης – βασική πιστοποίηση, API key και OAuth2. Το μέρος αυτό χρησιμοποιείται για να ορίσει τους διαθέσιμους τρόπους πιστοποίησης χωρίς όμως να λέει ποιος απαιτείται στα διάφορα σημεία. Οι ουσιαστικοί περιορισμοί της πιστοποίησης αναφέρονται στο τμήμα API declaration.

Το τμήμα Authorizations είναι ένα αντικείμενο που περιέχει τους ορισμούς άλλων αντικειμένων και για τον λόγο αυτό δομείται ως εξής:

```
{
   "Authorization1" : {...},
   "Authorization2" : {...},
   ...,
   "AuthorizationN" : {...}
}
```

3.1.2.2 API Declaration

Το API Declaration παρέχει πληροφορίες για ένα API σχετικά με έναν πόρο. Πρέπει να υπάρχει ένα αρχείο για κάθε πόρο που περιγράφεται. Το αρχείο πρέπει να βρίσκεται στο URL που δίνεται από το πεδίο `path`.

Όνομα πεδίου	Τύπος	Περιγραφή
swaggerVersion	`string`	**Απαραίτητο.** Προσδιορίζει την έκδοση του Swagger Specification που χρησιμοποιείται. Μπορεί να χρησιμοποιηθεί από το Swagger UI και άλλους clients για να ερμηνεύσει τη δομή του API. Η τιμή πρέπει να είναι μία υπάρχουσα έκδοση του Swagger specification. Τώρα, οι `"1.0"`, `"1.1"`, `"1.2"` είναι επιτρεπτές τιμές.
apiVersion	`string`	Παρέχει την έκδοση της εφαρμογής του API
basePath	`string`	**Απαραίτητο.** Το βασικό URL του API. Αυτό το πεδίο είναι σημαντικό γιατί ενώ είναι σύνηθες να υπάρχει το Resource Listing και το API Declaration στους server που παρέχουν τα ίδια τα APIs, εντούτοις δεν είναι δεσμευτικό. Η τεκμηρίωση του API μπορεί να δίνεται μέσω στατικών αρχείων και όχι αρχείων που παράγονται από τον server του API, έτσι το URL που δείχνει στο API δεν μπορεί πάντα να παραχθεί από το URL που δείχνει στο API specification. Η τιμή πρέπει να είναι σε μορφή URL.
resourcePath	`string`	Το σχετικό μονοπάτι για το resource, από το `basePath`, που περιγράφει το API Specification. Η τιμή πρέπει να ξεκινάει από slash (`"/"`).
apis	[API Object]	**Απαραίτητο.** Μία λίστα από τα APIs που βρίσκονται σε αυτό το resource. Δεν μπορεί να βρίσκεται πάνω από ένα API Object ανά `path` στον πίνακα.
models	Models Object	Μία λίστα των μοντέλων που είναι διαθέσιμα σε αυτό το resource.
produces	[`string`]	Μία λίστα από τύπους MIME που το API μπορεί να παράξει. Αυτό είναι global για όλα τα APIs αλλά μπορεί να παρακαμφθεί σε ορισμένες κλήσεις API.
consumes	[`string`]	Μία λίστα από τύπους MIME που το API μπορεί να καταναλώσει. Αυτό είναι global για όλα τα APIs αλλά μπορεί να παρακαμφθεί σε ορισμένες κλήσεις API.
authorizations	Authorizations Object	Μία λίστα από τρόπους πιστοποίησης που απαιτούνται για τις λειτουργίες που αναφέρονται σε αυτό το API declaration. Ορισμένες λειτουργίες μπορεί να παρακάμπτουν αυτή τη ρύθμιση. Αν εδώ περιγράφονται πολλαπλοί τρόποι πιστοποίησης σημαίνει ότι εφαρμόζονται όλοι.

Παράδειγμα:


```
{
  "apiVersion": "1.0.0",
  "swaggerVersion": "1.2",
  "basePath": "http://petstore.swagger.wordnik.com/api",
  "resourcePath": "/store",
  "produces": [
    "application/json"
  ],
  "authorizations": {},
  "apis": [
    {
      "path": "/store/order/{orderId}",
      "operations": [
        {
          "method": "GET",
          "summary": "Find purchase order by ID",
          "notes": "For valid response try integer IDs with value <= 5.
Anything above 5 or nonintegers will generate API errors",
          "type": "Order",
          "nickname": "getOrderById",
          "authorizations": {},
          "parameters": [
            {
              "name": "orderId",
              "description": "ID of pet that needs to be fetched",
              "required": true,
              "type": "string",
              "paramType": "path"
            }
          ],
          "responseMessages": [
            {
              "code": 400,
              "message": "Invalid ID supplied"
            },
            {
              "code": 404,
              "message": "Order not found"
            }
          ]
        },
        {
          "method": "DELETE",
          "summary": "Delete purchase order by ID",
          "notes": "For valid response try integer IDs with value < 1000.
Anything above 1000 or nonintegers will generate API errors",
          "type": "void",
          "nickname": "deleteOrder",
          "authorizations": {
```

```
          "oauth2": [
            {
              "scope": "test:anything",
              "description": "anything"
            }
          ]
        },
        "parameters": [
          {
            "name": "orderId",
            "description": "ID of the order that needs to be deleted",
            "required": true,
            "type": "string",
            "paramType": "path"
          }
        ],
        "responseMessages": [
          {
            "code": 400,
            "message": "Invalid ID supplied"
          },
          {
            "code": 404,
            "message": "Order not found"
          }
        ]
      }
    ]
  },
  {
    "path": "/store/order",
    "operations": [
      {
        "method": "POST",
        "summary": "Place an order for a pet",
        "notes": "",
        "type": "void",
        "nickname": "placeOrder",
        "authorizations": {
          "oauth2": [
            {
              "scope": "test:anything",
              "description": "anything"
            }
          ]
        },
        "parameters": [
          {
            "name": "body",
            "description": "order placed for purchasing the pet",
            "required": true,
```

```
            "type": "Order",
            "paramType": "body"
          }
        ],
        "responseMessages": [
          {
            "code": 400,
            "message": "Invalid order"
          }
        ]
      }
    ]
  }
 ],
 "models": {
   "Order": {
     "id": "Order",
     "properties": {
       "id": {
         "type": "integer",
         "format": "int64"
       },
       "petId": {
         "type": "integer",
         "format": "int64"
       },
       "quantity": {
         "type": "integer",
         "format": "int32"
       },
       "status": {
         "type": "string",
         "description": "Order Status",
         "enum": [
           "placed",
           " approved",
           " delivered"
         ]
       },
       "shipDate": {
         "type": "string",
         "format": "date-time"
       }
     }
   }
 }
}
```

API Object

Το API Object περιγράφει μία ή περισσότερες λειτουργίες σε ένα `path`. Στον πίνακα `apis` πρέπει να υπάρχει μόνο ένα `API Object` ανά `path`.

Όνομα Πεδίου	Τύπος	Περιγραφή
path	`string`	**Απαραίτητο.** Το σχετικό μονοπάτι για την λειτουργία από το `basePath`. Η τιμή πρέπει να έχει τη μορφή URL μονοπατιού.
description	`string`	*Προτείνεται.* Μία σύντομη περιγραφή του resource.
operations	[Operation Object]	**Απαραίτητο.** Μία λίστα από τις λειτουργίες που είναι διαθέσιμες σε αυτό το μονοπάτι. Ο πίνακας μπορεί να περιέχει 0 ή περισσότερες λειτουργίες.

Operation Object

Το Operation Object περιγράφει μία λειτουργία σε ένα μονοπάτι. Στον πίνακα `operations` πρέπει να υπάρχει μόνο ένα `Operation Object` ανά `method`.

Όνομα Πεδίου	Τύπος	Περιγραφή
method	`string`	**Απαραίτητο.** Η HTTP μέθοδος που απαιτείται για να γίνει η λειτουργία. Η τιμή πρέπει να είναι μία από τις: `"GET"`, `"HEAD"`, `"POST"`, `"PUT"`, `"PATCH"`, `"DELETE"`,`"OPTIONS"`.
summary	`string`	Μία σύντομη περίληψη του τι κάνει η λειτουργία. Καλό θα είναι να είναι μικρότερη από 120 χαρακτήρες.
notes	`string`	Μία μακροσκελής εξήγηση της συμπεριφοράς της λειτουργίας.
nickname	`string`	**Απαραίτητο.** Ένα μοναδικό id για την λειτουργία που μπορεί να χρησιμοποιηθεί από εργαλεία που διαβάζουν την έξοδο για ευκολότερο χειρισμό.
authorizations	Authorizations Object	Μία λίστα από πιστοποιήσεις που απαιτούνται για την εκτέλεση αυτής της λειτουργίας.
parameters	[Parameter Object]	**Απαραίτητο.** Οι είσοδοι στην λειτουργία.
responseMessages	[Response Message Object]	Αναφέρει τα πιθανά απαντητικά μηνύματα που μπορούν να επιστρέψουν από τη λειτουργία.
produces	[`string`]	Μία λίστα από τύπους MIME που μπορεί να παράξει αυτή η λειτουργία.

consumes	[`string`]	Μία λίστα από τύπους MIME που μπορεί να καταναλώσει αυτή η λειτουργία.

Παράδειγμα:

```
{
  "method": "GET",
  "summary": "Find pet by ID",
  "notes": "Returns a pet based on ID",
  "type": "Pet",
  "nickname": "getPetById",
  "authorizations": {},
  "parameters": [
    {
      "name": "petId",
      "description": "ID of pet that needs to be fetched",
      "required": true,
      "type": "integer",
      "format": "int64",
      "paramType": "path",
      "minimum": "1.0",
      "maximum": "100000.0"
    }
  ],
  "responseMessages": [
    {
      "code": 400,
      "message": "Invalid ID supplied"
    },
    {
      "code": 404,
      "message": "Pet not found"
    }
  ]
}
```

Parameter Object

Το parameter object περιγράφει μία παράμετρο που στέλνεται σε μία λειτουργία.

Όνομα Πεδίου	Τύπος	Περιγραφή
paramType	`string`	**Απαραίτητο.** Ο τύπος της παραμέτρου. Μπορεί να είναι μία από τις ακόλουθες τιμές: `"path"`, `"query"`, `"body"`, `"header"`, `"form"`.
name	`string`	**Απαραίτητο.** Το μοναδικό όνομα της παραμέτρου. Κάθε όνομα πρέπει να είναι μοναδικό ακόμα και αν σχετίζεται με διαφορετικό `paramType`.

description	`string`	*Προτείνεται.* Μία σύντομη περιγραφή της παραμέτρου.
required	`boolean`	Μία σημαία που δείχνει αν αυτή η παράμετρος είναι απαραίτητη. Αν το πεδίο δεν υπάρχει η τιμή του θεωρείται `false`.
allowMultiple	`boolean`	Ένας άλλος τρόπος για να επιτρέπουμε πολλαπλές τιμές σε μία παράμετρο "query".

Παράδειγμα:

```
{
  "name": "body",
  "description": "Pet object that needs to be updated in the store",
  "required": true,
  "type": "Pet",
  "paramType": "body"
}
```

Response Message Object

Στο μέρος αυτό περιγράφεται ένα πιθανό απαντητικό μήνυμα που μπορεί να έρθει από την κλήση μίας λειτουργίας.

Όνομα Πεδίου	Τύπος	Περιγραφή
code	`integer`	**Απαραίτητο.** Ο κωδικός HTTP που δίνεται. Η τιμή του πρέπει να είναι μία από αυτές που περιγράφονται στο RFC 2616 - Section 10.
message	`string`	**Απαραίτητο.** Η εξήγηση για αυτόν τον κωδικό.
responseModel	`string`	Ο τύπος που επιστρέφεται για την συγκεκριμένη απόκριση.

Παράδειγμα:

```
{
    "code": 404,
    "message": "no project found",
    "responseModel": "ErrorModel"
  }
```

3.2 RAML

Το RAML βγαίνει από το RESTful API Modeling Language δηλαδή είναι μία γλώσσα μοντελοποίησης για RESTful APIs που βασίζεται στο YAML. Είναι ένας τρόπος περιγραφής των πρακτικά RESTful APIs ώστε να διαβάζονται εύκολα από τους ανθρώπους και τους υπολογιστές. Αναφέρουμε «πρακτικά RESTful» καθώς μέχρι σήμερα στον πραγματικό κόσμο πολύ λίγα APIs ικανοποιούν όλους τους περιορισμούς του REST. Η RAML δεν είναι αυστηρή. Προς το παρόν επικεντρώνεται στην ξεκάθαρη περιγραφή πόρων, μεθόδων, παραμέτρων, ανταποκρίσεων και άλλων HTTP κατασκευασμάτων που αποτελούν την βάση για τα σύγχρονα API τα οποία ακολουθούν πολλούς αλλά πιθανότατα όχι όλους τους RESTful περιορισμούς.

Η RAML δεν είναι ιδιοκτησιακή ούτε πωλείται. Στόχος είναι να βοηθήσει το υπάρχον οικοσύστημα των API και να λύσει άμεσα προβλήματα, προτού προωθήσει ακόμα καλύτερους τρόπους ανάπτυξης API.

Οι λόγοι που θα έπρεπε κάποιος να χρησιμοποιήσει RAML για το API του είναι πολλοί. Αρχικά υλοποιεί τον πηγαίο κώδικα με έναν καλά ορισμένο τρόπο και σε μία πραγματικά αναγνώσιμη από τον άνθρωπο μορφή. Στη συνέχεια, η δομή του API είναι φανερή και γίνεται εύκολα κατανοητή από προγραμματιστές, συνεργάτες και άλλους καταναλωτές API. Και τέλος η διαδικασία ανάπτυξης εφαρμογών (Application Programming eXperience) ή APX βελτιώνεται ευρέως γνωρίζοντας ότι υπάρχει μια επίσημη και ξεκάθαρη σύμβαση που αντικατοπτρίζει την δομή του API και είναι ανεξάρτητη από την εφαρμογή του. Καλά σχεδιασμένα και ξεκάθαρα δομημένα APIs με εσωτερική συνέπεια που είναι βασισμένα σε συγκεκριμένα μοτίβα διευκολύνουν και κάνουν πιο σταθερή την ανάπτυξη κώδικα από το μέρος του πελάτη αλλά και από την εφαρμογή στον server.

Η RAML περιγράφει τα APIs με τρόπο που είναι:

- Ξεκάθαρος: Η δομή ενός API πρέπει να είναι φανερή. Υποδείγματα αναπτύσσονται, τα δεδομένα και οι τρόποι αλληλεπίδρασης βρίσκονται στο επίκεντρο.
- Σωστός: Η περιγραφή ενός API αποτελεί σύμβαση. Οτιδήποτε γράφεται πρέπει να περιγράφει σωστά την συμπεριφορά του API.
- Ακριβής: Η ανάπτυξη του πελάτη μπορεί να προκύψει από την περιγραφή του API και να είναι πιστή και συμβατή με αυτό. Αυτοί που θα το εφαρμόσουν γνωρίζουν σε τι πρέπει να ανταποκριθούν.
- Συνεπής: Η RAML παρέχει μεγάλη υποστήριξη στην αποθήκευση υποδειγμάτων. Προωθεί την επαναχρησιμοποίηση, δίνει τη δυνατότητα για ανταλλαγή και ανακάλυψη νέων προτύπων και στοχεύει στην επικράτηση των καλύτερων τακτικών.
- Ευανάγνωστος και writable: Βελτιστοποιεί την δημιουργία και την ανάγνωση περιγραφών από δυναμικούς και έξυπνους ανθρώπους.
- Φυσικός και διαισθητικός: Είναι όσο πιο κοντά γίνεται στο πνευματικό μοντέλο που έχει κανείς για τα APIs. Γράφει σχεδόν όπως θα έγραφε εάν έστελνε e-mail σε κάποιον βοηθώντας τον να σχεδιάσει το API.

Τα API documentation generators, API client code generators και οι API servers δέχονται ένα έγγραφο RAML και παράγουν τεκμηρίωση για τους χρήστες, κώδικα για τους πελάτες (clients) και τμήματα για τον κώδικα του server αντίστοιχα. Η RAML ορίζει το media type "application/raml+yaml" για την περιγραφή και την τεκμηρίωση των πόρων του API. Επιπλέον παρέχει τη δυνατότητα για εκτενέστερη τεκμηρίωση ενός RESTful API, επιτρέποντας σε εργαλεία παραγωγής τεκμηρίωσης να εξάγουν την τεκμηρίωση του χρήστη και να την μεταφράσουν σε οπτικές μορφές όπως PDF, HTML κ.α.

Επιπλέον η RAML εισάγει την καινοτόμα ιδέα για τον χαρακτηρισμό πόρων και μεθόδων και έτσι ελαχιστοποιούνται οι επαναλήψεις που απαιτούνται για να καθοριστεί ένα σχέδιο για RESTful API.

3.2.1 Γλώσσα Markup

Οι ορισμοί API σε RAML είναι έγγραφα συμβατά με YAML. Ξεκινούν με μία γραμμή με YAML σχόλιο που δείχνει την έκδοση της RAML όπως

```
#%RAML 0.8
```

Η έκδοση της RAML πρέπει να βρίσκεται στην πρώτη γραμμή του εγγράφου. Όλες οι άλλες γραμμές που αρχίζουν με " #%" πρέπει να ερμηνεύονται ως σχόλια.

Τα έγγραφα RAML πρέπει να είναι σε θέση να χωριστούν σε πολλαπλά αρχεία. Για τον λόγο αυτό όλοι οι RAML parsers πρέπει να υποστηρίζουν την ένδειξη *include* που επιτρέπει την εισαγωγή RAML, YAML και αρχείων απλού κειμένου.

Σε αυτό το παράδειγμα το περιεχόμενο του myTextFile.txt εισάγεται σαν τιμή στην external property

```
#%RAML 0.8
external: !include myTextFile.txt
```

Όταν εισάγονται αρχεία RAML ή YAML οι parsers πρέπει όχι μόνο να διαβάζουν το περιεχόμενό τους αλλά να το προσθέτουν στην υπάρχουσα δομή σαν το περιεχόμενο να δηλωνόταν αναλυτικά.

Παράδειγμα. Το αρχείο properties.raml ορίζει δύο ιδιότητες. Το αρχείο big.raml εισάγει το αρχείο properties.raml

```
#%RAML 0.8
#properties.raml

propertyA: valueA
propertyB: valueB
```

```
#%RAML 0.8
#big.raml

external: !include properties.raml
```

Η τελική δομή είναι ισοδύναμη με το αν γράφαμε

```
#%RAML 0.8
external:
  propertyA: valueA
  propertyB: valueB
```

Η τοποθεσία του αρχείου που είναι να εισαχθεί, δηλαδή το δεξί μέρος του !include πρέπει να είναι στατική που σημαίνει ότι δεν μπορεί να περιέχει πόρους ή μεταβλητές. Κάτι τέτοιο θα αναζητηθεί στις μελλοντικές εκδόσεις της RAML.

3.2.2 Named Parameters

Το specification της RAML περιέχει ορισμένα ονόματα παραμέτρων που είναι χρήσιμα για την περιγραφή ιδιοτήτων

displayName	Προαιρετικό. Είναι ένα φιλικό όνομα για την παράμετρο το οποίο χρησιμοποιείται για λόγους τεκμηρίωσης και αναφοράς σε αυτή
description	Προαιρετικό. Περιγράφει την αναμενόμενη χρήση ή σημασία της παραμέτρου. Μπορεί να γραφεί σε Markdown
type	Προαιρετικό. Προσδιορίζει τον τύπο της τιμής της παραμέτρου. Αν ο τύπος που δίνεται είναι διαφορετικός από αυτόν που έχει οριστεί πρέπει να δίνεται μήνυμα λάθους. Αν δεν ορίζεται θεωρούμε ότι είναι string.
enum	Προαιρετικό, μόνο για τύπου string. Απαριθμεί τις αποδεκτές τιμές της παραμέτρου. Πρέπει να είναι πίνακας.
pattern	Προαιρετικό, μόνο για τύπου string. Είναι μία έκφραση με την οποία πρέπει να ταιριάζει η τιμή της παραμέτρου.
minLength	Προαιρετικό, μόνο για τύπου string. Δίνει τον ελάχιστο αριθμό χαρακτήρων για την τιμή της παραμέτρου.
maxLength	Προαιρετικό, μόνο για τύπου string. Δίνει τον μέγιστο αριθμό χαρακτήρων για την τιμή της παραμέτρου.
minimum	Προαιρετικό, μόνο για τύπου integer. Η ελάχιστη επιτρεπτή τιμή της παραμέτρου
maximum	Προαιρετικό, μόνο για τύπου integer. Η μέγιστη επιτρεπτή τιμή της παραμέτρου
example	Προαιρετικό. Μία ενδεικτική τιμή για την παράμετρο.
repeat	Προαιρετικό. Ορίζει ότι η παράμετρος μπορεί να επαναληφθεί. Αν η παράμετρος μπορεί να χρησιμοποιηθεί πολλές φορές τότε η τιμή του repeat πρέπει να τεθεί true.

required	Προαιρετικό εκτός αν ορίζεται διαφορετικά. Ορίζει εάν η παράμετρος και η τιμή της πρέπει να υπάρχουν στον ορισμό του API. Σε αυτή την περίπτωση η τιμή τίθεται true.
default	Προαιρετικό. Ορίζει την προκαθορισμένη τιμή μιας παραμέτρου αν η παράμετρος παραλείπεται ή η τιμή της δεν καθορίζεται.

3.2.3 Βασικές πληροφορίες

Root Section

Η ενότητα αυτή περιγράφει τις βασικές πληροφορίες για ένα API όπως τον τίτλο του και το βασικό URI και περιγράφει πως ορίζονται schema references. Παράδειγμα

```
#%RAML 0.8
title: GitHub API
version: v3
baseUri: https://api.github.com
mediaType:  application/json
schemas:
  - User:  schema/user.json
    Users: schema/users.json
    Org:   schema/org.json
    Orgs:  schema/orgs.json
```

API Title

(Απαιτείται) Η ιδιότητα title είναι μία σύντομη περιγραφή ενός RESTful API σε απλό κείμενο. Καλό θα είναι να είναι αντιπροσωπευτικός καθώς χρησιμοποιείται στο documentation.

API Version

(προαιρετικό) Αν ο ορισμός του RAML API αναφέρεται σε κάποια συγκεκριμένη έκδοση τότε αυτή πρέπει να αναφέρεται. Η ιδιότητα αυτή μπορεί να παραλειφθεί αν

- Το ίδιο το API δεν έχει εκδόσεις
- Ο ορισμός του API δεν αλλάζει στις διαφορετικές εκδόσεις του.

Base URI και baseUriParameters

(Προαιρετικό κατά την ανάπτυξη, απαραίτητο κατά την εφαρμογή) Οι πόροι ενός RESTful API καθορίζονται σχετικά με το base URI του API. Η τιμή της ιδιότητας base URI πρέπει να συμμορφώνεται με το URI specification [RFC2396] ή με το Level 1 Template URI όπως ορίζεται στο RFC 6570 [RFC6570].

```
#%RAML 0.8
title: Amazon S3 REST API
version: 1
baseUri: https://{bucketName}.s3.amazonaws.com
baseUriParameters:
  bucketName:
    description: The name of the bucket
```

Protocols

(Προαιρετικό) Ένα RESTful API μπορεί να χρησιμοποιεί HTTP, HTTPS ή και τα δύο. Η ιδιότητα protocols χρησιμοποιείται για να αναφέρει τα πρωτόκολλα που το API υποστηρίζει. Αν η ιδιότητα αυτή δεν προσδιορίζεται τότε χρησιμοποιείται το πρωτόκολλο που αναφέρεται στο base URI. Η τιμή του είναι πίνακας από strings με τις τιμές “HTTP” και/ή “HTTPS”.

```
#%RAML 0.8
title: Salesforce Chatter REST API
version: v28.0
protocols: [ HTTP, HTTPS ]
baseUri: https://na1.salesforce.com/services/data/{version}/chatter
```

Default Media Type

(Προαιρετικό) Τα media types που δίνονται σε API responses και αναμένονται από API requests μπορεί να είναι προκαθορισμένα από την ιδιότητα mediaType. Αυτή η ιδιότητα καθορίζεται στο βασικό επίπεδο του ορισμού ενός API. Η τιμή της μπορεί να είναι ένα string με ένα έγκυρο media type:

- Ένα από τα ακόλουθα YAML media types: * text/yaml * text/x-yaml * application/yaml * application/x-yaml*.
- Οποιοδήποτε media type από την λίστα IANA MIME Media Types, http://www.iana.org/assignments/media-types
- Έναν ειδικό τύπο που περιγράφεται από την έκφραση "application\/[A-Za-z.-0-1]*+?(json|xml)"

```
#%RAML 0.8
title: Stormpath REST API
version: v1
baseUri: https://api.stormpath.com/{version}
mediaType: application/json
```

Schemas

(Προαιρετικό) Για μεγαλύτερη συνέπεια και απλότητα ο ορισμός του API καλό είναι να περιλαμβάνει μία προαιρετική ιδιότητα schemas στην ενότητα root. Η ιδιότητα schemas καθορίζει συλλογές από schemas που μπορούν να χρησιμοποιηθούν οπουδήποτε στον ορισμό του API. Η τιμή της είναι ένας πίνακας χαρτών. Σε κάθε χάρτη το κλειδί είναι το όνομα του schema και η τιμή είναι ορισμοί schema.

```
#%RAML 0.8
baseUri: https://api.example.com
title: Filesystem API
version: 0.1
schemas:
  - !include path-to-canonical-schemas/canonicalSchemas.raml
  - File:       !include path-to-schemas/filesystem/file.xsd
    FileUpdate: !include path-to-schemas/filesystem/fileupdate.xsd
    Files:      !include path-to-schemas/filesystem/files.xsd
    Dir:        !include path-to-schemas/filesystem/dir.xsd
    Dirs:       !include path-to-schemas/filesystem/dirs.xsd
```

```
/files:
  get:
    responses:
      200:
        body:
          application/xml:
            schema: Files
```

URI Parameters

(προαιρετικό) Εκτός από τις δεσμευμένες παραμέτρους URI που περιγράφηκαν στην ενότητα για την ιδιότητα baseUri, μία Level 1 Template URI μπορεί να ενσωματώσει ειδικές παραμέτρους URI που χρησιμοποιούνται σε μία πληθώρα περιπτώσεων. Για παράδειγμα ας δούμε τον ακόλουθο πάροχο API που παραμετροποιεί το base URI με πληροφορίες των πελατών όπως το όνομα της εταιρίας.

```
#%RAML 0.8
title: FreshBooks API
version: 2.1
baseUri: https://{companyName}.freshbooks.com/api/{version}/xml-in
```

Οι παράμετροι URI μπορούν να καθοριστούν περισσότερο χρησιμοποιώντας την ιδιότητα uriParameters, η οποία είναι προαιρετική. Η ιδιότητα αυτή πρέπει να είναι ένας χάρτης στον οποίο κάθε κλειδί είναι το όνομα της παραμέτρου URI όπως ορίζεται στην ιδιότητα baseUri. Το uriParameters δεν μπορεί να περιέχει κλειδί με το όνομα version καθώς είναι όνομα από μια δεσμευμένη παράμετρο URI.

```
#%RAML 0.8
title: Salesforce Chatter Communities REST API
version: v28.0
baseUri: https://{communityDomain}.force.com/{communityPath}
uriParameters:
 communityDomain:
   displayName: Community Domain
   type: string
 communityPath:
   displayName: Community Path
   type: string
   pattern: ^[a-zA-Z0-9][-a-zA-Z0-9]*$
   minLength: 1
```

User Documentation

(Προαιρετικό) Ο ορισμός ενός API μπορεί να περιλαμβάνει μία πληθώρα από έγγραφα που λειτουργούν σαν οδηγοί χρήσης και σαν reference documentation για το API. Τέτοια έγγραφα διευκρινίζουν πώς δουλεύει το API και παρέχουν business context.

Οι documentation-generators πρέπει να περιλαμβάνουν όλες τις ενότητες που έχει η ιδιότητα documentation στην έξοδο της τεκμηρίωσης και να διατηρούν τη σειρά κατά την οποία συντάσσεται.

Για να προσθέσουμε τεκμηρίωση χρήστη στο API, εισάγουμε την ιδιότητα documentation στο root του ορισμού του API. Η τιμή της ιδιότητας αυτής είναι ένας πίνακας από έγγραφα. Κάθε έγγραφο πρέπει να περιλαμβάνει τον προσδιορισμό title και content που είναι απαραίτητοι. Αν υπάρχει η ιδιότητα documentation πρέπει να περιέχει τουλάχιστον ένα έγγραφο.

Οι documentation-generators πρέπει να επεξεργάζονται το περιεχόμενο όπως θα το έκαναν αν οριζόταν χρησιμοποιώντας Markdown.

Παράδειγμα

```
#%RAML 0.8
title: ZEncoder API
baseUri: https://app.zencoder.com/api
documentation:
 - title: Home
   content: |
     Welcome to the _Zencoder API_ Documentation. The _Zencoder API_
     allows you to connect your application to our encoding service
     and encode videos without going through the web  interface. You
     may also benefit from one of our
     [integration
libraries](https://app.zencoder.com/docs/faq/basics/libraries)
     for different languages.
```

Το documentation μπορεί να γράφεται αναλυτικά όπως παραπάνω ή χρησιμοποιώντας το !include ώστε να εισάγει κάποιο εξωτερικό περιεχόμενο. Δηλαδή

```
#%RAML 0.8
title: ZEncoder API
baseUri: https://app.zencoder.com/api
documentation:
 - title: Home
   content: !include zencoder-home.md
```

Δεν υπάρχει περιορισμός στον αριθμό των σελίδων που μπορούν να εισαχθούν στον ορισμό του RAML API. Βέβαια, εάν το content της ιδιότητας documentation είναι αρκετά μεγάλο ώστε να κάνει δύσκολη την ανάγνωση του ορισμού του API, καλό θα είναι να χρησιμοποιείται το include.

Το παρακάτω παράδειγμα δείχνει τον ορισμό ενός RAML API με πολλαπλές σελίδες τεκμηρίωσης

```
#%RAML 0.8
title: GitHub API
version: v3
baseUri: https://api.github.com
documentation:
 - title: Getting Started
   content: !include github-3-getting-started.md
 - title: Basics of Authentication
   content: !include github-3-basics-of-authentication.md
```

```
 - title: Rendering Data as Graphs
   content: !include github-3-rendering-data-as-graphs.md
```

Resources και εμφωλευμένα resources

Τα resources προσδιορίζονται από το σχετικό τους URI που πρέπει να ξεκινάει με slash (/).

Ένα resource που ορίζεται ως ιδιότητα στο επίπεδο root ονομάζεται top-level resource. Το κλειδί αυτής της ιδιότητας είναι το URI του resource σε σχέση με το baseUri. Ένα resource που ορίζεται σαν παιδί κάποιου άλλου resource ονομάζεται εμφωλευμένο resource και το κλειδί της ιδιότητάς του είναι το URI του σε σχέση με το URI του resource – γονέα.

Το παράδειγμα αυτό δείχνει τον ορισμό ενός API με ένα top-level resource, /gists και ένα εμφωλευμένο resource, /public.

```
#%RAML 0.8
title: GitHub API
version: v3
baseUri: https://api.github.com
/gists:
  displayName: Gists
  /public:
    displayName: Public Gists
```

Display Name

Το πεδίο displayName προσφέρει ένα φιλικό όνομα στο resource και μπορεί να χρησιμοποιηθεί από εργαλεία που παράγουν τεκμηριώσεις. Το πεδίο αυτό είναι προαιρετικό. Αν το πεδίο displayName δεν υπάρχει για κάποιο resource, τότε η τεκμηρίωση πρέπει να αναφέρεται στο resource αυτό με το σχετικό του URI (π.χ. “/jobs”).

Description

Κάθε resource είτε top-level είτε εμφωλευμένο μπορεί να έχει την ιδιότητα description στην οποία περιγράφεται σύντομα το resource. Προτείνεται όλα τα resources να έχουν την ιδιότητα description.

Template URIs και URI Parameters

Template URIs που περιέχουν παραμέτρους URI μπορούν να χρησιμοποιηθούν για να ορίσουν το σχετικό URI ενός resource όταν έχει μεταβλητά στοιχεία. Το ακόλουθο παράδειγμα δείχνει ένα top-level resource με το κλειδί /jobs και ένα εμφωλευμένο resource με το κλειδί /{jobId}

```
#%RAML 0.8
title: ZEncoder API
version: v2
baseUri: https://app.zencoder.com/api/{version}
/jobs: # its fully-resolved URI is
https://app.zencoder.com/api/{version}/jobs
  displayName: Jobs
  description: A collection of jobs
  /{jobId}: # its fully-resolved URI is
https://app.zencoder.com/api/{version}/jobs/{jobId}
    description: A specific job, a member of the jobs collection
```

Ένα resource μπορεί να περιλαμβάνει την ιδιότητα uriParameters η οποία καθορίζει τις παραμέτρους URI στο σχετικό URI του resource. Το παρακάτω παράδειγμα δείχνει δύο top-level resources (/user και /users) και ένα εμφωλευμένο resource που καθορίζεται από το URI template (“/{userId}”). Η παράμετρος URI “userId” ορίζεται ξεκάθαρα και έχει το displayName “User ID” και τύπο integer.

```
#%RAML 0.8
title: GitHub API
version: v3
baseUri: https://api.github.com
/user:
  displayName: Authenticated User
/users:
  displayName: Users
  /{userId}:
   displayName: User
   uriParameters:
     userId:
       displayName: User ID
       type: integer
```

Base URI parameters

Ένα resource ή μία μέθοδος μπορεί να παρακάμψει τις τιμές μίας base URI template. Αυτό είναι χρήσιμο για να περιορίσει ή να αλλάξει κανείς την επιλογή των παραμέτρων στο base URI. Η ιδιότητα baseUriParameters μπορεί να χρησιμοποιηθεί για να παρακαμφθούν οι παράμετροι που ορίζονται στο επίπεδο root.

Στο ακόλουθο παράδειγμα κλήσεις στο resource /files πρέπει να γίνουν στο "https://api-content.dropbox.com/{version}". Όλες οι άλλες κλήσεις στο API γίνονται στο "https://api.dropbox.com/{version}".

```
#%RAML 0.8
title: Dropbox API
version: 1
baseUri: https://{apiDomain}.dropbox.com/{version}
baseUriParameters:
  apiDomain:
    description: |
      The sub-domain at which the API is accessible. Most API calls are
sent to https://api.dropbox.com
    enum: [ "api" ]
/account/info:
  displayName: Account Information
/files:
  displayName: Download files
  baseUriParameters:
    apiDomain:
      enum: [ "api-content" ]
```

Absolute URI

Τα απόλυτα URIs δεν προσδιορίζονται ξεκάθαρα. Σχηματίζονται ξεκινώντας από το baseUri και στη συνέχεια προσθέτουν το σχετικό URI του top-level resource. Μετά, προσθέτουν

διαδοχικά τις τιμές των σχετικών URI για κάθε εμφωλευμένο resource μέχρι να φτάσουμε στο resource που επιθυμούμε.

```
   "https://api.github.com"            <--- baseUri
              +
          "/gists"                     <--- gists resource relative URI
              +
          "/public"                    <--- public gists resource relative
URI
              =
"https://api.github.com/gists/public"  <--- public gists absolute URI
```

Ένα εμφωλευμένο resource μπορεί και αυτό να έχει μέσα του ένα άλλο εμφωλευμένο resource οπότε δημιουργούνται πολλαπλά εμφωλευμένα resources.

Σε αυτό το παράδειγμα, το /user είναι ένα top-level resource που δεν έχει παιδιά. Το /users είναι ένα top-level resource που έχει ένα εμφωλευμένο resource, /{userId}. Το εμφωλευμένο resource /{userId} έχει τρία εμφωλευμένα resources, /followers, /following, και /keys.

```
#%RAML 0.8
title: GitHub API
version: v3
baseUri: https://api.github.com
/user:
/users:
  /{userId}:
    uriParameters:
      userId:
        type: integer
    /followers:
    /following:
    /keys:
      /{keyId}:
        uriParameters:
          keyId:
            type: integer
```

Και τα απόλυτα URI για αυτά τα resources στην ίδια σειρά με τον ορισμό τους είναι:

```
https://api.github.com/user
https://api.github.com/users
https://api.github.com/users/{userId}
https://api.github.com/users/{userId}/followers
https://api.github.com/users/{userId}/following
https://api.github.com/users/{userId}/keys
https://api.github.com/users/{userId}/keys/{keyId}
```

Methods

Σε ένα RESTful API, methods είναι οι ενέργειες που γίνονται σε ένα resource. Κάθε μέθοδος πρέπει να είναι μία από τις HTTP μεθόδους που ορίζονται στο HTTP version 1.1 specification [RFC2616] και την επέκτασή του, RFC5789 [RFC5789].

Description

Κάθε ορισμένη μέθοδος μπορεί να έχει το πεδίο description το οποίο περιγράφει σύντομα τι κάνει η μέθοδος στο resource. Προτείνεται όλες οι μέθοδοι που ορίζονται στο API να έχουν το πεδίο description.

Το παράδειγμα αυτό το resource /jobs στο οποίο ορίζονται οι μέθοδοι POST και GET

```
#%RAML 0.8
title: ZEncoder API
version: v2
baseUri: https://app.zencoder.com/api/{version}
/jobs:
  post:
    description: Create a Job
  get:
    description: List Jobs
```

Βέβαια η τιμή της ιδιότητας description μπορεί να γραφεί και χρησιμοποιώντας Markdown.

Headers

Μία μέθοδος ενός API μπορεί να υποστηρίζει ή να απαιτεί μη-καθιερωμένες HTTP επικεφαλίδες. Στον ορισμό του API προσδιορίζονται αυτές οι επικεφαλίδες με την ιδιότητα headers. Η ιδιότητα αυτή είναι ένας χάρτης στον οποίο το κλειδί είναι το όνομα της επικεφαλίδας και η τιμή είναι ένας άλλος χάρτης που καθορίζει τα πεδία της επικεφαλίδας, σύμφωνα με την ενότητα Named Parameters.

Αυτό το παράδειγμα δείχνει μία μέθοδο POST με μία HTTP επικεφαλίδα

```
#%RAML 0.8
title: ZEncoder API
version: v2
baseUri: https://app.zencoder.com/api/{version}
/jobs:
  post:
    description: Create a Job
    headers:
      Zencoder-Api-Key:
        displayName: ZEncoder API Key
```

Protocols

Μία μέθοδος μπορεί να παρακάμψει την τιμή του protocols για εκείνη τη συγκεκριμένη μέθοδο και μόνο, ορίζοντας μία διαφορετική τιμή στο πεδίο. Στο παράδειγμα που ακολουθεί η μέθοδος GET είναι προσβάσιμη και από HTTP και από HTTPS ενώ το υπόλοιπο API μόνο από HTTPS.

```
#%RAML 0.8
title: Twitter API
version: 1.1
baseUri: https://api.twitter.com/{version}
/search/tweets.json:
  displayName: Tweet Search
  get:
    description: Returns a collection of relevant Tweets matching a
specified query
    protocols: [HTTP, HTTPS]
```

Query Strings

Οι πόροι ενός API μπορούν να φιλτραριστούν (ώστε να επιστρέψουν ένα υποσύνολο αποτελεσμάτων) ή να τροποποιηθούν (όπως να μετατραπεί το body από JSON σε XML μορφή) χρησιμοποιώντας τα query strings. Εάν το resource ή οι μέθοδοί του υποστηρίζουν ένα query string, τότε αυτό πρέπει να ορίζεται από την ιδιότητα queryParameters.

Η ιδιότητα αυτή είναι ένας χάρτης στον οποίο το κλειδί είναι το όνομα του query parameter και η τιμή του είναι πίνακας που ορίζει τα πεδία του query parameters, σύμφωνα με την ενότητα Named Parameters.

```
#%RAML 0.8
title: GitHub API
version: v3
baseUri: https://api.github.com/{version}
/users:
  get:
    description: Get a list of users
    queryParameters:
      page:
        description: Specify the page that you want to retrieve
        type: integer
        required: true
        example: 1
      per_page:
        description: Specify the amount of items that will be retrieved per
page
        type: integer
        minimum: 10
        maximum: 200
        default: 30
        example: 50
```

Body

Ορισμένες μέθοδοι αναμένουν το resource να τους σταλεί σαν request body. Για παράδειγμα, για να δημιουργηθεί ένα resource, πρέπει το αίτημα να περιλαμβάνει πληροφορίες για το resource που πρόκειται να δημιουργηθεί.

Τα resources μπορούν να έχουν διαφορετικές αναπαραστάσεις. Για παράδειγμα ένα API μπορεί να υποστηρίζει JSON και XML. Το body μιας μεθόδου ορίζεται από στην ιδιότητα body στην οποία το κλειδί είναι ένα έγκυρο media type.

Αυτό το παράδειγμα δείχνει ένα απόσπασμα από το API Zencoder σχετικά με το resource /jobs, το οποίο δέχεται ως εισόδους είτε JSON είτα XML.

```
/jobs:
  post:
    description: Create a Job
    body:
      text/xml: !!null
      application/json: !!null
```

Responses

Οι μέθοδοι μπορεί να έχουν μία ή περισσότερες απαντήσεις. Οι απαντήσεις αυτές μπορούν να περιγραφούν χρησιμοποιώντας την ιδιότητα description και μπορεί να περιέχουν το πεδίο example ή την ιδιότητα schema.

Αυτό το παράδειγμα δείχνει έναν ορισμό για την απάντηση 200 σε GET μέθοδο

```
/media/popular:
  displayName: Most Popular Media
  get:
    description: |
      Get a list of what media is most popular at the moment.
    responses:
      200:
        body:
          application/json:
            example: !include examples/instagram-v1-media-popular-
example.json
```

Οι απαντήσεις πρέπει να είναι ένας χάρτης από έναν ή περισσότερους κωδικούς κατάστασης HTTP, όπου κάθε κωδικός είναι ένας χάρτης που περιγράφει τι σημαίνει ο κωδικός.

Κάθε απάντηση μπορεί να περιέχει την ιδιότητα body η οποία έχει την ίδια δομή όπως αυτή του request body. Απαντήσεις που μπορούν να στείλουν πάνω από έναν κωδικό μπορούν να ορίσουν πολλαπλά body.

Για API χωρίς a priori γνώση του τύπου της απάντησης που θα δώσουν, το “*/*” μπορεί να χρησιμοποιηθεί για να δείξει ότι οι απαντήσεις που δεν ταιριάζουν με τους τύπους δεδομένων που έχουν οριστεί, πρέπει να γίνουν αποδεκτές.

Οι απαντήσεις πιθανόν να έχουν την ιδιότητα description που εξηγεί αναλυτικότερα γιατί δόθηκε αυτή η απάντηση. Η περιγραφή αυτή είναι ιδιαίτερα χρήσιμη σε περιπτώσεις λαθών. Για παράδειγμα

```
/media/popular:
  displayName: Most Popular Media
  get:
    description: |
      Get a list of what media is most popular at the moment.
    responses:
      503:
        description: |
          The service is currently unavailable or you exceeded the maximum
requests per hour allowed to your application.
```

Resource Types και Traits

Οι ορισμοί για resources και μεθόδους είναι συχνά επαναλαμβανόμενοι. Για παράδειγμα αν ένα API απαιτεί πιστοποίηση OAuth, ο ορισμός του API πρέπει να περιλαμβάνει το query string access_token στις περιγραφές όλων των μεθόδων. Επιπλέον υπάρχουν πολλά πλεονεκτήματα στην επαναχρησιμοποίηση μοτίβων σε πολλά resources και μεθόδους. Η

χρήση αυτών των μοτίβων συμβάλει στη συνέπεια και μειώνει την πολυπλοκότητα για τους servers και για τους πελάτες.

Ένα Resource type είναι ένας μερικός ορισμός ενός resource που μπορεί να προσδιορίσει την περιγραφή και τις μεθόδους με τις ιδιότητές τους.

Ένα Trait είναι ένας μερικός ορισμός μίας μεθόδου που μπορεί να παρέχει ιδιότητες στο επίπεδο της μεθόδου όπως description, headers, query string parameters και responses.

Οι ιδιότητες resourceTypes και traits ορίζονται στο επίπεδο root του ορισμού του API. Η τιμή τους είναι ένας πίνακας χαρτών. Σε κάθε χάρτη τα κλειδιά είναι τα ονόματα των resourceTypes και traits και οι τιμές τους είναι οι ορισμοί των resourceTypes και traits αντίστοιχα.

```
#%RAML 0.8
title: Example API
version: v1
resourceTypes:
  - collection:
      usage: This resourceType should be used for any collection of items
      description: The collection of <<resourcePathName>>
      get:
        description: Get all <<resourcePathName>>, optionally filtered
      post:
        description: Create a new <<resourcePathName | !singularize>>
traits:
  - secured:
      usage: Apply this to any method that needs to be secured
      description: Some requests require authentication.
      queryParameters:
        access_token:
          description: Access Token
          type: string
          example: ACCESS_TOKEN
          required: true
```

3.2.4 Ασφάλεια

Τα περισσότερα REST APIs έχουν έναν ή περισσότερους μηχανισμούς για να ασφαλίσουν την πρόσβαση στα δεδομένα, για αιτήματα ταυτοποίησης και για να αποφασίζουν το επίπεδο πρόσβασης και ορατότητας των δεδομένων.

Η ιδιότητα securitySchemes ορίζεται στο επίπεδο root του API. Πρέπει να χρησιμοποιείται για να καθορίζει τους μηχανισμούς ασφαλείας ενός API, συμπεριλαμβανομένων των απαραίτητων ρυθμίσεων και μεθόδων πιστοποίησης που το API υποστηρίζει.

Στο ακόλουθο παράδειγμα το API του Dropbox υποστηρίζει πιστοποίηση μέσω OAuth 2.0 και OAuth 1.0.

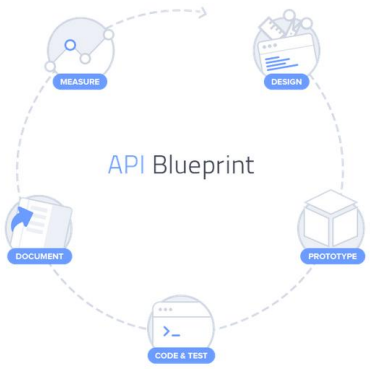
```
#%RAML 0.8
title: Dropbox API
version: 1
baseUri: https://api.dropbox.com/{version}
securitySchemes:
    - oauth_2_0:
        description: |
            Dropbox supports OAuth 2.0 for authenticating all API requests.
        type: OAuth 2.0
        describedBy:
            headers:
                Authorization:
                    description: |
                       Used to send a valid OAuth 2 access token. Do not
use
                       with the "access_token" query string parameter.
                    type: string
            queryParameters:
                access_token:
                    description: |
                       Used to send a valid OAuth 2 access token. Do not
use                                    together with the "Authorization"
header
                    type: string
            responses:
                401:
                    description: |
                        Bad or expired token. This can happen if the user
or Dropbox revoked or expired an access token. To fix, you should re-
authenticate the user.
                403:
                    description: |
                        Bad OAuth request (wrong consumer key, bad nonce,
expired timestamp...). Unfortunately, re-authenticating the user won't help
here.
        settings:
          authorizationUri: https://www.dropbox.com/1/oauth2/authorize
          accessTokenUri: https://api.dropbox.com/1/oauth2/token
          authorizationGrants: [ code, token ]
    - oauth_1_0:
        description:|
            OAuth 1.0 continues to be supported for all API requests, but
OAuth 2.0 is now preferred.
        type: OAuth 1.0
        settings:
          requestTokenUri: https://api.dropbox.com/1/oauth/request_token
          authorizationUri: https://www.dropbox.com/1/oauth/authorize
          tokenCredentialsUri: https://api.dropbox.com/1/oauth/access_token
    - customHeader:
        description:|
            A custom
```

3.3 API-Blueprint

Το API Blueprint είναι μια περιγραφική γλώσσα για APIs που βασίζεται στην τεκμηρίωση (documentation). Είναι ένα σύνολο σημασιολογικών υποθέσεων πέρα από το απλό Markdown. (Το Markdown είναι ένα εργαλείο για web developers που μετατρέπει κείμενο σε HTML. Με αυτόν τον τρόπο επιτρέπει την μετατροπή απλού κειμένου που είναι ευανάγνωστο και εύκολο να γραφτεί σε σωστά δομημένη XHTML ή HTML.)

Το API Blueprint είναι ιδιαίτερα χρήσιμο όχι μόνο κατά τον σχεδιασμό του web API και της κατανοητής τεκμηρίωσής του αλλά και κατά την γρήγορη προτυποποίηση και συνεργασία. Είναι εύκολο να το μάθει κανείς και ακόμα ευκολότερο να το διαβάσει καθώς είναι μια μορφή απλού κειμένου.

Τα περισσότερα εργαλεία του API Blueprint έχουν πλήρως ελεύθερο κώδικα οπότε δεν χρειάζεται να «εγκλωβιστεί κάποιος σε μη επιθυμητές αγορές» για να υλοποιήσει το API του. Αυτό σημαίνει ότι ο καθένας μπορεί να ενσωματώσει το API Blueprint σε οποιοδήποτε προϊόν εμπορικό ή μη.



Επιπλέον το API Blueprint καλύπτει όλες τις απαιτήσεις του API σε όλη τη διάρκεια ζωής του. Ο σχεδιασμός του API είναι μόνο η αρχή. Το API Blueprint μπορεί να χρησιμοποιηθεί από μία πληθώρα εργαλείων από διαδραστική τεκμηρίωση, δημιουργία SDK, αποσφαλμάτωση (debugging), από server για την δοκιμαστική λειτουργία του API μέχρι αναλυτικά εργαλεία.

Ορισμένα εργαλεία που μπορούν να χρησιμοποιηθούν με το API Blueprint είναι:

- Apiary.io REST API Platform για την εύρεση, δημιουργία και ανταλλαγή APIs
- Dredd δοκιμαστικό εργαλείο
- cURL trace parser εργαλείο για ενδοσκόπηση HTTP
- RSpec API Blueprint για την αυτόματη δημιουργία τεκμηρίωσης για API
- Και άλλα

Γιατί υπερτερεί το API Blueprint σε σχέση με το:

- «κάν' το μόνος σου»

Δεν υπάρχει λόγος για την εκ νέου ανακάλυψη του τροχού και την σπατάλη πολύτιμου χρόνου. Στόχος είναι η επαναχρησιμοποίηση έτοιμων και λειτουργικών στοιχείων και η συγκέντρωση στις λειτουργίες του API. Το API Blueprint παρέχει χρήσιμα εργαλεία για όλη την διάρκεια ζωής του API και προωθεί την συζήτηση κάθε API με άλλους προγραμματιστές και παράγει αυτόματα τεκμηρίωση.

- JSON

Το JSON είναι μία δομή για μηχανές. Χρησιμοποιώντας το API Blueprint λύνονται τα προβλήματα σχετικά με το ποιος θα διορθώσει τον κώδικα και ποιος θα ανανεώσει την τεκμηρίωση καθώς θα είναι εύκολο για όλους.

- Google Docs

Ο κώδικας εξελίσσεται, δημιουργούνται νέες εκδόσεις και ελέγχεται. Το ίδιο πρέπει να γίνεται και με την τεκμηρίωση του API. Το API Blueprint είναι συνδεδεμένο με τον κώδικα, έχει τον ίδιο κύκλο ζωής και η ανανέωση γίνεται παράλληλα.

3.3.1 Η γλώσσα της API-Blueprint

3.3.1.1 Έγγραφο API Blueprint

Ένα έγγραφο σε API Blueprint είναι στην ουσία ένα έγγραφο απλού κειμένου Markdown που περιγράφει ένα web API ή ένα μέρος του. Αυτό το έγγραφο είναι δομημένο σε λογικές ενότητες. Κάθε ενότητα έχει το χαρακτηριστικό της νόημα, περιεχόμενο και θέση στο έγγραφο. Όλες οι ενότητες είναι προαιρετικές όμως όταν μία ενότητα υπάρχει τότε πρέπει υποχρεωτικά να ακολουθεί τη δομή του εγγράφου API Blueprint.

Δομή εγγράφου Blueprint

- `0-1` **Metadata** section
- `0-1` **API Name & overview** section
- `0+` **Resource** sections
  - `0-1` **URI Parameters** section
  - `0-1` **Model** section
    - `0-1` **Headers** section
    - `0-1` **Body** section
    - `0-1` **Schema** section
  - `1+` **Action** sections
    - `0-1` **URI Parameters** section
    - `0+` **Request** sections
      - `0-1` **Headers** section
      - `0-1` **Body** section
      - `0-1` **Schema** section
    - `1+` **Response** sections
      - `0-1` **Headers** section
      - `0-1` **Body** section
      - `0-1` **Schema** section
- `0+` **Resource Group** sections
  - `0+` *Resource* ***sections***

Ο αριθμός πριν από το όνομα κάθε ενότητας δείχνει πόσες φορές μπορεί να εμφανιστεί η συγκεκριμένη ενότητα στο έγγραφο.

3.3.1.2 Ενότητα Blueprint

Κάθε ενότητα αντιπροσωπεύει μία λογική μονάδα του API Blueprint. Για παράδειγμα την περίληψη, ένα σύνολο από resources ή ορισμούς των resources. Γενικά μία ενότητα ορίζεται χρησιμοποιώντας μία λέξη κλειδί στην οντότητα του Markdown. Ανάλογα με την ενότητα η λέξη κλειδί γράφεται είτε ως επικεφαλίδα οντότητας Markdown είτε ως μέλος μίας λίστας. Ο ορισμός μιας ενότητας μπορεί να περιέχει επιπλέον μεταβλητά στοιχεία όπως το αναγνωριστικό της.

Υπάρχουν δύο ειδικές ενότητες που αναγνωρίζονται από τη θέση τους στο έγγραφο αντί από μία λέξη κλειδί. Η ενότητα Metadata και η ενότητα API Name and Overview.

Παράδειγμα: ενότητες που καθορίζονται από τον τίτλο

```
# <keyword>

 ...

# <keyword>

 ...
```

Παράδειγμα: ενότητες που καθορίζονται από λίστες

```
+ <keyword>

 ...

+ <keyword>

 ...
```

Δομή ενοτήτων

Υπάρχουν πολλοί και διαφορετικοί τύποι API Blueprint ενοτήτων. Μία γενική δομή ενότητας που καθορίζεται από μία λέξη κλειδί περιλαμβάνει ένα αναγνωριστικό όνομα, μία ενότητα περιγραφής και εμφωλευμένες ενότητες ή ειδικά σχεδιασμένο περιεχόμενο.

Παράδειγμα: δομή ενοτήτων που καθορίζονται από τον τίτλο

```
# <keyword> <identifier>

<description>

<specific content>

<nested sections>
```

Παράδειγμα: δομή ενοτήτων που καθορίζονται από λίστες

```
+ <keyword> <identifier>

    <description>

    <specific content>

    <nested sections>
```

Λέξεις κλειδιά

Οι ακόλουθες δεσμευμένες λέξεις κλειδιά χρησιμοποιούνται στον ορισμό ενοτήτων:

Λέξεις κλειδιά για επικεφαλίδες

- `Group`
- HTTP methods (π.χ `GET, POST, PUT, DELETE`...)
- URI templates (π.χ `/resource/{id}`)
- Συνδυασμός μίας μεθόδου HTTP και ενός URI Template (π.χ `GET /resource/{id}`)

Λέξεις κλειδιά για λίστες

- `Request`
- `Response`
- `Body`
- `Schema`
- `Model`
- `Header` & `Headers`
- `Parameter` & `Parameters`
- `Values`

Αναγνωριστικό (Identifier)

Ο ορισμός μιας ενότητας μπορεί ή πρέπει να περιέχει το αναγνωριστικό της ενότητας. Το αναγνωριστικό αυτό είναι οποιαδήποτε ακολουθία μη κενών χαρακτήρων εκτός από τους [,], (,). Επιπλέον ένα αναγνωριστικό δεν μπορεί να περιέχει καμία από τις λέξεις κλειδιά.

Παράδειγμα

```
Adam's Message 42
```

```
my-awesome-message_2
```

Περιγραφή

Η περιγραφή μιας ενότητας είναι οποιοδήποτε αυθαίρετο περιεχόμενο σε μορφή Markdown που ακολουθεί την ενότητα του ορισμού. Είναι δυνατό να χρησιμοποιηθεί οποιαδήποτε Markdown επικεφαλίδα ή στοιχείο λίστας στην ενότητα περιγραφής αρκεί να μην συγκρούεται με κάποια από τις δεσμευμένες λέξεις κλειδιά.

Εμφωλευμένες ενότητες

Μία ενότητα μπορεί να περιέχει μία ή περισσότερες άλλες εμφωλευμένες ενότητες. Για να εμφωλευτεί μία ενότητα πρέπει ανάλογα με τον τύπο της να αυξηθεί το επίπεδο του τίτλου της ή η θέση της μέσα στη λίστα. Οτιδήποτε ανάμεσα στην αρχή της τρέχουσας ενότητας μέχρι την αρχή της επόμενης που βρίσκεται στο ίδιο επίπεδο θεωρείται μέρος της τρέχουσας ενότητας. Το ποιες ενότητες μπορούν να εμφωλευθούν και που, εξαρτάται από την συγκεκριμένη ενότητα που θα τις περιέχει.

Παράδειγμα: Εμφωλευμένη header-defined ενότητα

```
# <section definition>
 ...

## <nested section definition>
 ...
```

Παράδειγμα: Εμφωλευμένη list-defined ενότητα

```
+ <section definition>
     ...

    + <nested section definition>
     ...
```

3.3.2 Πρότυπα ενοτήτων

Στη συνέχεια υπάρχει ανάλυση κάθε ενότητας. Ορισμένες που χαρακτηρίζονται ως abstract αποτελούν τη βάση για άλλες ενότητες και επομένως δεν μπορούν να χρησιμοποιηθούν άμεσα.

1. Named section

- Abstract
- Ενότητες – γονείς: ποικίλουν
- Εμφωλευμένες ενότητες: ποικίλουν
- Οντότητα Markdown: επικεφαλίδα, λίστα
- Κληρονομεί από: κανέναν

Ορισμός

Ορίζεται από μία λέξη κλειδί που ακολουθείται από ένα προαιρετικό όνομα ενότητας – αναγνωριστικό και βρίσκεται σε οντότητα Markdown ως κεφαλίδα ή λίστα.

```
# <keyword> <identifier>
```

```
+ <keyword> <identifier>
```

Περιγραφή

Η ενότητα Named είναι η βάση για τις περισσότερες ενότητες του API Blueprint. Αποτελεί την γενική ενότητα και έτσι αποτελείται από το όνομα της ενότητας, την περιγραφή και εμφωλευμένα τμήματα ή περιεχόμενο ειδικής μορφής.

Παράδειγμα

```
# <keyword> Section Name
This the `Section Name` description.

- one
- **two**
- three

<nested sections> |  <formatted content>
```

2. Asset section

- Abstract
- Ενότητες – γονείς : ποικίλουν
- Εμφωλευμένες ενότητες: καμία
- Οντότητα Markdown: λίστα
- Κληρονομεί από :κανέναν

Ορισμός

Ορίζεται από μία λέξη κλειδί στην λίστα του Markdown

```
+ <keyword>
```

Περιγραφή

Η ενότητα Asset είναι η βασική ενότητα για ατομικά δεδομένα στο API Blueprint. Το περιεχόμενο σε αυτή την ενότητα αναμένεται να είναι ένα block προεπεξεργασμένου κώδικα.

Παράδειγμα

```
+ <keyword>

        {
            "message": "Hello"
        }
```

3. Payload section

- Abstract
- Ενότητες – γονείς : ποικίλουν
- Εμφωλευμένες ενότητες: 0-1 Headers section, 0-1 Body section, 0-1 Schema section
- Οντότητα Markdown: λίστα
- Κληρονομεί από : Named section

Ορισμός

Ορίζεται από μία λέξη κλειδί στην λίστα Markdown. Η λέξη κλειδί μπορεί να ακολουθείται από κάποιο αναγνωριστικό. Ο ορισμός μπορεί να περιέχει το media-type του payload σε παρένθεση.

```
+ <keyword> <identifier> (<media type>)
```

Περιγραφή

Η ενότητα payload αντιπροσωπεύει την πληροφορία που μεταφέρεται σε αιτήσεις και ανταποκρίσεις HTTP μηνυμάτων. Το payload αποτελείται από προαιρετικές μετά-πληροφορίες σε μορφή HTTP επικεφαλίδων και από προαιρετικό περιεχόμενο στη μορφή σώματος HTTP. Επιπλέον, στο API Blueprint το payload περιλαμβάνει μία μορφή περιγραφής και επικύρωσης.

Η ενότητα payload μπορεί να έχει συνδεδεμένο το media type της. Το media type του payload αντιπροσωπεύει τα δεδομένα που λαμβάνονται ή στέλνονται στη μορφή μίας HTTP `Content-Type` επικεφαλίδας. Όταν καθορίζεται καλό θα είναι το payload να περιλαμβάνει μία εμφωλευμένη ενότητα Body.

Αυτή η ενότητα πρέπει να περιλαμβάνει τουλάχιστον μία από τις ακόλουθες εμφωλευμένες ενότητες:

- Headers section
- 0-1 Body section
- 0-1 Schema section

Αν δεν υπάρχει εμφωλευμένη ενότητα το περιεχόμενο της ενότητας payload θεωρείται περιεχόμενο της ενότητας Body

Παράδειγμα

```
+ <keyword> Payload Name (application/json)

    This the `Payload Name` description.

    + Headers
      ...

    + Body
```

```
    ...

   + Schema
   ...
```

4. Headers section

- Ενότητες – γονείς : payload section
- Εμφωλευμένες ενότητες: καμία
- Οντότητα Markdown: λίστα
- Κληρονομεί από : καμία

Ορισμός

Ορίζεται από τη λέξη κλειδί `Headers` στη λίστα Markdown

```
+ Headers
```

Περιγραφή

Προσδιορίζει το HTTP message-headers από την ενότητα payload του γονέα. Το περιεχόμενο αυτής της ενότητας αναμένεται να είναι ένα block προεπεξεργασμένου κώδικα με το ακόλουθο συντακτικό:

```
<HTTP header name>: <value>
```

Ένα HTTP header ανά γραμμή.

Παράδειγμα

```
+ Headers

        Accept-Charset: utf-8
        Connection: keep-alive
        Content-Type: multipart/form-data, boundary=AaB03x
```

5. Body section

- Ενότητες – γονείς : payload section
- Εμφωλευμένες ενότητες: καμία
- Οντότητα Markdown: λίστα
- Κληρονομεί από : Asset section

Ορισμός

Ορίζεται από τη λέξη κλειδί `Body` στη λίστα Markdown

```
+ Body
```

Περιγραφή

Προσδιορίζει το HTTP message-body του payload section

Παράδειγμα

```
+ Body
        {
            "message": "Hello"
        }
```

6. Schema section

- Ενότητες – γονείς : payload section
- Εμφωλευμένες ενότητες: καμία
- Οντότητα Markdown: λίστα
- Κληρονομεί από : Asset section

Ορισμός

Ορίζεται από τη λέξη κλειδί `Schema` στη λίστα Markdown

```
+ Schema
```

Περιγραφή

Προσδιορίζει μία μορφή επικύρωσης για το HTTP message-body του payload section του γονέα.

7. Metadata section

- Ενότητες – γονείς : καμία
- Εμφωλευμένες ενότητες: καμία
- Οντότητα Markdown: ειδική
- Κληρονομεί από : καμία

Ορισμός

Ζεύγη κλειδιού – τιμής. Το κλειδί χωρίζεται από την τιμή του με άνω κάτω τελεία (:). Ένα ζεύγος ανά γραμμή. Ξεκινάει στην αρχή του εγγράφου και τελειώνει με το πρώτο Markdown στοιχείο που δεν αναγνωρίζεται ως ζεύγος κλειδιού – τιμής.

Περιγραφή

Τα κλειδιά μεταδεδομένων και οι τιμές τους είναι tool-specific και πρέπει να γίνει αναφορά στην τεκμηρίωση του αντίστοιχου εργαλείου για την λίστα των υποστηριζόμενων κλειδιών.

Παράδειγμα

```
HOST: http://blog.acme.com
FORMAT: 1A
```

8. API name & overview section

- Ενότητες – γονείς : καμία
- Εμφωλευμένες ενότητες: καμία
- Οντότητα Markdown: ειδική, επικεφαλίδα
- Κληρονομεί από : Named section

Ορισμός

Ορίζεται από την πρώτη Markdown επικεφαλίδα στο έγγραφο blueprint εκτός αν αντιπροσωπεύει τον ορισμό κάποιας άλλης ενότητας.

Περιγραφή

Όνομα και περιγραφή του API

Παράδειγμα

```
# Basic ACME Blog API
Welcome to the **ACME Blog** API. This API provides access to the **ACME
Blog** service.
```

9. Resource group section

- Ενότητες – γονείς : καμία
- Εμφωλευμένες ενότητες: 0+ Resource section
- Οντότητα Markdown: επικεφαλίδα
- Κληρονομεί από : Named section

Ορισμός

Ορίζεται από την λέξη κλειδί `Group` ακολουθούμενη από το όνομα του group (αναγνωριστικό)

```
# Group <identifier>
```

Περιγραφή

Αυτή η ενότητα αντιπροσωπεύει ένα σύνολο πόρων (Resource Sections). Μπορεί να έχει μία ή περισσότερες εμφωλευμένες Resource Sections.

Παράδειγμα

```
# Group Blog Posts

## Resource 1 [/resource1]
 ...

# Group Authors
Resources in this groups are related to **ACME Blog** authors.

## Resource 2 [/resource2]
 ...
```

10. Resource section

- Ενότητες – γονείς :Resource group section
- Εμφωλευμένες ενότητες: 0-1 Parameters section, 0-1 Model section, 1+ Action section
- Οντότητα Markdown: επικεφαλίδα
- Κληρονομεί από : Named section

Ορισμός

Ορίζεται από ένα RFC 6570 URI πρότυπο

```
# <URI template>
```

ή ορίζεται από το όνομα ενός πόρου ακολουθούμενο από το πρότυπο URI μέσα σε αγκύλες [].

```
# <identifier> [<URI template>]
```

ή ορίζεται από μία μέθοδο HTTP ακολουθούμενη από το πρότυπο URI

```
# <HTTP request method> <URI template>
```

Περιγραφή

Ένα resource API όπως καθορίζεται από το URI του ή ένα σύνολο από resources που αντιστοιχούν στο πρότυπο URI τους.
Αυτή η ενότητα περιλαμβάνει τουλάχιστον μία εμφωλευμένη ενότητα Action και επιπλέον μπορεί να περιλαμβάνει τις ακόλουθες εμφωλευμένες ενότητες:

- Model section
- URI parameters section
- Επιπλέον Action sections

Παράδειγμα

```
# Blog Posts [/posts/{id}]
Resource representing **ACME Blog** posts.
```

```
# /posts/{id}
```

```
# GET /posts/{id}
```

11. Resource model section

- Ενότητες – γονείς :Resource section
- Εμφωλευμένες ενότητες: όπως στο payload section
- Οντότητα Markdown: λίστα
- Κληρονομεί από : Payload section

Ορισμός

Ορίζεται από την λέξη κλειδί `Model` ακολουθούμενη προαιρετικά από ένα media type

```
+ Model (<media type>)
```

Περιγραφή

Μία υποδειγματική παρουσίαση των πόρων στην μορφή payload.

Παράδειγμα

```
# My Resource [/resource]
+ Model (text/plain)

        Hello World

## Retrieve My Resource [GET]
+ Response 200

    [My Resource][]
```

12. URI parameters section

- Ενότητες – γονείς :Resource section |Action section
- Εμφωλευμένες ενότητες: καμία
- Οντότητα Markdown: λίστα
- Κληρονομεί από : καμία, ειδικά

Ορισμός

Ορίζεται από την λέξη κλειδί `Parameters` γραμμένη σαν αντικείμενο λίστας Markdown

```
+ Parameters
```

Περιγραφή

Αναφορά των παραμέτρων URI της ενότητας γονέα.

Αυτή η ενότητα πρέπει να αποτελείται μόνο από αντικείμενα εμφωλευμένων λιστών και δεν πρέπει να περιέχει άλλα στοιχεία. Μία λίστα για κάθε παράμετρο URI.

```
+ <parameter name> = `<default value>` (required | optional , <type>,
`<example value>`) ... <description>

    <additional description>

    + Values
```

```
        + `<enumeration element 1>`
        + `<enumeration element 2>`
        ...
        + `<enumeration element N>`
```

Όπου,

`<parameter name>`	είναι το όνομα της παραμέτρου όπως γράφεται στο URI του Resource Section (π.χ. "id").
`<description>`	είναι οποιαδήποτε προαιρετική περιγραφή της παραμέτρου σε μορφή Markdown.
`<additional description>`	είναι οποιαδήποτε επιπρόσθετη προαιρετική περιγραφή της παραμέτρου σε μορφή Markdown.
`<default value>`	είναι μια προαιρετική προεπιλεγμένη τιμή της παραμέτρου – μία τιμή που χρησιμοποιείται όταν δεν ορίζονται ξεκάθαρα άλλες τιμές.
`<example value>`	είναι μία προαιρετική ενδεικτική τιμή της παραμέτρου (π.χ. `1234`).
`<type>`	είναι ο προαιρετικός τύπος της παραμέτρου όπως αναμένεται από το API (π.χ. "number").
`Values`	είναι η προαιρετική αρίθμηση των πιθανών τιμών
`<enumeration element n>`	αντιπροσωπεύει ένα στοιχείο της αρίθμησης.
`required`	είναι προαιρετικό και αναφέρεται όταν μία παράμετρος είναι απαραίτητη (αυτό είναι προεπιλεγμένο).
`optional`	είναι προαιρετικό και αναφέρεται όταν μία παράμετρος είναι προαιρετική.

Παράδειγμα

```
# GET /posts/{id}
```

```
+ Parameters
    + id ... Id of a post.
```

```
+ Parameters
    + id (number) ... Id of a post.
```

```
+ Parameters
```

```
    + id (required, number, `1001`) ... Id of a post.
```

```
+ Parameters
    + id = `20` (optional, number, `1001`) ... Id of a post.
```

```
+ Parameters
    + id (string)

        Id of a Post

        + Values
            + `A`
            + `B`
            + `C`
```

13. Action section

- Ενότητες – γονείς :Resource section |Action section
- Εμφωλευμένες ενότητες: καμία
- Οντότητα Markdown: λίστα
- Κληρονομεί από : καμία, ειδικά

Ορισμός

Ορίζεται από μία μέθοδο HTTP

```
## <HTTP request method>
```

ή ορίζεται από το όνομα μιας δράσης ακολουθούμενο από μία μέθοδο HTTP σε αγκύλες [].

```
# <identifier> [<HTTP request method>]
```

Περιγραφή

Ορισμός μίας τουλάχιστον ολοκληρωμένης HTTP συναλλαγής όπως γίνεται με την ενότητα του γονέα. Μία ενότητα Action μπορεί να αποτελείται από πολλαπλά παραδείγματα HTTP συναλλαγών για μία συγκεκριμένη μέθοδο HTTP.

Η ενότητα αυτή μπορεί να περιλαμβάνει μία εμφωλευμένη ενότητα URI Parameters η οποία θα περιγράφει τις URI παραμέτρους ειδικά για την δράση. Καλό θα ήταν να περιλαμβάνει τουλάχιστον μία εμφωλευμένη ενότητα Response. Επιπλέον όμως μπορεί να περιέχει και άλλες εμφωλευμένες Request και Response ενότητες.

Οι εμφωλευμένες Request και Response ενότητες μπορούν να διαταχθούν σε ομάδες όπου κάθε ομάδα θα αντιπροσωπεύει ένα παράδειγμα συναλλαγής. Η ομάδα του πρώτου παραδείγματος ξεκινά με την πρώτη εμφωλευμένη Request ή Response ενότητα. Οι επόμενες ομάδες ξεκινούν με την πρώτη εμφωλευμένη Request ενότητα που ακολουθεί μία ενότητα Response.

Πολλαπλές εμφωλευμένες ενότητες με Request και Response μέσα στο ίδιο παράδειγμα συναλλαγής πρέπει να έχουν διαφορετικά αναγνωριστικά.

Παράδειγμα

```
# Blog Posts [/posts{?limit}]
 ...

## Retrieve Blog Posts [GET]
Retrieves the list of **ACME Blog** posts.

+ Parameters
    + limit (optional, number) ... Maximum number of posts ot retrieve

+ Response 200
        ...

### Create a Post [POST]
+ Request
        ...

+ Response 201
        ...
```

Για πολλαπλά παραδείγματα

```
# Resource [/resource]
## Create Resource [POST]

+ request A
        ...

+ response 200
        ...

+ request B
        ...

+ response 200
        ...

+ response 500
        ...

+ request C
        ...

+ request D
        ...
```

```
+ response 200
        ...
```

Στο παράδειγμα αυτό έχουμε τρία παραδείγματα συναλλαγών για μία δράση:

1. 1ο παράδειγμα: request A, response 200
2. 2ο παράδειγμα: request B, responses 200 and 500
3. 3ο παράδειγμα: requests C and D, response 200

14. Request Section

- Ενότητες – γονείς : Action section
- Εμφωλευμένες ενότητες: όπως στην ενότητα payload
- Οντότητα Markdown: λίστα
- Κληρονομεί από : Payload section

Ορισμός

Ορίζεται από την λέξη κλειδί `Request` ακολουθούμενη από ένα προαιρετικό αναγνωριστικό

```
+ Request <identifier> (<Media Type>)
```

Περιγραφή

Ένα παράδειγμα από HTTP request-message και payload

Παράδειγμα

```
+ Request Create Blog Post (application/json)

        { "message" : "Hello World." }
```

15. Response section

- Abstract
- Ενότητες – γονείς : Action section
- Εμφωλευμένες ενότητες: όπως στην ενότητα payload
- Οντότητα Markdown: λίστα
- Κληρονομεί από : Payload section

Ορισμός

Ορίζεται από την λέξη κλειδί `Response` . Ο ορισμός της ενότητας Response πρέπει να περιλαμβάνει έναν HTTP status code σαν αναγνωριστικό της.

```
+ Response <HTTP status code> (<Media Type>)
```

Περιγραφή
Ένα παράδειγμα από HTTP response-message και payload

Παράδειγμα

```
+ Response 201 (application/json)

            { "message" : "created" }
```

3.4 Σύγκριση

Στην συνέχεια ακολουθούν πίνακες στους οποίους συγκρίνονται και αντιστοιχίζονται συνοπτικά τα τρία πρότυπα που αναλύθηκαν προηγουμένως.

3.4.1 Βασικές πληροφορίες

	Swagger	**RAML**	**API-Blueprint**
Format	JSON	YAML	Markdown
Spec License	ASL 2.0	ASL 2.0 / TM	MIT
Διατίθεται σε	GitHub	GitHub	GitHub
Χρηματοδοτείται από	Reverb	Mulesoft	Apiary
Τελευταία έκδοση	1.2	0.8	1A3
Πρώτη εμφάνιση	Ιούλιος 2011	Σεπτέμβριος 2013	Απρίλιος 2013
Εμπορική προσφορά	όχι	Ναι	Ναι
Προσέγγιση στον σχεδιασμό API	Bottom - up	Top-down	Top-down

3.4.2 Στοιχεία σχετικά με την μοντελοποίηση του REST

	Swagger	RAML	API-Blueprint
Resources	√	√	√
Methods/Actions	√ (operation)	√ (method)	√ (action)
Query Parameters	√	√	√
URL Parameters / path	√	√	√
Header Parameters	√	√	√
Representations (κωδικοί κατάστασης, τύποι MIME)	√	√	√
Τεκμηρίωση	√	√	√
Πιστοποίηση	Βασική, API key, OAuth 2	Βασική, Digest, OAuth 1&2	
Representation Metadata	JSON Schema	Οποιαδήποτε (inline / external)	Οποιαδήποτε (inline)
Εμφωλευμένα resources		√	√
Composition/ Inheritance		Traits, Resource Types	Resource Models
Εισαγωγή αρχείων		√	
API version metadata	√	√	
Sample Representations		√	√

3.4.3 Στατιστικά χρήσης

	Swagger	RAML	API-Blueprint
Ερωτήσεις στο stackoverflow	596	18	37
Τα περισσότερα αστέρια GitHub (σε ένα project)	1859	478	613
Συνολικά project στο GitHub	340	52	72
Αναζήτηση στο Google με το όνομά τους + REST	5 εκατομμύρια	64 χιλιάδες	807 χιλίαδες

Από τον παραπάνω πίνακα είναι ξεκάθαρο ότι το πρότυπο που είναι περισσότερο διαδεδομένο και έχει τους περισσότερους χρήστες είναι το Swagger. Στη δεύτερη θέση βρίσκεται το API-Blueprint και τελευταίο το RAML. Βέβαια αξίζει να αναφέρουμε ότι το Swagger έχει εμφανιστεί από το 2011 ενώ τα άλλα πρότυπα με τα οποία συγκρίνεται έχουν τεθεί στην διάθεση των προγραμματιστών το 2013. Το γεγονός αυτό δίνει ένα μεγάλο πλεονέκτημα στο Swagger και αυτό φαίνεται από τους αριθμούς που παρουσιάζονται στον πίνακα.

3.5 Συμπερασματικά

Swagger:

- Πλεονεκτήματα: μεγάλος αριθμός χρηστών, καλά εργαλεία στο επίπεδο του σχεδιασμού κώδικα και μεγάλος ρυθμός εισόδου νέων χρηστών.
- Μειονεκτήματα: βασίζεται στη σχεδίαση bottom-up και υπάρχει έλλειψη από εξελιγμένες δομές μεταδεδομένων.

RAML:

- Πλεονεκτήματα: καλά σχεδιαστικά εργαλεία online και ώριμη υποστηρικτική υποδομή. Επίσης υπάρχουν χρήσιμες και εξελιγμένες δομές
- Μειονεκτήματα: Ιδιαίτερα νέο, μικρός ρυθμός εισόδου νέων χρηστών και υπάρχει έλλειψη εργαλείων στο επίπεδο ανάπτυξης κώδικα

API-Blueprint:

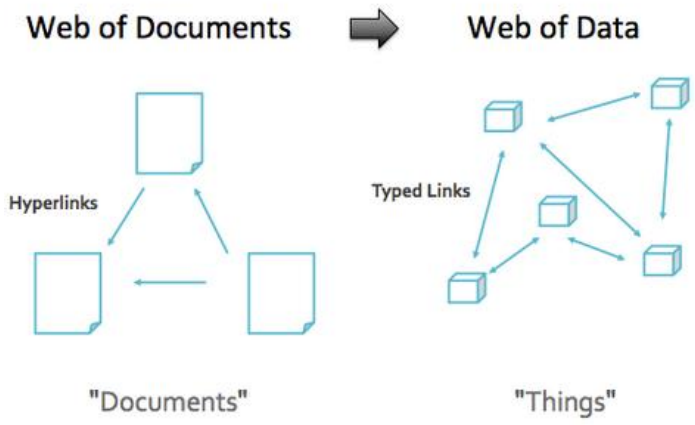
- Πλεονεκτήματα: καλά σχεδιαστικά εργαλεία online και κοινότητα χρηστών αρκετά βοηθητική
- Μειονεκτήματα: Ιδιαίτερα νέο, μικρός ρυθμός εισόδου νέων χρηστών και υπάρχει έλλειψη εργαλείων στο επίπεδο ανάπτυξης κώδικα καθώς και έλλειψη σε σύνθετες δομές.

Στο σημείο αυτό πρέπει να αναφερθεί ότι τα τρία πρότυπα που αναλύθηκαν δεν είναι τα μοναδικά για την μοντελοποίηση των REST APIs. Υπάρχουν και άλλα, τα οποία δεν είναι τόσο διαδεδομένα και για τον λόγο αυτό δεν αναφέρονται εκτενώς στην παρούσα εργασία.

4. Hydra – Σημασιολογικός ιστός

4.1 Σημασιολογικός ιστός

Την τελευταία δεκαετία έχει παρατηρηθεί ένα έντονο ενδιαφέρον για τον σημασιολογικό ιστό, ο οποίος επεκτείνει το δίκτυο των εγγράφων (web of documents) σε δίκτυο των δεδομένων (web of data). Αυτή η τεχνολογία εφαρμόζει πρότυπα που βασίζονται στο web για την κωδικοποίηση συνόλων δεδομένων και την σύνδεσή τους με άλλα δημοσιευμένα δεδομένα ώστε οι εφαρμογές να μπορούν να τα ανακτούν από πολλές και διαφορετικές πηγές. Επίσης παρέχει πρότυπα για την κωδικοποίηση γενικών γνώσεων πάνω σε οντολογίες βελτιώνοντας την αυτόματη συλλογιστική (για παράδειγμα βελτιωμένες αναζητήσεις). Στο κεφάλαιο αυτό θα εισαχθούν τα Linked Data και οι σχετικές σημασιολογικές τεχνολογίες καθώς και πώς αυτά μπορούν να χρησιμοποιηθούν σε εφαρμογές web. Το παράδειγμα που θα χρησιμοποιηθεί είναι η δημιουργία ενός μουσικού portal.



4.1.1 Σημασιολογικές Τεχνολογίες και Linked Data

Στο σημείο αυτό, θα περιγραφούν ένα σύνολο από τεχνολογίες που επιτρέπουν στα δεδομένα να δημοσιεύονται στο διαδίκτυο και να αναζητούνται αποτελεσματικά. Συγκριτικά με τις μηχανές αναζήτησης όπως η Google και η Yahoo, που βασίζονται στην εύρεση κειμένου, αυτές οι τεχνολογίες είναι σημασιολογικές. Αυτό σημαίνει ότι οι πληροφορίες δεν αναπαριστώνται σε φυσική γλώσσα π.χ. αγγλικά, ελληνικά αλλά σε μοντέλα δεδομένων που βασίζονται σε γράφους και επιτρέπουν επεκτάσεις, ενσωματώσεις, συμπεράσματα και ομοιόμορφη αναζήτηση. Όπως είπαμε σαν μία ρεαλιστική εφαρμογή των σημασιολογικών τεχνολογιών θα θεωρήσουμε ένα μουσικό portal που ο καθένας θα μπορεί να αναζητά πληροφορίες σχετικές με τον κόσμο της μουσικής. Για παράδειγμα:

- Εύρεση ενός κονσέρτου του Beethoven από κινέζικη ορχήστρα
- Εύρεση μίας φωτογραφίας του μαέστρου αυτού του κονσέρτου
- Εύρεση Βρετανών rock μουσικών που παντρεύτηκαν Σκανδιναβές

Η προσπάθεια για να απαντηθούν αυτά τα ερωτήματα μέσω μηχανών αναζήτησης που βασίζονται σε κείμενο είναι αναξιόπιστη καθώς μπορούμε εξίσου να πάρουμε σαν αποτελέσματα ένα κονσέρτο που ο σολίστ ήταν Κινέζος ή στην τρίτη περίπτωση έναν rock μουσικό που παίζει σκανδιναβική μουσική. Με τη χρήση σημασιολογικών τεχνολογιών πόροι όπως το αρχείο ήχου της παράστασης ή η φωτογραφία του μαέστρου μπορούν να αποθηκευτούν χρησιμοποιώντας το Resource Description Framework (RDF). Σε αυτό το πλαίσιο, μπορούν να δοθούν στα resources του Beethoven, του κονσέρτου του, της ορχήστρας και του μαέστρου ονόματα. Ονόματα μπορούν επίσης να δοθούν σε τύπους ή κλάσεις από resources (συνθέτες, κονσέρτα…) και σε σχέσεις ή ιδιότητες που συνδέουν τα resources (π.χ. το "γράφτηκε από " ανάμεσα στο κονσέρτο και τον συνθέτη). Με έναν συλλογισμό πάνω σε δεδομένα που είναι κωδικοποιημένα με αυτόν τον τρόπο ένα σύστημα αναζήτησης μπορεί να επιβεβαιώσει ότι μία παράσταση δόθηκε από την συμφωνική ορχήστρα του Πεκίνου, ότι η ορχήστρα αυτή βρίσκεται στο Πεκίνο, ότι το Πεκίνο βρίσκεται στην Κίνα κλπ συνδυάζοντας γεωγραφικές και μουσικές γνώσεις ώστε να βρει την απάντηση.

Στον σχεδιασμό αυτών των σημασιολογικών τεχνολογιών, σημείο κλειδί ήταν να μείνει ελεύθερη η ονοματολογία των resources και οι ιδιότητές τους ώστε τα ονόματα να συμμορφώνονται με το format των web resource names, δεδομένου ότι είναι Uniform Resource Identifiers (URIs).

http://rdf.freebase.com/ns/en.ludwig_van_beethoven
http://dbpedia.org/resource/Ludwig_van_Beethoven
http://musicbrainz.org/artist/1f9df192-a621-4f54-8850-2c5373b7eac9#_
http://data.nytimes.com/N30866506154608358173

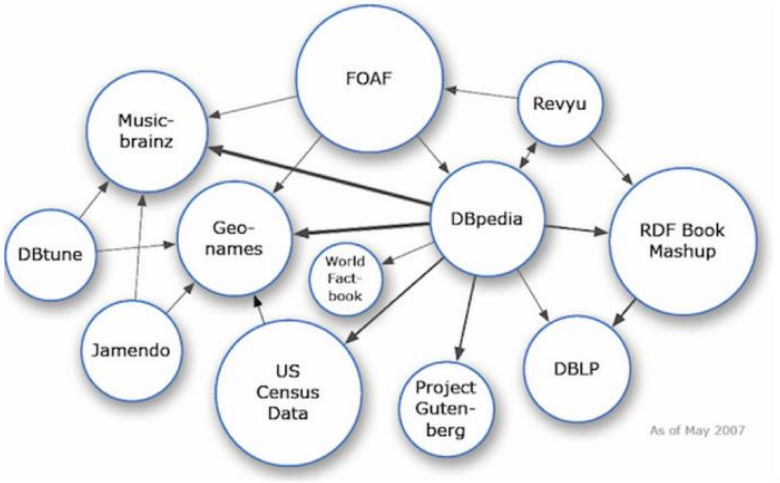
Και τα τέσσερα αυτά θα μπορούσαν να ήταν ονόματα για τον Beethoven, δείχνοντας ότι ένα URI δεν χρειάζεται να μπορεί να διαβαστεί από τον άνθρωπο. Παρόλα αυτά τα αναγνωριστικά πρέπει να είναι επιλύσιμα από τις αναπαραστάσεις RDF που περιλαμβάνουν ευανάγνωστες για τον άνθρωπο ετικέτες. Αν πρέπει να συνδυαστούν δεδομένα από διαφορετικές πηγές είναι σημαντικό να εδραιωθούν links που να δηλώνουν για παράδειγμα ότι τα προηγούμενα τέσσερα URIs είναι συνώνυμα. Αυτές οι δηλώσεις που μπορούν να εκφραστούν σε RDF παρέχουν ένα μέσο ώστε δεδομένα που δημοσιεύονται από πολλούς ανθρώπους και οργανισμούς να μπορούν να συνδυαστούν σε Linked Data.

Τα Linked Data είναι αποτέλεσμα της ανάμειξης προηγούμενων ιδεών και τεχνολογιών όπως hypertext, βάσεις δεδομένων, οντολογίες, γλώσσες markup, internet και παγκόσμιος ιστός. Οι τεχνολογίες αυτές εφαρμόζονται μέσα από συγκεκριμένα πρωτόκολλα και γλώσσες με οικεία ονόματα όπως είναι τα HTTP, URI, XML, RDF, RDFS, OWL και SPARQL

4.1.2 Εισαγωγή στα Linked Data

Το 2006 ο Berners-Lee πρότεινε ορισμένες αρχές για την έκδοση δεδομένων στον σημασιολογικό ιστό. Από τότε ο όγκος των δεδομένων έχει αυξηθεί από περίπου 2 δις τριπλέτες το 2007 σε πάνω από 3 δις το 2011 συνδεδεμένες μεταξύ τους με πάνω από 500

εκατομμύρια RDF links ο κύριος σκοπός των οποίων είναι να εδραιώσουν τις σχέσεις ανάμεσα σε URIs που αναφέρονται στο ίδιο πράγμα. Μέσα από αυτά τα links τα δεδομένα συνδυάζονται σε ένα απέραντο σώμα δεδομένων γνωστό ως cloud.



Στην εικόνα αυτή παρουσιάζεται το διάγραμμα του cloud των Linked Data για το 2007, στο οποίο οι κόμβοι αντιπροσωπεύουν σύνολα δεδομένων και τα links αντιπροσωπεύουν σύνολα από RDF τριπλέτες μέσω των οποίων τα URIs ενός συνόλου αντιστοιχίζονται με τα άλλα σύνολα. Επομένως το link ανάμεσα στο DBpedia και το MusicBrainz σημαίνει ότι το DBpedia περιέχει όχι μόνο RDF που δίνουν πληροφορίες για τον κόσμο αλλά και RDF που συνδέουν κάποια ονόματα του DBpedia με τα συνώνυμά τους στο MusicBrainz. Η επόμενη τριπλέτα συνδέει τα δύο ονόματα για τους Beatles.

```
<http://musicbrainz.org/artist/b10bbbfc-cf9e-42e0-be17-e2c3e1d2600d>
<http://www.w3.org/2002/07/owl#sameAs>
<http://dbpedia.org/resource/The_Beatles>.
```

Επειδή η σχέση sameAs είναι αντιμεταθετική και μεταβατική το X sameAs Y και το Y sameAs Z σημαίνει ότι και X sameAs Z. Με αυτόν τον τρόπο δημιουργούνται λίστες από συνώνυμα.

Αρχές

Το 2006 ο Berners-Lee διατύπωσε τέσσερις απλές αρχές για την έκδοση δεδομένων στο διαδίκτυο. Οι αρχές αυτές αποτελούν περισσότερο μία οδηγία παρά κανόνες που πρέπει υποχρεωτικά να ακολουθούνται. Η ιδέα είναι ότι όσο περισσότεροι ακολουθούν αυτές τις αρχές τόσο περισσότερα δεδομένα θα μπορούν να χρησιμοποιηθούν από άλλους.

Σε συντομία οι αρχές αυτές είναι:

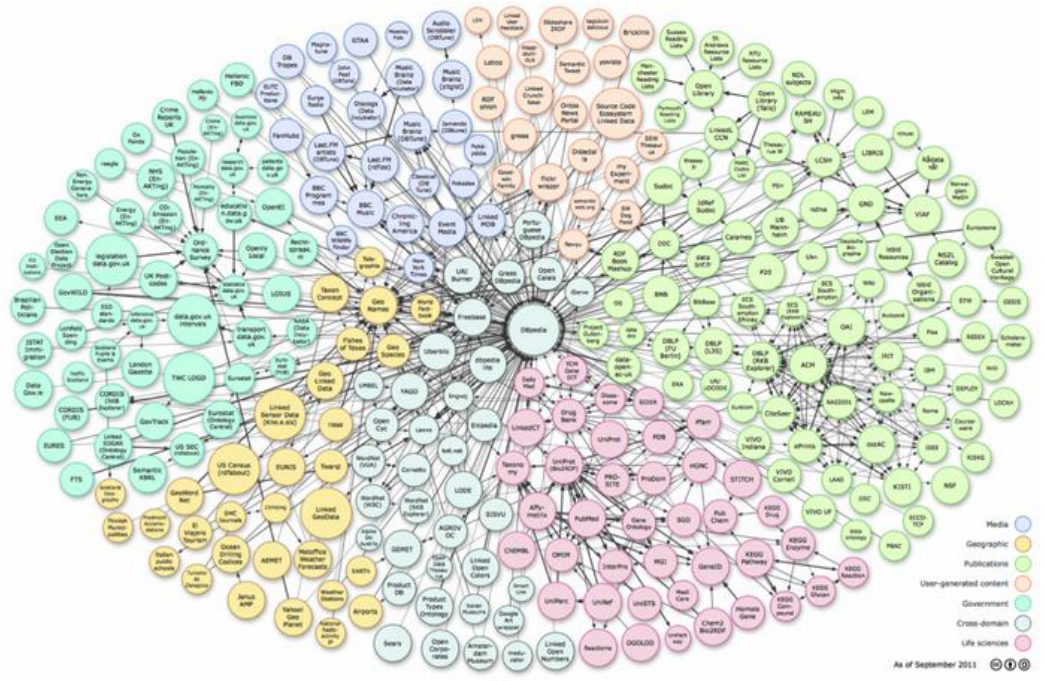
1. Η χρήση URIs για τον χαρακτηρισμό πραγμάτων
2. Η χρήση HTTP URIs ώστε οι άνθρωποι να μπορούν να ψάχνουν αυτά τα ονόματα
3. Όταν κάποιος αναζητά ένα URI να παρέχονται χρήσιμες πληροφορίες χρησιμοποιώντας πρότυπα όπως RDF, RDFS, SPARQL
4. Η εισαγωγή link για άλλα URIs ώστε να ανακαλύπτονται περισσότερα πράγματα.

Η λογική για αυτές τις αρχές είναι προφανής. Χρησιμοποιώντας URI για να ταυτοποιήσουμε ανεξάρτητα στοιχεία, κλάσεις και ιδιότητες αποκτάμε ονόματα με διπλή σημασία. Εκτός από την αναφορά στο σχετικό αντικείμενο, μας δίνουν μία τοποθεσία στο διαδίκτυο που μπορούμε να ψάξουμε για πληροφορίες για αυτό το αντικείμενο. Οι άλλοι τρόποι ονοματολογίας έχουν μόνο την πρώτη σημασία. Βέβαια για να επωφεληθούμε από ένα όνομα που λειτουργεί και ως web διεύθυνση θα πρέπει το URI να μην είναι ένα "σπασμένο" link. Πρέπει να δείχνει σε σχετικές πληροφορίες που θα είναι κωδικοποιημένες σε μία από τις αναμενόμενες μορφές.

Επιπλέον, για την αξιολόγηση των υπαρχόντων δομών δεδομένων ο Berners-Lee εισήγαγε το 2010 μία βαθμολογική κλίμακα με πέντε αστέρια όπως αυτή των ξενοδοχείων η οποία βασιζόταν στις αρχές που αναφέρθηκαν προηγουμένως.

- One-star (*): τα δεδομένα είναι διαθέσιμα στο διαδίκτυο για όλους
- Two-star (**): τα δεδομένα είναι δομημένα και μπορούν να διαβαστούν από μηχανή
- Three-star (***): τα δεδομένα δεν έχουν ειδική- μοναδική μορφή
- Four-star(****): τα δεδομένα χρησιμοποιούν μόνο ανοιχτά πρότυπα από το W3C (RDF, SPARQL)
- Five-star (*****): τα δεδομένα είναι συνδεδεμένα με άλλους παρόχους δεδομένων.

Σε σύγκριση με το διάγραμμα των linked Data για το 2007 που παρουσιάστηκε προηγουμένως ακολουθεί το αντίστοιχο διάγραμμα για το 2011 που δείχνει την επέκταση που έχει λάβει χώρα αυτά τα χρόνια



Ορισμένα web sites που βασίζονται στην τεχνολογία του σημασιολογικού δικτύου είναι τα http://marbles.sourceforge.net/#sthash.2cj4ka4H.dpuf http://sig.ma/#sthash.2cj4ka4H.dpuf , http://wiki.dbpedia.org/DBPediaMobile#sthash.2cj4ka4H.dpuf αλλά και άλλα από τους New York Times, το BBC Music κλπ.

4.2 HYDRA

Η Hydra είναι μία προσπάθεια που γίνεται για την απλοποίηση της ανάπτυξης των hypermedia-driven Web APIs. Τα δύο δομικά στοιχεία της Hydra είναι το JSON-LD και το Hydra Core Vocabulary.

Το JSON-LD είναι μία μορφή σειριοποίησης (serialization) που χρησιμοποιείται στην επικοινωνία μεταξύ του server και των πελατών του. Το Hydra Core Vocabulary αντιπροσωπεύει το κοινό λεξιλόγιο που μοιράζονται. Ο προσδιορισμός ορισμένων concepts που χρησιμοποιούνται συχνά στα web APIs, μπορεί να λειτουργήσει ως βάση για την δημιουργία υπηρεσιών web που διαθέτουν τα πλεονεκτήματα του REST. Δηλαδή χαλαροί δεσμοί, εύκολη συντήρηση, δυνατότητα εξέλιξης και ανάπτυξης. Επιπλέον επιτρέπει την δημιουργία γενικών πελατών API σε αντίθεση με την ανάγκη για εξειδικευμένους πελάτες για κάθε απλό API.

Η Hydra είναι ένα project που βρίσκεται σε εξέλιξη και δεν έχει ολοκληρωθεί ακόμη. Για τον λόγο αυτό τα στοιχεία που παρουσιάζονται στη συνέχεια περιγράφουν την πιο πρόσφατη έκδοσή της και είναι πιθανό να αλλάξουν στο μέλλον.

4.2.1 Hydra Core Vocabulary

Εισαγωγή

Το να τα βγάζει κανείς πέρα με την όλο και αυξανόμενη ποσότητα δεδομένων αποτελεί μία σημαντική πρόκληση. Για να μειωθεί ο φόρτος πληροφοριών που τίθεται στους ανθρώπους τα συστήματα αρχίζουν να συνδέονται άμεσα μεταξύ τους. Ανταλλάσσουν, αναλύουν και χρησιμοποιούν τεράστιες ποσότητες δεδομένων χωρίς καμία ανθρώπινη παρέμβαση. Οι λύσεις που υπάρχουν σήμερα όμως, δεν αξιοποιούν όλες τις δυνατότητες που προσφέρει το διαδίκτυο και αγνοούν την δυναμική των τεχνολογιών Linked Data.

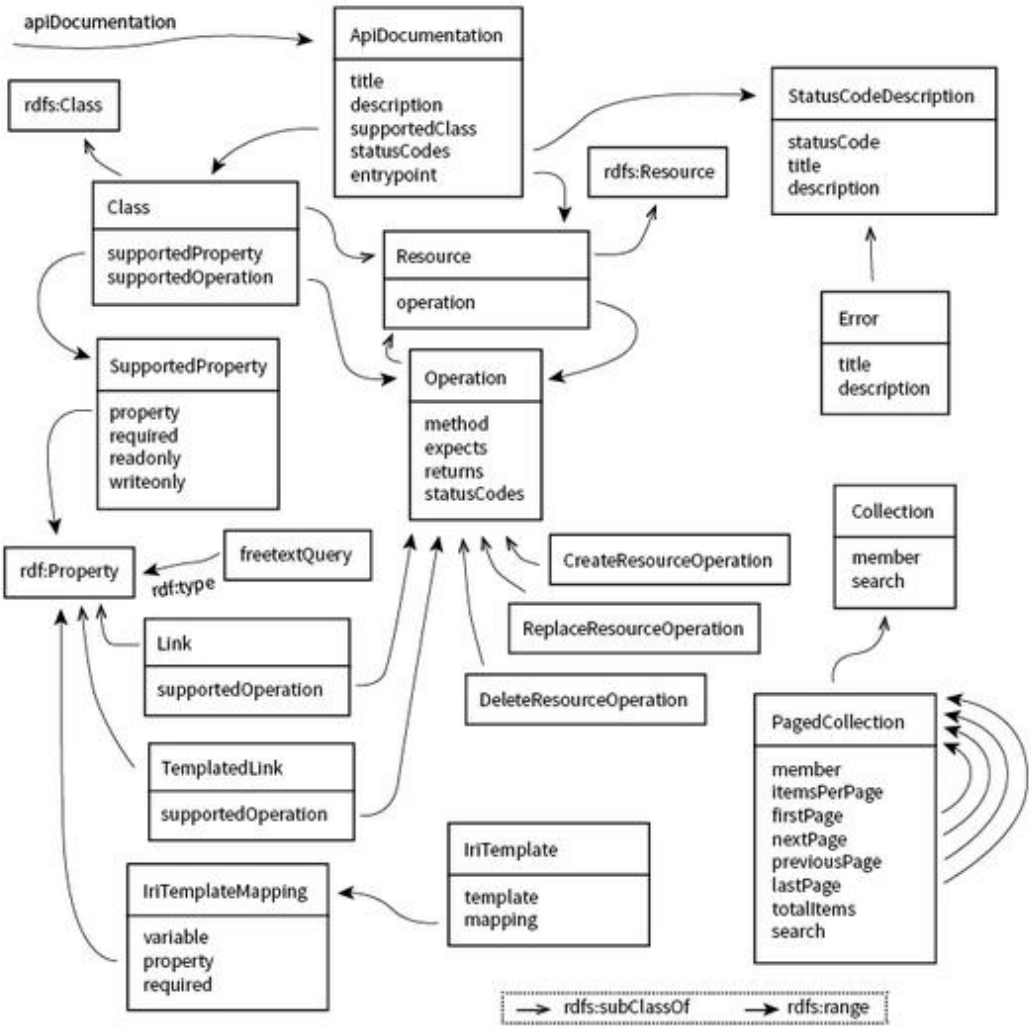
Ο συνδυασμός του αρχιτεκτονικού στυλ REST και οι αρχές των Linked Data προσφέρουν ευκαιρίες για την εξέλιξη του διαδικτύου των μηχανών με τρόπο όμοιο με αυτόν που προσέφερε το hypertext στο διαδίκτυο των ανθρώπων. Τα περισσότερα στοιχεία για αυτό υπάρχουν ήδη και βρίσκονται σε εφαρμογή αλλά σπάνια χρησιμοποιούνται μαζί. Η Hydra προσπαθεί να γεφυρώσει αυτό το κενό. Επιτρέπει στα δεδομένα να εμπλουτιστούν με στοιχεία που μπορούν να διαβαστούν από μηχανές και επιτρέπουν την αλληλεπίδραση. Αυτό όχι μόνο αντιμετωπίζει το πρόβλημα των Linked Data που είναι σχεδόν αποκλειστικά για ανάγνωση αλλά ετοιμάζει το έδαφος για νέα web APIs. Το γεγονός ότι επιτρέπει την δημιουργία σύνθετων τμημάτων κώδικα σημαίνει ότι τα μοντέλα αλληλεπίδρασης των web APIs θα μπορούν να επαναχρησιμοποιηθούν σε μεγάλο βαθμό.

Μία γρήγορη ματιά στην Hydra

Η βασική ιδέα πίσω από την Hydra είναι η δημιουργία ενός λεξιλογίου που θα επιτρέπει σε έναν server να δείχνει μία έγκυρη αλλαγή κατάστασης σε έναν client. Ο client τότε μπορεί να χρησιμοποιήσει αυτή την πληροφορία ώστε να κάνει HTTP requests τα οποία τροποποιούν την κατάσταση του server ώστε να επιτευχθεί ένας συγκεκριμένος στόχος. Αφού όλες οι πληροφορίες σχετικά με τις έγκυρες αλλαγές καταστάσεων ανταλλάσσονται μεταξύ μηχανών κατά την διάρκεια της εκτέλεσής τους αντί να είναι καταγεγραμμένες στον

κώδικα του client από τον σχεδιασμό του, οι clients μπορούν να αποσυζευχθούν από τον server και να προσαρμόζονται στις αλλαγές ευκολότερα.

Στο παρακάτω διάγραμμα παρουσιάζεται το βασικό λεξιλόγιο της Hydra. Σκοπός του είναι να δείξει πως χρησιμοποιείται η Hydra και όχι τον ακριβή ορισμό της.



The Hydra core vocabulary

Χρησιμοποιώντας την Hydra

Σε αυτή την ενότητα θα παρουσιαστεί η χρήση της Hydra μέσα από ένα παράδειγμα. Στο API του παραδείγματος οι χρήστες μπορούν να εισάγουν νέο θέμα, να επεξεργάζονται ή να διαγράφουν τα υπάρχοντα και να τα σχολιάζουν.

Προσθέτοντας δυνατότητες στις αναπαραστάσεις

Το API του παραδείγματος πρέπει να δείχνει αναπαραστάσεις από θέματα και σχόλια. Για να επιτρέψει την αλληλεπίδραση με αυτούς τους πόρους, ένας client πρέπει να ξέρει ποιες λειτουργίες υποστηρίζει ο server. Σε websites που προορίζονται για χρήση του ανθρώπου αυτές οι λειτουργίες συνήθως παρουσιάζονται ως links ή φόρμες και περιγράφονται σε φυσική γλώσσα. Όμως οι πληροφορίες αυτές δεν μπορούν να αναγνωριστούν εύκολα από μηχανές. Η λύση για αυτό είναι να μειωθεί η γλώσσα σε ένα μικρό σύνολο από ξεκάθαρες

ενότητες που μπορούν εύκολα να αναγνωριστούν από την μηχανή του client. Η Hydra προτυποποιεί τις ενότητες αυτές.

Ο πιο απλός και σημαντικός τρόπος για να προσφέρουμε δυνατότητες στο web είναι μέσω hyperlinks. Χωρίς αυτά θα ήταν αδύνατο να έχουμε πρόσβαση στο διαδίκτυο. Οι χρήστες επιλέγουν συνήθως το link ανάλογα με το κείμενο που το περιγράφει. Για να δώσουμε στις μηχανές μία αντίστοιχη ικανότητα κατανόησης, τα link μπορούν να σηματοδοτούνται με έναν τύπο link relation – ένα registered token ή ένα URI που προσδιορίζει την σημασιολογία του link. Στο ακόλουθο παράδειγμα βλέπουμε πως ένα link χρησιμοποιείται σε HTML για να δείξει σε ένα stylesheet

```
<link rel="stylesheet" href="http://www.example.com/styles.css" />
```

Στα Linked Data, ο τύπος link relation αντιστοιχεί στην ίδια την ιδιότητα. Γι αυτό ένα παράδειγμα σε JSON-LD θα γραφόταν

```
{
  "urn:iana:link-relations:stylesheet":
  { "@id": "http://www.example.com/styles.css" }
}
```

Γενικά, ένας client αποφασίζει εάν θα ακολουθήσει ένα link ή όχι με βάση τη σχέση του link που καθορίζει την σημασιολογία του. Υπάρχουν όμως clients όπως οι web crawlers που απλά ακολουθούν οποιοδήποτε link. Στην HTML αυτό σημαίνει ότι όλα τα links με την ετικέτα <a> ακολουθούνται.

Ενώ τα links είναι αρκετά για την δημιουργία APIs μόνο για ανάγνωση, για την δημιουργία web APIs που διαβάζονται και γράφονται απαιτούνται ισχυρότερα μέσα. Για τον λόγο αυτό η Hydra εισάγει την έννοια των λειτουργιών. Μία λειτουργία - Operation αντιπροσωπεύει τις πληροφορίες που είναι απαραίτητες για έναν client προκειμένου να δημιουργήσει ένα έγκυρο HTTP request ώστε να αλλάξει την κατάσταση του server. Επομένως η μόνη απαραίτητη ιδιότητα για μία λειτουργία είναι η HTTP μέθοδός της. Προαιρετικά είναι δυνατό να περιγράφει τις πληροφορίες που ο server αναμένεται να επιστρέψει συμπεριλαμβανομένων επιπλέον πληροφοριών σχετικά με τους κωδικούς κατάστασης HTTP που μπορεί να επιστραφούν. Αυτό βοηθά έναν προγραμματιστή στο να κατανοήσει τι να περιμένει όταν εκτελεί μία λειτουργία. Βέβαια αυτή η πληροφορία δεν θεωρείται πλήρης αλλά αποτελεί μία βοήθεια. Οι προγραμματιστές για παράδειγμα πρέπει να περιμένουν ότι μπορεί να επιστρέψουν και διαφορετικοί κωδικοί κατάστασης HTTP ώστε να προγραμματίσουν ανάλογα τους clients.

Η Hydra έχει τρείς κλάσεις προκαθορισμένων λειτουργιών με ονόματα *CreateResourceOperation*, *ReplaceResourceOperation*, και *DeleteResourceOperation*. Όπως λένε και τα ονόματά τους, μπορούν να χρησιμοποιηθούν για να εισάγουν απλή CRUD λειτουργικότητα. Περισσότερο εξειδικευμένες λειτουργίες μπορούν να δημιουργηθούν εύκολα με την τροποποίηση της βασικής κλάσης Operation.

Το ακόλουθο παράδειγμα δείχνει πώς αναπαραστάσεις μπορούν να επαυξηθούν με πληροφορίες που διευκολύνουν τους clients να αλληλεπιδράσουν μαζί τους. Ένας client θα

μπορούσε να καταλάβει ότι το resource στο ακόλουθο παράδειγμα μπορεί να διαγραφεί στέλνοντας ένα HTTP DELETE request.

```
{
  "@context": "http://www.w3.org/ns/hydra/context.jsonld",
  "@id": "/an-issue",
  "title": "An exemplary issue representation",
  "description": "This issue can be deleted with an HTTP DELETE request",
  "operation": [
    {
      "@type": "DeleteResourceOperation",
      "method": "DELETE"
    }
  ]
}
```

Στο παράδειγμα αυτό βλέπουμε ότι η Hydra συνδυάζει ιδιότητες όπως operation και method και τιμές όπως DeleteResourceOperation με URL που ξεκάθαρα αναφέρονται σε αυτές. Το ευρύτερο πλαίσιο είναι κοινό για πολλές αναπαραστάσεις σε web APIs και για τον λόγο αυτό έχει νόημα να μειώσουμε το μέγεθος της απάντησης σε ένα πλαίσιο που θα μπορεί εύκολα να φορτωθεί από τους clients.

Τεκμηρίωση ενός Web API

Στα web APIs οι περισσότερες αναπαραστάσεις μοιάζουν πολύ. Επιπλέον τα resources συχνά υποστηρίζουν τις ίδιες λειτουργίες. Για τον λόγο αυτό γίνεται μία προσπάθεια για την συγκέντρωση όλων αυτών των πληροφοριών σε ένα κεντρικό documentation. Παραδοσιακά, αυτό υπάρχει στην φυσική γλώσσα και οι προγραμματιστές ενσωματώνουν αυτή τη γνώση στους clients. Η Hydra πλησιάζει αυτό το ζήτημα προσπαθώντας να κάνει αυτό το κεντρικό documentation επεξεργάσιμο από μηχανές. Το γεγονός ότι όλοι οι ορισμοί μπορούν να αναγνωριστούν από URL δίνει την δυνατότητα επαναχρησιμοποίησης σε μεγάλο βαθμό.

Η κλάση της Hydra ApiDocumentation θέτει τις βάσεις για την περιγραφή ενός web API. Όπως φαίνεται στο ακόλουθο παράδειγμα, η Hydra περιγράφει ένα API δίνοντάς του έναν τίτλο, μία σύντομη περιγραφή και τεκμηριώνοντας το κύριο σημείο πρόσβασης σε αυτό. Επιπλέον, μπορούν να περιγραφούν οι κλάσεις που υποστηρίζονται από το web API και άλλες πληροφορίες σχετικά με τους κωδικούς κατάστασης που μπορούν να επιστραφούν. Αυτές οι πληροφορίες μπορούν να χρησιμοποιηθούν για την αυτόματη δημιουργία τεκμηριώσεων σε φυσική γλώσσα.

```
{
  "@context": "http://www.w3.org/ns/hydra/context.jsonld",
  "@id": "http://api.example.com/doc/",
  "@type": "ApiDocumentation",
  "title": "The name of the API",
  "description": "A short description of the API",
  "entrypoint": "URL of the API's main entry point",
  "supportedClass": [
```

```
    ... Classes known to be supported by the Web API ...
  ],
  "statusCodes": [
    ... Additional information about HTTP status codes ...
  ]
}
```

Αφού η Hydra χρησιμοποιεί κλάσεις για να περιγράψει τις πληροφορίες που αναμένονται ή στέλνονται από μία λειτουργία, ορίζει και έναν τρόπο για να περιγράφει τις ιδιότητες που υποστηρίζονται από μία κλάση. Αυτό φαίνεται στο ακόλουθο παράδειγμα. Αντί να αναφερθούν οι ιδιότητες άμεσα, το supportedProperty αποτελεί μία ενδιάμεση δομή δεδομένων. Αυτό δίνει την δυνατότητα να ορίσουμε εάν μία συγκεκριμένη ιδιότητα απαιτείται, εάν είναι read-only ή write-only ανάλογα με την κλάση με την οποία έχει συσχετιστεί.

```
{
  "@context": "http://www.w3.org/ns/hydra/context.jsonld",
  "@id": "http://api.example.com/doc/#Comment",
  "@type": "Class",
  "title": "The name of the class",
  "description": "A short description of the class.",
  "supportedProperty": [
    ... Properties known to be supported by the class ...
    {
      "@type": "SupportedProperty",
      "property": "#property", // The property
      "required": true, // Is the property required in a request to be
valid?
      "readonly": false, // Can the property's value be modified or is it
read-only?
      "writeonly": true // Can the property's value be retrieved or is it
write-only?
    }
  ]
}
```

Εύρεση ενός Web API που βασίζεται στην Hydra

Το πρώτο βήμα όταν προσπαθούμε να αποκτήσουμε πρόσβαση σε ένα web API είναι να βρούμε ένα σημείο εισόδου. Συνήθως αυτό γίνεται κοιτάζοντας την τεκμηρίωση στην κεντρική σελίδα του εκδότη του API. Η Hydra επιτρέπει το κεντρικό σημείο εισόδου του API να εντοπίζεται αυτόματα εάν ο εκδότης του API το σηματοδοτήσει με ένα ειδικό HTTP Link Header όπως ορίζεται στο [RFC5988]. Ένας client της Hydra θα έψαχνε για ένα Link Header με relation type http://www.w3.org/ns/hydra/core#apiDocumentation .

Στο ακόλουθο παράδειγμα, ένας client της Hydra απλά μπαίνει στο homepage ενός εκδότη API (http://www.example.com) για να βρει το σημείο εισόδου του API. Ο client μπορεί να κάνει ένα HTTP GET ή HEAD request. Η διαφορά μεταξύ των δύο είναι ότι το πρώτο μπορεί να επιστρέψει ένα message-body ως απάντηση ενώ το δεύτερο όχι.

```
HEAD / HTTP/1.1
Host: www.example.com

====================================

HTTP/1.1 200 OK
...
Content-Type: application/ld+json
Link: <http://api.example.com/doc/>;
rel="http://www.w3.org/ns/hydra/core#apiDocumentation"
```

Η απάντηση στο προηγούμενο παράδειγμα περιέχει ένα HTTP Link Header που δείχνει στο http://api.example.com/doc/ . Ανακτώντας αυτό το resource ο client παίρνει ένα Hydra API documentation που ορίζει το κύριο σημείο εισόδου του API:

```
GET /doc/ HTTP/1.1
Host: api.example.com
Accept: application/ld+json

====================================

HTTP/1.1 200 OK
...
Content-Type: application/ld+json

{
  "@context": "http://www.w3.org/ns/hydra/context.jsonld",
  "@id": "http://api.example.com/doc/",
  "title": "The example.com API",
  "entrypoint": "http://api.example.com/",
  ...
}
```

Βέβαια στις περισσότερες περιπτώσεις το σημείο εισόδου είναι ήδη γνωστό στον client. Για τον λόγο αυτό η εύρεση του API documentation χρησιμοποιώντας HTTP Link Headers συνήθως δεν είναι απαραίτητη.

Collections

Σε πολλές περιπτώσεις έχει νόημα να παρουσιάζονται μαζί resources που αναφέρονται σε ένα σύνολο από συσχετιζόμενα με κάποιο τρόπο resources. Τα αποτελέσματα μιας αναζήτησης ή ένα βιβλίο διευθύνσεων είναι δύο παραδείγματα. Για να απλοποιηθούν αυτές οι περιπτώσεις η Hydra ορίζει δύο κλάσεις: *hydra:Collection* και *hydra:PagedCollection*.

Η *hydra:Collection* χρησιμοποιείται για να αναφερθούμε σε ένα σύνολο από resources με τον εξής τρόπο:

```
{
  "@context": "http://www.w3.org/ns/hydra/context.jsonld",
  "@id": "http://api.example.com/an-issue/comments",
  "@type": "Collection",
  "member": [
    {
      "@id": "/comments/429"
    },
    {
      "@id": "/comments/781",
      "title": "Properties may be embedded directly in the collection"
    },
    ...
  ]
}
```

Όπως φαίνεται, τα στοιχεία-μέλη μπορούν να αποτελούνται είτε μόνο από ένα link ή να περιέχουν και κάποιες ιδιότητες. Σε μερικές περιπτώσεις το να εισάγονται ορισμένες ιδιότητες των μελών άμεσα στη συλλογή είναι ιδιαίτερα βοηθητικό καθώς έτσι μπορεί να μειωθεί ο αριθμός των HTTP requests που απαιτούνται ώστε να ληφθούν οι απαραίτητες πληροφορίες για την επεξεργασία του αποτελέσματος.

Επειδή οι συλλογές μπορεί να γίνουν πολύ μεγάλες, τα web APIs συχνά επιλέγουν να χωρίσουν μία συλλογή σε σελίδες. Στην Hydra αυτό μπορεί να υλοποιηθεί με το *hydra:PagedCollection.* Εκτός από την ιδιότητα member μία *PagedCollection* μπορεί να περιέχει links για την *firstPage*, *nextPage*, *previousPage*, ή *lastPage* όπως και πληροφορίες σχετικά με το *itemsPerPage* και το *totalItems* όπως παρουσιάζεται στο ακόλουθο παράδειγμα.

```
{
  "@context": "http://www.w3.org/ns/hydra/context.jsonld",
  "@id": "http://api.example.com/an-issue/comments?page=3",
  "@type": "PagedCollection",
  "totalItems": "4980",
  "itemsPerPage": "10",
  "firstPage": "/an-issue/comments?page=1",
  "nextPage": "/an-issue/comments?page=4",
  "previousPage": "/an-issue/comments?page=2",
  "lastPage": "/an-issue/comments?page=498",
  "member": [
    ... the members of this PagedCollection ...
  ]
}
```

Templated Links

Μερικές φορές είναι αδύνατον για έναν server να παράξει ένα URL γιατί το URL αυτό εξαρτάται από πληροφορίες που είναι γνωστές μόνο από τον client. Μία τυπική περίπτωση είναι τα URL που επιτρέπουν στον client να κάνει αίτημα για αναζήτηση στον server. Σε μία

τέτοια περίπτωση ο server δεν μπορεί να φτιάξει το URL επειδή δεν ξέρει το αίτημα για το οποίο ενδιαφέρεται ο client. Αυτό όμως που μπορεί να κάνει είναι να δώσει στον client ένα template για να δημιουργήσει ένα τέτοιο URL. Στη Hydra η κλάση *IriTemplate* χρησιμοποιείται για αυτόν τον σκοπό.

Το *IriTemplate* αποτελείται από ένα *template*, του οποίου το συντακτικό αναλύεται στο [RFC6570], και από ένα σύνολο από *mappings.* Κάθε *IriTemplateMapping* ταιριάζει τη μεταβλητή (*variable)* που χρησιμοποιείται στην φόρμα με ένα *property* και μπορεί προαιρετικά να ορίζει εάν αυτή η μεταβλητή είναι απαραίτητη (*required*) ή όχι .

```
{
  "@context": "http://www.w3.org/ns/hydra/context.jsonld",
  "@type": "IriTemplate",
  "template": "http://api.example.com/issues{?q}",
  "mapping": [
    {
      "@type": "IriTemplateMapping",
      "variable": "q",
      "property": "hydra:freetextQuery",
      "required": true
    }
  ]
}
```

Το παράδειγμα αυτό αντιστοιχίζει την μεταβλητή q στην ιδιότητα *freetextQuery* της Hydra και ορίζει ότι είναι απαραίτητη.

Όπως η κλάση *Link* της Hydra επιτρέπει τον ορισμό ιδιοτήτων που αντιπροσωπεύουν Hyperlinks (όπως περιγράφηκε στην παράγραφο Προσθέτοντας δυνατότητες στις αναπαραστάσεις), έτσι η κλάση *TemplatedLink* επιτρέπει τον ορισμό ιδιοτήτων που η τιμή τους είναι IRI templates. Η Hydra έχει μία τέτοια ιδιότητα που ονομάζεται *search* και χρησιμοποιείται για την περιγραφή διαθέσιμων διεπαφών αναζήτησης.

```
{
  "@context": "http://www.w3.org/ns/hydra/context.jsonld",
  "@id": "hydra:search",
  "@type": "hydra:TemplatedLink"
}
```

Περιγραφή των HTTP κωδικών κατάστασης και λαθών

Οι κωδικοί κατάστασης HTTP έχουν καλά ορισμένη σημασιολογία και μπορούν να χρησιμοποιηθούν για να δείξουν το αποτέλεσμα μίας λειτουργίας. Όμως αρκετές φορές οι κωδικοί αυτοί δεν είναι αρκετά εξειδικευμένοι κάνοντας δύσκολο το να καταλάβουμε την πραγματική αιτία ενός λάθους. Για παράδειγμα η απάντηση 429 Too Many Requests μας ενημερώνει πολύ γενικά. Για την αντιμετώπιση αυτής της κατάστασης η Hydra ορίζει την κλάση StatusCodeDescription που επιτρέπει να σταλούν επιπλέον πληροφορίες με τον κωδικό κατάστασης HTTP.

```
{
  "@context": "http://www.w3.org/ns/hydra/context.jsonld",
  "@type": "StatusCodeDescription",
  "statusCode": 429,
  "title": "Too Many Requests",
  "description": "A maximum of 500 requests per hour and user is allowed.",
  ...
}
```

Ένας server μπορεί να στείλει κατευθείαν σε μία απάντησή του StatusCodeDescription. Όταν γίνεται αυτό καλό είναι να γράφεται σε κάποια υποκλάση ώστε η σημασία του να γίνεται πιο ξεκάθαρη. Η Hydra ορίζει μόνο μία τέτοια υποκλάση με το όνομα Error. Αυτό προσφέρει εκτεταμένο πλαίσιο για να φτάσουν λεπτομέρειες για το λάθος στον client.

```
HTTP/1.1 400 Bad Request
Content-Type: application/ld+json

{
  "@context": "http://www.w3.org/ns/hydra/context.jsonld",
  "@type": "Error",
  "title": "An error occurred",
  "description": "Typically, a specialization of this class is used in
practice.",
  ...
}
```

4.2.2 JSON – LD

Το Linked Data είναι ένας τρόπος για να δημιουργήσουμε ένα δίκτυο από δεδομένα κατανοητά από μηχανές ανάμεσα σε διαφορετικά έγγραφα και web sites. Επιτρέπει σε μία εφαρμογή να ξεκινήσει από ένα σημείο των Linked Data και ακολουθώντας ενσωματωμένα links να φτάσει σε άλλα τμήματά τους που βρίσκονται σε διαφορετικές ιστοσελίδες του διαδικτύου.

Το JSON – LD είναι ένα “ελαφρύ” συντακτικό για την σειριοποίηση των Linked Data σε JSON. Ο σχεδιασμός του δίνει τη δυνατότητα σε υπάρχοντα JSON να ερμηνευθούν ως Linked Data με ελάχιστες αλλαγές. Κύριος σκοπός του JSON – LD είναι να αποτελέσει έναν τρόπο για την χρήση των Linked Data σε διαδικτυακά προγραμματιστικά περιβάλλοντα, για την δημιουργία υπηρεσιών web που θα συνεργάζονται μεταξύ τους και για την αποθήκευση Linked Data σε μηχανές αποθήκευσης που βασίζονται σε JSON. Επειδή το JSON-LD είναι 100% συμβατό με το JSON, ο μεγάλος αριθμός από εργαλεία και βιβλιοθήκες JSON που έχουν αναπτυχθεί ως σήμερα μπορούν να επαναχρησιμοποιηθούν. Εκτός από όλα τα χαρακτηριστικά που προσφέρει το JSON, το JSON-LD εισάγει:

- Έναν μηχανισμό αναγνώρισης για JSON objects μέσω της χρήσης IRIs
- Έναν μηχανισμό στον οποίο η τιμή ενός JSON object μπορεί να αναφέρεται σε ένα JSON object ενός άλλου web site.
- Έναν τρόπο να σχετίζονται τύποι δεδομένων με τιμές όπως ημερομηνίες και χρόνοι
- Την δυνατότητα να εκφραστεί ένας ή περισσότεροι γράφοι, όπως ένα κοινωνικό δίκτυο, σε ένα έγγραφο.

Το JSON-LD σχεδιάστηκε ώστε να χρησιμοποιείται κατευθείαν ως JSON χωρίς γνώση για RDF. Επίσης σχεδιάστηκε ώστε να χρησιμοποιείται αν το επιθυμεί κανείς ως RDF σε συνδυασμό με άλλες τεχνολογίες για Linked Data όπως την SPARQL. Το συντακτικό του δεν προκαλεί προβλήματα σε συστήματα που τρέχουν ήδη JSON αλλά παρέχει μία αναβάθμιση σε αυτά.

Το JSON-LD ικανοποιεί τους ακόλουθους σχεδιαστικούς στόχους:

- Απλότητα: Για να χρησιμοποιηθεί το JSON-LD στην βασική του μορφή δεν απαιτούνται επιπλέον επεξεργαστές ή βιβλιοθήκες λογισμικού. Η γλώσσα διευκολύνει τους προγραμματιστές κατά την εκμάθησή της καθώς το μόνο που χρειάζεται να ξέρουν είναι JSON και δύο λέξεις κλειδιά, @context και @id, για να εφαρμόσουν τα βασικά της χαρακτηριστικά.
- Συμβατότητα: Ένα έγγραφο JSON-LD είναι πάντα ένα έγκυρο έγγραφο JSON. Αυτό διαβεβαιώνει ότι όλες οι βιβλιοθήκες του JSON λειτουργούν εξίσου και για τα έγγραφα JSON-LD.
- Εκφραστικότητα: Το συντακτικό σειριοποιεί τους κατευθυνόμενους γράφους. Αυτό διαβεβαιώνει ότι σχεδόν οποιοδήποτε μοντέλο από τον πραγματικό κόσμο μπορεί να εκφραστεί.
- Λακωνικότητα: Το συντακτικό του JSON-LD είναι πολύ λακωνικό και ευανάγνωστο από τον άνθρωπο, απαιτώντας την ελάχιστη δυνατή προσπάθεια από τον προγραμματιστή.
- Ελάχιστες τροποποιήσεις: Το JSON-LD κάνει μία ομαλή μετάβαση από τα υπάρχοντα συστήματα σε JSON. Στις περισσότερες περιπτώσεις το μόνο που χρειάζεται είναι η προσθήκη μιας γραμμής στο HTTP response. Αυτό επιτρέπει στους οργανισμούς που έχουν μεγάλη υποδομή σε JSON να χρησιμοποιήσουν τα χαρακτηριστικά του JSON-LD με έναν τρόπο που δεν θα επηρεάσει την καθημερινή τους λειτουργία και είναι οικείος στους υπάρχοντες πελάτες. Βέβαια υπάρχουν φορές που η αντιστοίχιση ενός JSON σε μορφή γράφου είναι μία σύνθετη διαδικασία. Ενώ οι μηδενικές μετατροπές είναι στόχος του JSON-LD, μερικές φορές αυτό δεν είναι δυνατό χωρίς η γλώσσα να γίνει σύνθετη. Το JSON-LD επικεντρώνεται στην απλότητα όπου αυτό είναι δυνατό.
- Χρήση ως RDF: Το JSON-LD μπορεί να χρησιμοποιηθεί από τους προγραμματιστές σαν ιδιωματική JSON, χωρίς την ανάγκη κατανόησης RDF. Επιπλέον το JSON-LD μπορεί να χρησιμοποιηθεί ως RDF, έτσι οι άνθρωποι που πρόκειται να χρησιμοποιήσουν JSON-LD μαζί με άλλα εργαλεία για RDF θα ανακαλύψουν ότι μπορεί να χρησιμοποιηθεί ως οποιοδήποτε άλλο συντακτικό RDF.

Μοντέλο δεδομένων

Γενικά, το μοντέλο δεδομένων που χρησιμοποιείται από το JSON-LD είναι ένας labeled, κατευθυνόμενος γράφος. Ο γράφος περιέχει κόμβους που συνδέονται με ακμές. Ένας κόμβος είναι συνήθως δεδομένα όπως string, αριθμός, τυποποιημένες τιμές όπως ημερομηνίες και χρόνοι ή IRI. Επίσης υπάρχει μία ειδική κλάση κόμβου που ονομάζεται blank node (κενός κόμβος) και συνήθως χρησιμοποιείται για να εκφράσει δεδομένα που δεν έχουν ένα καθολικό αναγνωριστικό όπως ένα IRI. Οι blank nodes ταυτοποιούνται χρησιμοποιώντας ένα αναγνωριστικό κενού κόμβου. Αυτό το απλό μοντέλο δεδομένων είναι απίστευτα προσαρμόσιμο και ισχυρό, με δυνατότητα να μοντελοποιήσει σχεδόν όλα τα είδη δεδομένων.

Δύο χαρακτηριστικές λέξεις κλειδιά για ένα έγγραφο JSON-LD είναι οι @context και @id.

@context
Χρησιμοποιείται για να ορίσει τα σύντομα ονόματα που χρησιμοποιούνται σε ένα JSON-LD έγγραφο. Τα ονόματα αυτά ονομάζονται terms και βοηθούν τους προγραμματιστές να εκφράσουν ορισμένα αναγνωριστικά συνοπτικά.

@id
Χρησιμοποιείται για να ορίσει μοναδικά τα *things* που περιγράφονται στο έγγραφο με IRIs ή με αναγνωριστικά κενού κόμβου.

Βασικά χαρακτηριστικά

Το JSON είναι ένα “ελαφρύ”, ανεξάρτητο από γλώσσα format για την ανταλλαγή δεδομένων. Είναι εύκολο να αναλυθεί και να δημιουργηθεί. Βέβαια είναι δύσκολο να ενσωματώσεις JSON από διαφορετικές πηγές καθώς τα δεδομένα μπορεί να περιέχουν κλειδιά που συγκρούονται με άλλες πηγές δεδομένων. Επιπλέον το JSON δεν έχει ενσωματωμένη υποστήριξη για hyperlinks που είναι βασικό συστατικό του διαδικτύου. Ακολουθεί ένα παράδειγμα από απλό JSON

```
{
  "name": "Manu Sporny",
  "homepage": "http://manu.sporny.org/",
  "image": "http://manu.sporny.org/images/manu.png"
}
```

Για τους ανθρώπους είναι προφανές ότι πρόκειται για έναν άνθρωπο που το όνομά του (name) είναι “Manu Sporny” και ότι η ιδιότητα homepage περιέχει το URL αυτού του ανθρώπου. Μία μηχανή δεν μπορεί να καταλάβει κάτι τέτοιο και μερικές φορές είναι δύσκολο ακόμη και για τον άνθρωπο. Αυτό το πρόβλημα μπορεί να λυθεί χρησιμοποιώντας ξεκάθαρα αναγνωριστικά για να δείξουμε το διαφορετικό νόημα και όχι απλά σύμβολα όπως “name” και ”homepage”.

Τα Linked Data, και το διαδίκτυο γενικότερα, χρησιμοποιούν IRIs (Internationalized Resource Identifiers) για ξεκάθαρη ταυτοποίηση. Η ιδέα είναι να χρησιμοποιούμε IRIs για να ορίζουμε ξεκάθαρα αναγνωριστικά σε δεδομένα που μπορεί να χρησιμοποιηθούν και από άλλους προγραμματιστές. Είναι χρήσιμο για όρους όπως name και homepage να δημιουργηθούν IRIs ώστε οι προγραμματιστές να χρησιμοποιούν κοινή ορολογία και μαζί

με τις μηχανές να βλέπουν τη σημασία των όρων που αναζητούν. Η διαδικασία αυτή είναι γνωστή ως IRI dereferencing.

Αξιοποιώντας το δημοφιλές λεξιλόγιο schema.org, το προηγούμενο παράδειγμα θα μπορούσε να γραφεί ως:

```
{
  "http://schema.org/name": "Manu Sporny",
  "http://schema.org/url": { "@id": "http://manu.sporny.org/" },    ← The
'@id' keyword means 'This value is an identifier that is an IRI'
  "http://schema.org/image":                    {                    "@id":
"http://manu.sporny.org/images/manu.png" }
}
```

Εδώ βλέπουμε ότι κάθε ιδιότητα είναι ξεκάθαρα ορισμένη από ένα IRI και ότι όλες οι τιμές που αντιπροσωπεύουν IRIs σημειώνονται από την λέξη κλειδί @id. Ενώ το έγγραφο αυτό είναι ένα έγκυρο έγγραφο JSON-LD που περιγράφει τα δεδομένα του με μεγάλη ακρίβεια είναι ταυτόχρονα υπερβολικά μακροσκελές και δύσκολο για να δουλέψουν πάνω του άνθρωποι. Για να λυθεί αυτό το θέμα το JSON-LD εισάγει την έννοια του context.

Context

Όταν δύο άνθρωποι μιλάνε μεταξύ τους, η συζήτηση λαμβάνει χώρα σε ένα κοινό περιβάλλον που ονομάζεται το context της συζήτησης. Αυτό το κοινό context επιτρέπει στους συμμετέχοντες να χρησιμοποιούν συντομεύσεις όπως το μικρό όνομα ενός κοινού φίλου ώστε να επικοινωνούν γρηγορότερα χωρίς όμως να χάνουν σε ακρίβεια. Το context στο JSON-LD λειτουργεί με τον ίδιο τρόπο. Επιτρέπει σε δύο εφαρμογές να επικοινωνούν μεταξύ τους πιο αποδοτικά χωρίς έλλειψη ακρίβειας. Στην ουσία το context χρησιμοποιείται ώστε να αντιστοιχίσει τους όρους στα IRIs τους. Για το προηγούμενο παράδειγμα το context θα μπορούσε να είναι το:

```
{
  "@context":
  {
    "name": "http://schema.org/name",     ← This means that 'name' is
shorthand for 'http://schema.org/name'
    "image": {
      "@id": "http://schema.org/image",    ← This means that 'image' is
shorthand for 'http://schema.org/image'
      "@type": "@id"   ← This means that a string value associated with
'image' should be interpreted as an identifier that is an IRI
    },
    "homepage": {
      "@id": "http://schema.org/url",    ← This means that 'homepage' is
shorthand for 'http://schema.org/url'
      "@type": "@id"   ← This means that a string value associated with
'homepage' should be interpreted as an identifier that is an IRI
    }
  }
}
```

Όπως παρατηρούμε η τιμή του ορισμού ενός όρου μπορεί να είναι είτε ένα απλό string που ένωνε τον όρο με το IRI είτε ένα JSON object.

Το context μπορεί είτε να γραφτεί άμεσα μέσα στο έγγραφο είτε να κληθεί. Αν υποθέσουμε ότι το context του προηγούμενου παραδείγματος βρίσκεται στο http://json-ld.org/contexts/person.jsonld μπορεί να κληθεί προσθέτοντας στο JSON-LD έγγραφο μία μόνο σειρά όπως φαίνεται στο παράδειγμα:

```
{
  "@context": "http://json-ld.org/contexts/person.jsonld",
  "name": "Manu Sporny",
  "homepage": "http://manu.sporny.org/",
  "image": "http://manu.sporny.org/images/manu.png"
}
```

Έγγραφα JSON μπορούν να ερμηνευθούν ως JSON-LD χωρίς να τροποποιηθούν απλώς προθέτοντας ένα context σαν HTTP Link Header.

Στα έγγραφα JSON-LD το context μπορεί να οριστεί και μέσα στον κώδικα. Αυτό έχει το πλεονέκτημα ότι το έγγραφο μπορεί να επεξεργασθεί ακόμα και αν δεν υπάρχει σύνδεση με το δίκτυο.

```
{
  "@context":
  {
    "name": "http://schema.org/name",
    "image": {
      "@id": "http://schema.org/image",
      "@type": "@id"
    },
    "homepage": {
      "@id": "http://schema.org/url",
      "@type": "@id"
    }
  },
  "name": "Manu Sporny",
  "homepage": "http://manu.sporny.org/",
  "image": "http://manu.sporny.org/images/manu.png"
}
```

IRIs

Τα IRIs είναι δομικά στοιχεία των Linked Data καθώς αυτά ορίζουν τι είναι κάθε κόμβος και κάθε ιδιότητα. Στο JSON-LD τα IRIs μπορούν να αναπαρασταθούν ως απόλυτα ή σχετικά IRIs. Ένα απόλυτο IRI ορίζεται να περιέχει ένα scheme μαζί με ένα path και προαιρετικά πεδία query και fragment. Ένα σχετικό IRI είναι ένα IRI που είναι σχετικό ως προς κάποιο απόλυτο IRI. Στο JSON-LD όλα τα σχετικά IRIs είναι σχετικά ως προς το base IRI.
Ένα string ερμηνεύεται ως IRI όταν είναι τιμή του @id:

```
{
...
  "homepage": { "@id": "http://example.com/" }
...
}
```

Τιμές που ερμηνεύονται ως IRIs μπορούν να εκφραστούν και σαν σχετικά IRIs. Για παράδειγμα αν υποθέσουμε ότι το ακόλουθο έγγραφο βρίσκεται στο http://example.com/about/ , το σχετικό IRI ../ θα πήγαινε στο http://example.com/ .

```
{
...
  "homepage": { "@id": "../" }
...
}
```

Απόλυτα IRIs μπορούν να εκφραστούν άμεσα στην θέση του κλειδιού όπως:

```
{
...
  "http://schema.org/name": "Manu Sporny",
...
}
```

Στο προηγούμενο παράδειγμα το κλειδί `http://schema.org/name` ερμηνεύεται ως απόλυτο IRI.

Αναγνωριστικά κόμβων

Για να είναι δυνατό να αναφερθούμε εξωτερικά στους κόμβους ενός γράφου είναι σημαντικό ο καθένας να έχει το αναγνωριστικό του. Τα IRIs είναι βασικό στοιχείο των Linked Data ώστε οι κόμβοι να είναι πραγματικά συνδεδεμένοι. Η κλήση ενός αναγνωριστικού θα πρέπει να καταλήγει στην αναπαράσταση του κόμβου του. Αυτό μπορεί να επιτρέπει σε μία εφαρμογή να ανακτήσει περισσότερες πληροφορίες για έναν κόμβο.
Στο JSON-LD, ένας κόμβος ταυτοποιείται χρησιμοποιώντας την λέξη κλειδί @id:

```
{
  "@context":
  {
    ...
    "name": "http://schema.org/name"
  },
  "@id": "http://me.markus-lanthaler.com/",
  "name": "Markus Lanthaler",
  ...
}
```

Το παράδειγμα αυτό περιέχει έναν κόμβο που χαρακτηρίζεται από το IRI `http://me.markus-lanthaler.com/`.

Ορίζοντας τον Τύπο

Ο τύπος ενός συγκεκριμένου κόμβου μπορεί να οριστεί χρησιμοποιώντας την λέξη κλειδί @type. Στα Linked Data οι τύποι ορίζονται μοναδικά με ένα IRI.

```
{
...
  "@id": "http://example.org/places#BrewEats",
  "@type": "http://schema.org/Restaurant",
...
}
```

Ένας κόμβος μπορεί να έχει πάνω από έναν τύπο και για να οριστεί κάτι τέτοιο χρησιμοποιούμε πίνακα

```
{
...
  "@id": "http://example.org/places#BrewEats",
  "@type": [ "http://schema.org/Restaurant", "http://schema.org/Brewery" ],
...
}
```

Η τιμή ενός κλειδιού @type μπορεί επίσης να είναι ένας όρος που ορίζεται στο context:

```
{
  "@context": {
    ...
    "Restaurant": "http://schema.org/Restaurant",
    "Brewery": "http://schema.org/Brewery"
  }
  "@id": "http://example.org/places#BrewEats",
  "@type": [ "Restaurant", "Brewery" ],
  ...
}
```

5. Η προσφορά των APIs στις επιχειρήσεις

Το Internet συνεχίζει να επηρεάζει καθοριστικά κάθε επιχείρηση. Βέβαια οι υπάρχουσες επενδύσεις στο internet και τις τεχνολογίες web δεν είναι αρκετές για να διαβεβαιώσουν μία μελλοντική επιτυχία. Η νέα τεχνολογία εμφανίζεται για να αλλάξει την παρούσα κατάσταση. Οι επιχειρήσεις που δεν θα υιοθετήσουν τις νέες τάσεις θα αποδυναμωθούν.

Ανάμεσα στις πιο ισχυρές τάσεις που μορφοποιούν το διαδίκτυο σήμερα είναι τα κοινωνικά δίκτυα, τα κινητά, και υπηρεσίες βάσει της τοποθεσίας. Επιπλέον ιδιαίτερα σημαντικά είναι τα APIs.

> *[Πριν δημιουργήσουμε την API πλατφόρμα μας,] "δεν μπορούσαμε να φανταστούμε όλες τις ιδέες που είχαν οι άνθρωποι για την χρήση της Guardian και των δεδομένων της στον ψηφιακό κόσμο. … Με αυτό το αυξανόμενο δίκτυο συνεργατών μπορούμε να επεκτείνουμε το εύρος μας, να συνδεθούμε με τους χρήστες και να γίνουμε πιο σχετικοί καθώς οι ανάγκες και οι συνήθειες των ανθρώπων εξελίσσονται στο μέλλον."*
>
> *Source: Guardian.co.um Open Platform blog.*

Κατά το τέλος της δεκαετίας του '90 ήταν επιτακτική ανάγκη, κάθε επιχείρηση να επικοινωνεί με τους πελάτες της μέσω κάποιου site. Όσες επιχειρήσεις δεν εκμεταλλεύτηκαν αυτό το ισχυρό μέσο επικοινωνίας γρήγορα βρέθηκαν σε μειονεκτική θέση με συνέπεια την μείωση του ποσοστού τους στο μερίδιο αγοράς. Ήταν μία περίοδος που ήταν απλό να φτάσεις σε ένα online συνδεδεμένο κοινό. Το μόνο που χρειαζόταν ήταν ένα website και οι πελάτες θα έβρισκαν την επιχείρηση από τον υπολογιστή τους μέσω ενός browser.

Παρόλο που οι προκλήσεις για την προσέλκυση και την διατήρηση χρηστών παραμένουν, ο σημερινός κόσμος είναι πολύ πιο πολύπλοκος. Οι συσκευές πολλαπλασιάζονται και οι άνθρωποι έχουν πρόσβαση στο internet από αμέτρητα smartphones, tablets και άλλες συσκευές, χωρίς να αναφέρουμε και το κρυμμένο δίκτυο των διασυνδεδεμένων έξυπνων συσκευών. Ο web browser δεν είναι πια η μόνη πύλη για πρόσβαση στο περιεχόμενο του διαδικτύου. Σήμερα όλο και περισσότερο περιεχόμενο ανακτάται μέσα από νέα μέσα όπως για παράδειγμα widgets ή εφαρμογές για κινητά. Τέλος μία επιχείρηση δεν μπορεί να βασιστεί στο ότι οι πελάτες θα ψάξουν μόνοι τους να βρουν το site της. Αντί για αυτό είναι πια επιτακτικό να σιγουρευτεί ότι τα ψηφιακά δεδομένα και οι υπηρεσίες της είναι διαθέσιμες στο γενικό πλαίσιο που επιλέγουν οι χρήστες. Επειδή ο ιστός είναι μία πολύ αποτελεσματική πλατφόρμα για την μετάδοση πληροφοριών, οι χρήστες είναι πιο επιλεκτικοί στο πώς, πότε και που θα αναζητήσουν πληροφορίες από το web. Ενώνοντας όλες αυτές τις αποσπασματικές τάσεις καταλαβαίνουμε ότι οι επιχειρήσεις πρέπει να είναι

πιο επιθετικές ώστε να φτάσουν έμμεσα στο πιθανό κοινό τους που δεν μπαίνει στο site τους.

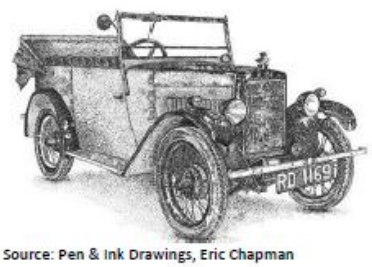
Η μεγάλη αλλαγή είναι η αναγνώριση ότι τα δεδομένα και οι υπηρεσίες, τα ψηφιακά περιουσιακά στοιχεία της επιχείρησης, πρέπει να αποδεσμευτούν από τους περιορισμούς ενός site. Από την στιγμή που θα είναι ελεύθερα και θα μπορούν να ανακτώνται από οπουδήποτε τότε η προοπτική για την ανάπτυξη της επιχείρησης είναι πολύ μεγάλη.

> *"Πλησιάζουμε την εποχή που το να ανοίξουμε την εφοδιαστική αλυσίδα μας στο Web δεν είναι μόνο καλή ιδέα αλλά απαραίτητο για την επιβίωση από τους ανταγωνιστές"*
> *Dion Hinchcliffe*

5.1 Δομικά στοιχεία στην εύρεση λύσεων

Για να κατανοήσουμε τη σημασία των APIs στις επιχειρήσεις καλό είναι να βρούμε παραλληλισμούς με άλλους κλάδους στη βιομηχανία. Αν αναλογιστούμε πως εξελίσσονται συνήθως οι επιχειρήσεις βλέπουμε ότι υπάρχει ένα μοτίβο που επαναλαμβάνεται. Κάθε βιομηχανία εξελίσσεται ώστε να παράγει πιο κομψές λύσεις με το πέρασμα του χρόνου μέσα από μία διαδικασία συναρμολόγησης και προτυποποίησης.

Είναι αδύνατο να φανταστούμε ότι κάποιος σήμερα θα έφτιαχνε ένα αυτοκίνητο χωρίς να εκμεταλλευτεί τα υπάρχοντα σχέδια και υποσυστήματα του οχήματος. Στις αρχές τις αυτοκινητοβιομηχανίας καινοτόμοι μηχανικοί έφτιαχναν την πλειοψηφία των εξαρτημάτων με δικά τους σχέδια. Με το πέρασμα του χρόνου άρχισαν να αναγνωρίζονται υποσυστήματα και στοιχεία όπως τα φρένα, η μηχανή, η ανάρτηση και το σώμα του αυτοκινήτου έγιναν περιοχές εξειδίκευσης. Ομάδες μηχανικών δούλευαν σε ξεχωριστά υποσυστήματα και συχνά ολόκληρες εταιρίες εξειδικεύονταν σε ένα ή περισσότερα. Καθώς οι μεγάλοι κατασκευαστές αυτοκινήτων έκαναν outsource σε όλο και περισσότερες εργασίες στην προσπάθειά τους για αποδοτικότητα ο ρόλος τους μετατράπηκε σε συντονιστή και συναρμολογητή του συστήματος. Χάρη στις καλά σχεδιασμένες διεπαφές των υποσυστημάτων, όταν αυτά ενώνονταν στη γραμμή συναρμολόγησης το αποτέλεσμα ήταν έναν αυτοκίνητο με εξαιρετικές επιδόσεις σε σχέση με το κόστος του.



Source: Pen & Ink Drawings, Eric Chapman

Αυτό το μοτίβο επαναλαμβανόταν ξανά και ξανά. Οι επιχειρήσεις διάλεγαν προσεκτικά την περιοχή δράσης τους είτε σε επίπεδο αρχιτεκτονικής είτε στη λύση προβλημάτων είτε ανάμεσα στα υποσυστήματα. Μετά συνεργάζονταν για να ενώσουν όλα τα τμήματα μαζί σε ένα πλήρες προϊόν ή υπηρεσία. Με αυτό τον τρόπο οι επιχειρήσεις πετύχαιναν

μεγαλύτερες οικονομίες κλίμακας, διατηρούσαν υψηλό επίπεδο στην έρευνα και ανάπτυξη και παρείχαν περισσότερες καινοτομίες στα προϊόντα τους.

Πέρα από την αυτοκινητοβιομηχανία, το ίδιο μοτίβο υπάρχει στους περισσότερους τομείς που βασίζονται σε engineering όπως για παράδειγμα κτίρια, αεροπλάνα, ρομποτική κλπ.

Αυτό το μοτίβο υπάρχει και στην βιομηχανία λογισμικού και υλοποιείται μέσω των APIs. Μάλιστα σε σύγκριση με τις άλλες βιομηχανίες, για το λογισμικό τα πλεονεκτήματα είναι εκθετικά μεγαλύτερα καθώς η δυνατότητα για συναρμολόγηση είναι ουσιαστικά απεριόριστη και καθώς τα κόστη εφαρμογής είναι σημαντικά χαμηλότερα.

5.2 Πρώτη περίοδος για τα APIs

Κατά την ανάπτυξη της βιομηχανίας των ηλεκτρονικών υπολογιστών τις τελευταίες δεκαετίες, -ακόμα και πριν τον παγκόσμιο ιστό- τα APIs βρίσκονταν στο κέντρο της δυναμικής της αγοράς. Συχνά “μάχες” μεταξύ ανταγωνιστικών επιχειρήσεων κρίνονταν με βάση τα APIs τους. Τα μερίδια ήταν υψηλά και συνήθως λίγες ήταν οι μεγάλες και ισχυρές επιχειρήσεις ή οργανισμοί που έδιναν κατευθύνσεις για την εξέλιξη των APIs. Η διεύθυνση ενός ανοιχτού ή ιδιοκτησιακού προτύπου API συνήθως εκτόξευε το μέγεθος μιας επιχείρησης.

Ο λόγος για τον οποίο τα APIs αποτέλεσαν σημαντικό στοιχείο για την ανάπτυξη ήταν το δίκτυο που δημιουργούνταν όταν πολλές εφαρμογές αναπτύσσονταν για μία πλατφόρμα, αυξάνοντας έτσι την αξία της ίδιας της πλατφόρμας. Από την στιγμή που μία μέθοδος ανάπτυξης εδραιώνεται η αλλαγή της σε κάποια άλλη εναλλακτική μέθοδο απαιτεί μεγάλα κόστη με δεδομένο ότι μερικές εφαρμογές μπορεί να μην είναι διαθέσιμες στην νέα πλατφόρμα.

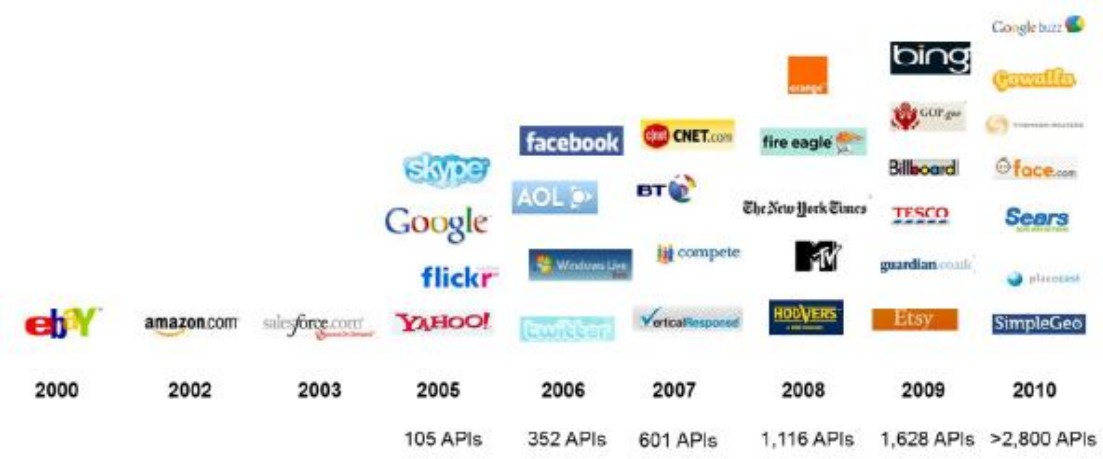
Η Microsoft ήταν μία από τις πιο επιτυχημένες εταιρίες που τα APIs της αποτέλεσαν πλεονέκτημα για την επιχείρηση. Έκαναν μεγάλες επενδύσεις για να προσελκύσουν την πλειοψηφία των προγραμματιστών εφαρμογών προκειμένου να γράψουν εφαρμογές για τα MS Windows και το Windows API. Μόλις έφτασαν σε ένα μεγάλο μέγεθος, ήταν σχεδόν αναγκαστικό για τους πελάτες να επιλέξουν τα Windows λόγω του μεγάλου αριθμού διαθέσιμων εφαρμογών. Αυτό οδήγησε ακόμα περισσότερους προγραμματιστές στο να αναπτύξουν εφαρμογές για αυτή την πλατφόρμα προκειμένου να προσελκύσουν το δυνατόν περισσότερους πελάτες.

Εκείνη την περίοδο, πριν το διαδίκτυο γίνει διαθέσιμο ως πλατφόρμα μετάδοσης, ήταν εξαιρετικά δύσκολο να δημιουργηθεί ένα “οικοσύστημα” από προγραμματιστές γύρω από τα APIs. Για να είναι αυτά επιτυχημένα θα έπρεπε να ξεπεραστούν η μεγάλη διανομή, ο ανταγωνισμός και άλλες σύνθετες προκλήσεις. Αυτές οι προκλήσεις ήταν δυνατό να αντιμετωπιστούν μόνο στις μεγάλες κλίμακες και για αυτό τον λόγο ήταν αναπόφευκτο ότι θα κυριαρχούσαν οι μεγάλες επιχειρήσεις.

5.3 Δεύτερη περίοδος για τα APIs

Το διαδίκτυο και συγκεκριμένα η ανάπτυξη των web APIs έφεραν μία ισορροπία σε σχέση με τις μεγάλες επιχειρήσεις. Ο καθένας τώρα μπορεί να ορίσει το υποσύστημά του ως μία υπηρεσία web που θα είναι διαθέσιμη να ενωθεί με άλλα συστήματα ή εφαρμογές στον συνδεδεμένο ιστό.

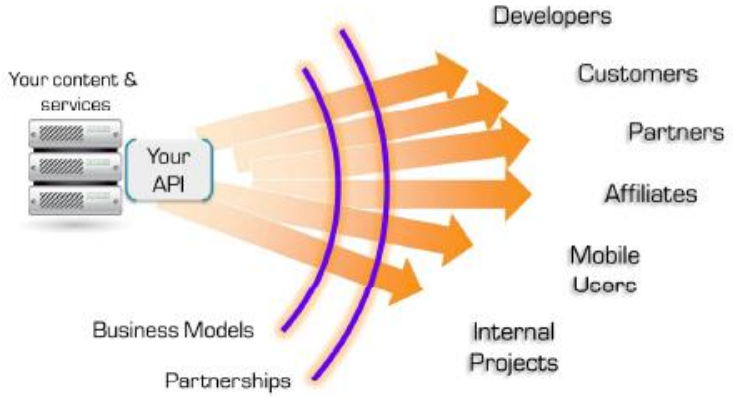
Η εμπορική δυνατότητα των APIs πια δεν είναι περιορισμένη σε έναν μικρό αριθμό μεγάλων επιχειρήσεων όπως η Microsoft. Οποιοσδήποτε μπορεί να εντοπίσει μία ευκαιρία για νέα υπηρεσία και μέσω των APIs να την εισάγει σε ένα μεγαλύτερο πλαίσιο. Με τον τρόπο αυτό η υπηρεσία αποκτά ευκολότερη πρόσβαση, γίνεται πιο χρήσιμη και πιο ισχυρή. Για προγραμματιστές που σχεδιάζουν υπηρεσίες web είναι τώρα ευκολότερο από ποτέ να εκμεταλλευτούν εξωτερικές υπηρεσίες και δεδομένα ώστε να ενδυναμώσουν την υπηρεσία που ήδη προσφέρουν. Το αποτέλεσμα αυτό είναι θετικό κυρίως για τους τελικούς χρήστες – καταναλωτές και για αυτό παρατηρήθηκε μία έκρηξη στον αριθμό των APIs.



Η έκρηξη των APIs σημαίνει ότι είναι δυνατό να δημιουργηθούν προϊόντα που καλύπτουν τις ανάγκες και επιθυμίες των καταναλωτών με μεγαλύτερη ακρίβεια. Μπορεί να είναι για να ικανοποιήσουν μία μικρή εξειδικευμένη αγορά, για να έχουν πρόσβαση οι χρήστες τους σε δεδομένα μέσα από ένα μοναδικό πλαίσιο ή για να καλύψουν την προτίμηση της αλληλεπίδρασής τους μέσα από smartphones. Κυρίως όμως δίνει στις επιχειρήσεις την ευελιξία να σχεδιάσουν εντελώς καινούρια και πρωτοποριακά business models. Το γεγονός αυτό που μπορεί να αποτελεί μία εξέλιξη στον ρόλο των APIs ουσιαστικά είναι μία επανάσταση στον τρόπο που οι επιχειρήσεις μπορούν να επωφεληθούν από αυτά.

Στην ακόλουθη εικόνα παρατηρούμε των αριθμών των κλήσεων API σε μερικές από τις μεγαλύτερες επιχειρήσεις.

5.4 APIs: Ο ακρογωνιαίος λίθος της online στρατηγικής



Τα APIs ανοίγουν ένα νέο κεφάλαιο για το διαδίκτυο. Το περιεχόμενο και οι υπηρεσίες είναι η ψηφιακή περιουσία μίας επιχείρησης και αποτελούν τον πυρήνα της. Ένα API ξεκλειδώνει την αξία των ψηφιακών στοιχείων της επιχείρησης και εξαπλώνεται πέρα από websites σε εφαρμογές για κινητά, συνεργάτες, προγραμματιστές κ.α. Αυτή η εξάπλωση επιτρέπει σε συνεργασίες να ανανεωθούν και δημιουργεί πολλαπλές χρήσεις για τα στοιχεία-κλειδιά. Με αυτό τον τρόπο δίνει την ευκαιρία για καινοτομία με εντελώς νέα business models. Οι ανταγωνιστές μένουν στατικοί ενώ οι πελάτες μπορούν να έχουν πρόσβαση σε περιεχόμενο και πληροφορίες ακριβώς με τον τρόπο που επιθυμούν.

Για τους λόγους αυτούς κάθε επιχείρηση πρέπει να αρχίσει να σκέφτεται την δημιουργία ενός API με τα ακόλουθα βήματα:

- Αναγνώριση των βασικών ψηφιακών κτημάτων της επιχείρησης
- Σκέψη για τις λύσεις που θα μπορούσαν να βρεθούν με την βοήθεια των ψηφιακών στοιχείων
- Ορισμός κάποιων περιπτώσεων για στρατηγική και business models βασισμένα σε API
- Εύρεση προδιαγραφών για την εισαγωγή τους στο API
- Έναρξη με μία στρατηγική και ένα business model και ετοιμότητα για επαναπροσαρμογή και αλλαγή του

Βιβλιογραφία

1. “RESTful Web APIs”, Leonard Richardson, Mike Amundsen, Sam Ruby, Εκδόσεις O’Reilly
2. “APIs: A Strategy Guide”, Daniel Jacobson, Greg Brail, Dan Woods, Εκδόσεις O’Reilly
3. “RESTful Web Services Cookbook”, Subbu Allamaraju, Εκδόσεις O’Reilly
4. “REST API Design Rulebook”, Mark Masse, Εκδόσεις O’Reilly
5. “PHP Web Services: APIs for the Modern Web” , Lorna Jane Mitchell, Εκδόσεις O’Reilly
6. “The REST API Design Handbook”, George Reese
7. http://history.apievangelist.com/
8. http://swagger.wordnik.com/
9. https://github.com/wordnik/swagger-spec
10. https://helloreverb.com/developers/swagger
11. https://github.com/wordnik/swagger-spec/blob/master/versions/1.2.md
12. http://raml.org/spec.html
13. http://apiblueprint.org/
14. https://github.com/apiaryio/api-blueprint/blob/master/API%20Blueprint%20Specification.md
15. http://www.slideshare.net/mpetyx/api-standards-2562014
16. http://www.markus-lanthaler.com/hydra/
17. http://www.w3.org/TR/json-ld/
18. www.3scale.net