Εθνικο Μετσοβιο Πολυτεχνειο
Σχολη Ηλεκτρολογων Μηχανικων και Μηχανικων Υπολογιστων
Τομεας Τεχνολογιας Πληροφορικης και Υπολογιστων

Αυτόματη Παραγωγή Τεκμηρίωσης Εξαρτήσεων Μεταξύ Κλήσεων Endpoints ενός REST API

Μελέτη και υλοποίηση

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

του

ΠΑΝΑΓΙΩΤΗ Μ ΠΑΠΑΔΕΑ

Επιβλέπων: Βασίλειος Βεσκούκης

Καθηγητής ΕΜΠ

Αθήνα, Ιούλιος 2023

Εθνικο Μετσοβιο Πολυτεχνειο
Σχολη Ηλεκτρολογων Μηχανικων και Μηχανικων Υπολογιστων
Τομεας Τεχνολογιας Πληροφορικης και Υπολογιστων

Αυτόματη Παραγωγή Τεκμηρίωσης Εξαρτήσεων Μεταξύ Κλήσεων Endpoints ενός REST API

Μελέτη και υλοποίηση

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

του

ΠΑΝΑΓΙΩΤΗ Μ ΠΑΠΑΔΕΑ

Επιβλέπων: Βασίλειος Βεσκούκης

Καθηγητής ΕΜΠ

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή την 4η Ιουλίου 2023.

Βασίλειος Βεσκούκης Καθηγητής ΕΜΠ	Παναγιώτης Τσανάκας Καθηγητής ΕΜΠ	Αριστείδης Παγουρτζής Καθηγητής ΕΜΠ

Αθήνα, Ιούλιος 2023

ΠΑΝΑΓΙΩΤΗΣ Μ ΠΑΠΑΔΕΑΣ

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

Περίληψη

Στην εποχή της πληροφορίας, χρειαζόμαστε τεχνολογίες που να χειρίζονται τα δεδομένα γρήγορα και αποδοτικά και να είναι συμβατές με τις περισσότερες εφαρμογές. Τα APIs είναι ιδανικά για αυτό το σκοπό. Τα REST APIs ειδικότερα, αποκτούν όλο και μεγαλύτερη δημοφιλία λόγω της ευχρηστίας και της κλιμακωσιμότητάς τους. Για να χρησιμοποιήσουμε όμως ένα REST API σωστά πρέπει πρώτα να το κατανοήσουμε.

Το API Documentation αποτελεί ένα έγγραφο που περιγράφει πώς θα χρησιμοποιηθεί ένα API. Ωστόσο, πολλές φορές η πληροφορία που περιέχει δεν επαρκεί. Συγκεκριμένα, τα διαφορετικά endpoints ενός REST API εμφανίζουν συχνά σχέσεις εξάρτησης μεταξύ τους οι οποίες επιβάλλουν μια σειρά προτεραιότητας στον τρόπο κλήσης τους. Θέμα της εργασίας μας αποτελεί η αναγνώριση αυτών των εξαρτήσεων, η αυτόματη παραγωγή του Documentation και η δημιουργία ενός γράφου που θα τις αναδεικνύει και θα βοηθάει τον χρήστη να κατανοήσει τον τρόπο με τον οποίο συνδέονται τα endpoints σε ένα REST API.

Σε αυτή την εργασία θα παρουσιαστεί μία μέθοδος η οποία συνδυάζει τα εργαλεία Postman και Visual Paradigm με σκοπό την αυτοματοποίηση της διαδικασίας παραγωγής και παρουσίασης ενός API Documentation το οποίο περιλαμβάνει την κατάλληλη πληροφορία για την ανάδειξη των σχέσεων εξάρτησης. Με το Postman μπορούμε να δημιουργήσουμε συλλογές από requests σε ένα REST API με όλες τις απαραίτητες πληροφορίες όπως request bodies, response bodies, headers κ.α. Με το εργαλείο Visual Paradigm μπορεί να παραχθεί ένα OpenAPI Documentation ή να προκύψει μια API αναπαράσταση από αυτό.

Μέσω λοιπόν της εργασίας αυτής δημιουργούμε ένα σύστημα το οποίο παράγει αυτόματα OpenAPI Documentations για REST APIs, τα οποία περιέχουν πληροφορία για τις εξαρτήσεις μεταξύ των endpoints. Ακόμη, επεξεργάζεται υπάρχοντα OpenAPI Documentations προκειμένου να εισάγει σε αυτά την απαραίτητη πληροφορία. Τέλος παρουσιάζουμε το αποτέλεσμα σε έναν αναλυτικό γράφο όπου είναι φανερές οι εξαρτήσεις μεταξύ των διαφορετικών endpoints ενός REST API.

Λέξεις κλειδιά

API, REST, HTTP, XML, JSON, RPC, SOAP, Postman, Visual Paradigm, OpenAPI, JSON, API Documentation, Python3

Abstract

In the era of information, we need technologies that can handle data fast and efficiently and are compatible with most applications. APIs are ideal for this purpose. REST APIs specifically, are getting more and more popular because of their ease of use and their scalability. However, in order to use a REST API correctly we first need to understand it.

The concept of API Documentation is a file that describes how an API should be used. However, in many cases the information it contains is insufficient. Different endpoints of the same REST API are often connected with each other through dependency relationships which show the order in which these endpoints should be called. Our goal is to find these dependencies between endpoints, automatically generate the Documentation and create a graph that will highlight them and help the user understand the way endpoints are connected with each other in a REST API.

In this thesis we will present a method that combines and enhances the tools Postman and Visual Paradigm in order to achieve automatic generation and presentation of an API Documentation which contains information required to highlight these dependency relationships. By using Postman we can create collections of requests on a REST API with all the necessary information such as request bodies, response bodies, headers etc. By using Visual Paradigm we can generate an OpenAPI Documentation or create an API visualization from it.

Through our project we create a system that automatically generates OpenAPI Documentations for REST APIs, which contain information about dependencies between endpoints. Furthermore, it processes existing OpenAPI Documentations in order to include the necessary information. Finally, we present the result in a graph where the dependencies between endpoints of a REST API are highly visible.

Keywords

API, REST, HTTP, XML, JSON, RPC, SOAP, Postman, Visual Paradigm, OpenAPI, JSON, API Documentation, Python3

στους γονείς μου και στον αδερφό μου,

Ευχαριστίες

Αρχικά, θα ήθελα να ευχαριστήσω τον αξιότιμο κύριο Βεσκούκη για την επίβλεψη της παρούσας διπλωματικής εργασίας, την βοήθεια και την ευκαιρία που μου έδωσε να ασχοληθώ με το παρόν θέμα. Επίσης θα ήθελα να ευχαριστήσω τους Υποψήφιους Διδάκτορες και Ερευνητές Χρήστο Χατζιχριστοφή και Ιωάννη Τζαννέτο για την συνεργασία τους και τη συμβολή τους στην πραγματοποίηση της εργασίας αυτής. Τέλος, θα ήθελα να ευχαριστήσω τους γονείς μου και τον αδερφό μου για τη στήριξη και την ηθική συμπαράσταση που μου προσέφεραν όλα αυτά τα χρόνια.

Περιεχόμενα

Κατάλογος εικόνων

1 Εισαγωγή

Στις μέρες μας, η πληροφορία αποκτάει όλο και μεγαλύτερη αξία και τα δεδομένα πολλαπλασιάζονται με ταχύτατους ρυθμούς. Χρειαζόμαστε τεχνολογίες ώστε να μπορούμε να χειριζόμαστε εύκολα αυτά τα δεδομένα αλλά και να δημιουργούμε εφαρμογές που επικοινωνούν με άλλες εφαρμογές χωρίς να απαιτείται ιδιαίτερη προγραμματιστική δυσκολία. Στο παρελθόν έχουν αναπτυχθεί πολλές τεχνολογίες για να γίνεται αυτό, αρκετές από τις οποίες χρησιμοποιούνται ακόμη και σήμερα. Ωστόσο, τα τελευταία χρόνια η χρήση προγραμματιστικών διεπαφών (Application Programming Interfaces - APIs) πάνω από πρωτόκολλο HTTP έχει κερδίσει σημαντικό έδαφος, λόγω της απλότητας που τη χαρακτηρίζει. Αυτός είναι και ο βασικός λόγος που το API έχει γίνει τόσο δημοφιλές στον κόσμο της ανάπτυξης λογισμικού.

1.1 Ιστορία του API - Ορισμός

Για να καταλάβουμε γιατί τα APIs είναι τόσο σημαντικά, πρέπει πρώτα να ρίξουμε μια ματιά στην ιστορία τους [1]. Τα APIs δημιουργήθηκαν στις αρχές του 2000 με σκοπό να εισάγουν το εμπόριο στο διαδίκτυο. Δημιουργήθηκαν δηλαδή, με σκοπό να κάνουν προϊόντα και υπηρεσίες διαθέσιμα στους πελάτες μέσω ενός ιστοτόπου και να επιτρέπουν σε συνεργάτες και μεταπωλητές να επεκτείνουν την εμβέλεια των πλατφορμών τους. Σε αυτή την προσπάθεια, καθοριστικό ρόλο έπαιξαν μεγάλες εταιρείες όπως eBay και Amazon.

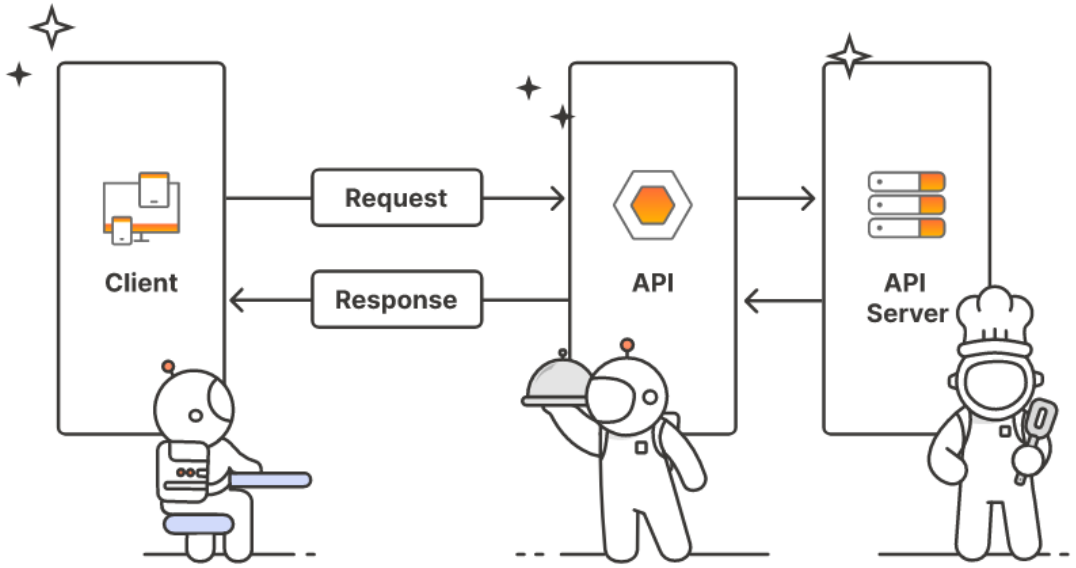
Από το 2004 και μετά, τα APIs δεν παρέμειναν άμεσα συνδεδεμένα με την αξία του εμπορίου, αλλά εξελίχτηκαν σε προσοδοφόρες πλατφόρμες. Ειδικότερα, από το 2010 και μετά όπου κυριάρχησαν τα μέσα διαδικτύωσης, τα APIs αποτέλεσαν την «υποδομή» τους και επέτρεψαν την επικοινωνία μεταξύ δύο ατόμων καθώς και την ανταλλαγή εικόνων και βίντεο. Εταιρείες όπως το Facebook και το Twitter στηρίχτηκαν στα APIs για να δημιουργήσουν επιτυχημένες πλατφόρμες.

Όσο περνούσαν τα χρόνια, τα APIs εισάχθηκαν στον κόσμο του cloud. Εταιρείες όπως η Amazon απαίτησαν σε όλους τους ψηφιακούς τους πόρους να συμπεριλαμβάνουν ένα API, ενώ διαδικτυακές υπηρεσίες που πουλούσαν, π.χ. αποθηκευτικό χώρο στο cloud, ήταν προσβάσιμες μέσω ενός API. Όταν μάλιστα η εταιρεία Apple κυκλοφόρησε το iPhone και η Google το Android, τα APIs έφτασαν στα κινητά μας και εφαρμογές που στηρίχτηκαν στα APIs έθεσαν νέα σύνορα στον τρόπο που δημιουργούμε, αποθηκεύουμε και μεταδίδουμε δεδομένα διαδικτυακά. Φυσικά, τα APIs δεν περιορίστηκαν μόνο στα smartphones, αλλά επεκτάθηκαν σε όλα τα «έξυπνα» αντικείμενα που έχουν πρόσβαση στο διαδίκτυο.

Για να κατανοήσουμε καλύτερα την έννοια του API, θα πρέπει πρώτα να εξηγήσουμε την έννοια του RPC [2]. RPC (remote protocol control) ονομάζουμε το πρωτόκολλο που επιτρέπει σε έναν υπολογιστή ή πρόγραμμα να επικοινωνεί με έναν άλλο υπολογιστή ή πρόγραμμα. Ένα RPC πρέπει να ξεκινήσει από τον client ο οποίος στέλνει μήνυμα στο server. Στη συνέχεια, ο server απαντάει στον client ο οποίος συνεχίζει με τη διεργασία που έτρεχε αρχικά.

API [3, 4] (application programming interface), ονομάζουμε ένα σύνολο από κανόνες που καθορίζει τον τρόπο με τον οποίο εφαρμογές και συσκευές μπορούν να συνδέονται και να επικοινωνούν μεταξύ τους. Τα APIs χρησιμοποιούνται ως ένα μέσο

για να έχουμε πρόσβαση σε δεδομένα, λειτουργίες και υπηρεσίες διαφόρων εφαρμογών και συστημάτων που ανήκουν σε τρίτους είτε για να εκθέσουμε δεδομένα και υπηρεσίες από δικές μας εφαρμογές σε άλλους προγραμματιστές και εφαρμογές. Μία χρήσιμη παρομοίωση είναι να θεωρήσουμε το API σαν ένα εστιατόριο [4]. Το API είναι ο σερβιτόρος που παίρνει την παραγγελία από τον πελάτη (χρήστη) και τη μεταφράζει σε απλές οδηγίες στην κουζίνα που αποτελεί τον API server .Το προσωπικό της κουζίνας πραγματοποιεί την παραγγελία και την δίνει στον σερβιτόρο για να τη μεταφέρει πίσω στον πελάτη.



Εικόνα 1: Τρόπος λειτουργίας ενός API [4]

Υπάρχουν τρεις κατηγορίες API:

- Private APIs, τα οποία συνδέουν συστατικά λογισμικού εντός ενός οργανισμού
- Public APIs, τα οποία παρέχουν δημόσια πρόσβαση σε δεδομένα και υπηρεσίες ενός οργανισμού και διάφοροι προγραμματιστές μπορούν να τα εντάξουν στις εφαρμογές τους
- Partner APIs, τα οποία επιτρέπουν σε διαφορετικές εταιρείες να μοιράζονται δεδομένα και υπηρεσίες για να συνεργαστούν σε ένα κοινό πρότζεκτ

Αν και οι όροι RPC και API συχνά συγχέονται, πρόκειται για δύο εντελώς διαφορετικά αντικείμενα. Τα APIs αποτελούν τον σκελετό που επιτρέπει υπολογιστές και προγράμματα να επικοινωνούν μεταξύ τους σε ένα κοινό δίκτυο, ενώ τα RPCs είναι τα μέσα με τα οποία επικοινωνούν. Αν θέλουμε να κάνουμε αυτή τη διάκριση πιο ξεκάθαρη, τα RPCs μας συνδέουν σε ένα δίκτυο ενώ τα APIs μας επιτρέπουν να επικοινωνούμε με άλλες συσκευές στο ίδιο δίκτυο [2].

1.2 HTTP protocol

Το HTTP (Hypertext Transfer Protocol) [5, 6] είναι ένα πρωτόκολλο επικοινωνίας επιπέδου εφαρμογής, το οποίο στηρίζεται στην αρχιτεκτονική client-

server για την ανταλλαγή μηνυμάτων μεταξύ ενός πελάτη και ενός διακομιστή. Υπάρχουν δύο είδη μηνυμάτων: request, το οποίο στέλνεται από τον πελάτη και response, το οποίο στέλνεται από τον διακομιστή. Ένας HTTP πελάτης στέλνει ένα request μήνυμα στον HTTP server ο οποίος του επιστρέφει ένα response μήνυμα. Το HTTP είναι ένα πρωτόκολλο τύπου «pull» καθώς ο πελάτης «τραβάει» δεδομένα από τον διακομιστή, αντί ο διακομιστής να σπρώχνει δεδομένα στον πελάτη.

Το request μήνυμα αποτελείται από τέσσερα μέρη:

- request line
- header (το οποίο περιέχει πληροφορίες για τον host)
- empty line
- eventual body (το οποίο περιέχει επιπλέον πληροφορίες, αν χρειάζεται)

Στο request line ορίζουμε την μέθοδο του request. Υπάρχουν πολλές μέθοδοι για διαφορετικές περιπτώσεις χρήσης. Ορισμένες από τις πιο βασικές παρουσιάζονται παρακάτω:

- GET (για να λάβουμε πληροφορία από τον server)
- POST (για να δώσουμε πληροφορία στον server)
- PUT (για να μεταβάλλουμε δεδομένα στον server)
- DELETE (για να διαγράψουμε δεδομένα στο server)
- PATCH (για να εφαρμόσουμε τροποποιήσεις σε υπάρχοντες πόρους)

Αντίστοιχα το response αποτελείται από τέσσερα μέρη:

- status line
- header (το οποίο περιέχει πληροφορίες για τον server)
- empty line
- eventual body (το οποίο περιέχει επιπλέον πληροφορίες, αν χρειάζεται)

Στο status line περιλαμβάνεται ένας κωδικός, ο οποίος μας δίνει πληροφορίες για την εκτέλεση του request και συγκεκριμένα αν το αποτέλεσμα ήταν επιτυχές ή απέτυχε. Ορισμένοι από τους πιο συνήθεις κωδικούς παρουσιάζονται παρακάτω:

- 200: OK, SUCCESS
- 400: BAD REQUEST
- 404: NOT FOUND
- 500: INTERNAL SERVER ERROR
- 502 BAD GATEWAY

Ένα API που χρησιμοποιεί HTTP requests συνήθως περιλαμβάνει παραμέτρους [7] σε κάθε request οι οποίες θα επηρεάσουν το response. Υπάρχουν τρεις κατηγορίες παραμέτρων:

- Query string parameters, περιλαμβάνουν το όνομα και την τιμή της παραμέτρου και βρίσκονται στο τέλος του endpoint μετά το χαρακτήρα «?»

```
http://mydomain.com/example/myparam1=123&myparam2=abc
```

Εικόνα 2: Query string parameters

- Path parameters, περιλαμβάνουν την τιμή της παραμέτρου, βρίσκονται πριν τον χαρακτήρα «?» στο endpoint και τοποθετούνται ανάμεσα σε αγκύλες

http://mydomain.com/service/user/{user}/bicycles/{bicycleid}

Εικόνα 3: Path parameters

- Request body parameters, περιλαμβάνουν το όνομα της παραμέτρου και την τιμή της και βρίσκονται στο request body του request

{
“myparam1”=”123”,
“myparam2”=”456”,
“myparam3”=”789”
}

Εικόνα 4: Request body parameters

1.3 XML

XML [8, 9], δηλαδή Extensible Markup Language, ονομάζουμε μια αυτοπεριγραφική γλώσσα με την οποία μπορούμε να αποθηκεύουμε και να μεταφέρουμε δεδομένα. Δημιουργήθηκε με σκοπό να ξεπεράσει τους περιορισμούς της HTML [10] και χρησιμοποιείται ως βασική γλώσσα για διάφορα πρωτόκολλα επικοινωνίας. Η βασική διαφορά της XML με την HTML είναι ότι η πρώτη σχεδιάστηκε για να μεταφέρει δεδομένα ενώ η δεύτερη για να παρουσιάζει δεδομένα. Ακόμη, η XML δεν χρησιμοποιεί προκαθορισμένες ετικέτες (tags). Είναι κλιμακώσιμη και συμβάλλει στην απλοποίηση διαφόρων διεργασιών όπως στην μεταφορά δεδομένων, στην αποθήκευσή τους και στην διαθεσιμότητά τους.

Το συντακτικό [11] της XML ακολουθεί τους εξής κανόνες:

- Κάθε έγγραφο XML πρέπει να περιέχει ένα στοιχείο root που είναι ο γονιός όλων των υπόλοιπων στοιχείων

```
<root>
  <child>
    <subchild>.....</subchild>
  </child>
</root>
```

Εικόνα 5: Root element XML

- Όλα τα στοιχεία XML πρέπει να περιέχουν μια ετικέτα τέλους

```
<p>This is a paragraph.</p>
<br />
```

Εικόνα 6: Tags XML

- Κάθε XML στοιχείο πρέπει να είναι σωστά εμφωλιασμένο

```
<b><i>This text is bold and italic</i></b>
```

Εικόνα 7: Nested elements XML

Η XML συνήθως χρησιμοποιείται για να διαχωρίσει τα δεδομένα από την παρουσίαση δεδομένων [12]. Δεν περιλαμβάνει καμία πληροφορία για τον τρόπο με τον οποίο πρέπει να παρουσιαστούν τα δεδομένα και μπορεί να χρησιμοποιηθεί σε διαφορετικά σενάρια παρουσιάσεων. Συχνά χρησιμοποιείται ως συνοδευτική γλώσσα της HTML. Η HTML παρουσιάζει τα δεδομένα και αν αυτά αλλάξουν δεν απαιτείται αλλαγή του κώδικα της HTML αλλά μόνο της XML. Συνεπώς επιτυγχάνεται πλήρης διαχωρισμός των δεδομένων και του τρόπου παρουσίασής τους.

```xml
<address>
  <name>
    <title>Mrs.</title>
    <first-name>
      Mary
    </first-name>
    <last-name>
      McGoon
    </last-name>
  </name>
  <street>
    1401 Main Street
  </street>
  <city state="NC">Anytown</city>
  <postal-code>
    34829
  </postal-code>
</address>

<!-- Source http://www.ibm.com/developerworks/xml/tutorials/xmlintro/xmlintro.html-->
```

Εικόνα 8: Παράδειγμα αρχείου XML

1.4 JSON

JSON [13], δηλαδή JavaScript Object Notation, ονομάζουμε μια αυτοπεριγραφική γλώσσα που χρησιμοποιούμε για να αποθηκεύουμε και να μεταφέρουμε δεδομένα και αποτελεί μια εναλλακτική λύση της XML. Είναι ανεξάρτητη γλωσσών και μπορεί να αξιοποιηθεί σε διάφορες γλώσσες προγραμματισμού. Αποτελεί έναν ελαφρύ μορφότυπο ανταλλαγής δεδομένων.

Το συντακτικό του JSON κληρονομείται από το συντακτικό της Javascript [14, 15]:

- Τα δεδομένα έχουν την μορφή ζευγών ονόματος/τιμής.

```
"name":"John"
```

Εικόνα 9: JSON δεδομένα σε ζεύγος ονόματος/τιμής

- Τα δεδομένα μπορούν να πάρουν τιμές που ανήκουν στις εξής κατηγορίες: string, number, object, array, boolean, null.
- Οι δύο βασικές δομές είναι: Objects και Arrays. Ένα object είναι μια συλλογή από ζευγάρια ονομάτων/τιμών και ένα array είναι μια ταξινομημένη λίστα από τιμές.

```
{
        "id": 4911,
        "firstName":"John",
        "lastName":"Walter",
        "isStudent": true
};
```

Εικόνα 10: JSON Object

```
[
     "Belgium", "Brazil", "Cameroon", "Canada", "Chile", "Denmark"
]
```

Εικόνα 11: JSON Array

Τόσο η γλώσσα JSON όσο και η XML μπορούν να χρησιμοποιηθούν για να λάβουμε δεδομένα από έναν διαδικτυακό server [16]. Και οι δύο γλώσσες είναι αυτοπεριγραφικές, δηλαδή είναι εύκολα αναγνώσιμες από άνθρωπο. Επίσης μπορούν να γίνουν «parse» από διάφορες γλώσσες προγραμματισμού (δηλαδή να μετατραπούν σε object που θα χρησιμοποιηθεί σε ένα πρόγραμμα). Η JSON όμως δεν χρησιμοποιεί ετικέτες τέλους (end tags), είναι συντομότερη, διαβάζεται και γράφεται πιο γρήγορα και χρησιμοποιεί arrays. Σε γενικές γραμμές, η JSON θεωρείται καλύτερη επειδή είναι πιο εύκολο να γίνει «parse», είναι πιο γρήγορη και πιο εύκολη από την XML. Στο παρακάτω παράδειγμα χρησιμοποιούμε την JSON και την XML για να δημιουργήσουμε ένα object υπαλλήλου που περιλαμβάνει έναν πίνακα από τρεις υπαλλήλους.

JSON Example

```
{"employees":[
  { "firstName":"John", "lastName":"Doe" },
  { "firstName":"Anna", "lastName":"Smith" },
  { "firstName":"Peter", "lastName":"Jones" }
]}
```

Εικόνα 12: Παράδειγμα ενός JSON Object που αναπαριστά υπαλλήλους και περιλαμβάνει Array από παραδείγματα τριών υπαλλήλων [16]

XML Example

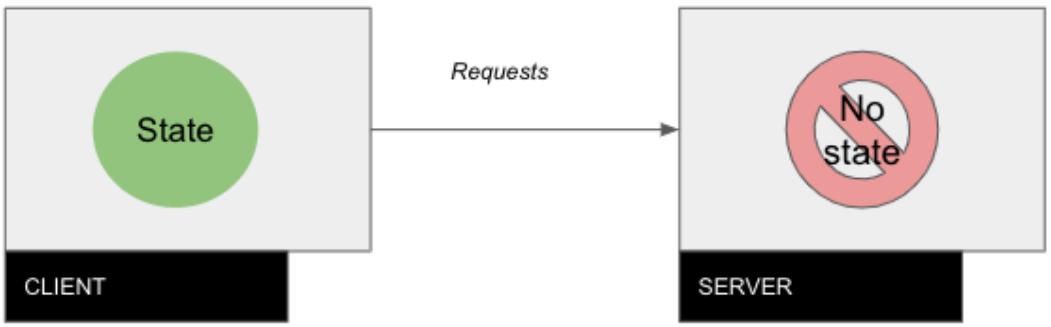
```xml
<employees>
  <employee>
    <firstName>John</firstName> <lastName>Doe</lastName>
  </employee>
  <employee>
    <firstName>Anna</firstName> <lastName>Smith</lastName>
  </employee>
  <employee>
    <firstName>Peter</firstName> <lastName>Jones</lastName>
  </employee>
</employees>
```

Εικόνα 13: Παράδειγμα XML για την αναπαράσταση υπαλλήλων που περιλαμβάνει παραδείγματα τριών υπαλλήλων [16]

1.5 Stateful vs Stateless API

Υπάρχουν δύο κατηγορίες στις οποίες χωρίζονται τα APIs: stateful και stateless [17, 18, 19, 20]. Σε ένα stateful API ο server είναι εκείνος που διατηρεί όλη την πληροφορία για την κατάσταση της συνεδρίας του client, δηλαδή περιέχει την απαραίτητη πληροφορία για την ταυτοποίηση του client και την κατάσταση που βρίσκεται κάθε στιγμή. Με αυτό τον τρόπο επιτρέπει στον χρήστη να χρησιμοποιεί την υπηρεσία αδιάκοπα καθώς δεν χρειάζεται να ξαναστέλνει την ίδια πληροφορία σε κάθε request που εκτελεί.

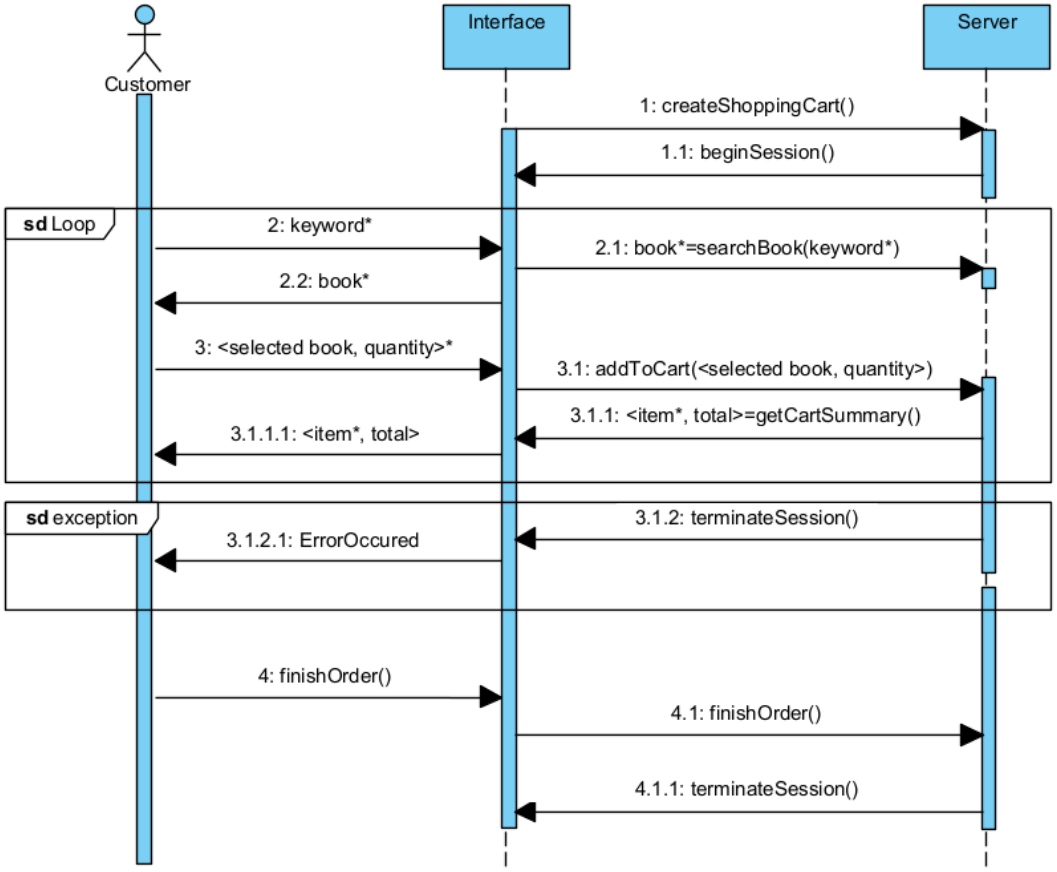
Σε ένα stateless API, κάθε request που εκτελεί ένας χρήστης περιέχει όλη την πληροφορία για την ταυτοποίηση του και την παρούσα κατάσταση της συνεδρίας. Ο server επεξεργάζεται κάθε request ξεχωριστά και δεν αποθηκεύει καμία πληροφορία σχετική με την συνεδρία του client.



Εικόνα 14: Τρόπος λειτουργίας ενός stateless API [18]

Κάθε κατηγορία εμφανίζει πλεονεκτήματα και μειονεκτήματα. Ένα stateful API εμφανίζει καλύτερη επίδοση καθώς ο server αποθηκεύει όλη την πληροφορία η οποία μπορεί να αξιοποιηθεί από τους προγραμματιστές για να την προσαρμόσουν στις

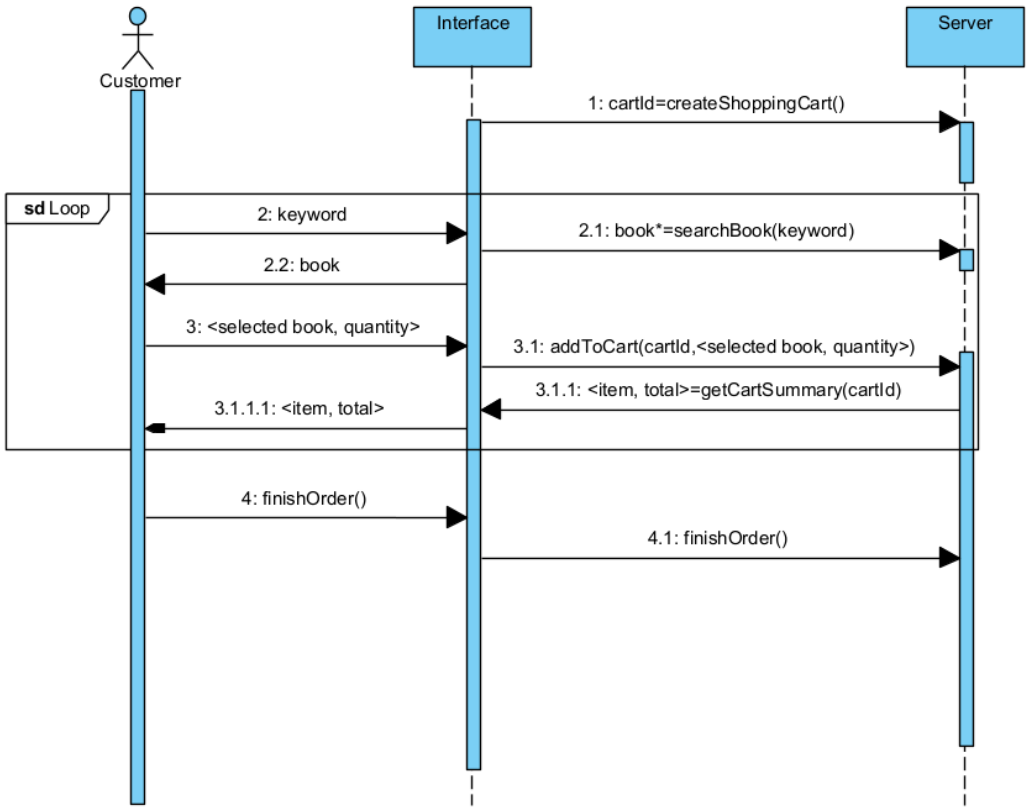
ανάγκες της εφαρμογής τους και να δημιουργήσουν αξιόπιστες εφαρμογές χωρίς να στηρίζονται σε τρίτους. Ακόμη επιτρέπει ενημερώσεις σε ζωντανό χρόνο λόγω της γρήγορης ανταπόκρισης των εφαρμογών, ενώ προσφέρει και μια αδιάκοπη εμπειρία χρήσης στους καταναλωτές. Ωστόσο, πέρα από το ότι είναι πιο περίπλοκα από ένα stateless API, είναι επίσης λιγότερο κλιμακώσιμα λόγω της πληροφορίας που αποθηκεύουν σε κάθε request και μπορεί να δημιουργήσουν καθυστερήσεις στην ανταπόκριση σε περίπτωση που υπάρχουν πολλαπλά requests. Τέλος, χρησιμοποιούν μεγαλύτερο εύρος ζώνης στο δίκτυο καθώς απαιτείται μεγαλύτερη ανταλλαγή πληροφοριών μεταξύ του client και του server. Ένα χαρακτηριστικό παράδειγμα stateful API είναι το SOAP API, το οποίο όμως μπορεί να γίνει και stateless.



Εικόνα 15: Stateful API sequence diagram για την ολοκλήρωση παραγγελίας βιβλίων

Από την άλλη, τα stateless APIs είναι πιο εύκολα και γρήγορα στην υλοποίηση από τα stateful APIs. Είναι cacheable, δηλαδή μπορούν να αποθηκευτούν σε τοπική cache για ταχύτερη πρόσβαση σε αυτά ενώ εμφανίζουν και μεγαλύτερη συμβατότητα με άλλες εφαρμογές χωρίς να απαιτούν ιδιαίτερη προσπάθεια διατήρησης. Στα αρνητικά τους είναι ότι εμφανίζουν γενικά χαμηλότερη επίδοση σε σχέση με τα stateful APIs και δεν προτείνονται για εφαρμογές που απαιτούν συχνές ενημερώσεις σε πραγματικό χρόνο.

Στα stateful APIs ο server ανοίγει τη συνεδρία και ο server την τερματίζει. Όλη η πληροφορία είναι κλειδωμένη στη συνεδρία και ο server έχει την ευθύνη να την διατηρεί. Αντίθετα, στα stateless APIs η ευθύνη μεταφέρεται στον client και ο φόρτος διαμοιράζεται. Για παράδειγμα, σε ένα stateless API που πραγματοποιεί παραγγελίες βιβλίων, ο server δεν χρειάζεται να συγκρατεί το id του καροτσιού του πελάτη αλλά ο client είναι αυτός που στέλνει σε κάθε request το id, γεγονός που ελαφρύνει σημαντικά τον φόρτο του server [Εικόνα 16]. Ένα χαρακτηριστικό παράδειγμα stateless API το οποίο θα χρησιμοποιήσουμε και εμείς στην έρευνά μας είναι το REST API [21].



Εικόνα 16: Stateless API sequence diagram για την ολοκλήρωση παραγγελίας βιβλίων

1.5.1 SOAP API

SOAP (Simple Object Access Protocol) [22, 23] ονομάζουμε ένα πρωτόκολλο βασισμένο σε XML το οποίο χρησιμοποιείται για την πρόσβαση υπηρεσιών διαδικτύου μέσω HTTP. SOAP API ονομάζουμε το API που βασίζεται σε αυτό το πρωτόκολλο και επιτρέπει εφαρμογές να αλληλεπιδρούν και να δημιουργούν, ενημερώνουν, διαγράφουν εγγραφές όπως κωδικούς, λογαριασμούς και άλλα αντικείμενα. Ένα SOAP API επιτρέπει εφαρμογές γραμμένες σε διαφορετικές γλώσσες προγραμματισμού να επικοινωνούν μεταξύ τους.

Με δεδομένο ότι η πολυπλοκότητα και η φύση του πρωτοκόλλου το καθιστούν εκτός πεδίου της παρούσας εργασίας, τα βασικά του χαρακτηριστικά συνοψίζονται ως ακολούθως:

- Αποτελεί ένα πρωτόκολλο βασισμένο εξ ολοκλήρου σε XML
- Είναι ανεξάρτητο πλατφόρμας
- Είναι «open standard» πρωτόκολλο δηλαδή μπορούν να το χρησιμοποιήσουν όλοι
- Είναι επέκταση του πρωτοκόλλου HTTP για αποστολή μηνυμάτων με XML
- Είναι χρήσιμο για μηνύματα broadcast από έναν υπολογιστή σε πολλούς
- Επιτρέπει τη αρχιτεκτονική client-server
- Επιτρέπει την μεταφορά δεδομένων για υπηρεσίες διαδικτύου
- Λειτουργεί στέλνοντας έναν «φάκελο» (envelope) ο οποίος περιέχει πληροφορίες για το τι πρόκειται να γίνει με τις υπηρεσίες διαδικτύου. Ο φάκελος αυτός στέλνεται στον πάροχο υπηρεσιών και για αυτό απαιτεί μεγαλύτερο εύρος ζώνης

1.5.2 REST API

REST (Representational State Transfer) είναι μια μορφή αρχιτεκτονικής λογισμικού η οποία επιβάλλει κάποιους περιορισμούς όταν δημιουργούμε υπηρεσίες διαδικτύου. REST API [21, 24] είναι ένα API το οποίο υπακούει σε αυτούς τους περιορισμούς και χρησιμοποιεί HTTP requests (GET, POST, PUT, PATCH, DELETE) για την επεξεργασία των δεδομένων.

Τα χαρακτηριστικά της αρχιτεκτονικής REST είναι:

- Γρήγορες υπηρεσίες διαδικτύου που καταναλώνουν μικρότερο εύρος ζώνης και λιγότερους πόρους
- Μπορεί να γραφεί σε οποιαδήποτε γλώσσα προγραμματισμού
- Οι υπηρεσίες μπορούν να εκτελεστούν σε οποιαδήποτε πλατφόρμα, δηλαδή ο server (πάροχος του API) και οι client (καταναλωτές του API) δεν δεσμεύονται από το ίδιο οικοσύστημα.
- Μια υπηρεσία χτισμένη με βάση την αρχιτεκτονική REST είναι ελαφριά και κλιμακώσιμη
- Χρησιμοποιεί HTTP μεθόδους όπως GET, PUT, POST, PATCH, DELETE για λειτουργίες CRUD (Create, Read, Update, Delete)
- Υποστηρίζει βασικές κρυπτογραφήσεις επικοινωνίας με TLS (Transport Layer Security)
- Είναι απλή στην ανάπτυξή της

Οι κανόνες που διέπουν την δημιουργία και λειτουργία ενός REST API είναι:

- Το πρόγραμμα του client πρέπει να είναι ανεξάρτητο του προγράμματος του server. Ο client πρέπει να γνωρίζει μόνο το URI του ζητούμενου πόρου και δεν πρέπει να έχει καμία άλλη αλληλεπίδραση με τον server
- Όλα τα API queries για την ίδια υπηρεσία πρέπει να έχουν όμοια μορφή
- Είναι stateteless
- Είναι cacheable
- Σε περίπτωση που τα responses περιλαμβάνουν εκτελέσιμο κώδικα να εκτελείται μόνο αν είναι απαραίτητο

1.5.3 SOAP VS REST

Το SOAP είναι προτιμότερο όταν θέλουμε να πραγματοποιήσουμε δοσοληψίες (transactions) οι οποίες απαιτούν πολλαπλές κλήσεις σε μια υπηρεσία για να ολοκληρώσουμε μια διεργασία, με δεδομένο ότι οι κλήσεις αυτές ελέγχονται από τον πάροχο της υπηρεσίας (server). Είναι πιο ασφαλές από το REST καθώς υποστηρίζει WS-security (web services security) και WS-AtomicTransactions (web services atomic transactions), για αυτό και προτιμάται όταν χρειαζόμαστε αξιοπιστία και χρησιμοποιείται συχνά στην τραπεζική βιομηχανία, σε χρηματιστήρια και σε άλλες κρίσιμες οικονομικές εφαρμογές. Σε περίπτωση μιας αποτυχημένης δοσοληψίας, το SOAP θα ξαναπροσπαθήσει αυτόματα να πραγματοποιήσει τη συναλλαγή σε αντίθεση με το REST όπου απαιτείται χειροκίνητος χειρισμός από την εφαρμογή που έκανε το request, δηλαδή από τον client.

Αν όμως δεν ασχολούμαστε με συναλλαγές και δεν αποτελεί η αξιοπιστία την βασική μας προτεραιότητα, ένα REST API και γενικότερα τα stateless APIs, ίσως να αποτελούν μια καλύτερη επιλογή. Σε γενικές γραμμές, τα stateless APIs γίνονται όλο και πιο δημοφιλή λόγω της απλότητας και της κλιμακωσιμότητας τους καθώς αποτελούν την ιδανική λύση για εταιρίες που επιθυμούν να υλοποιήσουν εύκολα και γρήγορα τις εφαρμογές τους και θέλουν να είναι ελαστικές στις απαιτήσεις τους. Ο άλλος βασικός λόγος που όλο και περισσότεροι τα επιλέγουν, είναι η συμβατότητα τους με διαφορετικές εφαρμογές και συστήματα, καθώς οι εφαρμογές αυτές είναι ευθύνη των πελατών (καταναλωτών) του REST API. Το REST API είναι μια ελαφριά και κλιμακώσιμη υπηρεσία η οποία κάνει αποδοτική χρήση του εύρους ζώνης του διαδικτύου, ενώ υποστηρίζει και πολλούς τύπους δεδομένων. Αν και το SOAP είναι πιο ασφαλές από το REST και προτιμάται σε επίπεδο επιχείρησης, το REST κυριαρχεί όταν χρησιμοποιούμε public APIs καθώς είναι πιο εύκολο να γραφεί και να το κατανοήσουμε [22, 25].

1.6 API Documentation

Όταν θέλουμε να χρησιμοποιήσουμε ένα API που διατίθεται από ένα τρίτο μέρος, πρέπει πρώτα να κατανοήσουμε πώς λειτουργεί. Αλλά ακόμα και όταν πρόκειται για τη δική μας εφαρμογή χρειαζόμαστε έναν εύκολο τρόπο να μεταφέρουμε την πληροφορία για τον σωστό τρόπο χρήσης ενός API στους συνεργάτες μας. Σε αυτή τη διαδικασία μας βοηθάει το Documentation. Ένα API Documentation [26] είναι ένα έγγραφο, γενικά ένα κείμενο οδηγιών, που περιγράφει πώς πρέπει να χρησιμοποιηθεί ένα API.

Ένα API χωρίς το απαραίτητο Documentation, δεν είναι πάντα απλό να χρησιμοποιηθεί. Ένα API μπορεί να αποτελείται από πολλά endpoints και κάθε ένα από αυτά να επιστρέφει διαφορετική πληροφορία και να δέχεται διαφορετικές παραμέτρους. Υπάρχουν πολλοί λόγοι που χρειαζόμαστε ένα αναλυτικό documentation με την κατάλληλη πληροφορία [26]:

- Αυξάνει την πιθανότητα προγραμματιστές να «υιοθετήσουν» το API, να το επεκτείνουν και να το βελτιώσουν. Το API Documentation βελτιώνει την εμπειρία χρήσης για τους προγραμματιστές και το καθιστά πιο προσιτό σε αυτούς ώστε να θέλουν να βοηθήσουν στην ανάπτυξή του.

- Αυξάνει την δημοφιλία του API. Όσο πιο εύκολα και γρήγορα ένας χρήστης μαθαίνει να χρησιμοποιεί το API, τόσο πιο πιθανό είναι να το προτείνει και σε άλλα άτομα.
- Κερδίζει χρόνο και χρήματα. Με ένα καλό Documentation, μειώνεται ο χρόνος που απαιτείται προκειμένου ένας νέος χρήστης να μάθει να χρησιμοποιεί το API. Επίσης κερδίζει χρήματα καθώς δεν χρειάζεται οι προγραμματιστές να εξηγούν στους χρήστες πώς να το χρησιμοποιήσουν.
- Ευκολότερη συντήρηση. Οι προγραμματιστές γνωρίζουν με λεπτομέρεια κάθε λειτουργία του API και μπορούν να προβούν πιο γρήγορα σε επιδιορθώσεις και ενημερώσεις.

Γενικεύοντας, αξίζει να σημειωθεί ότι ένα API δεν αποτελεί μόνο ένα μέσο που συνδέει εφαρμογές μεταξύ τους, αλλά μπορεί και το ίδιο να αποτελεί και μια υπηρεσία, δηλαδή να είναι προϊόν το οποίο προσφέρεται με χρέωση. Για παράδειγμα, μπορούμε να δημιουργήσουμε και να πουλήσουμε ένα API που παρέχει μετεωρολογικά δεδομένα όπως θερμοκρασία, υγρασία κ.α. για μια περιοχή. Από εκεί και πέρα μπορεί να αξιοποιηθεί τόσο από εταιρείες που παρέχουν μετεωρολογικές προβλέψεις όσο και από αγροτικές εταιρείες που επιθυμούν να γνωρίζουν το κλίμα ανά πάσα στιγμή για να καλλιεργήσουν τα προϊόντα τους. Η παροχή ενός API με το κατάλληλο documentation, το καθιστά πιο προσιτό σε μελλοντικούς πελάτες και προσελκύει μεγαλύτερο ενδιαφέρον. Ακόμη, αυξάνει τη δημοφιλία του προϊόντος και συμβάλλει στην αύξηση της τελικής του αξίας. Υπάρχουν πολλοί τρόποι με τους οποίους μπορούμε να γράψουμε το documentation για ένα API. Ένα παράδειγμα προδιαγραφής για API Documentation αποτελεί το OpenAPI [27].

1.7 OpenAPI

Το OpenAPI Specification (OAS) [27], αποτελεί μία ανοικτή («open source») προδιαγραφή για περιγραφή και documentation των APIs που δίνει τη δυνατότητα σε άτομα να ανακαλύψουν και να κατανοήσουν τις δυνατότητες και τον τρόπο που λειτουργεί ένα API χωρίς να έχουν πρόσβαση στον ίδιο τον πηγαίο κώδικα αυτού. Μπορεί στη συνέχεια να αξιοποιηθεί από ποικίλα εργαλεία για να παρουσιάσουν το API. Ένα OpenAPI documentation μπορεί να παρασταθεί ως ένα JSON object και να απεικονιστεί σε yaml ή json αρχείο. Τα παραδείγματα που θα αναλύσουμε είναι σε yaml [28].

Η yaml [28] είναι μια γλώσσα σειριοποίησης (serialization) δεδομένων η οποία είναι αναγνώσιμη από άνθρωπο. Χρησιμοποιείται για αρχεία διαμόρφωσης και σε εφαρμογές που αποθηκεύονται ή μεταδίδουν δεδομένα.

Ένα OpenAPI documentation μπορεί να περιέχει πληροφορίες που σχετίζονται με τις εξής κατηγορίες:

- Meta information
- Endpoints: Request bodies, Response bodies, parameters
- Reusable Components: Schemas, Parameters, Responses, Other Components

1.7.1 Meta Information

Στην απλή του μορφή ένα OpenAPI περιέχει μεταδεδομένα όπως ο τίτλος, η περιγραφή, το url του server και η έκδοση.

```
22   title: Sample Pet Store App
23   summary: A pet store manager
24   description: This is a sample server for a pet store.
25   TermsOfService: https://example.com/terms/
26   contact:
27     name: API Support
28     url: https://www.example.com/support
29     email: support@example.com
30   license:
31     name: Apache 2.0
32     url: https://www.apache.org/licenses/LICENSE-2.0.html
33   version: 1.0.1
```

Εικόνα 17: OpenAPI μεταδεδομένα

1.7.2 Path Items

Κάθε path item αποτελεί ένα endpoint του API και περιλαμβάνει τη μέθοδο του request. Στο παράδειγμα η μέθοδος είναι GET:

```
pets:
 get:
   description: Returns all pets from the system that the user has access to
```

Εικόνα 18: OpenAPI path items

1.7.3 Responses

Στο response του endpoint περιέχεται πληροφορία σχετικά με το τι επιστρέφει το endpoint όταν καλείται. Περιέχει το status code, την περιγραφή, ενώ γίνεται και αναφορά σε κάποιο component το οποίο αναπαριστά το object και τα attributes που περιέχει.

```yaml
/pets
  get:
    description: Returns all pets from the system that the user has access to
    responses:
      '200':
        description: A list of pets.
        content:
          application/json:
            schema:
              type: array
              items:
                $ref: '#/components/schemas/pet'
```

Εικόνα 19: OpenAPI responses

1.7.4 Parameters

Ιδιαίτερη αξία για την εργασία μας έχουν τα parameters καθώς περιέχουν πληροφορία που χρειάζεται ένα endpoint για να κληθεί σωστά.

Query parameters

Τα query parameters εμφανίζονται στο τέλος του URL, χωρίζονται από αυτό με τον χαρακτήρα «?» και περιλαμβάνουν το attribute μαζί με την τιμή που δίνουμε ως παράμετρο.

```
http://mydomain.com/example/myparam1=123&myparam2=abc
```

Εικόνα 20: Query parameters

```yaml
name: username
in: path
description: username to fetch
required: true
schema:
  type: string
```

Εικόνα 21: OpenAPI query parameters

Path parameters

Τα path parameters εμφανίζονται στο τέλος του URL, πριν από τον χαρακτήρα «?» και περιλαμβάνουν μόνο την τιμή του attribute.

```
http://mydomain.com/service/user/{user}/bicycles/{bicycleid}
```

Εικόνα 22: Path parameters

```yaml
name: id
in: query
description: ID of the object to fetch
required: false
schema:
  type: array
  items:
    type: string
style: form
explode: true
```

Εικόνα 23: OpenAPI path parameters

Request body

Με κάθε request μεθόδου POST PUT ή PATCH συνήθως απαιτείται ένα request body. Τα request bodies αποτελούν ολοκληρωμένα JSON objects ίδιας μορφής με τα response bodies. Περιέχουν την περισσότερη πληροφορία και είναι αυτά με τα οποία θα ασχοληθούμε κυρίως στην εργασία μας. Κάθε request body στο OAS μπορεί να κάνει αναφορά σε component που αναπαριστά το object αυτό μαζί με τα attributes που περιέχει ενώ μπορεί να περιέχει και παραδείγματα.

```
{
“myparam1”=”123”,
“myparam2”=”456”,
“myparam3”=”789”
}
```

Εικόνα 24: Request body parameters

```yaml
description: user to add to the system
content:
  'application/json':
    schema:
      $ref: '#/components/schemas/User'
    examples:
      user:
        summary: User Example
        externalValue: 'https://foo.bar/examples/user-example.json'
  'application/xml':
    schema:
      $ref: '#/components/schemas/User'
    examples:
      user:
        summary: User example in XML
        externalValue: 'https://foo.bar/examples/user-example.xml'
  'text/plain':
    examples:
      user:
        summary: User example in Plain text
        externalValue: 'https://foo.bar/examples/user-example.txt'
  '*/*':
    examples:
      user:
        summary: User example in other format
        externalValue: 'https://foo.bar/examples/user-example.whatever'
```

Εικόνα 25: OpenAPI Request body

1.7.4.4 Components, schemas

Στα components ορίζονται τα schemas τα οποία αναπαριστούν objects με τα attributes που περιέχουν. Συχνά αναπαριστούν τα responses και τα request bodies των endpoints.

```yaml
components:
  schemas:
    Category:
      type: object
      properties:
        id:
          type: integer
          format: int64
        name:
          type: string
    Tag:
      type: object
      properties:
        id:
          type: integer
          format: int64
        name:
          type: string
```

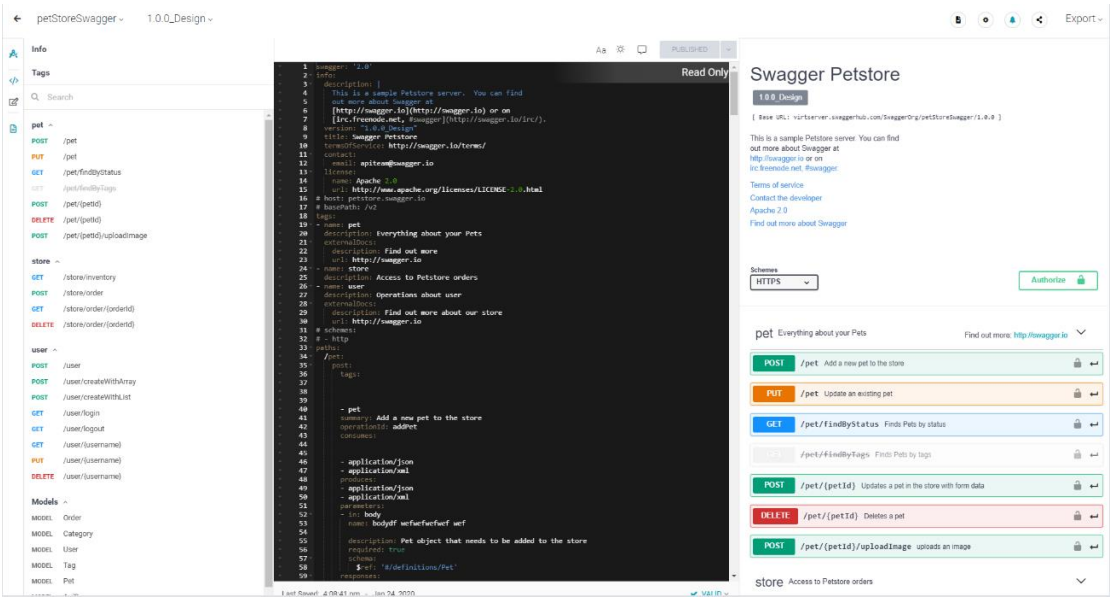
Εικόνα 26: OpenAPI components, OpenAPI schemas

1.8 Εργαλεία για την παραγωγή API Documentation

Πολλές προσπάθειες έχουν γίνει για την κατά το δυνατόν αυτόματη παραγωγή του Documenation καθώς είναι αυτό που μπορεί να αναδείξει την αξία ενός API. Στον κόσμο του λογισμικού, έχουν αναπτυχθεί πολλά εργαλεία που εξειδικεύονται σε αυτό το σκοπό. Μολονότι τα εργαλεία αυτά παριστάνουν τα δομικά στοιχεία του API, η εννοιολογία (semantics) παραμένει δύσκολο να εξαχθεί εντελώς αυτόματα.

1.8.1 Swaggerhub

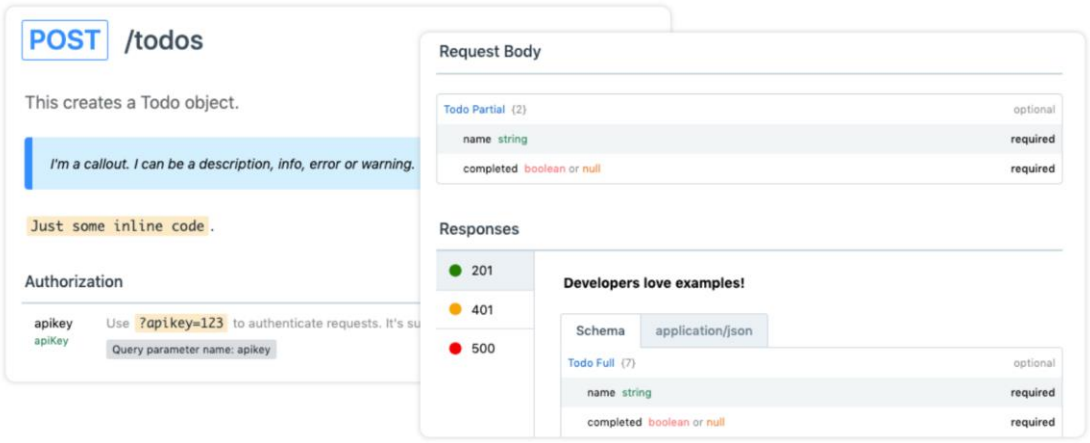
Το swaggerhub [29] αποτελεί ένα εργαλείο που εξειδικεύεται στην επιτάχυνση και απλοποίηση της διαδικασίας του Documentation ενός API. Χρησιμοποιώντας τον API editor μπορούμε να επιτύχουμε Documentation συμβατά με το OpenAPI specification (OAS). Προτείνεται για το Documentation των APIs μεγάλης κλίμακας.



Εικόνα 27: Swaggerhub API Documentation

1.8.2 Stoplight

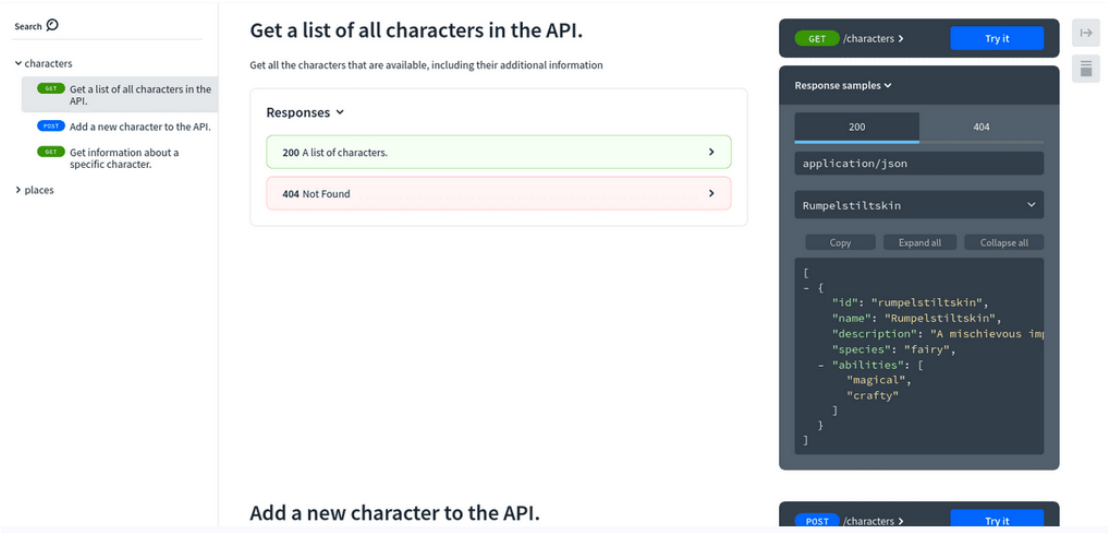
Με το εργαλείο REST API documentation του Stoplight [30] μπορούμε να δημιουργήσουμε και να διατηρήσουμε API Documentations online. Δίνει τη δυνατότητα να δημιουργήσουμε διαδραστικά API documentations στα οποία μπορούμε να συμπεριλάβουμε tutorials, οδηγούς για νέους χρήστες καθώς και παραδείγματα παραγωγής κώδικα.



Εικόνα 28: Stoplight API Documentation

1.8.3 Redocly

Το Redocly [31] είναι ένα εργαλείο που χρησιμοποιείται για την αυτοματοποίηση της διαδικασίας του API documentation pipeline. Με το Redocly μπορούμε να αποθηκεύσουμε documentations σε ένα λογισμικό τύπου version-control, το οποίο επιτρέπει την προεπισκόπηση μιας έκδοσης του documentation πριν περάσει στην παραγωγή.



Εικόνα 29: Redocly API Documentation

1.9 Εργαλεία για την παρουσίαση του API Documentation

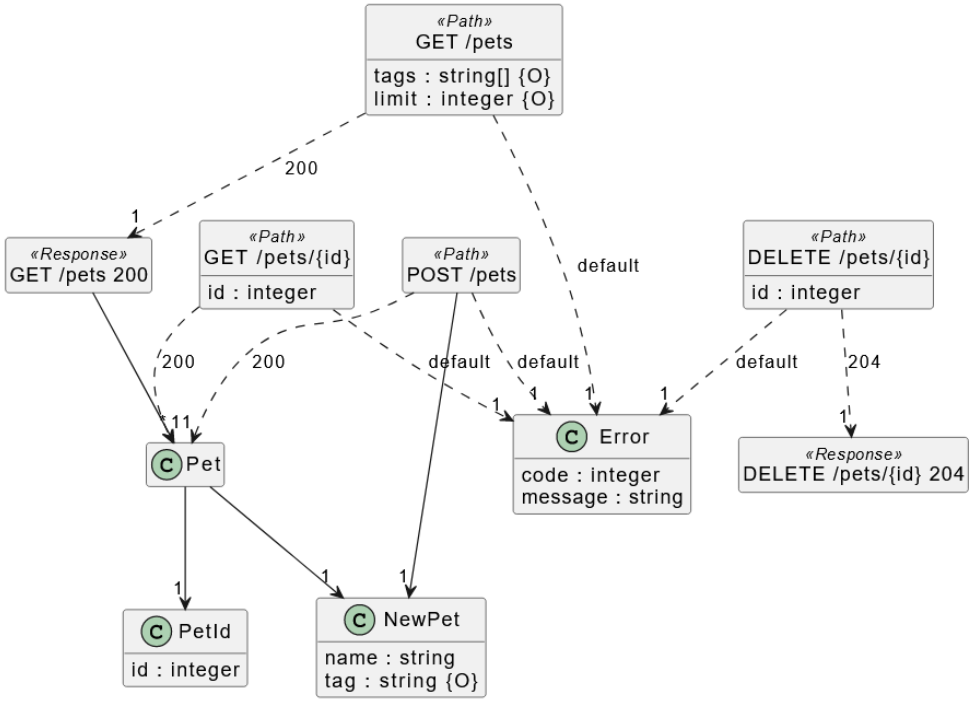
Αντίστοιχα υπάρχουν εργαλεία για την παρουσίαση των REST APIS. Ένα documentation αποκτάει μεγαλύτερη αξία όταν μπορεί να παρουσιαστεί και μέσω ενός γράφου.

1.9.1 Plant UML

Ένα παράδειγμα αποτελεί το plantUML [32] το οποίο χρησιμοποιεί ένα απλό plantUML Text για να παραγάγει UML [33] διαγράμματα.

Εικόνα 30: PlantUML UI

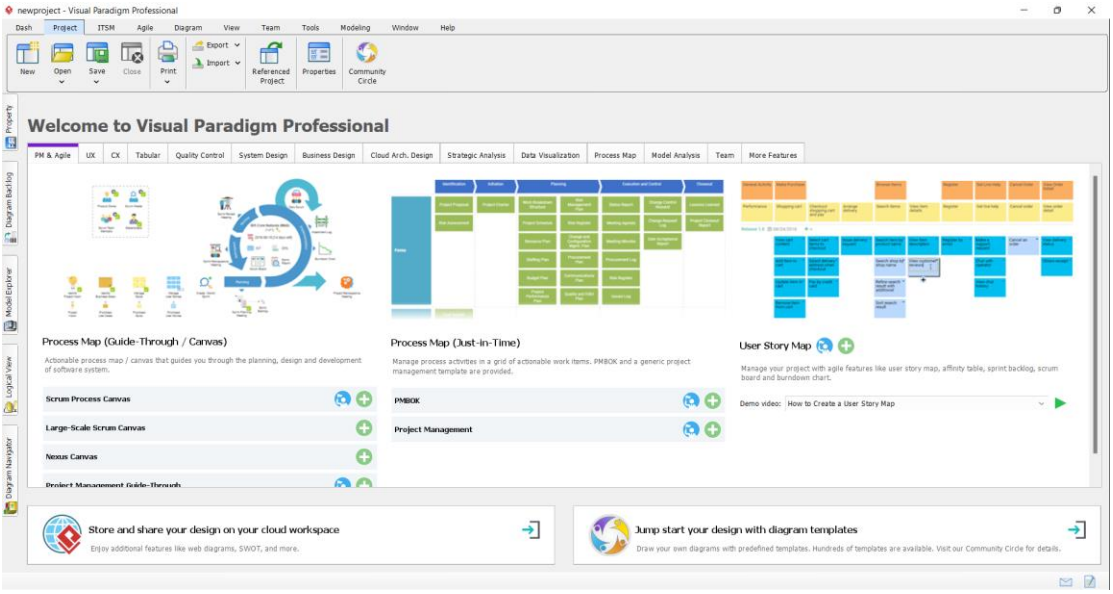
Ακολουθεί ένα παράδειγμα παρουσίασης ενός REST API μέσω του εργαλείου plantUML:



Εικόνα 31: Παράδειγμα παρουσίασης REST API μέσω του PlantUML

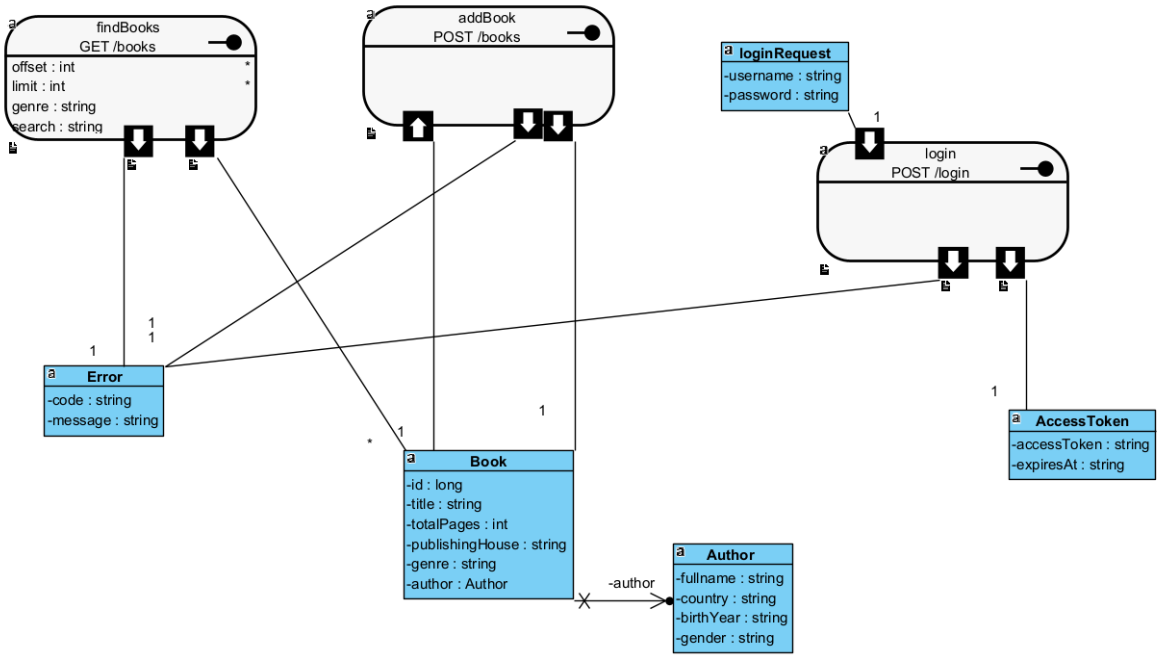
1.9.2 Visual Paradigm

Το Visual Paradigm [34] είναι ένα ισχυρό εργαλείο σχεδίασης και διαχείρισης για συστήματα Information Technology. Αποτελεί ένα εργαλείο UML CASE που υποστηρίζει το UML 2, το SysML και το Business Process Modeling Notation. Επιτρέπει τη δημιουργία πλούσιων διαγραμμάτων UML διαφόρων τύπων όπως Class Diagrams [35] και Use Case diagrams [36]. Επίσης επιτρέπει την αυτόματη παραγωγή OpenAPI documentations για REST APIs που έχουν σχεδιαστεί στο εργαλείο αυτό. Στο επόμενο κεφάλαιο περιγράφονται αναλυτικά οι δυνατότητες που προσφέρει αυτό το εργαλείο.



Εικόνα 32: Visual Paradigm UI

Ακολουθεί ένα παράδειγμα παρουσίασης ενός REST API μέσω του εργαλείου Visual Paradigm:



Εικόνα 33: Παράδειγμα παρουσίασης REST API μέσω του Visual Paradigm

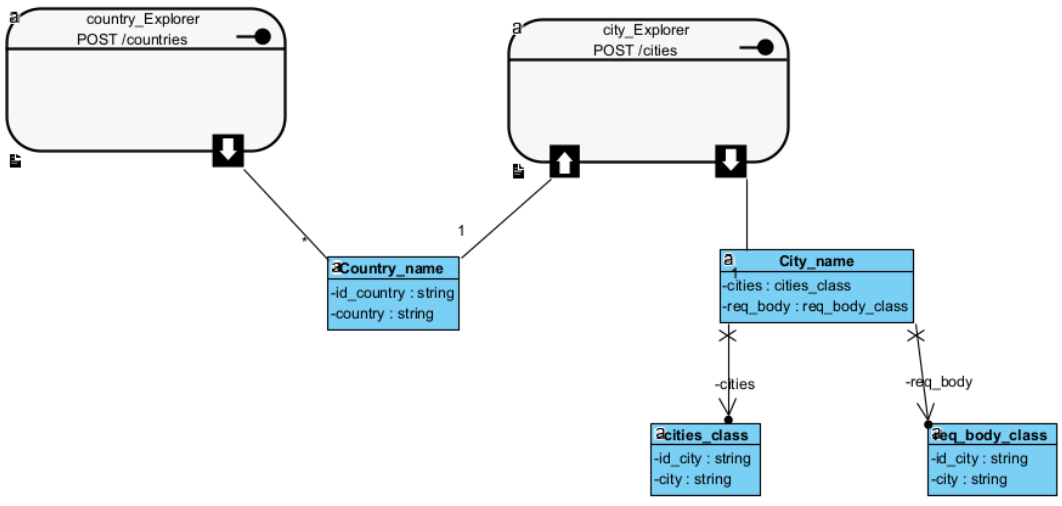
Βλέπουμε λοιπόν ότι η αυτόματη παραγωγή ενός ολοκληρωμένου Documentation για ένα REST API και η παρουσίασή του αποτελεί ένα φλέγον ζήτημα και έχει προσελκύσει το ενδιαφέρον πολλών εμπλεκόμενων στην ανάπτυξη λογισμικού

1.10 Εξαρτήσεις μεταξύ των endpoints ενός API

Ακόμα και ένα αναλυτικό documentation μπορεί να μην είναι αρκετό. Τι συμβαίνει όταν τα endpoints εξαρτώνται το ένα από το άλλο; Για παράδειγμα, ένα endpoint προτού κληθεί, μπορεί να προϋποθέτει την κλήση ενός διαφορετικού endpoint, το οποίο θα διαθέσει τις τιμές των παραμέτρων που χρειάζεται προκειμένου να κληθεί σωστά. Αυτή η πληροφορία είναι σημαντική τόσο για τους χρήστες που επιθυμούν να καλέσουν αυτά τα endpoints όσο και για προγραμματιστές που επιθυμούν να επεκτείνουν ένα REST API και οφείλουν να γνωρίζουν αυτές τις εξαρτήσεις μεταξύ των endpoints προκειμένου να διατηρηθεί η ομαλή λειτουργία του.

Οι εξαρτήσεις αυτές, μάλιστα, μπορεί να εκδηλώνονται και σε περιπτώσεις χρήσης μέσω ενός user interface που καλεί APIs. Για παράδειγμα, θέλουμε να πραγματοποιήσουμε μία παραγγελία και έχουμε 3 API endpoints στη διάθεσή μας. Το πρώτο επιστρέφει τον κωδικό του προϊόντος, το δεύτερο ελέγχει το απόθεμα του προϊόντος με βάση τον κωδικό του και το τρίτο το τοποθετεί στην παραγγελία. Αν θέλουμε να εισαγάγουμε ένα προϊόν, πρέπει πρώτα να χτυπήσουμε το πρώτο endpoint για να πάρουμε τον κωδικό του, στη συνέχεια να θέσουμε αυτό τον κωδικό ως παράμετρο στο δεύτερο, προκειμένου να ελέγξουμε τη διαθεσιμότητα και μόνο αν βρίσκεται σε απόθεμα να το προσθέσουμε στην παραγγελία μέσω του τρίτου. Αυτές οι εξαρτήσεις έχουν αξία να είναι καταγεγραμμένες προκειμένου να γνωρίζουμε τη σωστή σειρά κλήσεων των API endpoints, προκειμένου να αποφεύγονται τα λάθη και να αυξάνεται η παραγωγικότητα.

Έχει αξία λοιπόν, χρησιμοποιώντας όσο λιγότερα εργαλεία μπορούμε, να ερευνήσουμε σε ποιες περιπτώσεις μπορούμε να διευκολύνουμε την αυτόματη παραγωγή ενός API Documentation στο οποίο θα αποτυπώνονται οι εξαρτήσεις των endpoints του REST API. Θα ερευνήσουμε ακόμη αν μπορούμε να ανιχνεύσουμε αυτές τις εξαρτήσεις μέσω του τελικού διαγράμματος καθώς και σε ποιες ακόμη περιπτώσεις είναι χρήσιμη αυτή η πληροφορία. Στο παρακάτω σχήμα βλέπουμε δύο endpoints ενός REST API τα οποία συνδέονται με σχέσεις εξάρτησης και θα τις αναλύσουμε λεπτομερώς σε επόμενο κεφάλαιο.



Εικόνα 34: Παράδειγμα REST API με εξαρτήσεις μεταξύ των endpoints

1.11 Αντικείμενο της εργασίας

Αντικείμενο της παρούσας διπλωματικής είναι η ανάπτυξη ενός εργαλείου λογισμικού που ανιχνεύει τις σχέσεις μεταξύ των endpoints και παράγει αυτόματα ένα κατάλληλο API Documentation το οποίο μπορεί να αναπαρασταθεί με έναν ευπαρουσίαστο τρόπο. Στην προσπάθεια αυτή θα μας βοηθήσουν σημαντικά τα εργαλεία Postman [37] και Visual Paradigm [34] καθώς και ένα εργαλείο που αναπτύχθηκε από τη συνάδελφο Ναυσικά Αμπατζή στη διπλωματική της [38]. Με το Postman δημιουργούμε συλλογές από requests σε ένα API. Με το εργαλείο της Ναυσικάς μετατρέπουμε αυτές τις συλλογές σε OpenAPI. Με το Visual Paradigm αναπαριστούμε OpenAPI documentations και τα παρουσιάζουμε σε έναν αναλυτικό γράφο με κόμβους τα endpoints.

Στο σύστημα που αναπτύξαμε, χειριζόμαστε υπάρχοντα OpenAPI αρχεία για να εισαγάγουμε σε αυτά την απαραίτητη πληροφορία και να εμφανιστούν οι σχέσεις εξάρτησης μεταξύ των endpoints στο Visual Paradigm. Ακόμη, χειριζόμαστε και την περίπτωση όπου δεν υπάρχει κάποιο OpenAPI αρχείο διαθέσιμο και έχουμε στη διάθεση μας μόνο το REST API.

1.12 Οργάνωση του τόμου

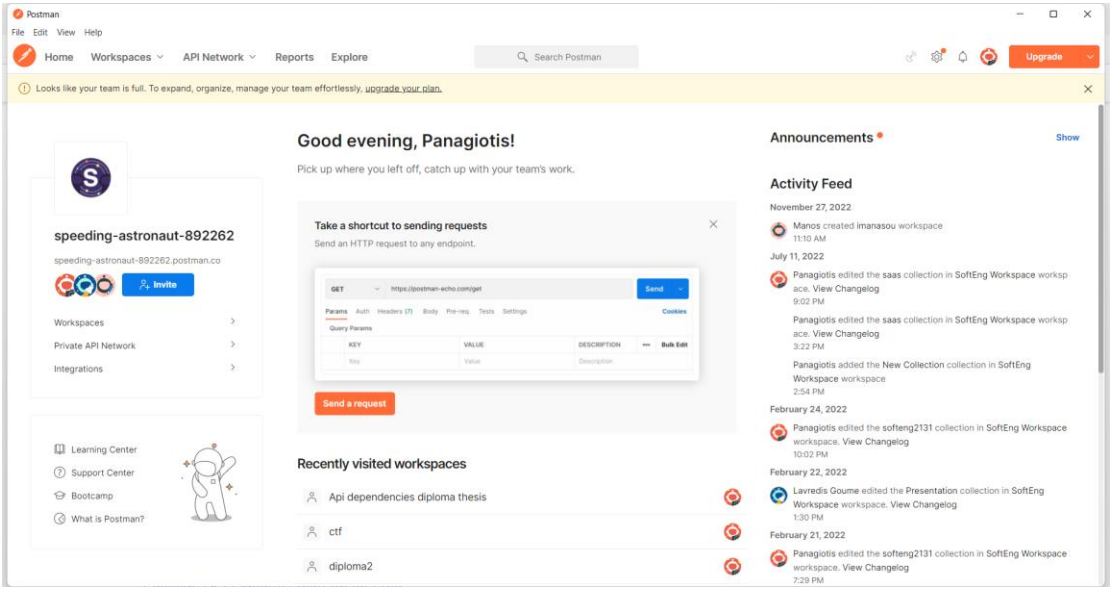
Η εργασία αυτή είναι οργανωμένη σε επτά κεφάλαια: Στο Κεφάλαιο 2 δίνεται το θεωρητικό υπόβαθρο των βασικών τεχνολογιών που σχετίζονται με τη διπλωματική αυτή. Αυτές είναι το Postman, το Visual Paradigm και το εργαλείο Postman to OpenAPI. Στο Κεφάλαιο 3 περιγράφεται η ιδανική μορφή που θέλουμε να έχει ένα API για να εμφανίζονται οι σχέσεις εξάρτησης που αναζητάμε στην διπλωματική μας. Στο Κεφάλαιο 4 παρουσιάζεται η ανάλυση και η σχεδίαση του συστήματος, δηλαδή η περιγραφή των υποσυστημάτων και των εφαρμογών του. Για κάθε περίπτωση παρουσιάζονται διαγράμματα χρήσης και τα αποτελέσματα που προκύπτουν για συγκεκριμένα παραδείγματα. Στο Κεφάλαιο 5 εφαρμόζουμε το σύστημά μας στο δημόσιο API της PayPal. Στο Κεφάλαιο 6 παρουσιάζεται η χρήση ενός generative AI tool ως εναλλακτικός τρόπος για την αυτοματοποίηση μέρους του συστήματος. Τέλος στο κεφάλαιο 7 παρουσιάζονται τα συμπεράσματα της διπλωματικής εργασίας μας και προτείνονται ιδέες για μελλοντικές επεκτάσεις.

2 Εργαλεία που αξιοποιήσαμε στο σύστημά μας

Στο κεφάλαιο αυτό παρουσιάζονται τα εργαλεία και οι τεχνολογίες που επιλέξαμε να χρησιμοποιήσουμε. Η επιλογή τους έγινε με βάση κριτήρια που αφορούν την ευχρηστία και την σωστή παρουσίαση του τελικού Documentation.

2.1 Postman

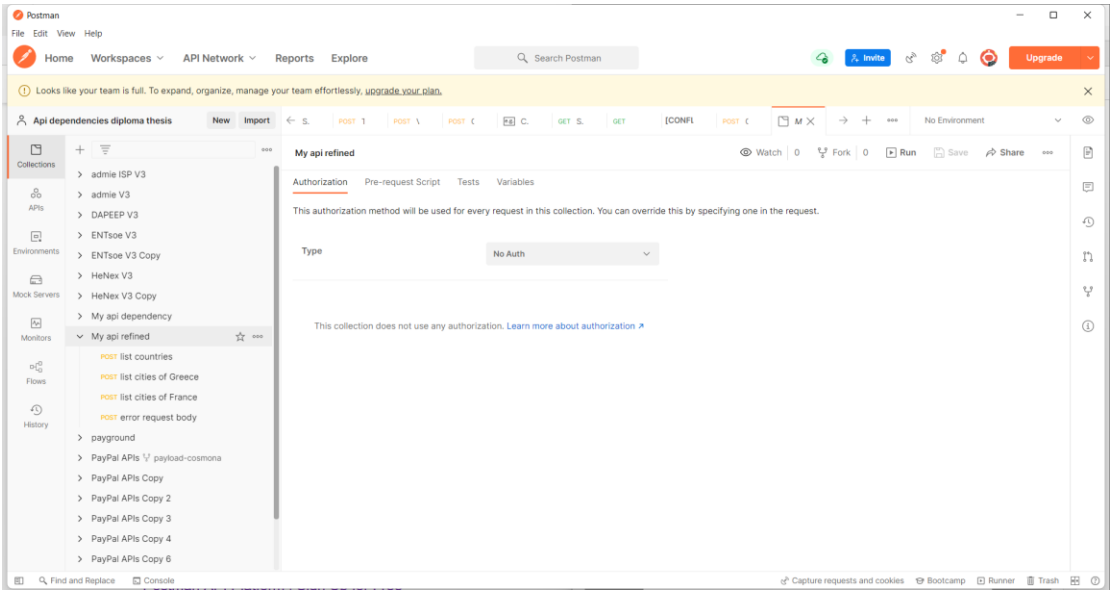
Το Postman [37] είναι μια πλατφόρμα API για προγραμματιστές με σκοπό να σχεδιάσουν, να δημιουργήσουν και να δοκιμάσουν τα APIs τους.



Εικόνα 35: Postman UI

2.1.1 Postman Collections

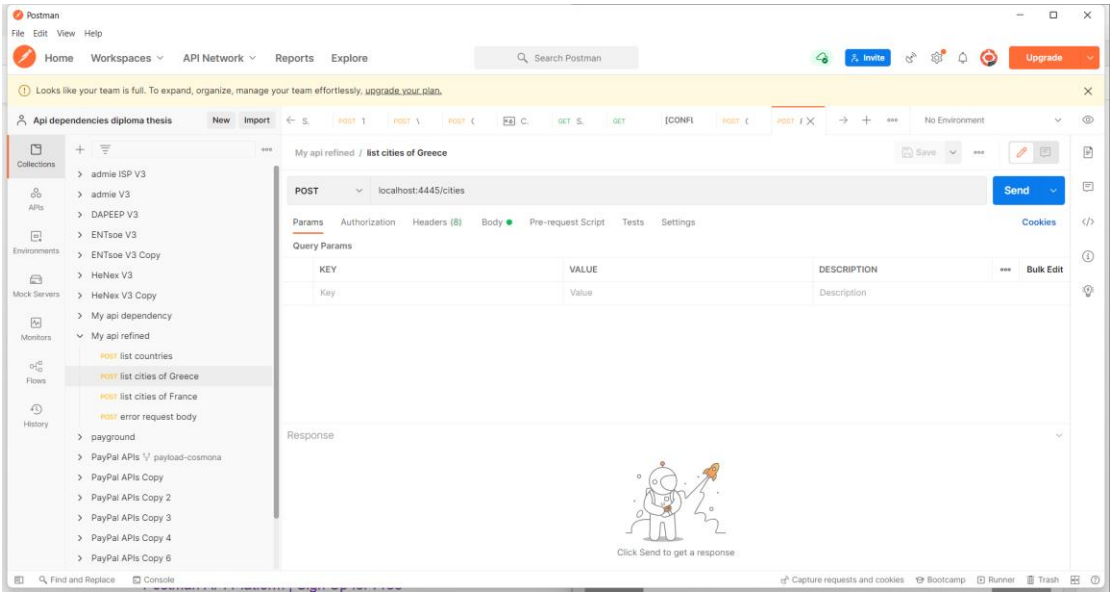
Μέσω των workspaces μπορούμε να δημιουργήσουμε δικές μας συλλογές [39] από requests σε REST APIs τις οποίες μπορούμε στη συνέχεια να κάνουμε export.



Εικόνα 36: Postman collections

2.1.2 Requests

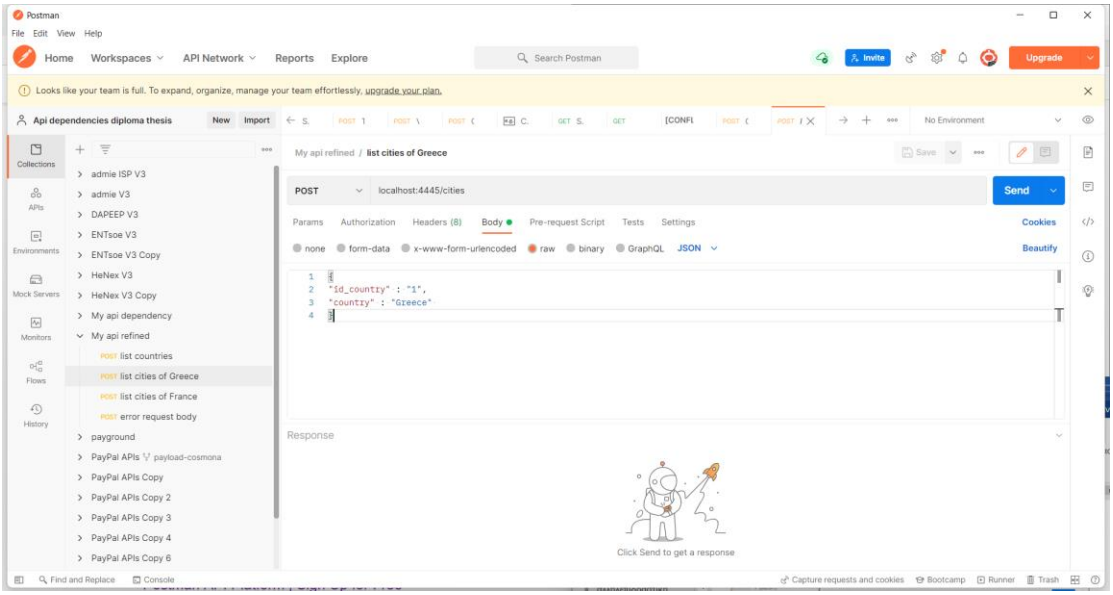
Σε κάθε request ορίζουμε τον τύπο της μεθόδου (GET, POST, PUT κ.α.) καθώς και το endpoint το οποίο επιθυμούμε να καλέσουμε.



Εικόνα 37: Postman requests

2.1.3 Body Data

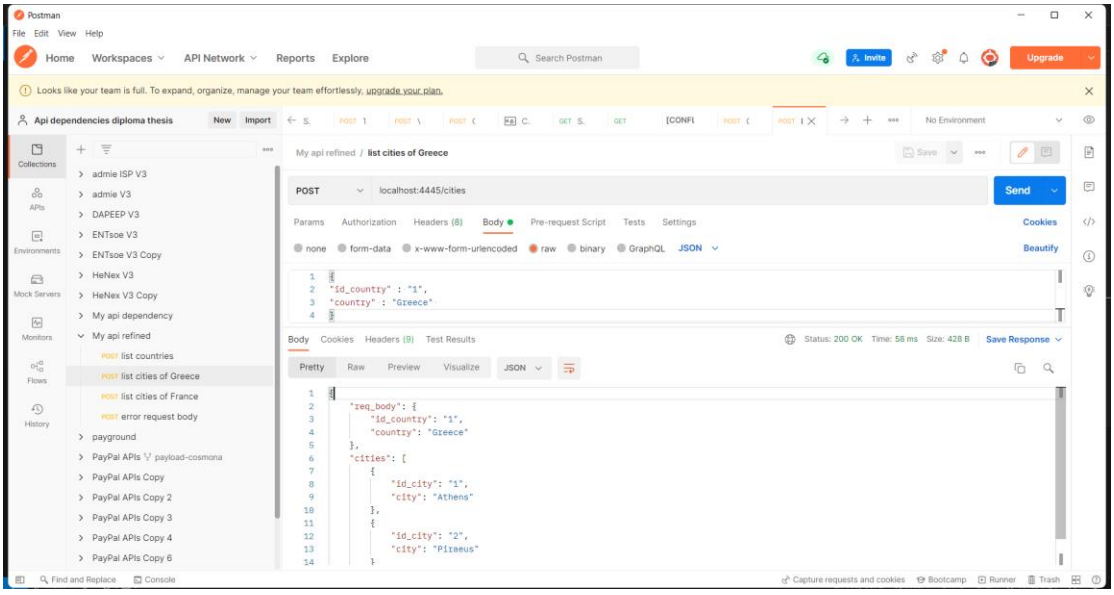
Μέσω του Body μπορούμε να ορίσουμε το JSON object που δίνουμε ως παράμετρο και αποτελεί το request body του request μας.



Εικόνα 38: Postman Body data

2.1.4 Responses

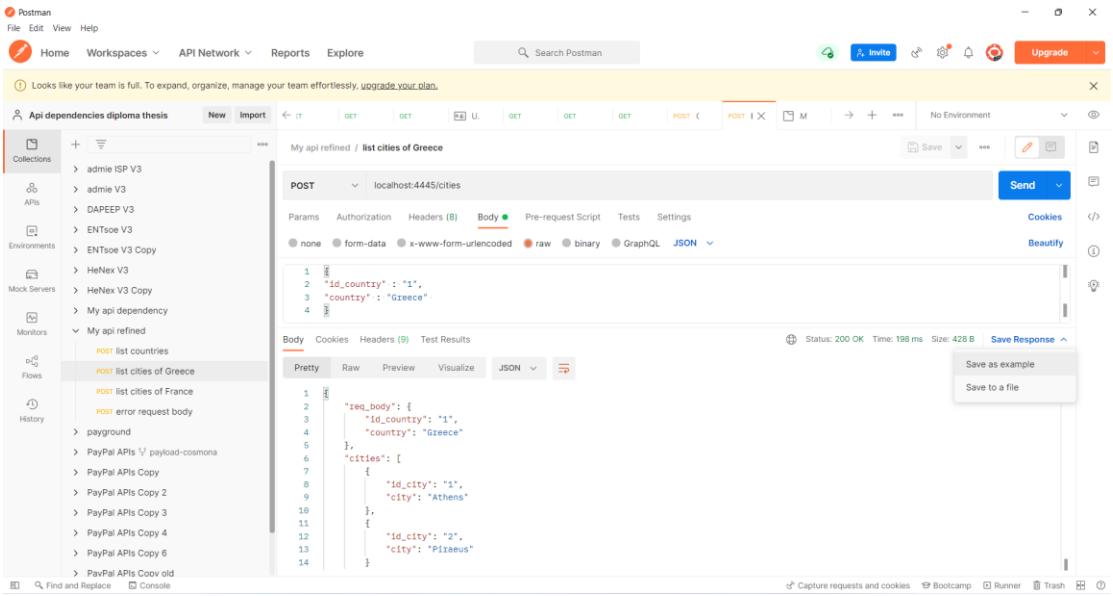
Μέσω το Body των responses βλέπουμε το response που λαμβάνουμε αφού καλέσουμε το endpoint.



Εικόνα 39: Postman responses

2.1.5 Postman Examples

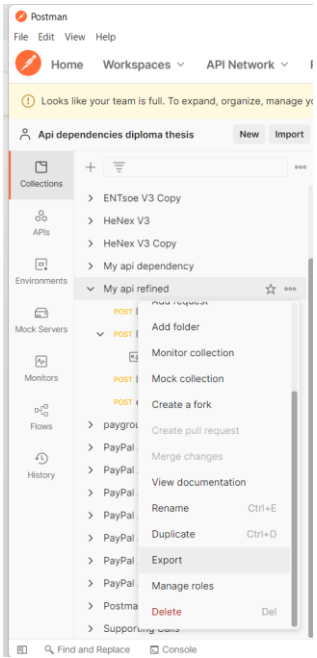
Επιπλέον έχουμε την δυνατότητα να αποθηκεύσουμε το response body ως παράδειγμα [40] για να είναι διαθέσιμη η πληροφορία όταν κάνουμε export τη συλλογή μας.



Εικόνα 40: Postman examples

2.1.6 Export postman collection

Τέλος μπορούμε να κάνουμε εξαγωγή το postman collection που δημιουργήσαμε και να αποθηκευτεί σε ένα αρχείο json.



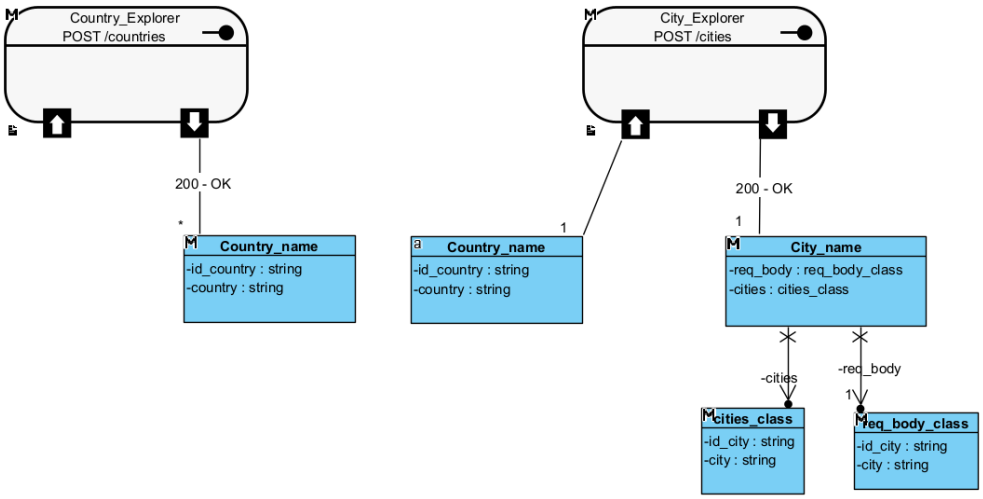
Εικόνα 41: Export Postman collection

2.2 Visual Paradigm

Ο λόγος που επιλέξαμε το συγκεκριμένο εργαλείο για την παρουσίαση του γράφου μας είναι επειδή περιέχει ειδικές κλάσεις για την αναπαράσταση των REST services ενώ περιέχει και χρήσιμα εργαλεία όπως το Reverse OpenAPI/Swagger μέσω του οποίου μπορούμε να κάνουμε import το επεξεργασμένο OpenAPI αρχείο μας.

2.2.1 Class Diagram in VP

Τα διαγράμματα κλάσεων [41] περιγράφουν τη δομή ενός συστήματος δείχνοντας τις κλάσεις του συστήματος, τις ιδιότητες τους (attributes), τις μεθόδους τους καθώς και τις σχέσεις μεταξύ των κλάσεων. Παρακάτω παρουσιάζεται ένα class diagram το οποίο θα εξηγήσουμε αναλυτικά σε επόμενο κεφάλαιο.



Εικόνα 42: Visual Paradigm Class Diagram

2.2.2 Σχέσεις κλάσεων

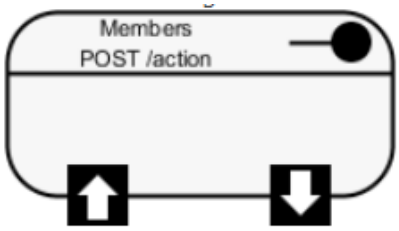
Οι σχέσεις κλάσεων που ορίζονται στο Visual Paradigm είναι οι παρακάτω:

- Inheritance (Generalization): Όταν η εξειδικευμένη κλάση αποτελεί μια έμμεση περίπτωση μιας πιο γενικής κλάσης.
- Association: Είναι η πιο απλή περίπτωση σχέσης μεταξύ δύο κλάσεων και σχεδιάζεται με μια απλή γραμμή μεταξύ των δύο κλάσεων.
- Aggregation: Όταν η μία κλάση αποτελεί μέρος της άλλης κλάσης.
- Dependency: Όταν δύο κλάσεις συνδέονται με σχέση εξάρτησης, δηλαδή αν αλλάξει κάτι στη μία κλάση μπορεί να προκαλέσει αλλαγές στην άλλη κλάση. Δεν ισχύει το αντίστροφο.
- Realization: Σχέση μεταξύ μιας κλάσης interface και μια κλάσης που υλοποιεί τη συγκεκριμένη κλάση τύπου interface.

- Composition: Όταν η μία κλάση εξαρτάται πλήρως από την άλλη και αν πεθάνει η μία δεν μπορεί να υπάρχει η άλλη κλάση μόνη της. Δεν ισχύει το αντίστροφο.

2.2.3 Σχεδιασμός REST API

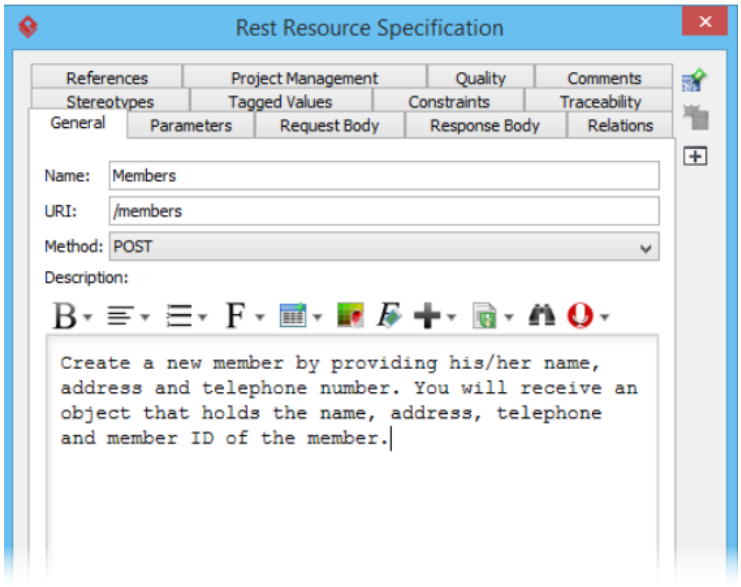
Μέσω του Visual Paradigm μπορούμε να σχεδιάσουμε ένα REST Resource [42], δηλαδή ένα endpoint του REST API, δημιουργώντας ένα Class Diagram και επιλέγοντας από το toolbar το component REST Resource.



Εικόνα 43: Visual Paradigm REST Resource

2.2.3.1 Specification

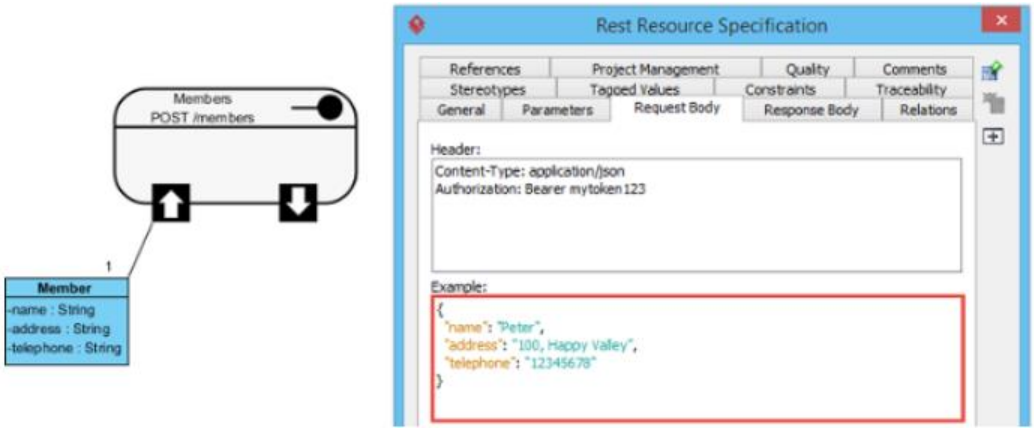
Μπορούμε μέσω του specification του REST Resource να ορίσουμε πεδία του endpoint όπως το όνομά του, το URI, τη μέθοδο και την περιγραφή.



Εικόνα 44: Visual Paradigm REST Resource specification

2.2.3.2 Request Body

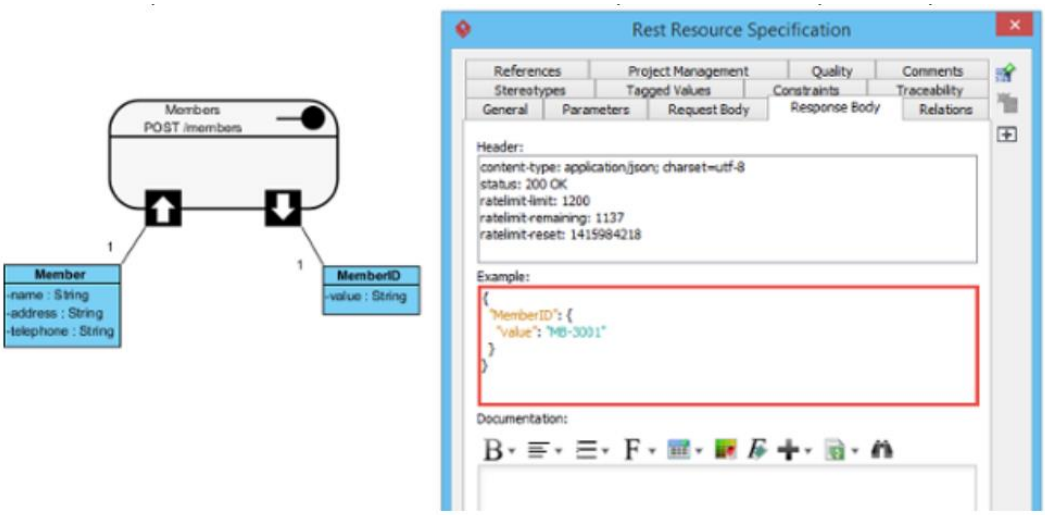
Μπορούμε να ορίσουμε το request body του REST Resource μαζί με τις ιδιότητες που το συνοδεύουν και μέσω του specification να ορίσουμε κάποιο παράδειγμα.



Εικόνα 45: Visual Paradigm REST Resource Request body

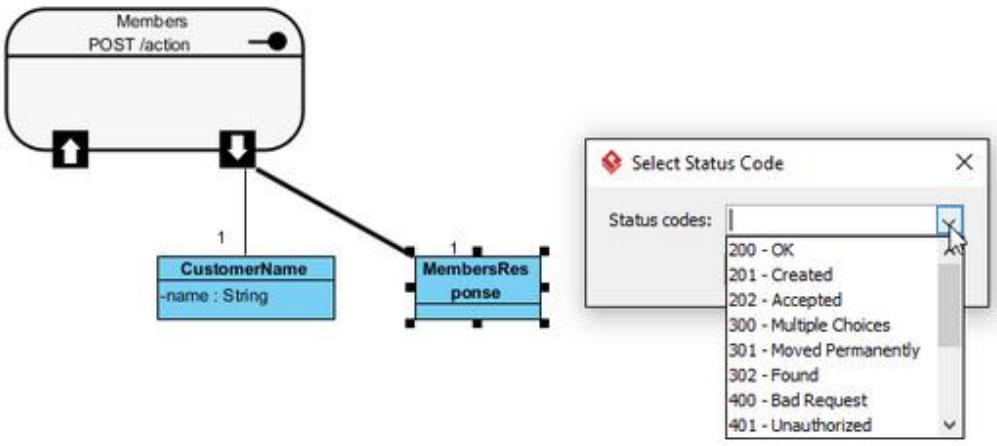
2.2.3.3 Response Body

Με αντίστοιχο τρόπο ορίζουμε και το response body, ενώ μέσω του specification μπορούμε να ορίσουμε και κάποιο παράδειγμα.



Εικόνα 46: Visual Paradigm REST Resource Response Body

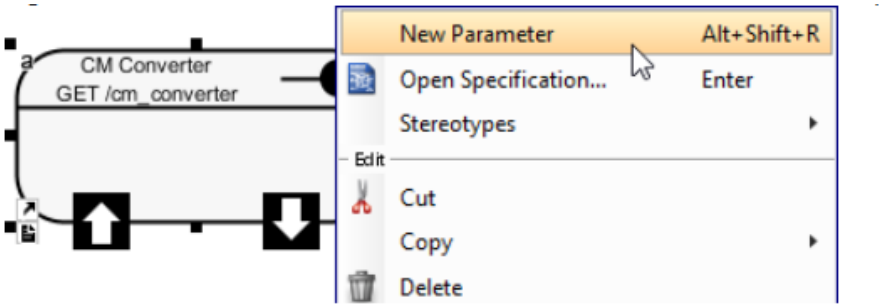
Επίσης, έχουμε τη δυνατότητα να ορίσουμε πολλαπλά responses για το ίδιο endpoint καταγράφοντας στο καθένα τον κωδικό που επιστρέφει.



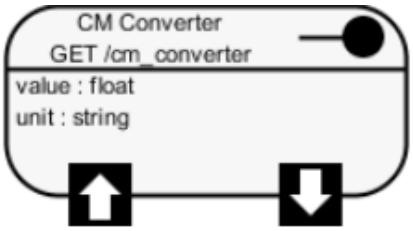
Εικόνα 47: Visual Paradigm REST Resource με πολλαπλά response bodies

2.2.3.4 Parameters

Τέλος μέσω της επιλογής New Parameter μπορούμε να ορίσουμε τις παραμέτρους που απαιτεί το endpoint μας.



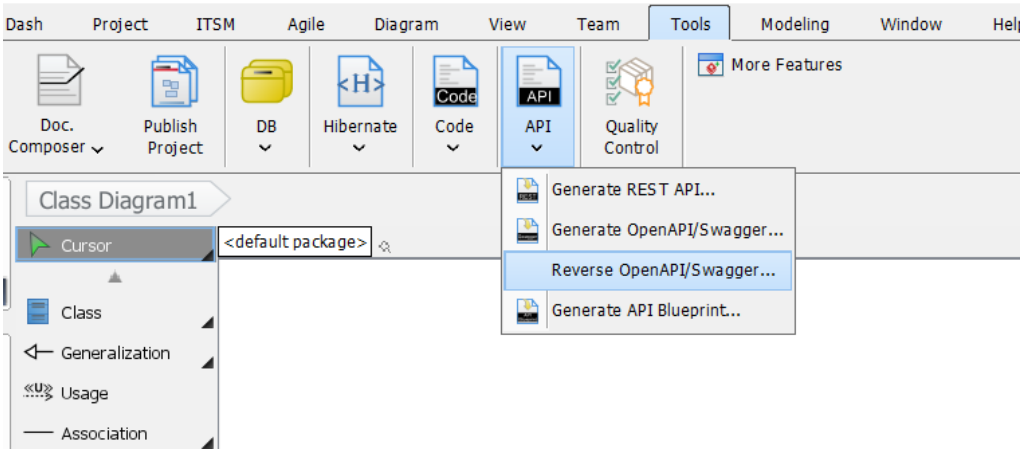
Εικόνα 48: Visual Paradigm REST Resource parameter 1



Εικόνα 49: Visual Paradigm REST Resource parameter 2

2.2.3.5 Reverse OpenAPI/Swagger

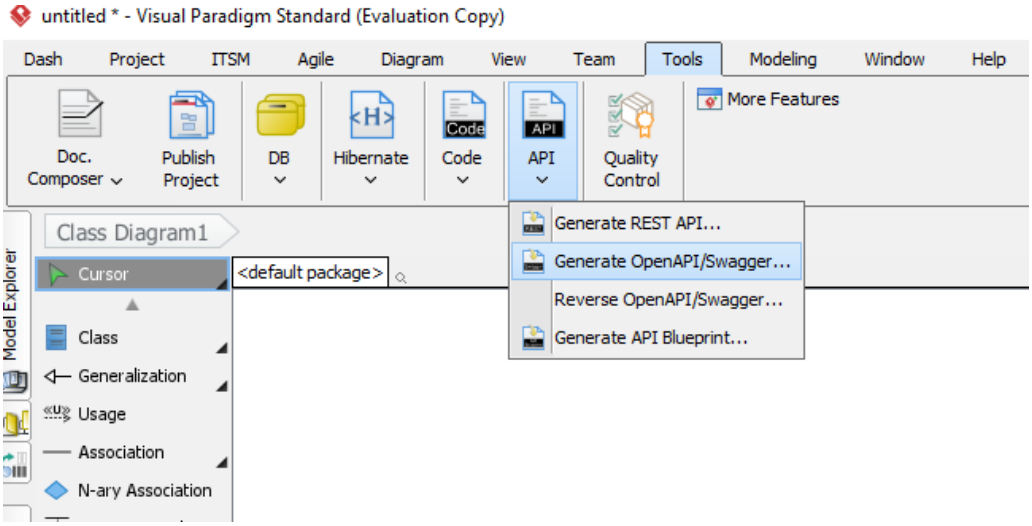
Μέσω του εργαλείου Reverse OpenAPI/Swagger μπορούμε να αυτοματοποιήσουμε τη διαδικασία σχεδίασης ενός REST API κάνοντας import ένα OpenAPI αρχείο.



Εικόνα 50: Visual Paradigm Reverse OpenAPI/Swagger Tool

2.2.3.6 Generate OpenAPI/Swagger

Αντίστοιχα μέσω της διαδικασίας Generate OpenAPI/Swagger μπορούμε να παραγάουμε ένα OpenAPI Documentation το οποίο θα περιλαμβάνει όλες τις πληροφορίες για το REST API μας. Θα περιλαμβάνει τις πληροφορίες που ορίσαμε στο specification, παραδείγματα, παραμέτρους καθώς και components με attributes που θα αντιστοιχούν στα request bodies και responses κάθε endpoint.



Εικόνα 51: Visual Paradigm Generate OpenAPI/Swagger Tool

2.3 Postman to OpenAPI

Όπως αναφέραμε προηγουμένως, τα APIs θέλουμε να συνοδεύονται από ένα documentation, αλλά αυτό δεν συμβαίνει πάντα. Ζητούμενο λοιπόν, είναι να διευκολύνουμε την παραγωγή documentation για υπάρχοντα APIs. Για αυτό το σκοπό αναπτύχθηκε ένα εργαλείο που αποτελεί προϊόν της διπλωματικής της συναδέλφου Ναυσικά Αμπατζή [38] και βοήθησε σημαντικά στην υλοποίηση ενός ολοκληρωμένου συστήματος.

Το εργαλείο αυτό παρέχεται μέσω ενός CLI (command line interface) στο οποίο αφού επιλέξουμε την λειτουργία Postman to OpenAPI δίνουμε σαν παράμετρο το αρχείο του postman collection που θέλουμε να μετατρέψουμε σε OpenAPI αρχείο. Με το εργαλείο αυτό μπορούμε να παραγάγουμε αυτόματα API Documentations για υπάρχοντα APIs τα οποία εξάγουμε μέσω του Postman. Στη συνέχεια το OpenAPI Documentation που παράγουμε μπορεί να γίνει import στο Visual Paradigm και να αναπαρασταθούν σωστά όλες οι κλάσεις που αντιστοιχούν στα endpoints του API.

Εικόνα 52: Postman To OpenAPI CLI

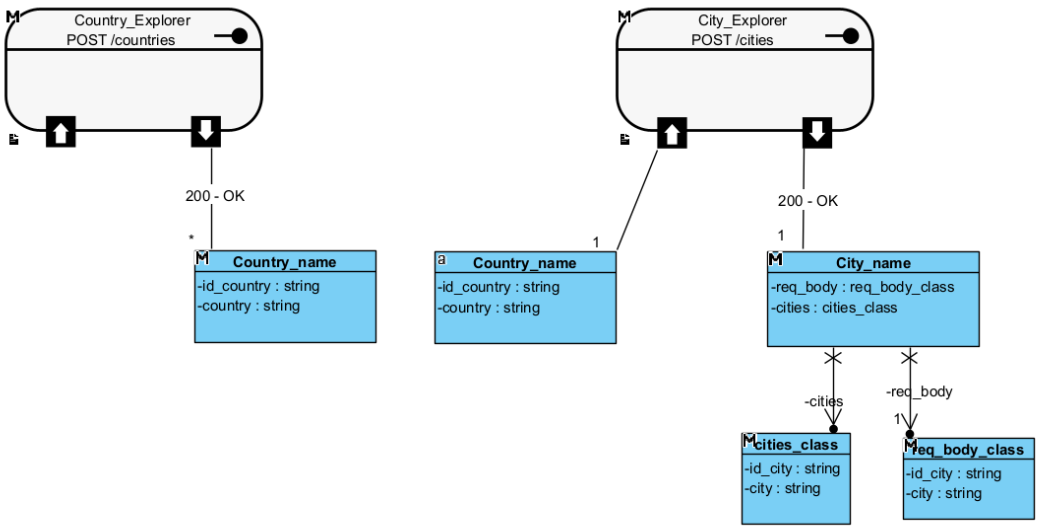
Το εργαλείο αυτό αποτελεί σημαντική βοήθεια για την έρευνά μας καθώς το χρησιμοποιούμε για να παραγάγουμε μια πρώτη μορφή OpenAPI Documentation για το API μας. Στη συνέχεια το επεκτείνουμε, εισάγοντας περισσότερες πληροφορίες στο τελικό documentation προκειμένου να είναι εμφανείς οι σχέσεις εξάρτησης μεταξύ των endpoints σε ένα REST API και να παρουσιάσουμε ένα πιο ολοκληρωμένο API Documentation μέσω του τελικού μας γράφου.

3 REST API με πλήρη «ιδανική» τεκμηρίωση

Στο κεφάλαιο αυτό παρουσιάζεται η ιδανική δομή για ένα REST API προκειμένου να είναι εμφανείς οι σχέσεις εξάρτησης μεταξύ των request και response bodies σε κάθε endpoint.

3.1 Δημιουργία REST API με ιδανική τεκμηρίωση

Πριν προγραμματίσουμε αυτό το API, θα σχεδιάσουμε την αρχιτεκτονική του μέσω του εργαλείου σχεδίασης και διαχείρισης: Visual Paradigm. [1]



Εικόνα 53: REST API με ιδανική τεκμηρίωση στο Visual Paradigm

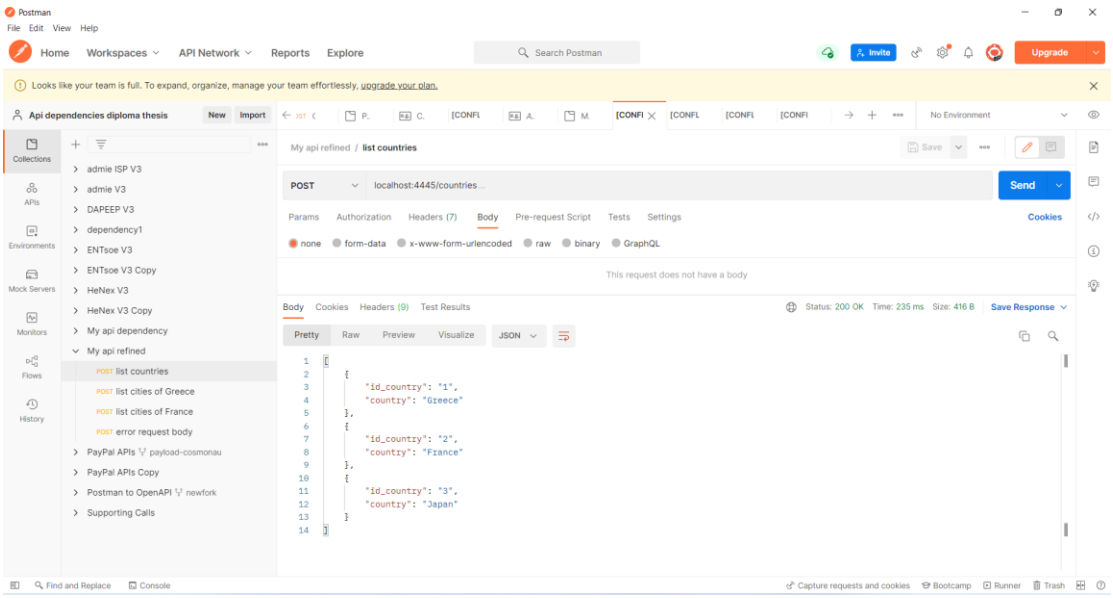
Στο παράδειγμα μας έχουμε δύο REST services. Το πρώτο REST service που ονομάζουμε Country_Explorer δεν απαιτεί κάποιο request body αλλά επιστρέφει στο response body μια λίστα από JSON objects όπου για κάθε object αναφέρεται το όνομα της χώρας και το αντίστοιχο id της.

Το δεύτερο REST service που ονομάζουμε City_Explorer περιέχει request body και response body. Στο request body δέχεται ως παράμετρο ένα JSON object στο όποιο περιλαμβάνεται το id της χώρας και το όνομά της. Στο response body επιστρέφει ένα root JSON object που αποτελείται από δύο nested objects. Το πρώτο nested object αποτελεί μια λίστα από ονόματα πόλεων με το αντίστοιχο id τους. Το δεύτερο nested object περιέχει απλά τις παραμέτρους που δόθηκαν στο request body. Η λίστα των πόλεων αλλάζει ανάλογα με τη χώρα που θέτουμε ως παράμετρο στο request body.

Είναι φανερό ότι υπάρχει μια σχέση εξάρτησης μεταξύ των δύο services καθώς το response body του Country_Explorer δίνει τα JSON objects που μπορούν να δοθούν ως παράμετροι στο request body του City_Explorer. Η μόνη διαφορά τους είναι ότι η πρώτη κλάση αποτελεί ένα array από objects σε αντίθεση με τη δεύτερη κλάση που αποτελεί ένα μοναδικό object. Το Visual Paradigm μας δείχνει αυτή την πληροφορία

ωστόσο ονομάζοντας τις δύο κλάσεις με το ίδιο όνομα αναγνωρίζει ότι αναφέρονται στην ίδια πληροφορία.

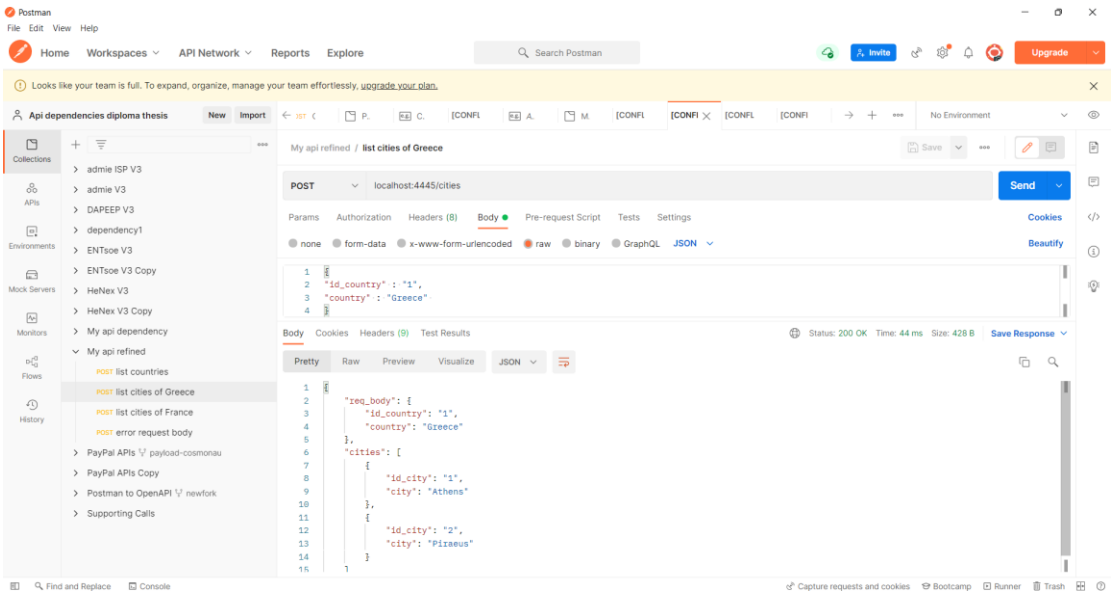
Στη συνέχεια, υλοποιούμε το REST API με NodeJS [43]. Χρησιμοποιούμε το λογισμικό δημιουργίας και χρήσης API: Postman, για να καλέσουμε τα endpoints και να λάβουμε τα responses. Αρχικά καλούμε το endpoint: /countries.



Εικόνα 54: Κλήση του endpoint countries μέσω του Postman

Βλέπουμε ότι επιστρέφει μια λίστα με 3 JSON objects όπου κάθε object έχει μοναδικό id και την χώρα στην οποία αντιστοιχεί.

Στη συνέχεια καλούμε το endpoint: /cities. Επιλέγουμε ένα object από το response της προηγούμενης κλήσης το οποίο θέτουμε ως παράμετρο στο request body.

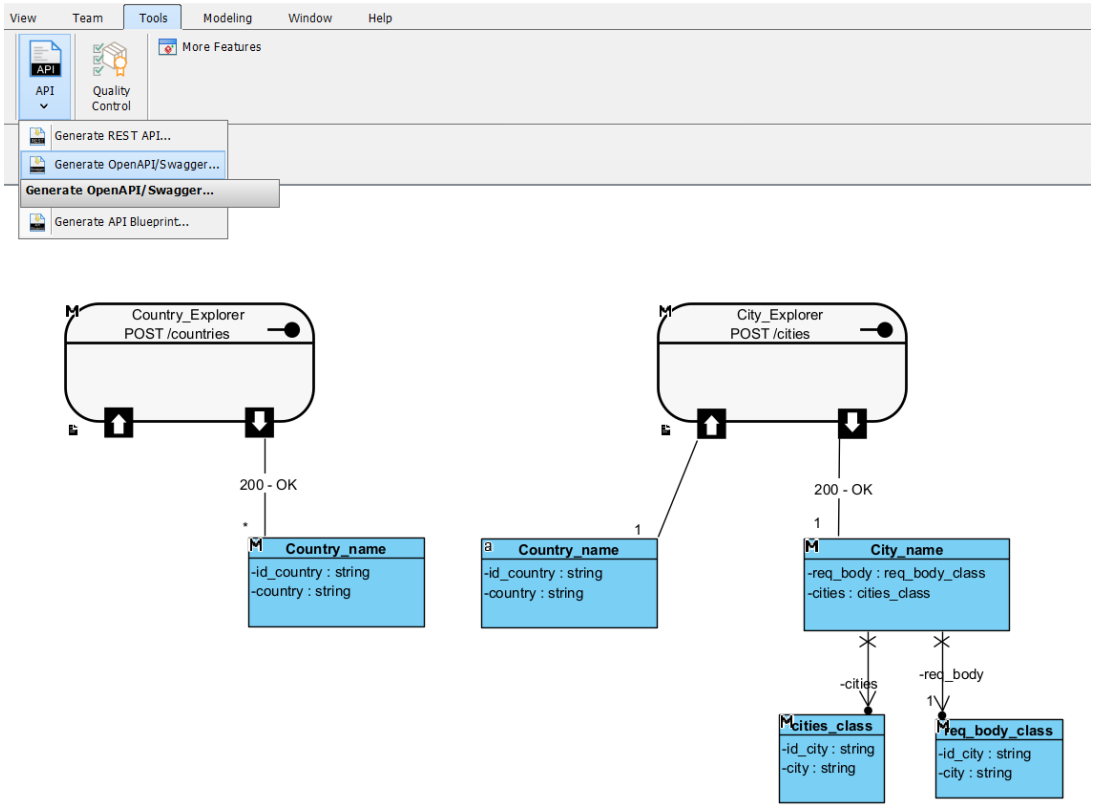


Εικόνα 55: Κλήση του endpoint cities μέσω του Postman

Βλέπουμε ότι επιστρέφει ένα object που περιλαμβάνει τις παραμέτρους του request body καθώς και ένα nested array με τις πόλεις που ανήκουν στην χώρα του request body.

Στο παράδειγμά μας βλέπουμε ότι υπάρχει ξεκάθαρη σχέση εξάρτησης μεταξύ των bodies των δύο endpoints. Η σχέση αυτή μάλιστα είναι τόσο σημαντική καθώς ένας χρήστης που επιθυμεί να χρησιμοποιήσει το endpoint /cities πρέπει πρώτα να γνωρίζει ποιες χώρες μπορεί να ορίσει στο request body του προκειμένου να λάβει την πληροφορία που επιθυμεί. Θα αναδείξουμε αυτή την εξάρτηση μέσω του εργαλείου Visual Paradigm.

Μέσω της διαδρομής Tools -> API -> Generate OpenAPI/Swagger παράγουμε αυτόματα το OpenAPI αρχείο για το REST API.



Εικόνα 56: Generate OpenAPI/Swagger ιδανικού REST API μέσω του Visual Paradigm

Παρατηρούμε τη δομή του αρχείου που δημιουργήθηκε:

```yaml
paths:
  /cities:
    post:
      description: Given a city and its corresponding ID return the country and its
        corresponding ID
      operationId: city_Explorer
      x-codegen-request-body-name: country_name
      requestBody:
        content:
          '*/*':
            schema:
              $ref: '#/components/schemas/Country_name'
            example: "{\r\n\"id_country\" : \"1\",\r\n\"country\" : \"Greece\" \r\n\
              }"
```

Εικόνα 57: OpenAPI ιδανικού REST API 1

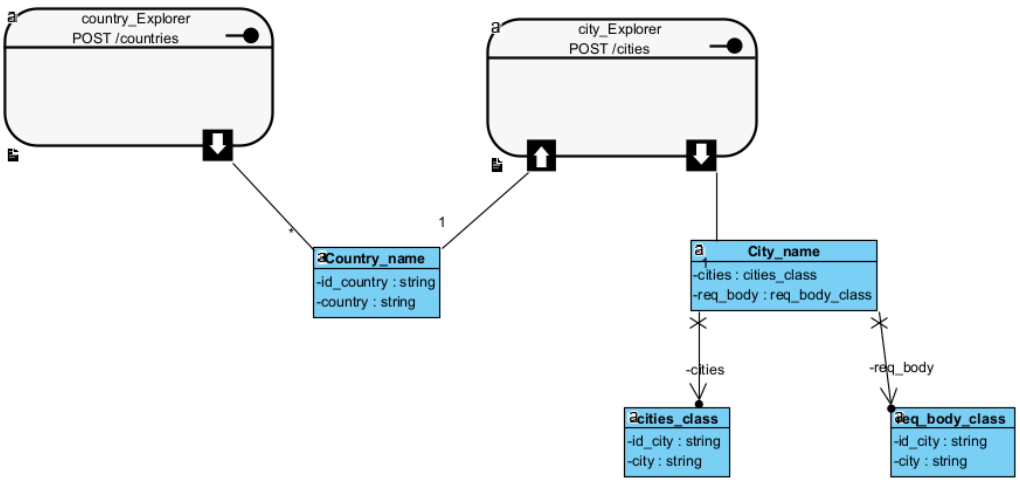
```yaml
  /countries:
    post:
      description: Get a list of countries with their corresponding IDs
      operationId: country_Explorer
      responses:
        200:
          description: OK
          content:
            '*/*':
              schema:
                type: array
                items:
                  $ref: '#/components/schemas/Country_name'
              example: "[\r\n    {\r\n        \"id_country\": \"1\",\r\n        \"\
                country\": \"Greece\"\r\n    },\r\n    {\r\n        \"id_country\"\
                : \"2\",\r\n        \"country\": \"France\"\r\n    },\r\n    {\r\n\
                \        \"id_country\": \"3\",\r\n        \"country\": \"Japan\"\r\
                \n    }\r\n]\r\n"
```

Εικόνα 58: OpenAPI ιδανικού REST API 2

Βλέπουμε ότι και τα δύο endpoints κάνουν αναφορά στο ίδιο component με το όνομα ‘Country_name’. Με αυτή την πληροφορία το Visual Paradigm αναγνωρίζει ότι πρόκειται για την ίδια κλάση.

Στη συνέχεια σε ένα καινούργιο diagram ακολουθούμε το μονοπάτι Tools->API->Reverse OpenAPI/Swagger και επιλέγουμε το OpenAPI αρχείο που δημιουργήσαμε προηγουμένως.

Το διάγραμμα μας είναι έτοιμο:

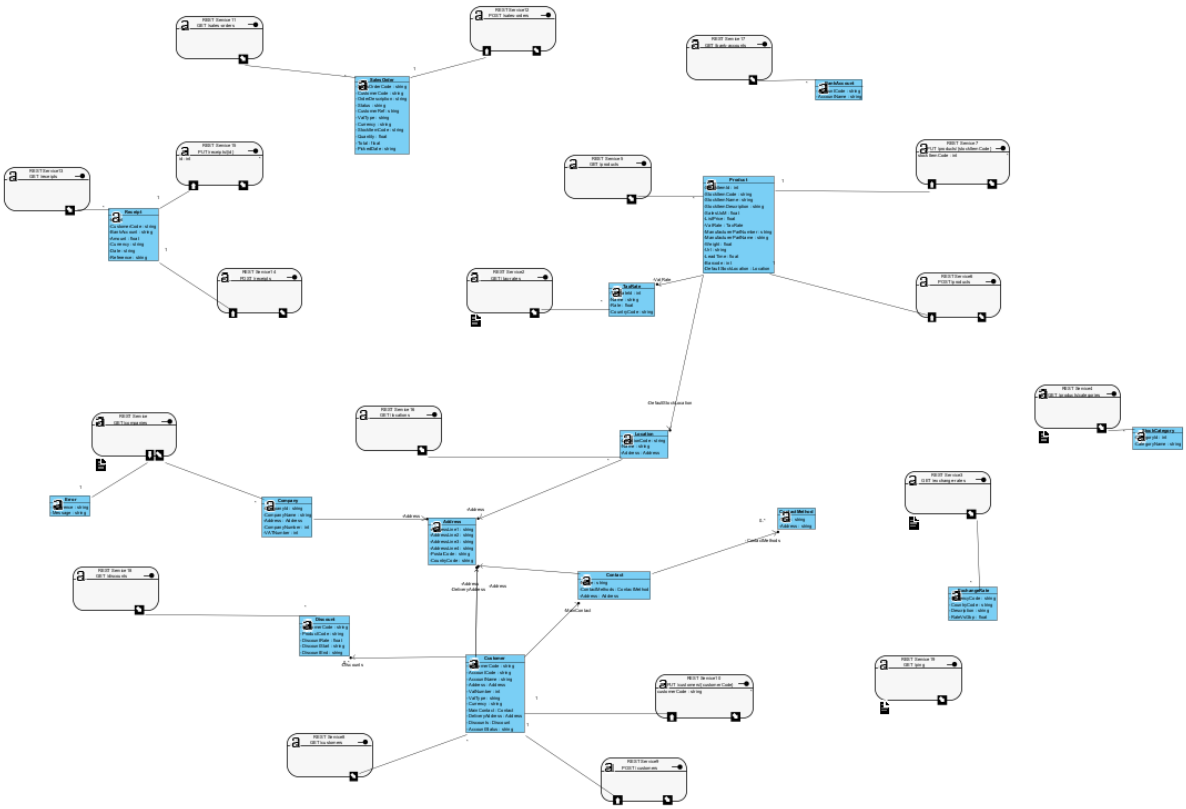


Εικόνα 59: Διάγραμμα ιδανικού REST API στο οποίο φαίνονται οι εξαρτήσεις μεταξύ των endpoints

Βλέπουμε ότι πλέον τα δύο REST services συνδέονται μεταξύ τους μέσω της ίδιας κλάσης. Έτσι μπορούμε εύκολα να διακρίνουμε ότι υπάρχει σχέση εξάρτησης μεταξύ του response body του ενός service με το request body του δεύτερου service.

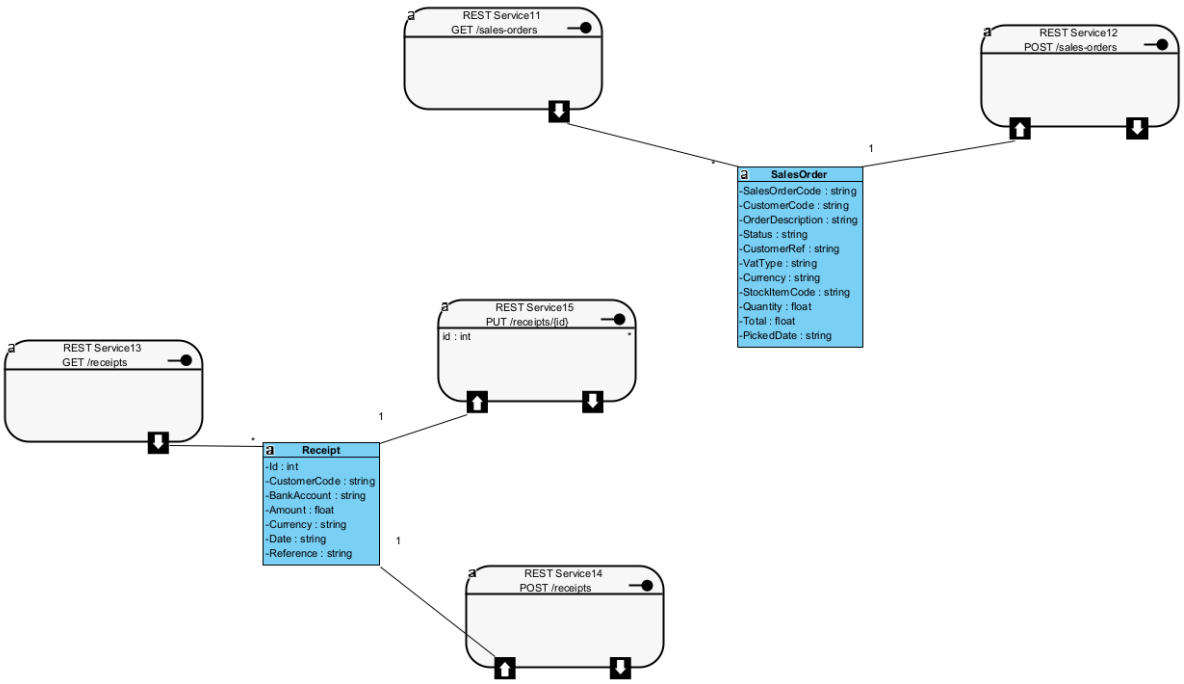
3.2 REST API για ecommerce

Στο παρακάτω διάγραμμα παρουσιάζεται η δομή ενός REST API για ένα ecommerce το οποίο παράχθηκε αυτόματα στο Visual Paradigm μέσω ενός OpenAPI αρχείου. Το OpenAPI αρχείο αυτό το βρήκαμε από ένα github repository ενός project που ασχολείται με παρουσίαση των OpenAPI αρχείων στο εργαλείο PlantUML [44]. Στο παράδειγμα αυτό παρατηρούμε ότι έχουν ληφθεί υπόψιν οι παραπάνω εξαρτήσεις και δεν χρειάζεται κάποια αλλαγή στο OpenAPI specification.



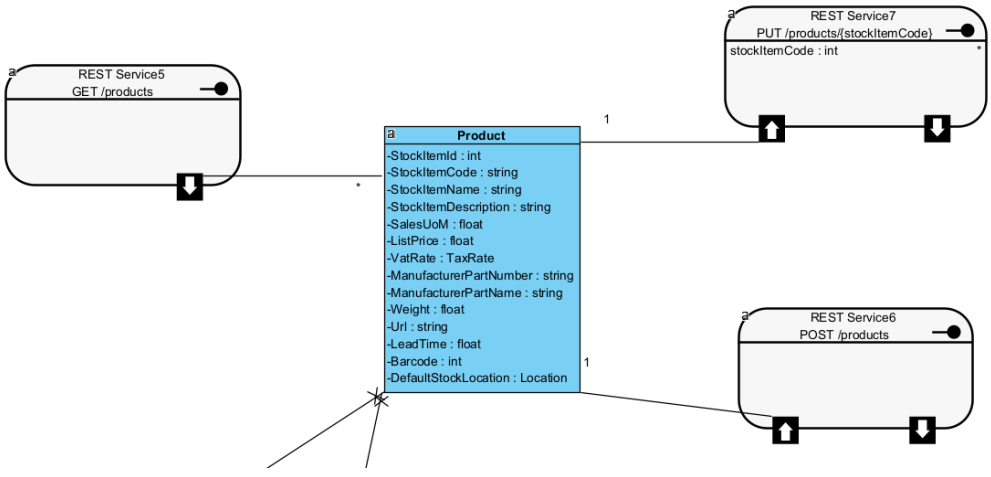
Εικόνα 60: Διάγραμμα REST API ecommerce

Βλέπουμε ότι αρκετά endpoints εμφανίζουν σχέσεις εξάρτησης μεταξύ τους. Αν κοιτάξουμε πιο προσεκτικά μάλιστα μπορούμε να πάρουμε κάποιες χρήσιμες πληροφορίες.



Εικόνα 61: REST API ecommerce εξαρτήσεις μεταξύ των endpoints 1

Μέσω του service GET /receipts μπορούμε να δούμε παραδείγματα αποδείξεων με όλα τα attributes και τις τιμές που περιέχουν και στη συνέχεια να δημιουργήσουμε εύκολα νέες αποδείξεις μέσω των endpoints POST /receipts και PUT /receipts. Ακόμη, βλέπουμε ότι μέσω του service GET /sales-order μπορούμε να δούμε παραγγελίες που δημιουργήθηκαν μέσω του service POST /sales-order.



Εικόνα 62: REST API ecommerce εξαρτήσεις μεταξύ των endpoints 2

Αντίστοιχη λογική είναι και με το endpoint GET /products που μας δίνει πληροφορίες για τις παραμέτρους που πρέπει να περιέχει ένα νέο προϊόν που θέλουμε

να εισαγάγουμε στο μαγαζί μας μέσω των endpoints PUT /products και POST /products.

Βλέπουμε λοιπόν, ότι στο συγκεκριμένο παράδειγμα τα services εμφανίζουν ισχυρές σχέσεις εξάρτησης μεταξύ τους και η πληροφορία αυτή είναι εμφανής στον τελικό γράφο. Μάλιστα, οι προγραμματιστές γνωρίζουν αυτές τις εξαρτήσεις και περιέχεται όλη η πληροφορία στο OpenAPI documentation που εισαγάγαμε στο Visual Paradigm. Το παράδειγμα αυτό έχει ιδιαίτερη αξία καθώς βλέπουμε ότι μπορούν να είναι φανερές οι εξαρτήσεις μεταξύ των endpoints στο τελικό Documentation αν έχει γίνει η σωστή ανάλυση κατά την ανάπτυξη του API. Αξία όμως, έχει να δούμε και αν μπορούμε να ανακαλύψουμε αυτές τις εξαρτήσεις όταν δεν είναι γνωστές από τους προγραμματιστές και δεν είναι τόσο εμφανείς.

4 Μελέτη και ανάπτυξη του συστήματος

4.1 Ανάλυση και απαιτήσεις

Πολλές φορές αυτή η σχέση εξάρτησης που αναλύσαμε στο προηγούμενο κεφάλαιο δεν είναι εμφανής. Πολλοί προγραμματιστές δεν γνωρίζουν ότι δύο endpoints μπορεί να εμφανίζουν πληροφορία που περιέχεται στην ίδια κλάση ενώ πολλές φορές μπορεί η εξάρτηση αυτή να υπάρχει μεταξύ κάποιων attributes και όχι σε ολόκληρη την κλάση. Θα αναλύσουμε αυτές τις περιπτώσεις όπου δεν είναι γνωστή αυτή η εξάρτηση και θα τις αντιμετωπίσουμε αναλόγως. Για το σύστημα που αναπτύξαμε έχουμε χρησιμοποιήσει python3 [45].

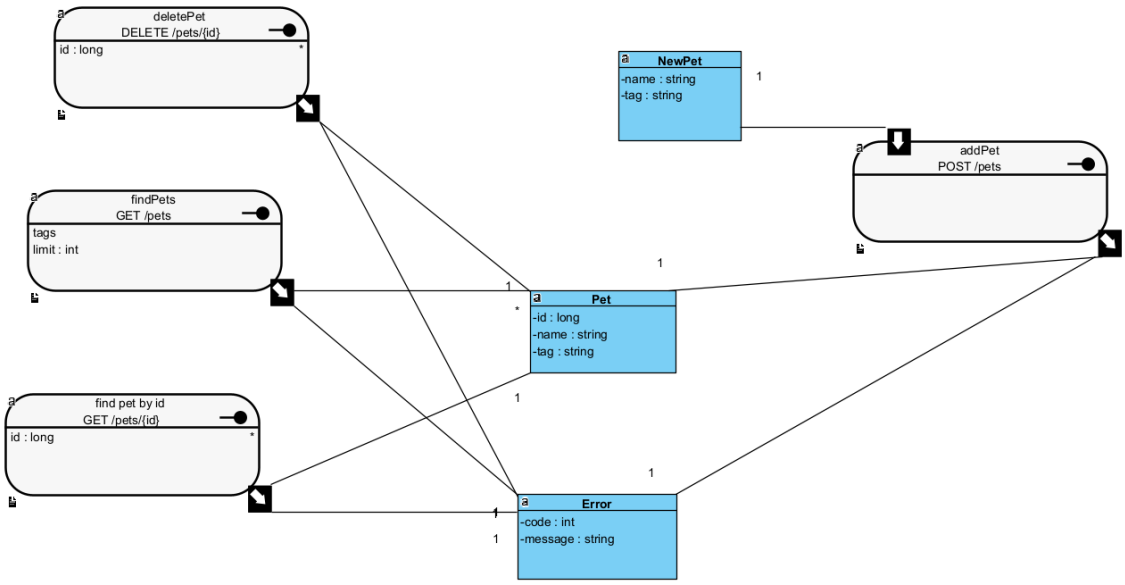
4.1.1 Τύποι παραμέτρων

Οι εξαρτήσεις ανάμεσα σε δύο endpoints μπορεί να είναι σε διάφορα σημεία. Μπορεί να είναι path parameter, query parameter ή και request body parameter. Εμείς στην έρευνά μας θα ασχοληθούμε με τις εξαρτήσεις που αφορούν το request body ενός API καθώς είναι αυτές που συνήθως περιέχουν την περισσότερη πληροφορία και έχουν τη μεγαλύτερη αξία. Τα path parameters δεν περιλαμβάνουν το όνομα του attribute συνεπώς δεν σχετίζονται άμεσα με την έρευνά μας. Από την άλλη, τα query parameters περιλαμβάνουν το attribute μαζί με την αντίστοιχη τιμή του όπως τα attributes στο request body του endpoint.

4.1.2 APIs υπό εξέταση

4.1.2.1 Petstore API

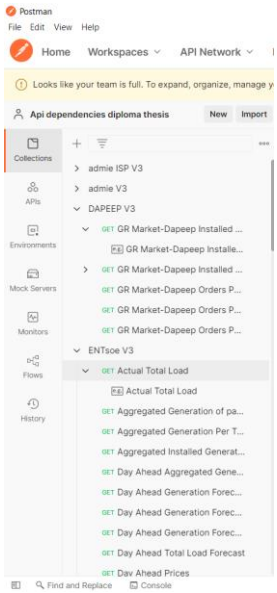
Εξετάζουμε κλιμακωτά κάποια APIs. Αρχικά, δοκιμάζουμε το σύστημά μας σε ένα έτοιμο OpenAPI αρχείο που αποτελεί μια έκδοση ενός petstore API που παρέχεται από την επίσημη σελίδα του Swagger [46] και αποτελεί πρότυπο αρχείο για OpenAPI. Το petstore αυτό αποτελείται από τέσσερα services τα addPet, deletePet, findPet και findPetbyId. Όλα τα services περιλαμβάνουν τα ίδια responses ενώ ιδιαίτερη αξία εμφανίζει το request body του service addPet. Εμείς έχουμε έτοιμο το OpenAPI documentation στη διάθεσή μας ωστόσο θα το βελτιώσουμε μέσω του συστήματός μας για να φανούν οι εξαρτήσεις.



Εικόνα 63: Petstore API

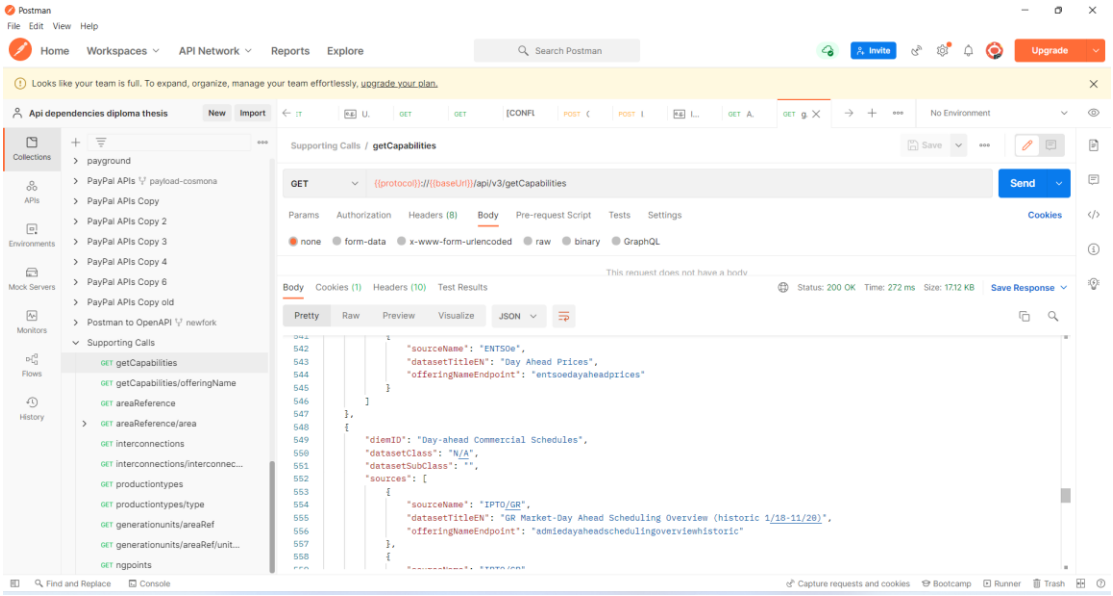
4.1.2.2 Energy API

Στη συνέχεια εφαρμόζουμε το σύστημά μας σε ένα δικό μας API όπου γνωρίζουμε τις εξαρτήσεις και αφορά πληροφορίες σχετικές με εταιρείες που χειρίζονται δεδομένα ενέργειας. Το API αυτό είναι χωρισμένο σε postman collections όπου κάθε ένα από αυτά αφορά μια συγκεκριμένη λειτουργία ή ομάδα λειτουργιών ενώ έχουμε και ένα ξεχωριστό collection με βοηθητικές κλήσεις οι οποίες συνήθως μας δίνουν τις παραμέτρους που απαιτούνται στα request bodies των services των εταιρειών.



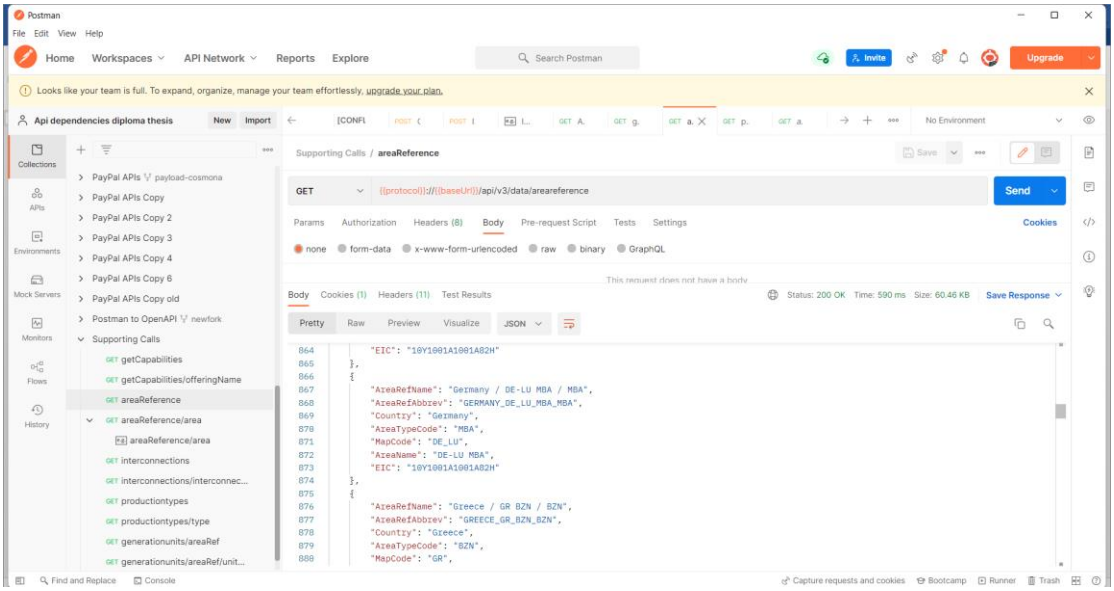
Εικόνα 64: Postman collections για το Energy API

Δεν περιέχει κάποιο Documentation ωστόσο μέσω του service /getCapabilities μπορούμε να δούμε πληροφορίες για κάθε endpoint.



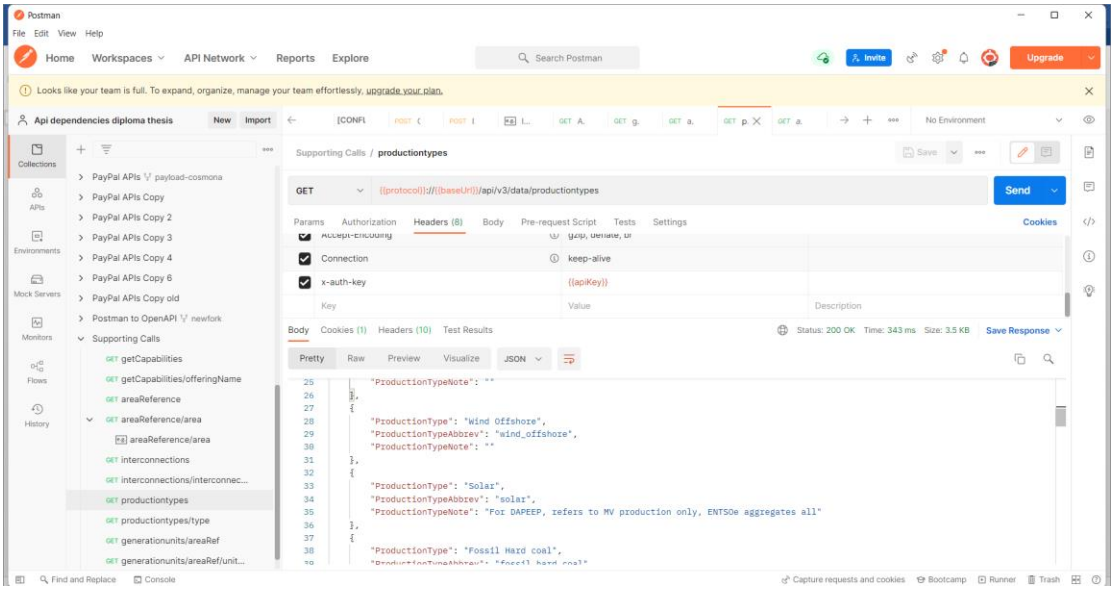
Εικόνα 65: Κλήση του endpoint getCapabilities του Energy API

Μέσω του endpoint areaReference λαμβάνουμε μια λίστα από ονόματα χωρών και τους κωδικούς που αφορούν την περιοχή αυτή.



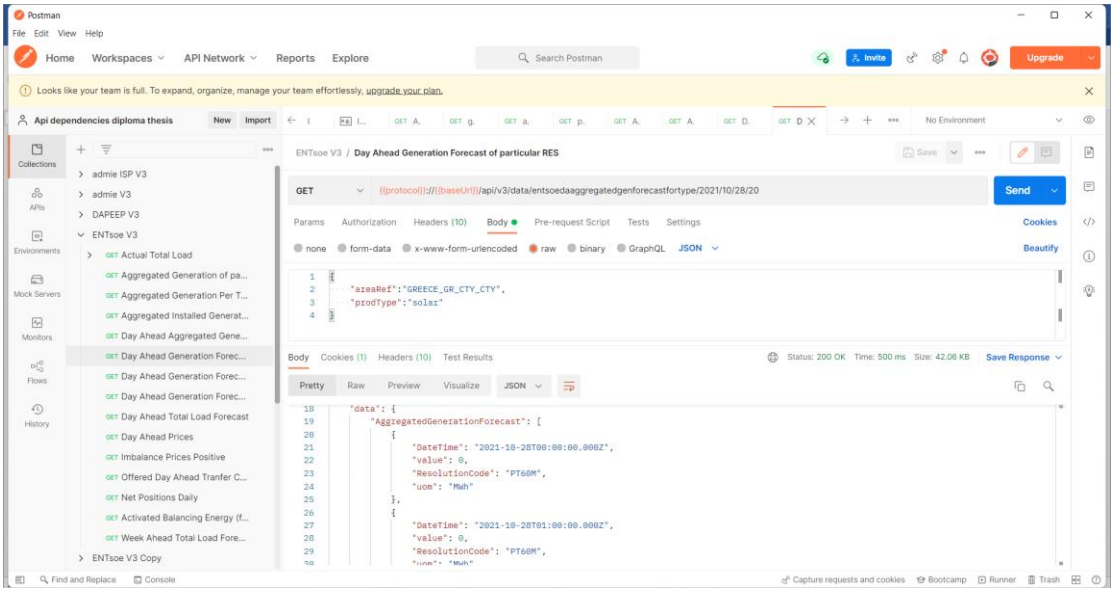
Εικόνα 66: Κλήση του endpoint areaReference του EnergyAPI

Αντίστοιχα, μέσω του endpoint productiontypes λαμβάνουμε μια λίστα από τύπους ενέργειας.



Εικόνα 67: Κλήση του endpoint productiontypes του Energy API

Στη συνέχεια, μπορούμε να επιλέξουμε κάποιες από τις παραπάνω τιμές και να τις θέσουμε ως παραμέτρους σε συγκεκριμένα endpoints των services των εταιρειών. Για παράδειγμα, καλώντας το endpoint ‘Day Ahead Generation Forecast’ του ευρωπαϊκού φορέα ENTSOe που ασχολείται με τη διαφάνεια στις αγορές ενέργειας, μπορούμε να ορίσουμε στο request body την χώρα και τον τύπο ενέργειας που επιθυμούμε και να λάβουμε τις αντίστοιχες πληροφορίες.

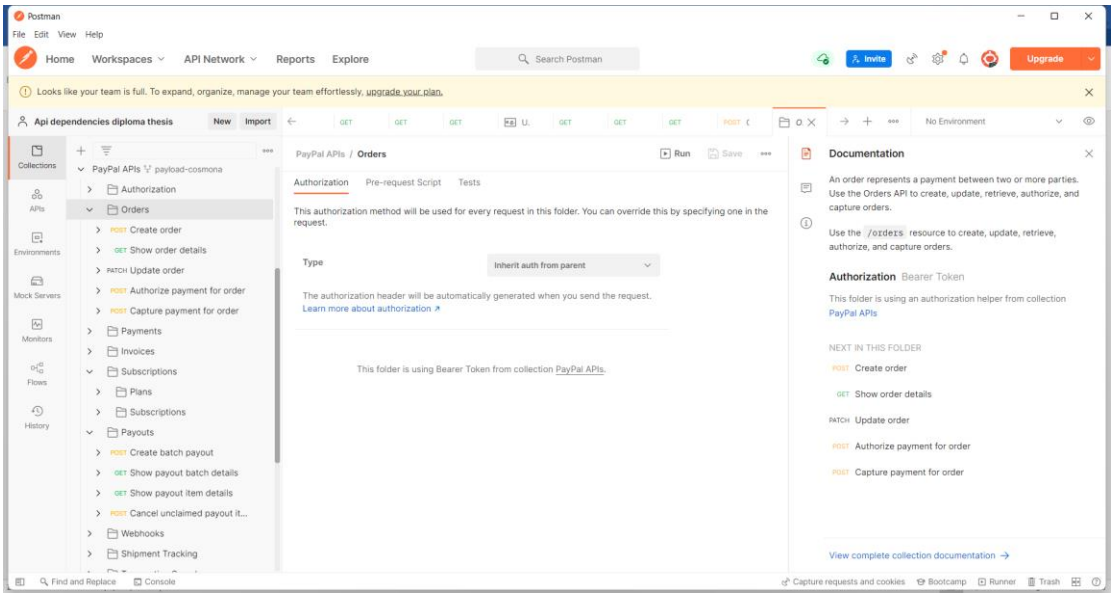


Εικόνα 68: Κλήση του endpoint Day Ahead Generation Forecast της εταιρείας ENTSOe του Energy API

Βλέπουμε δηλαδή, ότι η κλήση μερικών services των εταιρειών απαιτούν πρώτα την κλήση κάποιων βοηθητικών endpoints προκειμένου να λάβουμε τις πληροφορίες που χρειαζόμαστε για τη σωστή κλήση τους. Υπάρχει μια ισχυρή σχέση εξάρτησης μεταξύ αυτών των endpoints η οποία είναι σημαντικό να μπορεί να καταγραφεί και στο Documentation.

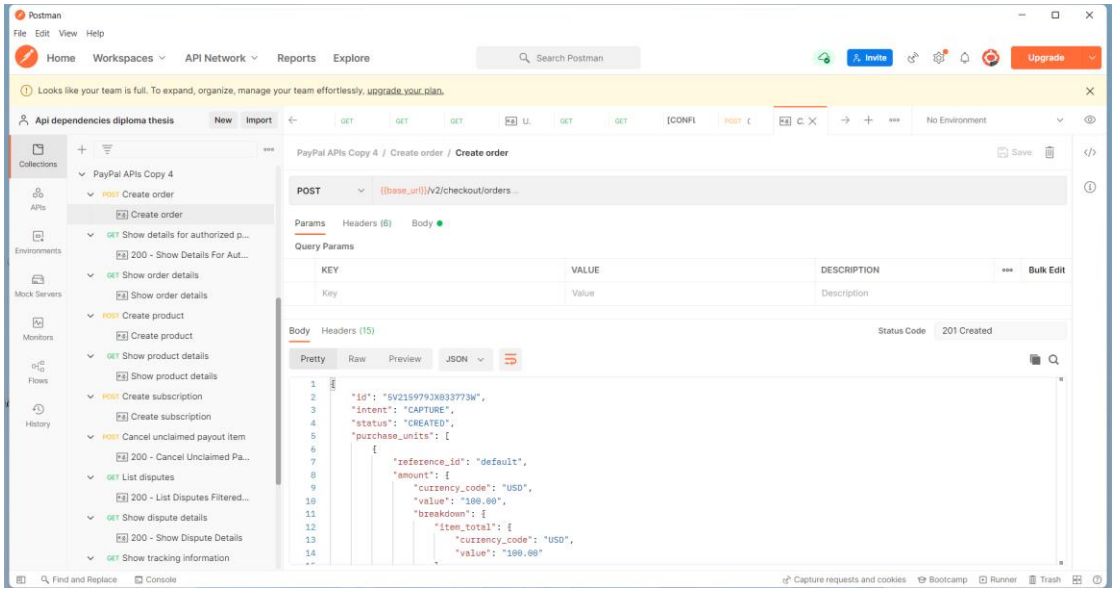
4.1.2.3 Paypal API

Τέλος, εφαρμόζουμε το σύστημά μας στο δημόσιο API της paypal [47] το οποίο δεν συνοδεύεται από κάποιο Documentation. Το REST API της paypal αποτελείται από πολλαπλά endpoints και είναι χωρισμένα σε φακέλους ανάλογα με την κατηγορία την οποία αφορούν. Δεν γνωρίζουμε με ποιον τρόπο συνδέονται τα endpoints μεταξύ τους και αν υπάρχουν σχέσεις εξάρτησης.



Εικόνα 69: Postman collection για το Paypal API

Εμείς δημιουργούμε ένα νέο collection στο οποίο επιλέγουμε κάποια από αυτά τα endpoints για να αναδείξουμε πιθανές εξαρτήσεις που υπάρχουν μεταξύ των bodies τους.



Εικόνα 70: Postman collection για το Paypal API 2

4.1.3 Μελέτη περιπτώσεων

4.1.3.1 OpenAPI με τεκμηρίωση που δεν περιλαμβάνει παραδείγματα

Επίσης, εξετάζουμε τα APIs ανά περιπτώσεις. Αρχικά δοκιμάζουμε το σύστημά μας όταν έχουμε ένα απλό OpenAPI αρχείο το οποίο δεν περιέχει παραδείγματα. Στην περίπτωση αυτή η μόνη πληροφορία που μπορούμε να πάρουμε είναι από τα ονόματα των attributes καθώς δεν κατέχουμε πληροφορίες σχετικά με τις τιμές που μπορούν να πάρουν.

```yaml
components:
  schemas:
    Pet:
      type: object
      required:
        - id
      properties:
        id:
          type: integer
          format: int64
        name:
          type: string
        tag:
          type: string

    NewPet:
      type: object
      required:
        - name
      properties:
        name:
          type: string
        tag:
          type: string

    Error:
      type: object
      required:
        - code
        - message
      properties:
        code:
          type: integer
          format: int32
        message:
          type: string
```

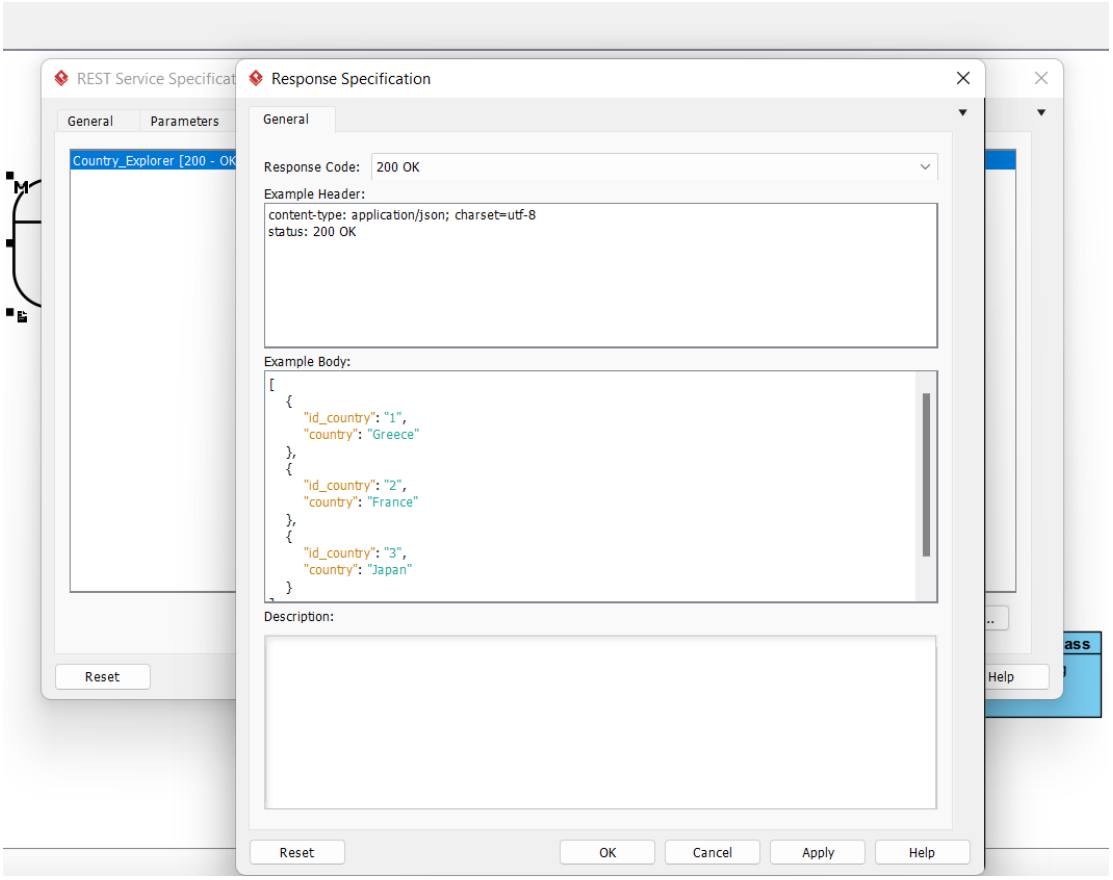
Εικόνα 71: OpenAPI attributes

Βλέπουμε ότι στο τέλος του αρχείου εμφανίζονται τα components με τα αντίστοιχα attributes τα οποία εμφανίζει το Visual Paradigm. Μπορούμε να διαπιστώσουμε ότι κάποια attributes με το ίδιο όνομα εμφανίζονται σε διαφορετικά components. Αυτό πιθανώς σημαίνει ότι αναφέρονται στην ίδια πληροφορία. Συνεπώς θα βρούμε αυτά τα κοινά attributes και θα μεταβάλλουμε το αρχείο για να φανούν αυτές οι εξαρτήσεις.

4.1.3.2 OpenAPI με τεκμηρίωση που περιλαμβάνει παραδείγματα

Στη συνέχεια δοκιμάζουμε το σύστημά μας όταν το OpenAPI αρχείο περιέχει παραδείγματα. Στην περίπτωση αυτή εκτός από τα ονόματα των attributes κατέχουμε πληροφορίες και για τις τιμές που μπορούν να πάρουν. Στην περίπτωση αυτή μπορούμε να είμαστε σίγουροι ότι δύο endpoints αναφέρονται σε κοινή πληροφορία όταν τα attributes τους εμφανίζουν όμοιες τιμές.

Το Visual Paradigm παρέχει τη δυνατότητα να ορίσουμε examples.



Εικόνα 72: Example response στο Visual Paradigm

Τα examples αυτά αναγράφονται στο OpenAPI αρχείο με την εξής μορφή:

```
example: "[\r\n    {\r\n        \"id_country\": \"1\",\r\n        \"\
  country\": \"Greece\"\r\n    },\r\n    {\r\n        \"id_country\"\
  : \"2\",\r\n        \"country\": \"France\"\r\n    },\r\n    {\r\n\
  \        \"id_country\": \"3\",\r\n        \"country\": \"Japan\"\r\
  \n    }\r\n]\r\n"
```

Εικόνα 73: OpenAPI examples

Βλέπουμε λοιπόν ότι μπορούμε να εξαγάγουμε αποτελέσματα για εξαρτήσεις μέσω των παραδειγμάτων καθώς μπορούμε να εντοπίσουμε περιπτώσεις όπου attributes διαφορετικών endpoints λαμβάνουν τις ίδιες τιμές.

4.1.3.2.1 OpenAPI examples εναλλακτική μορφή

Το OpenAPI Specification παρέχει τη δυνατότητα παρουσίασης των examples και με την παρακάτω μορφή.

```yaml
example:
  intent: CAPTURE
  purchase_units:
    - items:
        - name: T-Shirt
          description: Green XL
          quantity: '1'
          unit_amount:
            currency_code: USD
            value: '100.00'
      amount:
        currency_code: USD
        value: '100.00'
        breakdown:
          item_total:
            currency_code: USD
            value: '100.00'
  application_context:
    return_url: https://example.com/return
    cancel_url: https://example.com/cancel
```

Εικόνα 74: OpenAPI examples 2

Να σημειωθεί όμως ότι σε αυτή την περίπτωση το OpenAPI δεν μπορεί να καταγράψει αναλυτικά τα examples όταν το response κάποιου endpoint είναι array και συνεπώς περιορίζει τη δυνατότητα ανακάλυψης των εξαρτήσεων.

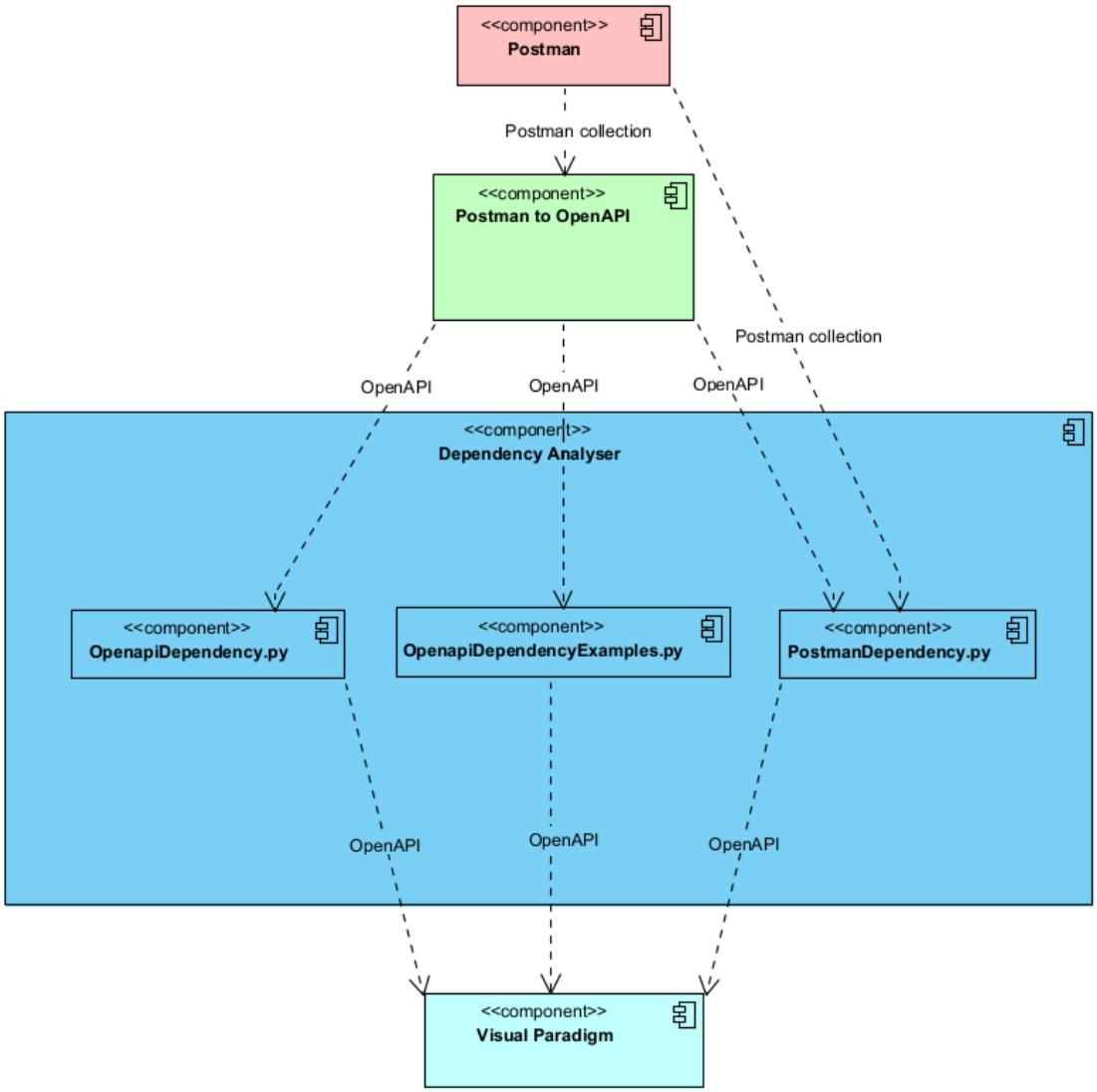
4.1.3.3 REST API χωρίς καθόλου τεκμηρίωση

Τέλος, αντιμετωπίζουμε την περίπτωση όπου δεν έχουμε εξαρχής κάποιο OpenAPI αρχείο και έχουμε μόνο το API διαθέσιμο. Στην περίπτωση αυτή δεν έχουμε καμία μορφή documentation. Θα επιχειρήσουμε να δημιουργήσουμε ένα αρχικό documentation με το εργαλείο της Ναυσικάς [38] και στη συνέχεια αξιοποιώντας πληροφορία από το Postman θα το εμπλουτίσουμε για να καταλήξουμε στο επιθυμητό αποτέλεσμα.

4.2 Ανάπτυξη συστήματος

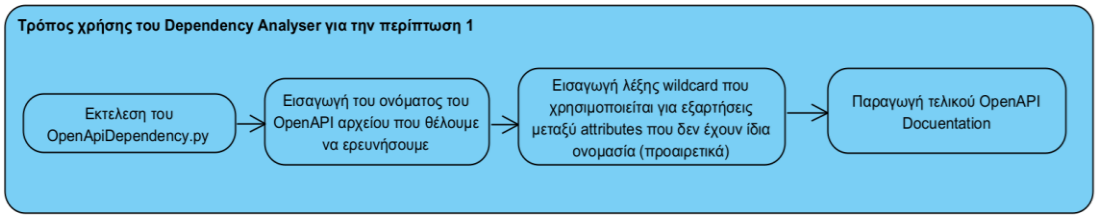
4.2.1 Δομή του λογισμικού Dependency Analyser

Το σύστημα που αναπτύξαμε ονομάζεται Dependency Analyser. Παρακάτω παρουσιάζεται ένα διάγραμμα με τα components που περιέχει το σύστημά μας καθώς και τον τρόπο με τον οποίο συνδέονται με τα υπόλοιπα εργαλεία.



Εικόνα 75: Component diagram με το λογισμικό Dependency Analyser και τα υπόλοιπα εργαλεία

Αρχικά παρουσιάζονται αναλυτικά διαγράμματα για τον τρόπο χρήσης του Dependency Analyser. Όπως αναφέραμε προηγουμένως αντιμετωπίζουμε 3 διαφορετικές περιπτώσεις, για κάθε μία από τις οποίες χρησιμοποιούμε ξεχωριστό component και υπάρχει ξεχωριστό διάγραμμα χρήσης. Τα βήματα που ακολουθούμε για κάθε περίπτωση παρουσιάζονται παρακάτω:



Εικόνα 76: Διάγραμμα ροής χρήσης του Dependency Analyser για την περίπτωση 1

Για την περίπτωση 1:

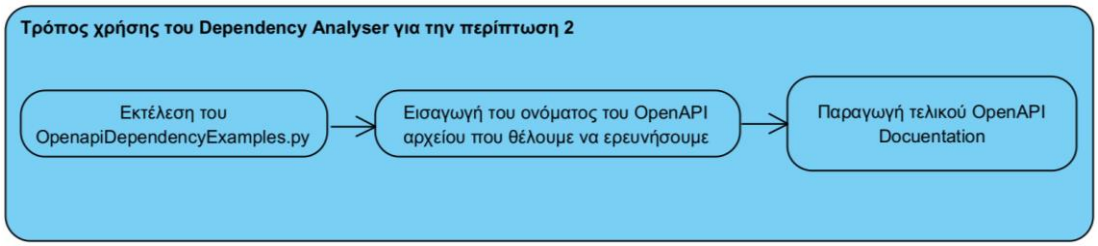
1. Εκτελλούμε το OpenapiDependency.py

2. Εισάγουμε το όνομα του OpenApi αρχείου

3. Εισάγουμε wildcard προαιρετικά (θα εξηγήσουμε τη χρήση του)

4. Παίρνουμε το τελικό OpenApi Documentation με πληροφορίες για τις εξαρτήσεις μεταξύ των endpoints

Όπως αναφέραμε προηγουμένως, η μόνη πληροφορία που έχουμε στη διάθεση μας σε αυτή την περίπτωση είναι τα ονόματα των attributes και όχι οι τιμές που μπορεί να πάρουν. Αυτό όμως, μας περιορίζει στην περίπτωση που attributes με κοινή πληροφορία έχουν διαφορετική ονομασία. Ένας τρόπος να ξεπεραστεί αυτό είναι ορίζοντας ένα string ως wildcard. Το wildcard αυτό χρησιμοποιείται στην περίπτωση που δύο attributes έχουν παρεμφερή ονόματα και γνωρίζουμε ότι αναφέρονται στην ίδια πληροφορία.



Εικόνα 77: Διάγραμμα ροής χρήσης του Dependency Analyser για την περίπτωση 2

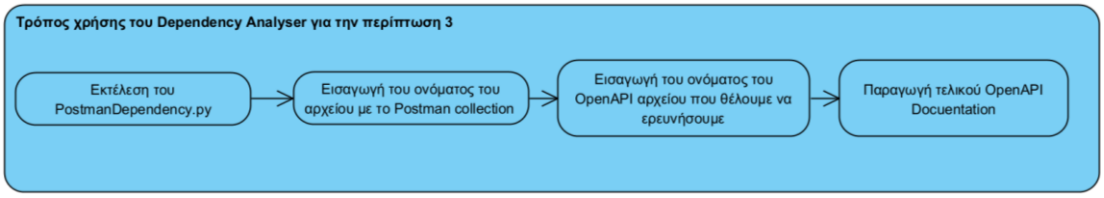
Για την περίπτωση 2:

1. Εκτελλούμε το OpenapiDependency.py

2. Εισάγουμε το όνομα του OpenApi αρχείου

3. Παίρνουμε το τελικό OpenApi Documentation με πληροφορίες για τις εξαρτήσεις μεταξύ των endpoints

Σε αυτή την περίπτωση έχουμε διαθέσιμες τις τιμές που λαμβάνουν τα attributes οπότε μπορούμε να είμαστε σίγουροι για τις εξαρτήσεις.



Εικόνα 78: Διάγραμμα ροής χρήσης του Dependency Analyser για την περίπτωση 3

Για την περίπτωση 3:

1. Εκτελλούμε το OpenapiDependency.py

2. Εισάγουμε το όνομα του αρχείου με το Postman Collection

3. Εισάγουμε το όνομα του OpenApi αρχείου

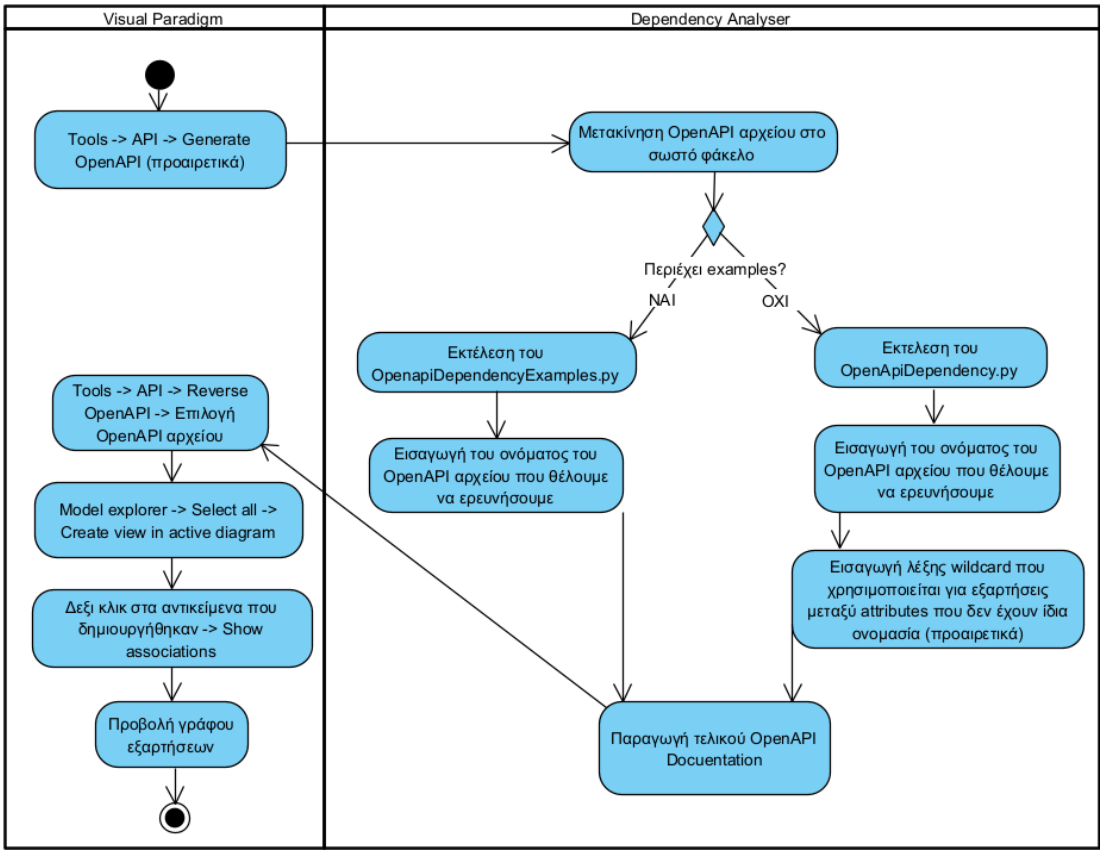
4. Παίρνουμε το τελικό OpenApi Documentation με πληροφορίες για τις εξαρτήσεις μεταξύ των endpoints

Στη συνέχεια αναλύουμε λεπτομερώς κάθε περίπτωση ξεχωριστά. Εξηγούμε πώς συνδέεται το σύστημά μας με τα υπόλοιπα εργαλεία και παρουσιάζουμε τα βήματα που ακολουθούμε σε κάθε εργαλείο, μέσω αναλυτικών διαγραμμάτων χρήσης, για να καταλήξουμε στην τελική παρουσίαση του Documentation.

4.2.1 Περίπτωση 1: OpenAPI με τεκμηρίωση που δεν περιλαμβάνει παραδείγματα

Το Visual Paradigm βασίζεται στην πληροφορία που περιέχεται σε ένα OpenAPI αρχείο για να σχεδιάσει το αντίστοιχο διάγραμμα. Το λογισμικό Dependency Analyser επεξεργάζεται κατάλληλα τέτοια αρχεία προκειμένου το Visual Paradigm να μπορέσει να αναπαραστήσει αυτές τις συσχετίσεις. Θα προσπαθήσουμε να βρούμε εξαρτήσεις με τις πληροφορίες που έχουμε στη διάθεσή μας, δηλαδή λαμβάνοντας υπόψιν τα ονόματα των attributes ανεξάρτητα από τις τιμές που μπορούν να πάρουν.

Τα βήματα που ακολουθούμε φαίνονται στο παρακάτω activity diagram:



Εικόνα 79: Διάγραμμα ροής χρήσης για την περίπτωση 1

Ακολουθούμε τα βήματα για την περίπτωση που δεν περιέχονται examples:

1. Παράγουμε το OpenAPI αρχείο (yaml) μέσω του Visual Paradigm αν δεν το έχουμε ήδη

2. Εισάγουμε το OpenAPI αρχείο προς επεξεργασία στο κατάλληλο φάκελο

3. Εκτελλούμε το OpenapiDependency.py για την επεξεργασία του

4. Εισάγουμε το όνομα του OpenAPI αρχείου που θέλουμε να ερευνήσουμε

5. Εισάγουμε wildcard προαιρετικά

6. Αποθηκεύουμε το αρχείο που δημιουργήθηκε για να το εμφανίσουμε στο Visual Paradigm

7. Εισάγουμε το αρχείο στο Visual Paradigm μέσω του εργαλείου Reverse OpenAPI

8. Επιλέγουμε τα components μέσω του Model explorer

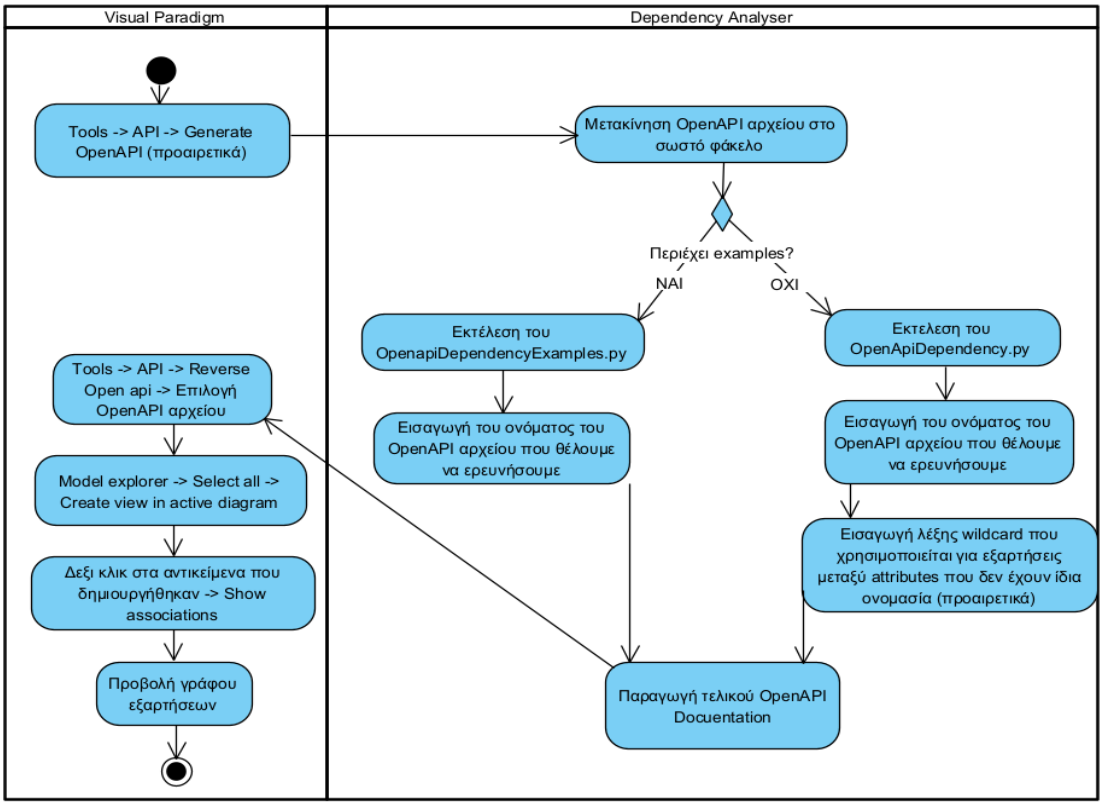
9. Στα components επιλέγουμε με δεξί κλικ show associations

10. Εμφανίζουμε τον γράφο

4.2.2 Περίπτωση 2: OpenAPI με τεκμηρίωση που περιλαμβάνει παραδείγματα

Στην περίπτωση αυτή έχουμε και πάλι το OpenAPI Documentation διαθέσιμο, ωστόσο γνωρίζουμε πληροφορίες για τις τιμές των attributes σε κάθε endpoint μέσω των παραδειγμάτων. Θα αξιοποιήσουμε αυτή την πληροφορία για να αναγνωρίσουμε αυτές τις εξαρτήσεις και να τις εμφανίσουμε στον τελικό μας γράφο. Σε αυτή την περίπτωση, ακόμα και για attributes που έχουν διαφορετικές ονομασίες, εμείς μπορούμε να εντοπίσουμε τις εξαρτήσεις μεταξύ των endpoints μέσω των τιμών που λαμβάνουν.

Τα βήματα που ακολουθούμε παρουσιάζονται μέσω του παρακάτω activity diagram:



Εικόνα 80: Διάγραμμα ροής χρήσης για την περίπτωαη 2

Ακολουθούμε τα βήματα για την περίπτωση που τα examples αναγράφονται στο OpenAPI αρχείο:

1. Παράγουμε το OpenAPI αρχείο (yaml) μέσω του Visual Paradigm αν δεν το έχουμε ήδη

2. Μετακινούμε το OpenAPI αρχείο στο κατάλληλο folder

3. Εκτελλούμε το OpenapiDependencyExamples.py

4. Εισάγουμε το όνομα του OpenAPI αρχείου που θέλουμε να ερευνήσουμε

5. Παίρνουμε το αρχείο που δημιουργήθηκε για να το εμφανίσουμε στο Visual Paradigm

6. Εισάγουμε το αρχείο στο Visual Paradigm μέσω του εργαλείου Reverse OpenAPI

7. Επιλέγουμε τα components μέσω του Model explorer

8. Στα components επιλέγουμε με δεξί κλικ show associations

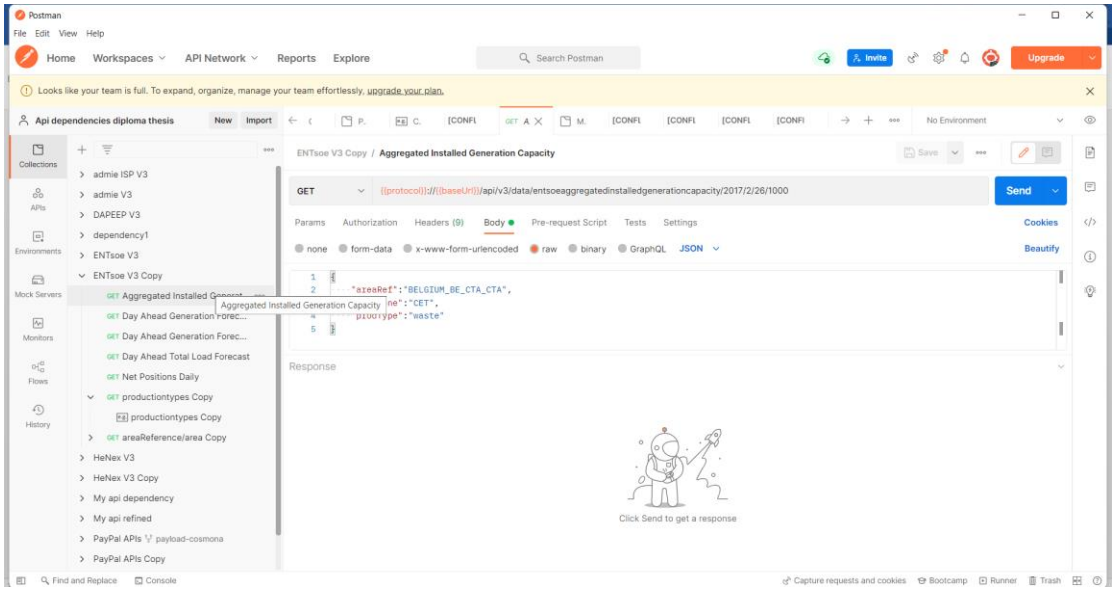
9. Εμφανίζουμε τον γράφο

4.2.3 Περίπτωση 3: REST API χωρίς καθόλου τεκμηρίωση

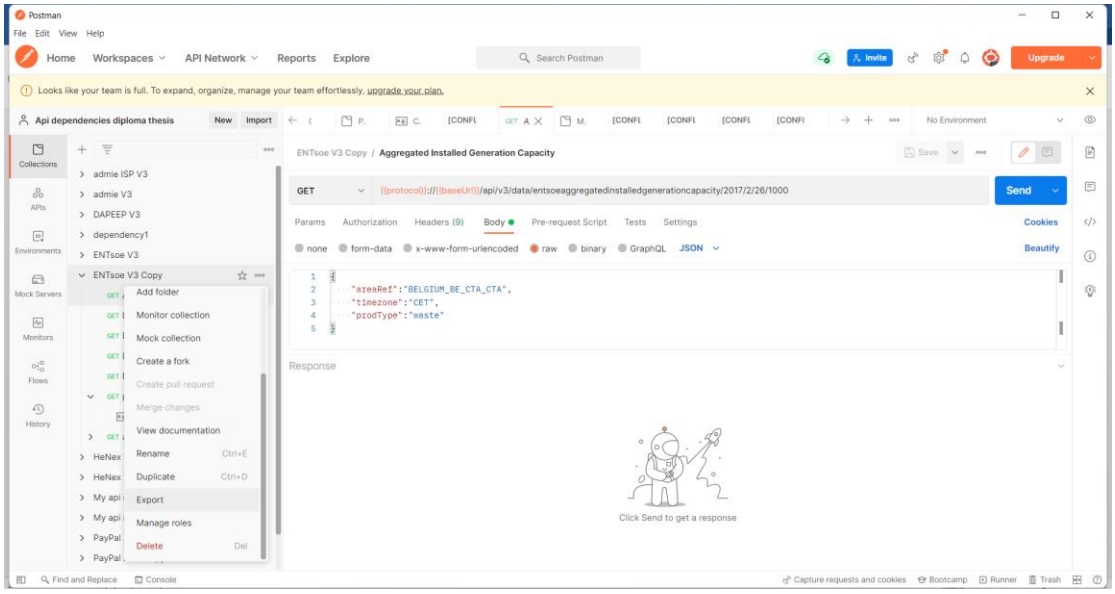
Οι παραπάνω περιπτώσεις προϋποθέτουν ότι υπάρχει ένα σωστά σχεδιασμένο OpenAPI αρχείο που μας δίνει πληροφορίες για το REST API που βρίσκεται υπό εξέταση. Πολλές φορές όμως αυτό δεν συμβαίνει. Θα δούμε την περίπτωση που έχουμε μόνο το REST API διαθέσιμο.

Σε αυτή την περίπτωση θα χρησιμοποιήσουμε το λογισμικό δημιουργίας και χρήσης API: postman [37], καθώς και ένα εργαλείο που αποτελεί αντικείμενο της διπλωματικής που είχε πραγματοποιήσει η Ναυσικά Αμπατζή και μας επιτρέπει να μετατρέπουμε postman collections σε OpenAPI αρχεία [38].

Το Postman μας δίνει τη δυνατότητα να δημιουργούμε συλλογές από κλήσεις στα endpoints ενός REST API τα οποία μπορούμε να κάνουμε export ως json αρχεία.

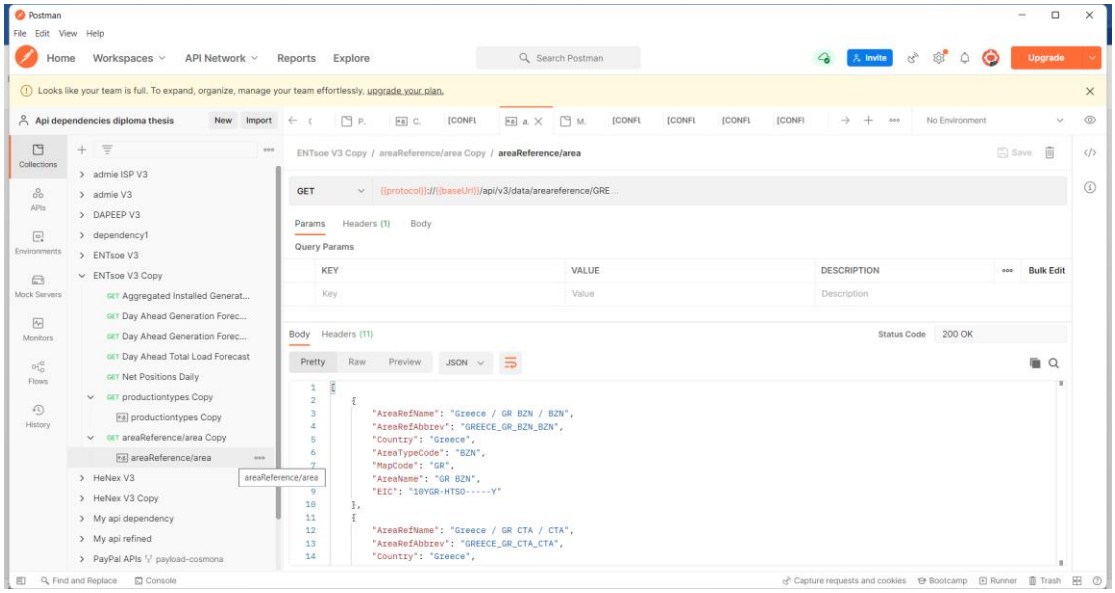


Εικόνα 81: Δημιουργία Postman collection



Εικόνα 82: Export Postman collection

Μία επίσης πολύ χρήσιμη λειτουργία που προσφέρει το postman είναι ότι μπορούμε να αποθηκεύσουμε τα responses που λαμβάνουμε από τα endpoints στη συλλογή μας ως παραδείγματα.



Εικόνα 83: Αποθήκευση response ως example

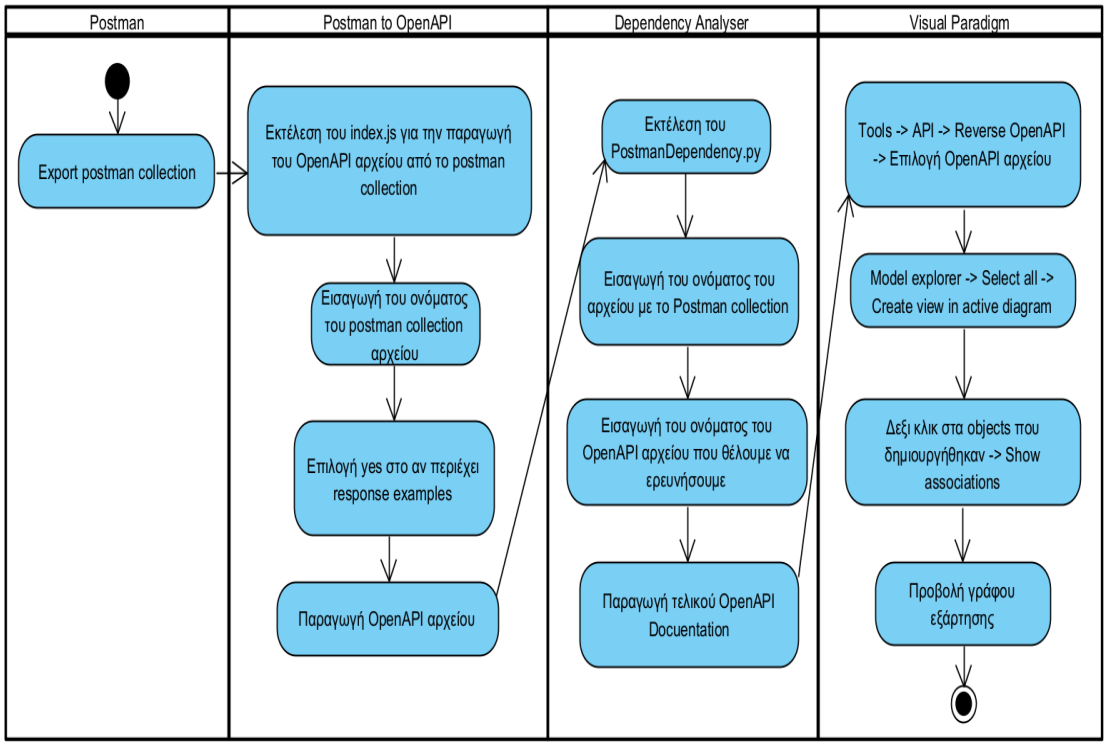
Τα examples αυτά υπάρχουν και στο παραγόμενο αρχείο όταν κάνουμε export, γεγονός που μας διευκολύνει να βρούμε τις εξαρτήσεις που αναζητούμε. Βλέπουμε ότι στο json αρχείο που γίνεται export, τα παραδείγματα αποθηκεύονται στο attribute ‘body’.

```
"cookie": [],
"body": "[\n    {\n        \"AreaRefName\": \"Greece / GR BZN / BZN\",\n        \"AreaRefAbbrev\": \"GREECE_GR_BZN
```


Εικόνα 84: Examples στο αρχείο με το εξαγόμενο postman collection

Στη συνέχεια παράγουμε το δεύτερο αρχείο με τη χρήση του εργαλείου Postman to OpenAPI [4] για μετατροπή postman collections σε OpenAPI καθώς θα χρειαστούμε και τα δύο για την ανάλυση των εξαρτήσεων και την εμφάνιση του τελικού γράφου στο Visual Paradigm.

Τα βήματα που ακολουθούμε φαίνονται αναλυτικά στο παρακάτω activity diagram.



Εικόνα 85: Διάγραμμα ροής χρήσης για την περίπτωση 3

1. Κάνουμε export το Postman collection ως json αρχείο
2. Από το εργαλείο Postman to OpenAPI εκτελούμε το index.js
3. Δίνουμε το όνομα του αρχείου που κάναμε export από το Postman
4. Επιλέγουμε yes στο αν περιέχει response examples
5. Αποθηκεύουμε το OpenAPI αρχείο
6. Εκτελλούμε το PostmanDependency.py
7. Εισάγουμε το όνομα του αρχείου με το Postman collection

8. Εισάγουμε το όνομα του OpenAPI αρχείου που θέλουμε να ερευνήσουμε

9. Αποθηκεύουμε το αρχείο που δημιουργήθηκε για να το εμφανίσουμε στο Visual Paradigm

10. Εισάγουμε το αρχείο στο Visual Paradigm μέσω του εργαλείου Reverse OpenAPI/Swagger

11. Επιλέγουμε τα components μέσω του Model explorer

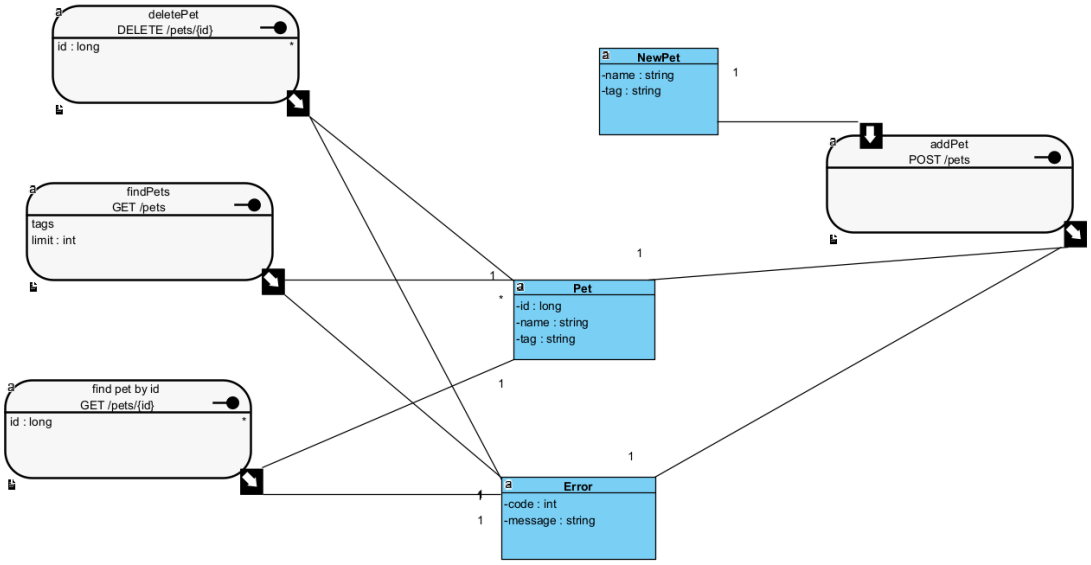
12. Στα components επιλέγουμε με δεξί κλικ show associations

13. Εμφανίζουμε τον γράφο

5 Χρήση του Dependency Analyser - Αποτελέσματα

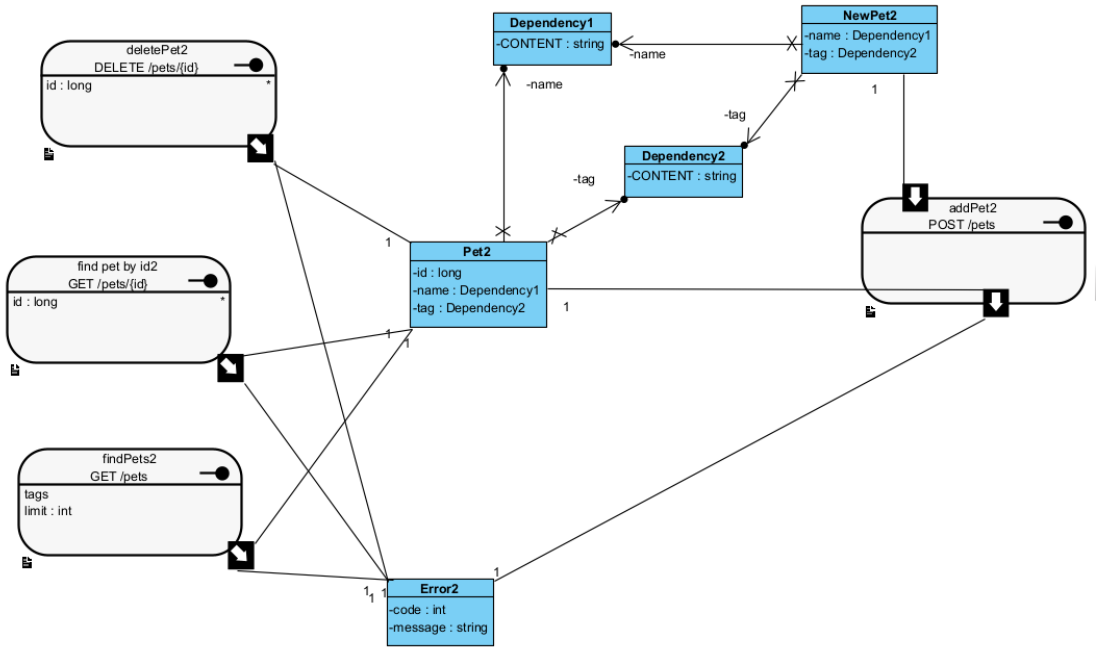
5.1 Petstore API

Θα χρησιμοποιήσουμε το σύστημα που αναπτύξαμε για να βρούμε τις εξαρτήσεις σε μια έκδοση του OpenAPI documentation για το Petstore API.



Εικόνα 86: Petstore API

Όπως αναφέραμε προηγουμένως, το OpenAPI Documentation σε αυτή την περίπτωση δεν περιέχει παραδείγματα. Συνεπώς ακολουθούμε το διάγραμμα της πρώτης περίπτωσης. Το αποτέλεσμα που καταλήγουμε παρουσιάζεται παρακάτω:



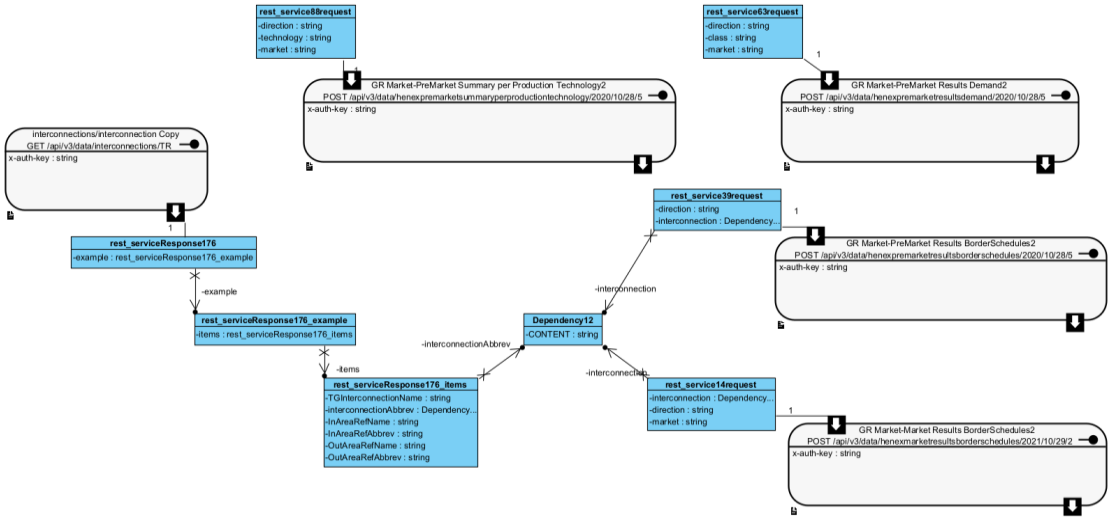
Εικόνα 87: Εξαρτήσεις στο Petstore API

Βλέπουμε ότι υπάρχουν δύο εξαρτήσεις στα attributes ‘name’ και ‘tag’ μεταξύ του request body του service addPet και του response των υπόλοιπων endpoints. Για να αναδείξουμε αυτές τις συσχετίσεις δημιουργήσαμε δύο νέες κλάσεις που ονομάσαμε Dependency1 και Dependency2. Τα σχετιζόμενα attributes κάνουν αναφορά σε αυτές τις κλάσεις οι οποίες περιέχουν ως πληροφορία τον τύπο των attributes, στη συγκεκριμένη περίπτωση ‘strings’.

Βλέπουμε λοιπόν ότι η πληροφορία που περιέχεται στο request body του service addPet συνδέεται άμεσα με τις πληροφορίες των responses στα υπόλοιπα services. Μέσω των services deletePet, findPet και findPetById μπροούμε να διαγράψουμε ή να βρούμε ένα κατοικίδιο το οποίο δημιουργήσαμε μέσω του service addPet. Αν και η πληροφορία αυτή ήταν σχετικά προφανής, εμείς καταφέραμε να την αναδείξουμε μέσω του συστήματός μας, γεγονός που αποδεικνύει ότι είναι εφικτό να εμφανίσουμε τις εξαρτήσεις μεταξύ των endpoints ενός REST API.

5.2 Energy API

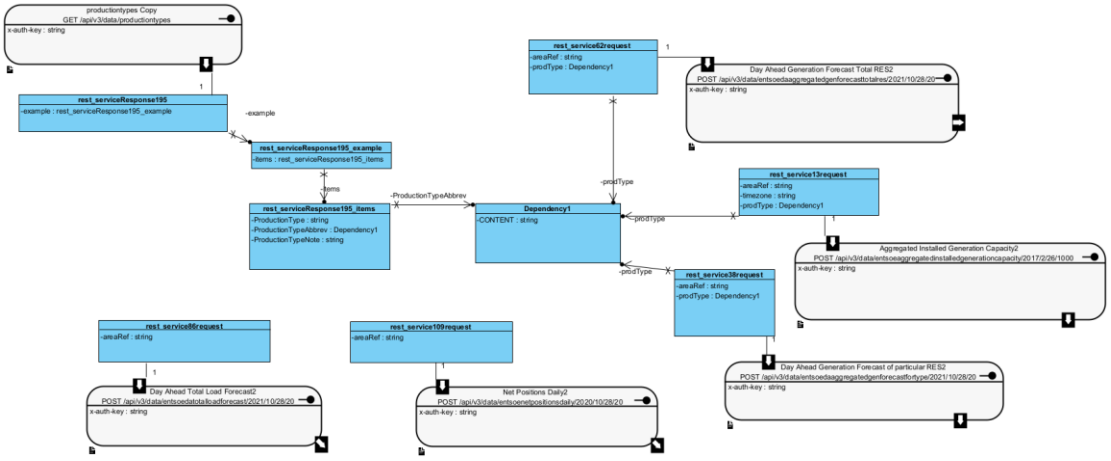
Στο παρακάτω παράδειγμα βλέπουμε μια περίπτωση μερικών services της εταιρείας Henex και της συσχέτισης τους με το βοηθητικό service interconnection. Στο παράδειγμα αυτό, γνωρίζουμε τις εξαρτήσεις και ξέρουμε ότι το attribute interconnectionAbbrev του βοηθητικού service δίνει τις τιμές των παραμέτρων interconnection στο request body κάποιων services της Henex. Επίσης κατέχουμε το OpenAPI Documentation μέσω του εργαλείου της Ναυσικάς. Συνεπώς ακολουθώντας πάλι το activity diagram της πρώτης περίπτωσης και ορίζοντας ως wildcard το string ‘abbrev’ μπορούμε να δούμε τις εξαρτήσεις που προκύπτουν στο παρακάτω διάγραμμα.



Εικόνα 88: Εξαρτήσεις στο Energy API 1

Ο χρήστης μπορεί να αξιοποιήσει αυτή την πληροφορία για να καλέσει πρώτα το endpoint του service ‘interconnection Copy’ και να δει τα πιθανές τιμές που μπορεί να θέσει ως παραμέτρους στα request bodies των υπόλοιπων services. Καθοδηγεί δηλαδή τον χρήστη στη σωστή σειρά κλήσης των endpoints και τον διευκολύνει στη χρήση του API.

Στη συνέχεια δημιουργούμε τον παρακάτω γράφο για τα services της εταιρείας Entsoe και του βοηθητικού endpoint ProductionTypes. Σε αυτή την περίπτωση κατέχουμε το OpenAPI Documentation και έχουμε εισάγει σε αυτό τα examples μέσω του Visual Paradigm. Συνεπώς ακολουθώντας το διάγραμμα για την δεύτερη περίπτωση μπορούμε να πάρουμε το αποτέλεσμα που επιθυμούμε.

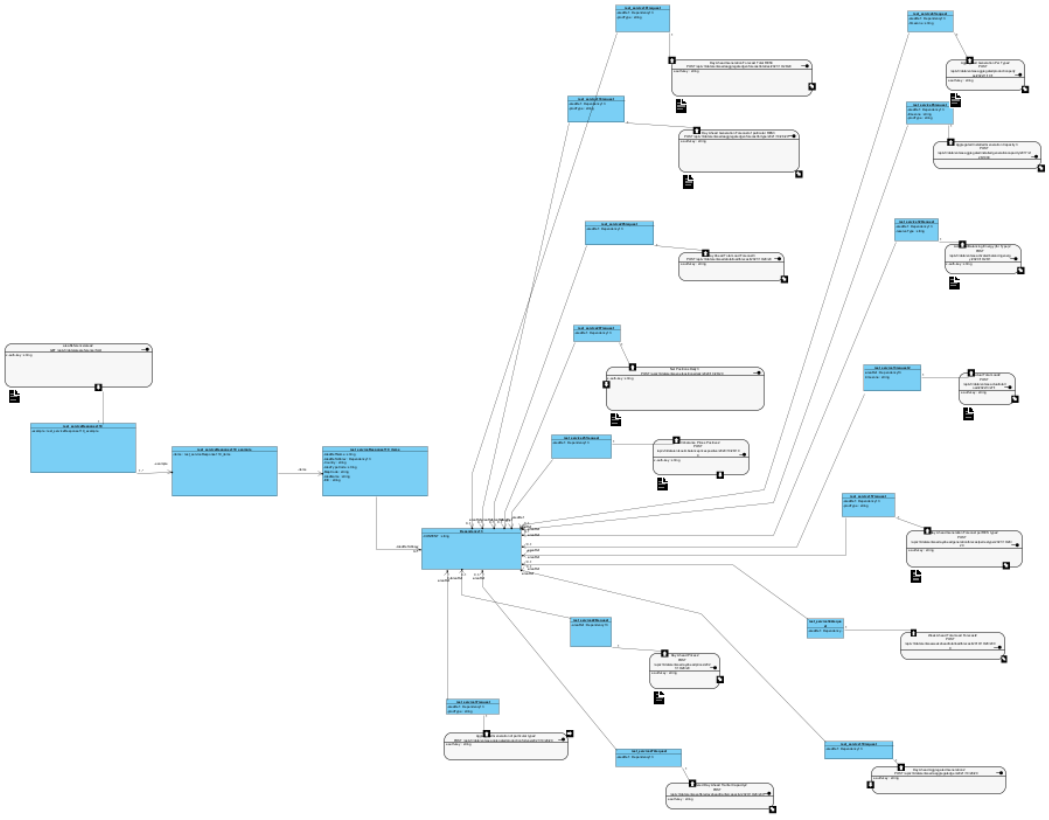


Εικόνα 89: Εξαρτήσεις στο Energy API 2

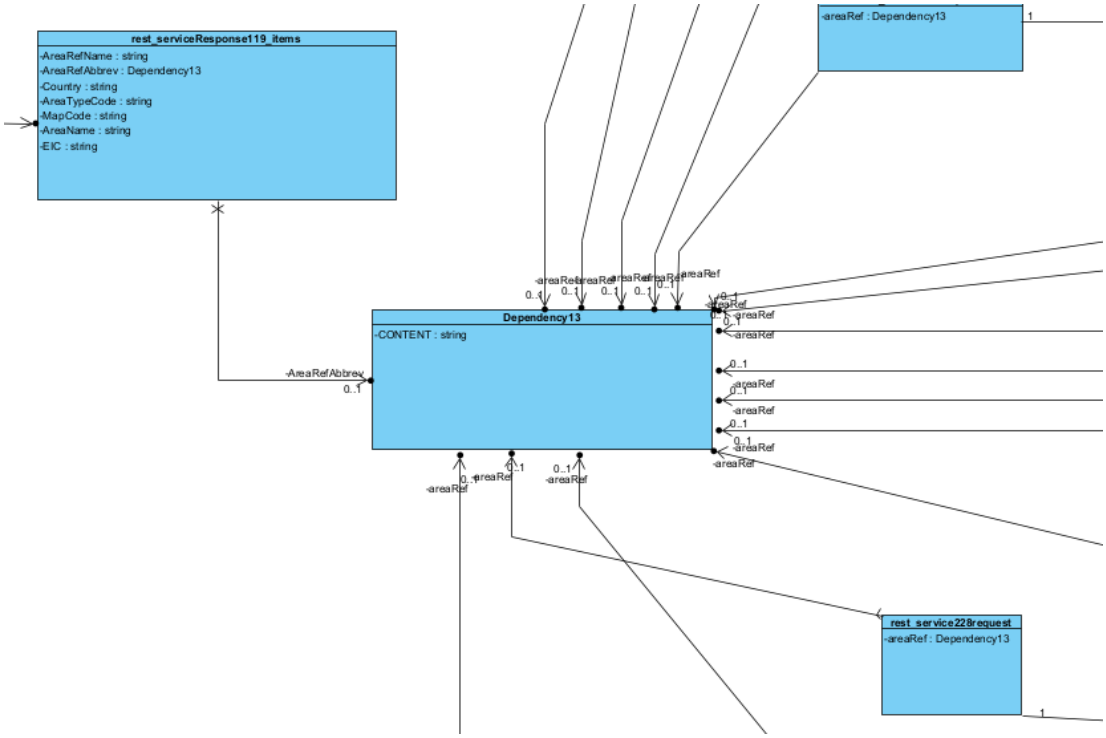
Παρατηρούμε ότι το ProductionTypeAbbrev του response body του service ‘productionstype Copy’ συνδέεται με το attribute prodType που ανήκει στο Request body 3 διαφορετικών REST services.

Ο χρήστης μπορεί να αξιοποιήσει αυτή την πληροφορία για να καλέσει πρώτα το endpoint του service ‘productionstype Copy’ και να δει τα πιθανές τιμές που μπορεί να θέσει ως παραμέτρους στα request bodies των υπόλοιπων services. Στη συγκεκριμένη περίπτωση, δεν θα μπορούσαμε να ακολουθήσουμε το διάγραμμα της πρώτης περίπτωσης καθώς τα εξαρτώμενα attributes έχουν αρκετά διαφορετικές ονομασίες. Ωστόσο μέσω των παραδειγμάτων καταφέραμε να ξεπεράσουμε αυτό το πρόβλημα και να υποδείξουμε και πάλι τη σωστή σειρά κλήσης των endpoints.

Τέλος δοκιμάζουμε το σύστημά μας για τα services της εταιρείας Entsoe και του βοηθητικού service areaReference. Σε αυτή την περίπτωση δεν έχουμε το OpenAPI αρχείο από πριν οπότε ακολουθούμε το διάγραμμα της τρίτης περίπτωσης και προκύπτει το παρακάτω αποτέλεσμα.



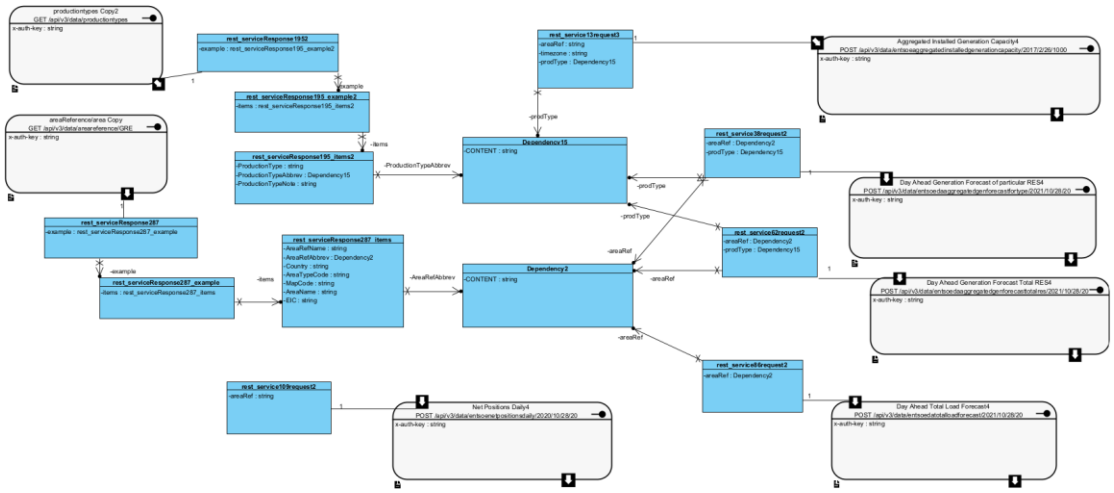
Εικόνα 90: Εξαρτήσεις στο Energy API 3



Εικόνα 91: Εξαρτήσεις στο Energy API 3 (Dependency Class)

Βλέπουμε την άμεση σύνδεση μεταξύ του attribute ‘AreaRefAbbrev’ που συμπεριλαμβάνεται στο response body του endpoint ‘areaReference/area’ και του attribute ‘arearef’ που περιλαμβάνεται στο request body πολλαπλών endpoints. Ο χρήστης μπορεί να αξιοποιήσει αυτή την πληροφορία για να καλέσει πρώτα το endpoint του service ‘areaReference/area’ και να δει τα πιθανές τιμές που μπορεί να θέσει ως παραμέτρους στα request bodies των υπόλοιπων services. Η περίπτωση αυτή είναι η πιο ενδιαφέρουσα καθώς καταφέραμε χωρίς κανένα Documentation και έχοντας μόνο το REST API στη διάθεσή μας να παράγουμε τον τελικό γράφο στον οποίο εμφανίζονται και οι σχέσεις εξάρτησης μεταξύ των endpoints.

Αντίστοιχα, ακολουθούμε την ίδια διαδικασία για τα services της Entsoe και των βοηθητικών services ‘production types’ και ‘areaReference’.

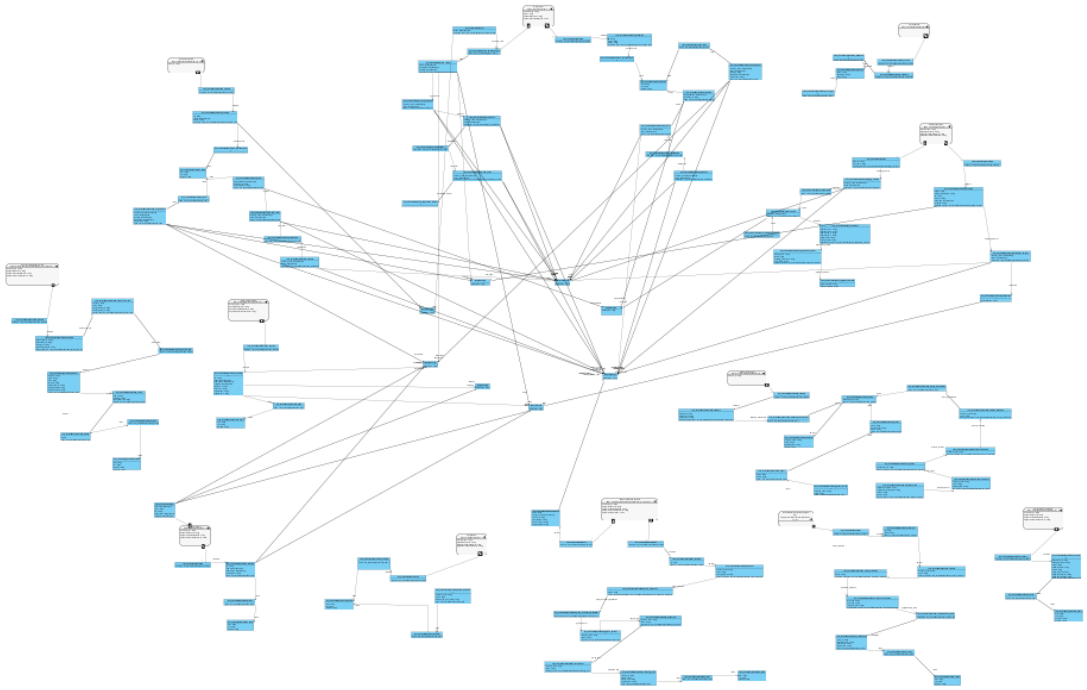


Εικόνα 92: Εξαρτήσεις στο Energy API 4

Βλέπουμε ότι περιλαμβάνονται πολλαπλές εξαρτήσεις μεταξύ των REST services τόσο μεταξύ των attributes prodType και productionTypeAbbreb καθώς και μεταξύ των areaRefAbbrev και areaRef. Ο χρήστης μπορεί να καλέσει τα services ‘productionType’ και ‘areaReference’ για να λάβει τις τιμές των παραμέτρων που μπορεί να εισαγάγει στα υπόλοιπα services. Συνεπώς, το σύστημά μας δεν περιορίζεται στο πλήθος των εξαρτήσεων αλλά δίνει τη δυνατότητα στο χρήστη να ανακαλύψει πολλαπλές εξαρτήσεις μεταξύ των endpoints του API.

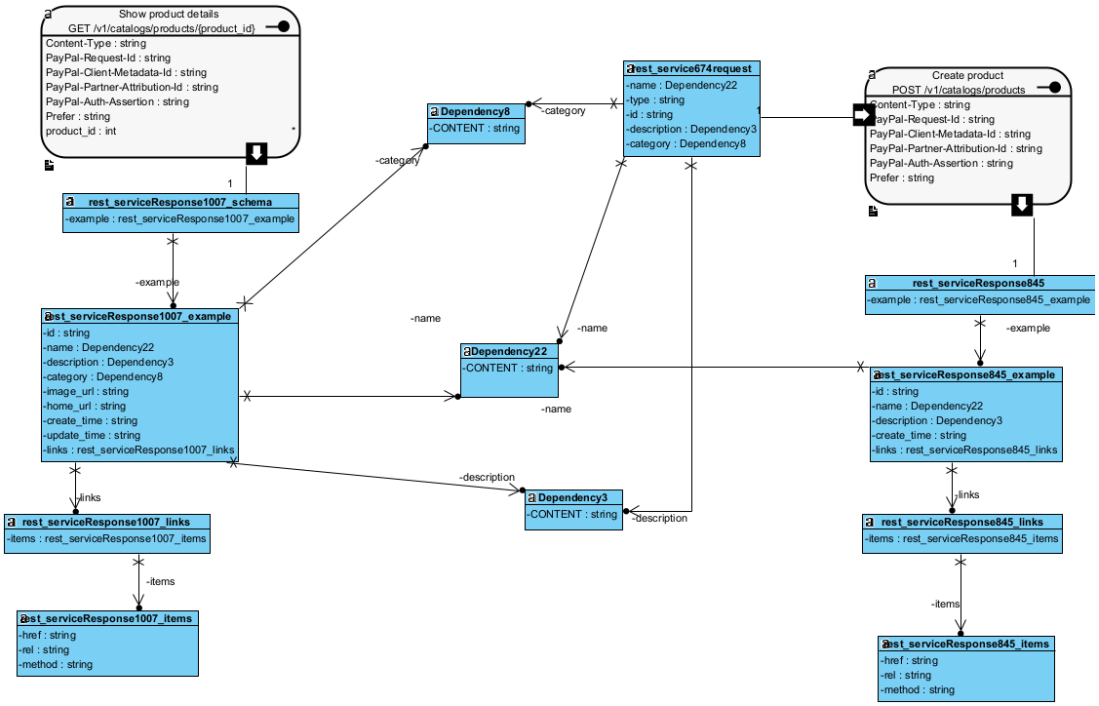
5.3 Paypal API

Ακολουθούμε τα βήματα του activity diagram [Εικόνα 85] του συστήματος στην περίπτωση που έχουμε μόνο το REST API για να καταλήξουμε στην παραγωγή του γράφου για το API της Paypal. Το αποτέλεσμα παρουσιάζεται παρακάτω:



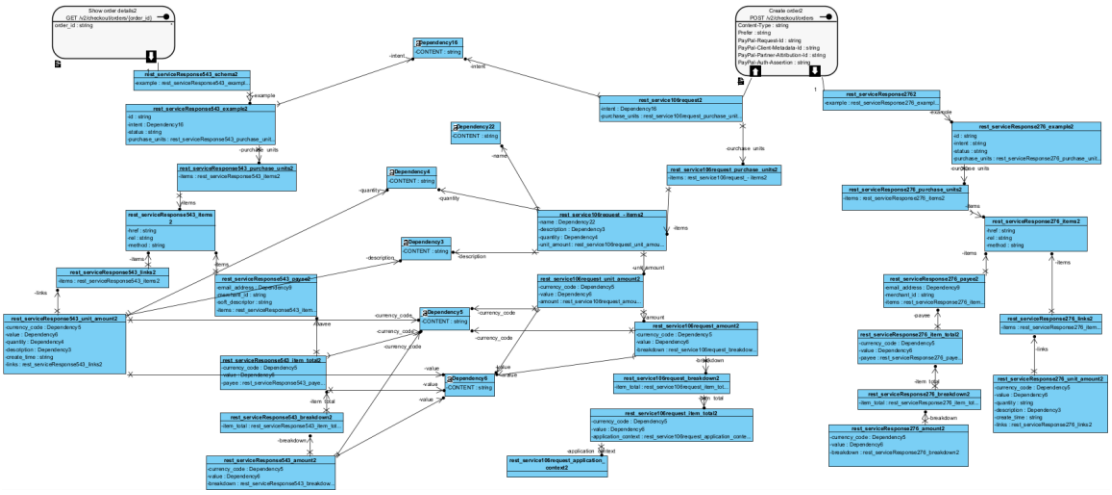
Εικόνα 93: Εξαρτήσεις στο Paypal API

Κοιτώντας πιο προσεκτικά βλέπουμε ότι με τη χρήση του εργαλείου μας αποκαλύπτονται αρκετές εξαρτήσεις μεταξύ των endpoints που επιλέξαμε.



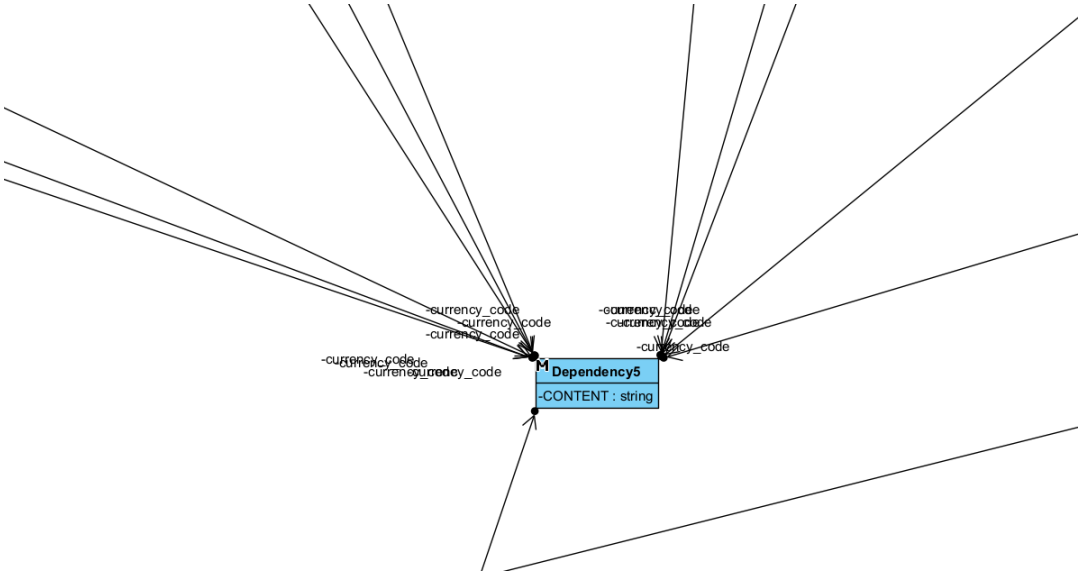
Εικόνα 94: Εξαρτήσεις στο Paypal API 1

Μεταξύ των Rest services ‘Show product details’ και ‘Create product’ βλέπουμε ότι υπάρχει εξάρτηση μεταξύ των attributes ‘name’ και ‘category’. Αναφέρονται δηλαδή στο ίδιο όνομα και στην ίδια κατηγορία προϊόντος. Με αυτό τον τρόπο ο χρήστης καταλαβαίνει ότι οι πληροφορίες του προϊόντος που βλέπει μέσω του ‘show product details’ αναφέρονται στα προϊόντα που δημιουργούνται μέσω του service ‘Create product’. Αντίστοιχα βλέπουμε ότι εμφανίζουν και την ίδια περιγραφή. Τέλος παρατηρούμε ότι και το attribute name του response body του Create product συνδέεται με το αντίστοιχο Dependency παρόλο που δεν υπάρχει το attribute αυτό στο request body του ‘Show product Details’. Αυτό συμβαίνει επειδή το attribute αυτό εμφανίζεται και στο request body ενός διαφορετικού REST service, συγκεκριμένα στο service ‘Create order’.



Εικόνα 95: Εξαρτήσεις στο Paypal API 2

Αντίστοιχα για τα services ‘create order’ και ‘show order details’ βλέπουμε ότι υπάρχουν πολλαπλές συσχετίσεις μεταξύ των attributes ‘value’ και ‘currency_code’. Με αυτό τον τρόπο ο χρήστης καταλαβαίνει ότι μέσω του service ‘show order details’ βλέπει τις πληροφορίες για την παραγγελία που δημιουργήθηκε μέσω του service ‘Create order’. Συγκεκριμένα είναι λογικό στις πληροφορίες μιας παραγγελίας να περιέχεται η ίδια τιμή και το ίδιο currency code για τα προϊόντα που επιλέχθηκαν κατά τη δημιουργία της. Ακόμη εμφανίζουν ίδια περιγραφή, ίδιο quantity και ίδιο intent. Το τελευταίο attribute ίσως να μην δίνει κάποια ξεκάθαρη πληροφορία στον χρήστη ωστόσο καταλαβαίνουμε ότι αυτά τα δύο services είναι στενά συνδεδεμένα μεταξύ τους. Τέλος βλέπουμε ότι το endpoint ‘Create order’ εμφανίζει εξάρτηση στο attribute ‘name’ του request body του, όπως είχαμε σχολιάσει προηγουμένως ότι εμφανίζει επιπλέον κάποια συσχέτιση με το REST service ‘Create product’. Η σχέση αυτή πιθανώς συνδέεται με το γεγονός ότι τα προϊόντα που δημιουργούνται από το service ‘Create product’ μπορούν στην συνέχεια να εισαχθούν στην παραγγελία μέσω του service ‘Create order’



Εικόνα 96: Εξαρτήσεις στο Paypal API 3

Τέλος, παρατηρούμε ότι υπάρχει συσχέτιση στα attributes ‘currency_code’ μεταξύ πολλαπλών REST services. Αυτό είναι λογικό καθώς όλα τα REST services ανήκουν στο REST API της paypal και όλα αναφέρονται στον ίδιο τύπο νομίσματος: ‘USD’ στη συγκεκριμένη περίπτωση.

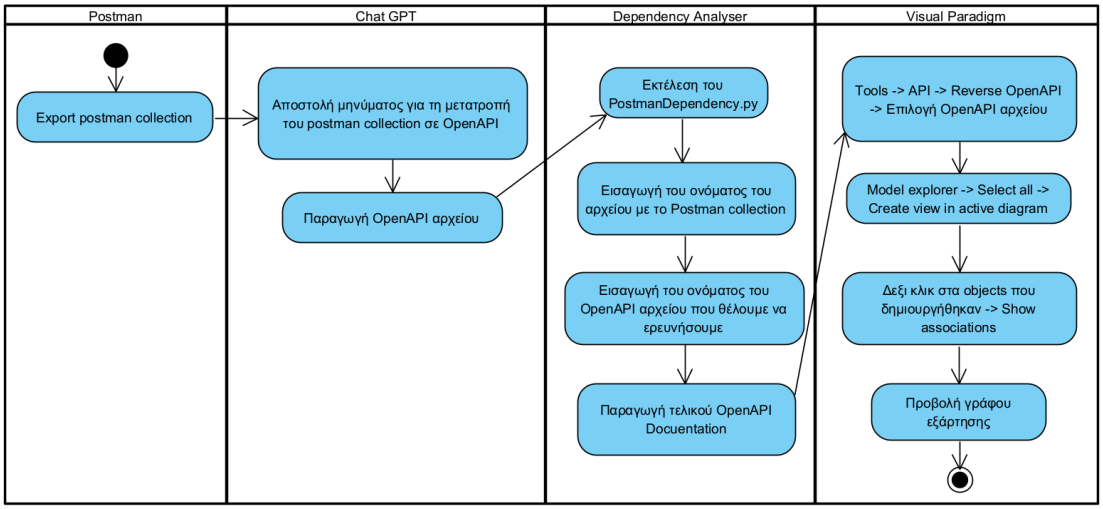
Βλέπουμε λοιπόν, ότι ακόμα και με ένα άγνωστο API για το οποίο δεν έχουμε κανένα είδος Documentation, καταφέραμε να δημιουργήσουμε ένα πλήρες Documentation στο οποίο εμφανίζονται οι εξαρτήσεις μεταξύ των διαφορετικών endpoints. Ο γράφος αυτός μας βοηθάει να αποκτήσουμε μια καλύτερη εικόνα για τον τρόπο χρήσης του συγκεκριμένου API, το καθιστά πιο προσιτό σε εμάς και μας διευκολύνει στην κατανόηση του τρόπου σύνδεσης των διαφορετικών endpoints.

6 Διερεύνηση αξιοποίησης Generative AI εργαλείου

Ο κόσμος του λογισμικού και του REST API εξελίσσεται διαρκώς, καθώς και τα εργαλεία που βελτιώνουν και διευκολύνουν την ανάπτυξή του, και δεν υπάρχει γενική ομοφωνία ούτε σωστή απάντηση στο ποιες τεχνολογίες πρέπει να επιλέξουμε όταν ασχολούμαστε με το συγκεκριμένο αντικείμενο. Τους τελευταίους μήνες έχουν φέρει την επανάσταση διάφορα generative AI εργαλεία τα οποία διευκολύνουν την ανάπτυξη λογισμικού. Θα επιχειρήσουμε να χρησιμοποιήσουμε ένα εργαλείο generative AI για να δούμε πώς θα μπορούσε να διευκολύνει τη χρήση του συστήματός μας.

Ένα από τα πιο σύγχρονα και ευρέως χρησιμοποιούμενα generative AI εργαλεία είναι το chat GPT [48]. Το chat GPT είναι ένα μοντέλο εκπαιδευμένο με τεχνικές μηχανικής μάθησης το οποίο μπορεί να μας δώσει απαντήσεις και να λύσει προβλήματα μέσω μιας διαδικασίας λήψης-αποστολής μηνυμάτων. Ένα χρήσιμο παράδειγμα στο οποίο θα επιχειρήσουμε να το αξιοποιήσουμε είναι στην παραγωγή OpenAPI documentation μέσω ενός postman collection του REST API.

Ακολουθούμε τα βήματα που φαίνονται στο παρακάτω activity diagram:



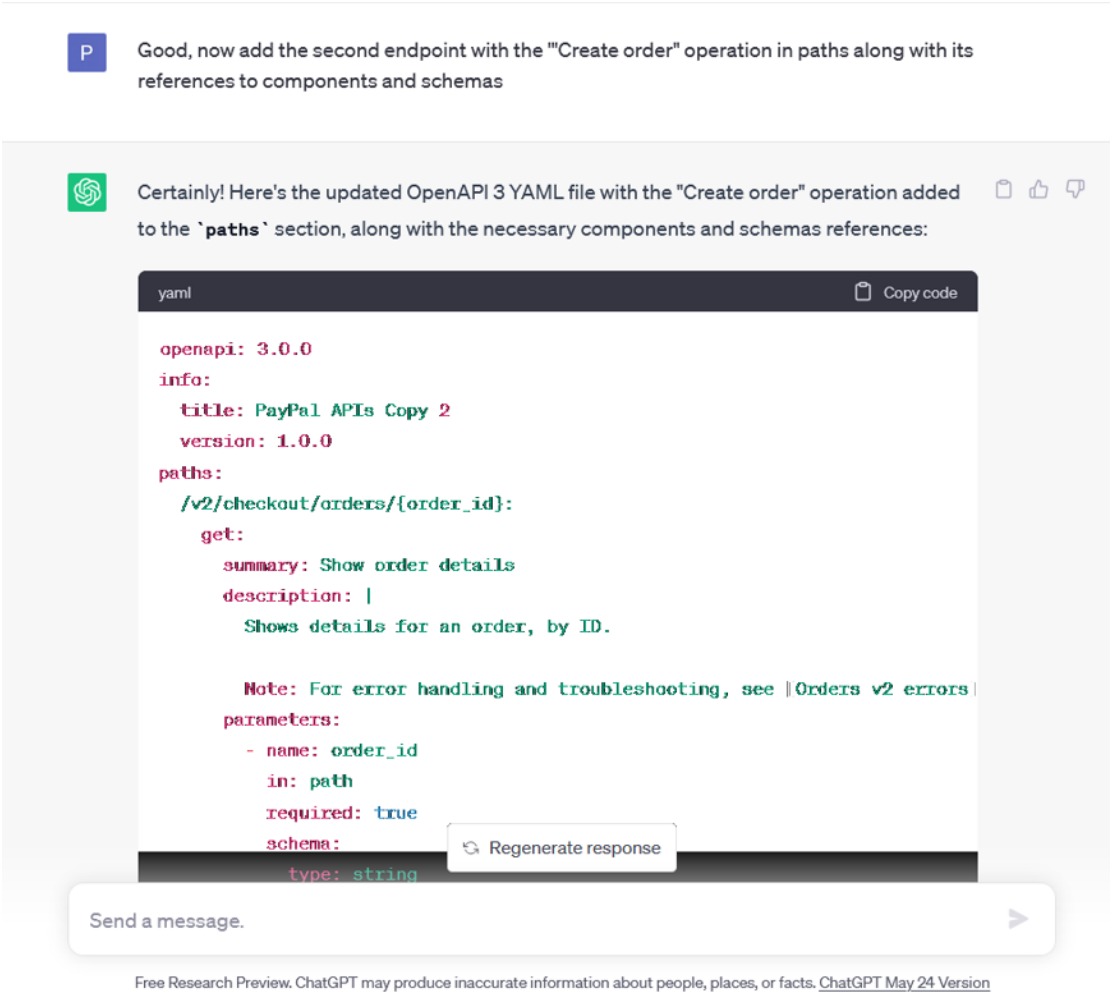
Εικόνα 97: Διάγραμμα ροής χρήσης του συστήματος σε συνδυασμό με το Chat GPT

1. Κάνουμε export το Postman collection ως json αρχείο
2. Κάνουμε επικόλληση το περιεχόμενο του αρχείου στο chat GPT και ζητάμε την μετατροπή του σε OpenAPI.
3. Αποθηκεύουμε το OpenAPI αρχείο
4. Εκτελλούμε το PostmanDependency.py
5. Εισάγουμε το όνομα του αρχείου με το Postman collection
6. Εισάγουμε το όνομα του OpenAPI αρχείου που θέλουμε να ερευνήσουμε
7. Αποθηκεύουμε το αρχείο που δημιουργήθηκε για να το εμφανίσουμε στο Visual Paradigm

8. Εισάγουμε το αρχείο στο Visual Paradigm μέσω του εργαλείου Reverse OpenAPI/Swagger
9. Επιλέγουμε τα components μέσω του Model explorer
10. Στα components επιλέγουμε με δεξί κλικ show associations
11. Εμφανίζουμε τον γράφο

Παρατηρούμε την απάντηση που λαμβάνουμε από το chat GPT αφού ζητήσουμε την μετατροπή του postman collection σε OpenAPI αρχείο.

Εικόνα 98: Αποστολή μηνύματος στο Chat GPT



Εικόνα 99: Λήψη απάντησης από το Chat GPT

Βλέπουμε ότι καταφέρνει να μετατρέψει το αρχείο μας στη μορφή που θέλουμε. Ωστόσο, η διαδικασία αυτή δεν εγγυάται πάντα επιτυχία καθώς το αποτέλεσμα που μας δίνει το chat GPT πολλές φορές δεν είναι το αναμενόμενο και χρειάζεται να δοκιμάσουμε διαφορετικές εντολές ακόμη και να δώσουμε επιπλέον οδηγίες όπως και στο παράδειγμα παραπάνω. Ακόμη, δεν διαθέτει τη δυνατότητα να εισάγουμε αρχεία και πρέπει να κάνουμε κάθε φορά επικόλληση το περιεχόμενο τους, γεγονός που μας περιορίζει καθώς έχει όριο στο συνολικό πλήθος των λέξεων που μπορεί να δεχτεί σε κάθε μήνυμα.

Σε γενικές γραμμές όμως, μας έδωσε μια ικανοποιητική λύση στο πρόβλημα αυτό. Το chat GPT είναι ένα εργαλείο που βελτιώνεται συνεχώς και ίσως στο άμεσο μέλλον να μπορούμε να ξεπεράσουμε τα προβλήματα που αντιμετωπίσαμε στην αυτοματοποίηση τέτοιων διεργασιών. Στη συνέχεια μπορούμε να επεξεργαστούμε αυτό το αρχείο μέσω του συστήματός μας όπως και προηγουμένως.

7 Συμπεράσματα - μελλοντική εργασία

7.1 Συμπεράσματα

Στην εργασία μας δημιουργήσαμε ένα ολοκληρωμένο εργαλείο που ξεκινάει την ανάλυση ενός REST API το οποίο είτε δεν διαθέτει καθόλου τεκμηρίωση, είτε διαθέτει ένα απλό OpenAPI documentation και καταλήγει σε ένα UML διάγραμμα κλάσεων το οποίο μπορούμε να το θεωρήσουμε ως ένα γράφο στον οποίο παρουσιάζονται πλήρως οι σχέσεις εξάρτησης μεταξύ των διαφορετικών endpoints. Το διάγραμμα αυτό προσφέρει έναν πρότυπο τρόπο για την βελτίωση του documentation ενός REST API. Συνδυάστηκαν πολλαπλά εργαλεία προκειμένου να μπορούμε εύκολα να φτάσουμε στην τελική παρουσίαση του documentation. Έχουν υπάρξει πολλαπλές απόπειρες παρουσίασης του documentation ενός REST API, ωστόσο ύστερα από αναζήτηση δεν βρήκαμε κάποιο σύστημα που να παρουσιάζει ταυτόχρονα τις εξαρτήσεις μεταξύ των endpoints.

Σημειώνεται ότι το εργαλείο που αναπτύχθηκε ανιχνεύει εξαρτήσεις μόνο μεταξύ endpoints τα οποία χρησιμοποιούν body parameters σε μορφή JSON objects. Ο λόγος για αυτό είναι ότι είναι εφικτό να εντοπιστεί η δομή ενός αντικειμένου JSON που περιέχεται στο body μιας κλήσης HTTP. Η ίδια η δομή αποτελεί τον ορισμό ενός τύπου, ο οποίος αποτυπώνεται ως κλάση στο αντίστοιχο διάγραμμα. Δεν είναι, όμως, δυνατό να εντοπιστούν τύποι μόνο από τα ονόματα url παραμέτρων, ενώ ακόμη δυσκολότερος, αν όχι αδύνατος, είναι ο εντοπισμός εξαρτήσεων μέσω παραμέτρων path.

Είδαμε ακόμη ότι οι σχέσεις εξάρτησης μεταξύ των endpoints είναι εμφανείς στον τελικό γράφο και μπορούμε να διακρίνουμε εύκολα endpoints που συνδέονται μεταξύ τους. Η πληροφορία αυτή είναι ιδιαίτερα σημαντική καθώς όπως είδαμε και στα παραδείγματα καταφέραμε να διακρίνουμε περιπτώσεις όπου απαιτούσαν να καλέσουμε τα endpoints με συγκεκριμένη σειρά για να λειτουργήσουν σωστά. Ακόμη, καταφέραμε να διακρίνουμε endpoints που συνδέονται λογικά μεταξύ τους και αναφέρονται σε κοινές πληροφορίες. Οι πληροφορίες αυτές δεν θα ήταν εύκολα ορατές χωρίς το κατάλληλο Documentation. Δυστυχώς, μολονότι το ιδανικό θα ήταν τέτοια τεκμηρίωση να είναι διαθέσιμη με κάθε API, αυτό δεν ισχύει, γεγονός που αναδεικνύει την αξία εργαλείων όπως αυτό που εισάγεται στην παρούσα εργασία.

7.2 Μελλοντικές επεκτάσεις

Η εργασία μας αποτελεί ένα πρώτο βήμα για τη βελτίωση του Documentation, ωστόσο υπάρχουν αρκετές ιδέες για μελλοντικές επεκτάσεις. Μια πιθανή πρόταση είναι η δημιουργία ενός λειτουργικού CLI για την καλύτερη ευχρηστία του συστήματος είτε ένα web app στο οποίο οι χρήστες θα μπορούν να χρησιμοποιήσουν το σύστημά μας χωρίς να απαιτείται να το εγκαταστήσουν τοπικά.

Ακόμη, θα μπορούσε να επιλεχθεί ένας διαφορετικός τρόπος παρουσίασης του τελικού γράφου. Μια ιδέα είναι η δημιουργία ενός γράφου που περιλαμβάνει μόνο τα REST services και τις κλάσεις των εξαρτήσεων χωρίς την πληροφορία των response bodies και των request bodies. Η ιδέα αυτή μπορεί να υιοθετηθεί αν θέλουμε να είναι πιο ξεκάθαρη η συσχέτιση των endpoints ανεξαρτήτως των attributes που τα συνδέουν.

Μπορούμε ακόμη να χρησιμοποιήσουμε διαφορετικά εργαλεία ή και να αξιοποιήσουμε generative AI τεχνολογίες (π.χ chatGpt) καθώς και διαφορετικό specification εκτός του OpenAPI. Ωστόσο, εμείς θέλαμε να δείξουμε έναν από τους τρόπους με τον οποίο μπορούν να είναι εμφανείς οι σχέσεις μεταξύ των endpoints και να μην αντιμετωπίζουμε κάθε endpoint αποκλειστικά ως ένα ξεχωριστό service. Από εκεί και πέρα οι επιλογές με τις οποίες μπορεί να εμπλουτιστεί και να παρουσιαστεί ένα documentation είναι πολλαπλές και αποτελούν μια ενδιαφέρουσα αφορμή για την επέκταση της έρευνάς μας.

Τέλος, έχει νόημα η διερεύνηση των εξαρτήσεων να γίνεται από δυναμική ανάλυση της συμπεριφοράς ενός API, δηλαδή μελετώντας τις κλήσεις που γίνονται κατά την πραγματική παραγωγική χρήση του. Στην περίπτωση αυτή ενδεχομένως να είναι δυνατός ο εντοπισμός εξαρτήσεων μεταξύ endpoints με χρήση URL και path parameters. Επίσης, θα είναι δυνατή η ανάλυση και τεκμηρίωση APIs που δεν απευθύνονται για χρήση από τρίτα μέρη, αλλά χρησιμοποιούνται εσωτερικά σε μια εφαρμογή, π.χ. για την επικοινωνία του frontend με το backend αυτής.

Αναφορές

[1] K. Lane, «Intro to APIs: History of APIs,» [Ηλεκτρονικό]. Available: https://blog.postman.com/intro-to-apis-history-of-apis/.

[2] M. Berg, «RPC vs API,» [Ηλεκτρονικό]. Available: https://medium.com/ankr-network/rpc-vs-api-whats-the-difference-455d253ca08f.

[3] «API,» [Ηλεκτρονικό]. Available: https://metamug.com/article/api-vs-rest-api.html.

[4] «What is an API?,» [Ηλεκτρονικό]. Available: https://www.postman.com/what-is-an-api/.

[5] M. Possamai, «HTTP,» [Ηλεκτρονικό]. Available: https://javascript.plainenglish.io/http-protocol-everything-you-need-to-know-d313b5544c93.

[6] «HTTP (HyperText Transfer Protocol),» [Ηλεκτρονικό]. Available: https://www3.ntu.edu.sg/home/ehchua/programming/webprogramming/HTTP_Basics.html.

[7] «What Are API Parameters? REST API URL Parameters [Explained],» [Ηλεκτρονικό]. Available: https://apipheny.io/what-are-api-parameters/.

[8] P. Sivakumar, «XML Basics,» [Ηλεκτρονικό]. Available: https://medium.com/@PrakhashS/xml-basics-a4ea2509f199.

[9] «Introduction to XML,» [Ηλεκτρονικό]. Available: https://www.w3schools.com/xml/xml_whatis.asp.

[10] «HTML Tutorial,» [Ηλεκτρονικό]. Available: https://www.w3schools.com/html/.

[11] «XML Syntax Rules,» [Ηλεκτρονικό]. Available: https://www.w3schools.com/xml/xml_syntax.asp.

[12] «How Can XML be Used?,» [Ηλεκτρονικό]. Available: https://www.w3schools.com/xml/xml_whatis.asp.

[13] O. Elgabry, «JSON In a Nutshell,» [Ηλεκτρονικό]. Available: https://medium.com/omarelgabrys-blog/json-in-a-nutshell-7d638dfea7cc.

[14] «JavaScript,» [Ηλεκτρονικό]. Available: https://www.javascript.com/.

[15] «JSON Syntax,» [Ηλεκτρονικό]. Available: https://www.w3schools.com/js/js_json_syntax.asp.

[16] «JSON vs XML,» [Ηλεκτρονικό]. Available: https://www.w3schools.com/js/js_json_xml.asp.

[17] «Stateful vs Stateless API,» [Ηλεκτρονικό]. Available: https://webo.digital/blog/stateful-vs-stateless-api-difference/.

[18] «Stateless APIs,» [Ηλεκτρονικό]. Available: https://restapilinks.com/stateless/.

[19] «Stateful strategy,» [Ηλεκτρονικό]. Available: https://ebrary.net/73340/computer_science/stateful_strategy.

[20] «Stateless strategy,» [Ηλεκτρονικό]. Available: https://ebrary.net/73339/computer_science/stateless_strategy.

[21] «REST API Definition,» [Ηλεκτρονικό]. Available: https://www.ibm.com/topics/rest-apis.

[22] «SOAP VS REST,» [Ηλεκτρονικό]. Available: https://www.interviewbit.com/blog/soap-vs-rest/.

[23] «SOAP API,» [Ηλεκτρονικό]. Available: https://www.indeed.com/career-advice/career-development/what-is-soap-api.

[24] S. Hati, «What is Rest API? Features, Principles, And Challenges,» [Ηλεκτρονικό]. Available: https://www.knowledgehut.com/blog/programming/rest-api.

[25] C. Staudinger, «SOAP vs. REST APIs,» [Ηλεκτρονικό]. Available: https://blog.postman.com/soap-vs-rest/.

[26] K. Vasudevan, «What is API Documentation, and Why It Matters?,» [Ηλεκτρονικό]. Available: https://swagger.io/blog/api-documentation/what-is-api-documentation-and-why-it-matters/.

[27] «OpenAPI Specification,» [Ηλεκτρονικό]. Available: https://swagger.io/specification/.

[28] M. Z. Mohammed, «Everything You Need To Know About YAML,» [Ηλεκτρονικό]. Available: https://medium.com/@mohsinzaheer1992/everything-you-need-to-know-about-yaml-9df0600c7255.

[29] «Swaggerhub,» [Ηλεκτρονικό]. Available: https://swagger.io/tools/swaggerhub/.

[30] «Stoplight,» [Ηλεκτρονικό]. Available: https://stoplight.io/.

[31] «Redocly,» [Ηλεκτρονικό]. Available: https://redocly.com/docs/.

[32] «PlantUML,» [Ηλεκτρονικό]. Available: https://plantuml.com/.

[33] «UML,» [Ηλεκτρονικό]. Available: https://en.wikipedia.org/wiki/Unified_Modeling_Language.

[34] «Visual Paradigm Tool,» [Ηλεκτρονικό]. Available: https://www.visual-paradigm.com/.

[35] «Class Diagram,» [Ηλεκτρονικό]. Available: https://en.wikipedia.org/wiki/Class_diagram.

[36] «Use Case Diagram,» [Ηλεκτρονικό]. Available: https://en.wikipedia.org/wiki/Use_case_diagram.

[37] «Postman Tool,» [Ηλεκτρονικό]. Available: https://www.postman.com/.

[38] Ν. Αμπατζή, «Postman to OpenAPI Tool,» [Ηλεκτρονικό]. Available: http://artemis.cslab.ece.ntua.gr:8080/jspui/handle/123456789/18349.

[39] «Postman collections,» [Ηλεκτρονικό]. Available: https://learning.postman.com/docs/collections/collections-overview/.

[40] «Postman Examples,» [Ηλεκτρονικό]. Available: https://learning.postman.com/docs/sending-requests/examples/.

[41] «Visual Paradigm Class Diagram,» [Ηλεκτρονικό]. Available: https://www.visual-paradigm.com/guide/uml-unified-modeling-language/uml-class-diagram-tutorial/.

[42] «Visual Paradigm Generating REST API,» [Ηλεκτρονικό]. Available: https://www.visual-paradigm.com/support/documents/vpuserguide/276/3420_generatingre.html.

[43] «Nodejs,» [Ηλεκτρονικό]. Available: https://nodejs.org/en.

[44] «Ecommerce github repository,» [Ηλεκτρονικό]. Available: https://github.com/davidmoten/openapi-to-plantuml/blob/master/src/test/resources/demo/ecommerce.yml.

[45] «Python3,» [Ηλεκτρονικό]. Available: https://www.python.org/downloads/.

[46] «Swagger Petstore,» [Ηλεκτρονικό]. Available: https://editor.swagger.io/?docExpansion=none.

[47] «Paypal's REST API,» [Ηλεκτρονικό]. Available: https://developer.paypal.com/api/rest/postman/.

[48] «Chat GPT,» [Ηλεκτρονικό]. Available: https://openai.com/blog/chatgpt.

[49] R. S. Wazlawick, «Stateless Strategy,» [Ηλεκτρονικό]. Available: https://www.sciencedirect.com/topics/computer-science/system-sequence-diagram.