
# ΠΑΝΕΠΙΣΤΗΜΙΟ ΚΡΗΤΗΣ

ΤΜΗΜΑ ΕΠΙΣΤΗΜΗΣ ΥΠΟΛΟΓΙΣΤΩΝ

**Αναφορά Πτυχιακής Εργασίας**

Θέμα:

Δημιουργία Web Service για εφαρμογή πρωτοκόλλου δημόσιας υπηρεσίας με χρήση OpenAPI

Εκπόνηση εργασίας:

Ιωάννης Χριστοδούλου, Α.Μ: 3503

Δημήτρης Λεοντιάδης, Α.Μ: 3393

Δήμητρα Κτιστάκη, Α.Μ: 3617

Επιβλέπων Καθηγητής:

Ιωάννης Γ. Τόλλης

Ηράκλειο, 2020

# ΠΕΡΙΕΧΟΜΕΝΑ

# 0. Εισαγωγή

Σκοπός της εργασίας είναι η υλοποίηση ενός web service που θα επικοινωνεί με μία υπάρχουσα βάση δεδομένων. Η βάση περιέχει πληροφορίες για τα έγγραφα και ειδικότερα για τα πρωτόκολλα που διαχειρίζεται μια δημόσια υπηρεσία. Για να επιτευχθεί η επικοινωνία ανάμεσα σε βάση και web service, δημιουργήθηκε ένα REST API το οποίο δίνει τη δυνατότητα σε κάποιον client να επιχειρήσει κάποιες προκαθορισμένες ενέργειες με τη χρήση του service.

## 0.1 Ορισμός του API

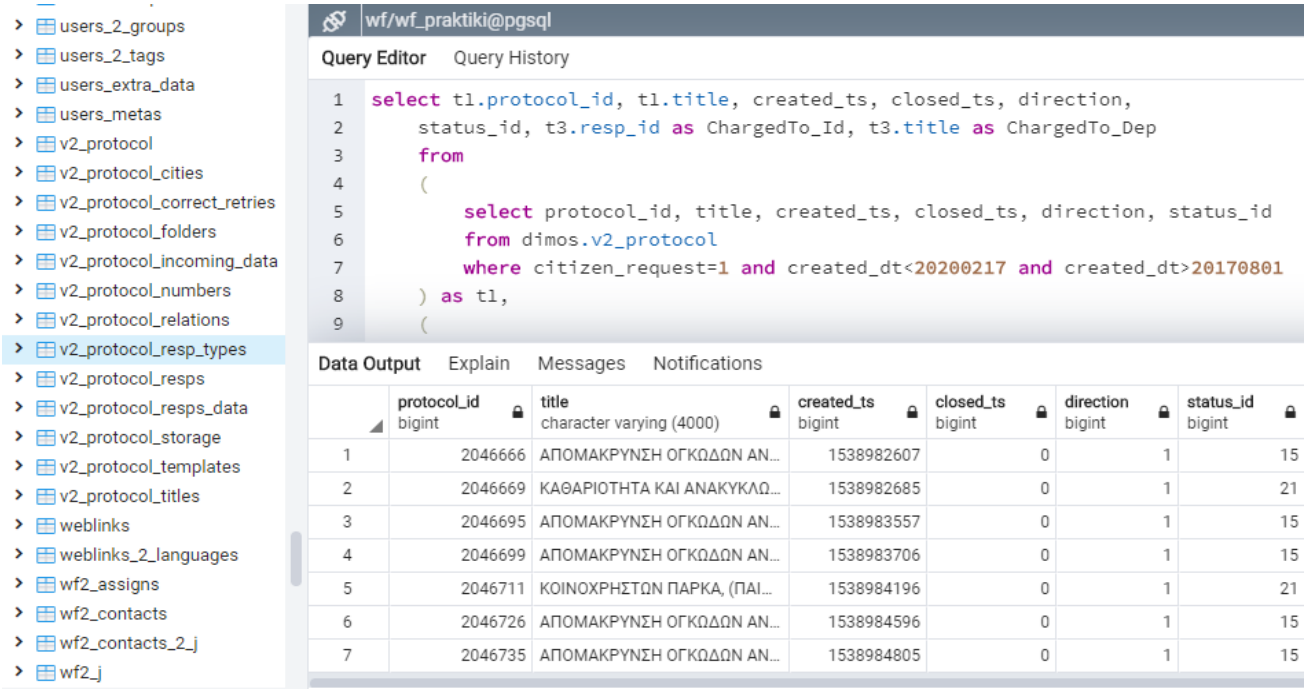
Το **application programming interface(API)** είναι ένα ενδιάμεσο λογισμικό το οποίο επιτρέπει την επικοινωνία μεταξύ πολλαπλών εφαρμογών. Με απλά λόγια, είναι ο φορέας που παραδίδει το ζητούμενο αίτημα στον εκάστοτε πάροχο και στη συνέχεια επιστρέφει την απάντηση πίσω στον client. Ένα API καθορίζει τις λειτουργίες οι οποίες είναι ανεξάρτητες από τις αντίστοιχες υλοποιήσεις τους. Αυτό επιτρέπει στις εν λόγω υλοποιήσεις και ορισμούς να διαφέρουν χωρίς να εκθέτει ο ένας τον άλλον. Εν κατακλείδι, το API ορίζει το σωστό τρόπο που ένας developer θα αναπτύξει μια εφαρμογή η οποία θα αιτείται υπηρεσίες από ένα λειτουργικό σύστημα ή μία άλλη εφαρμογή.

## 0.2 Λίγα λόγια για την αρχιτεκτονική REST

Το **REST(Representational State Transfer)** αποτελεί ένα σύνολο από αρχές σχεδίασης μιας δικτυακής υπηρεσίας που επικεντρώνεται στα resources ενός συστήματος. Η μεταβολή της κατάστασης των resources του συστήματος περιγράφεται και μεταφέρεται στο σύστημα μέσω του πρωτοκόλλου HTTP από διάφορους clients (ανεξαρτήτως της γλώσσας στην οποία έχουν υλοποιηθεί).
Με βάση το **REST**, το URI δεν χρησιμοποιείται πια για την περιγραφή της ενέργειας που θέλουμε να εκτελέσουμε αλλά μόνο για τον εντοπισμό του resource επί του οποίου θα ασκηθεί η ενέργεια. Τα δεδομένα δεν μεταφέρονται ως παράμετροι στο URI ενός GET αιτήματος αλλά ως XML ή JSON-formatted δεδομένα στο περιεχομενα μιας POST ή PUT μεθόδου. Με άλλα λόγια, ένα URI εκφράζει ένα αντικείμενο στο οποίο παρέχει πρόσβαση η υπηρεσία μέσω ενός HTTP αιτήματος. Το είδος του αιτήματος καθορίζει την ενέργεια που θέλουμε να εφαρμόσουμε στο αντικείμενο αυτό και το περιεχόμενο του αιτήματος περιέχει διάφορες εξειδικεύσεις της ενέργειας.
Οπότε ο σχεδιασμός μιας υπηρεσίας **REST** περιλαμβάνει κυρίως δύο βήματα. Πρώτον, να αποφασίσουμε τι αντικείμενα θα διαθέτουμε δια μέσω της υπηρεσίας και δεύτερον, τι ενέργεια θα εφαρμόζεται στο κάθε αντικείμενο για κάθε μία από τις GET, POST, PUT και DELETE μεθόδους.

# 1. Εξοικείωση με την υπάρχουσα βάση δεδομένων

Σε αυτό το στάδιο περιηγηθήκαμε στη **PostgreSQL** βάση δεδομένων αρχικά τρέχοντας κάποιες απλές επερωτήσεις προκειμένου να κατανοήσουμε τις σχέσεις μεταξύ των tables. Στη συνέχεια, υλοποιήσαμε πιο σύνθετες επερωτήσεις προκειμένου να πάρουμε τις πληροφορίες που ζητούσαν τα tasks τα οποία μας είχαν ανατεθεί. Η υλοποίηση των επερωτήσεων έγινε με τη χρήση του pgadmin. Ένα στιγμιότυπο από την χρήση του φαίνεται παρακάτω:



# 2. Δημιουργία documentation για το web service με τη χρήση του OpenApi Specification

Το OpenAPI Specification ορίζει ένα τυπικό language-agnostic interface σε RESTful APIs που επιτρέπει τόσο στους ανθρώπους όσο και στους υπολογιστές να ανακαλύψουν και να κατανοήσουν τις δυνατότητες της υπηρεσίας χωρίς πρόσβαση στον πηγαίο κώδικα, στην τεκμηρίωση ή μέσω επιθεώρησης κίνησης δικτύου. Όταν ορίζεται σωστά, ένας χρήστης μπορεί να κατανοήσει και να αλληλεπιδράσει με την απομακρυσμένη υπηρεσία με ένα ελάχιστο ποσό λογικής εφαρμογής.
Σε αυτήν τη φάση της εργασίας δημιουργήσαμε ένα documentation χρησιμοποιώντας το SwaggerHub και ακολουθώντας το OpenApi specification Version 3.

Το documentation ακολουθεί τη μορφή ενός yaml αρχείου καθώς είναι πιο εύκολα αναγνώσιμο. Το βασικό πλεονέκτημα του SwaggerHub σε αυτήν τη φάση είναι ότι παρέχει api mocks των resources και των endpoints τα οποία αλλάζουν δυναμικά έπειτα από ενδεχόμενες τροποποιήσεις στο yaml. Με άλλα λόγια, παράγεται ένα interactive documentation αυτόματα κατά τη σχεδίαση. Αυτό διευκολύνει τους API consumers και users να αλληλεπιδράσουν με το API. Για τον παραπάνω λόγο, ακολουθήθηκε η design-first προσέγγιση.

## 2.1 Δημιουργία σχημάτων και endpoints

Ορίσαμε μοντέλα-σχήματα τα οποία περιγράφουν τα resources του web service. Πολλά από τα μοντέλα προέκυψαν από συνδιασμό πινάκων της βάσης δεδομένων. Το κάθε resource αποτελείται από τρία models. Για παράδειγμα, το resource Protocol με τη βοήθεια του “allOf” αποτελείται από τα ProtocolId, ProtocolProperties, ProtocolRequiredProperties.

```
ProtocolRequiredProperties:
  required:
  - description
  - direction
  - number
  - priority
  - status
  - title
  - year
  type: object
Protocol:
  allOf:
  - $ref: '#/components/schemas/ProtocolId'
  - $ref: '#/components/schemas/ProtocolProperties'
  - $ref: '#/components/schemas/ProtocolRequiredProperties'
```

Επίσης περιγράφονται κάποια endpoints τα οποία αποτελούν τον τρόπο επικοινωνίας μεταξύ χρήστη και web service. Μέσω των endpoints δίνεται access στα resources. Ένα παράδειγμα endpoint από το οποίο μπορεί κανείς να αποκτήσει access στα τμήματα μιας υπηρεσίας είναι το εξής.

```
      - read:participants
    x-swagger-router-controller: Participant
/departments:
  get:
    tags:
    - department
    summary: Returns all departments.
    operationId: getDepartments
    parameters:
    - name: deptName
      in: query
      description: |
        The name of desired department.
      required: false
      style: form
      explode: true
      schema:
        type: string
    responses:
      "200":
        description: successful operation
        content:
```

Τα endpoints περιγράφονται με σεβασμό στη δομή ενός yaml. Σύμφωνα με OpenApi specification τα endpoints ορίζονται ως μέλη της γενικότερης οντότητας paths μέσα στην οποία το καθένα περιγράφεται με τον εξής τρόπο.

• Ορισμός του path το οποίο οδηγεί στο εκάστοτε api call από το οποίο μπορούν να προκύψουν ένα ή περισσότερα endpoints.
• Πρόσθεση του τύπου του http request (get, post, put, patch) ως attribute του αντίστοιχου path. Με αποτέλεσμα τη δημιουργία ενός endpoint.
• Προσθήκη ενός ή παραπάνω tag με σκοπό την κατηγοριοποίηση των endpoints ως attribute του αντίστοιχου request.
• Ορισμός μιας περιγραφής και ενός μοναδικού ονόματος μεθόδου που αντιστοιχεί στο endpoint.
• Ορισμός των παραμέτρων ως attribute του object parameters

• Ορισμός των πιθανών απαντήσεων(responses)

## 2.2 Status codes

Ανάλογα με το είδος του http request επιστέφονται οι κατάλληλες απαντήσεις. Για παράδειγμα, στα επιτυχημένα get requests επιστρέφεται το status code **200** μαζί με τα δεδομένα που ζητήθηκαν. Στα επιτυχημένα post requests επιστρέφεται το status code **201** το οποίο δηλώνει την επιτυχή δημιουργία μιάς εγγραφής στη βάση. Ενώ, όταν ο χρήστης στέλνει ένα request με μη έγκυρες παραμέτρους επιστρέφεται η απάντηση με status code **400**. Όταν οι παράμετροι που θα δώσει ο χρήστης είναι έγκυρες αλλά δεν υπαρχεί η ζητούμενη πληροφορία στη βάση, επιστρέφεται το status code **404**.

Παρακάτω ακολουθούν και τα υπόλοιπα status codes που έχουμε ορίσει μαζί με τα αντίστοιχα μηνύματα:

**204**: “The request has succeeded and there is no additional content to send”
**303**: “Redirecting the user agent to a different resource”
**409**: “Conflict with the current state of the target resource”
**415**: “Payload is in a format not supported by this method on the target resource”
**422**: ”Server is unable to process the contained instructions”
**500**: “Internal Server Error”

## 2.3 Τεχνικές σχεδίασης

Υπάρχουν κάποιες συνιστώμενες τεχνικές σχεδίασης τις οποίες ακολουθήσαμε:

- Ορισμός απλοποιημένων paths που είναι εύκολα κατανοήσιμα από τον χρήστη.
- Χρήση μόνο μικρών και όχι κεφαλαίων χαρακτήρων.
- Αντί για κενά διαστήματα (στα URI) παύλες ή κάτω παύλες.
- Χρήση ενικού για περιγραφή των resources στα post requests.
- Προσθήκη παραδειγμάτων στα πεδία των requests, όχι μόνο στο σχήμα των δεδομένων.
- Περιορισμός των επιστρεφόμενων αποτελεσμάτων όταν έχουν μεγάλο μέγεθος.
- Κατάλληλα error responses που ενημερώνουν τον client.

# 3. Υλοποίηση του REST API

Το SwaggerHub δίνει τη δυνατότητα παραγωγής κώδικα για τα Server Stub, από το specification που έχει δημιουργηθεί, μέσω του Swagger Codegen. Με τη βοήθεια του γίνεται εύκολη η μετάβαση από το design στο development στάδιο. Υπάρχει η δυνατότητα επιλογής γλώσσας του παραγόμενου κώδικα. Στην περίπτωση μας έγινε επιλογή του **nodejs**.

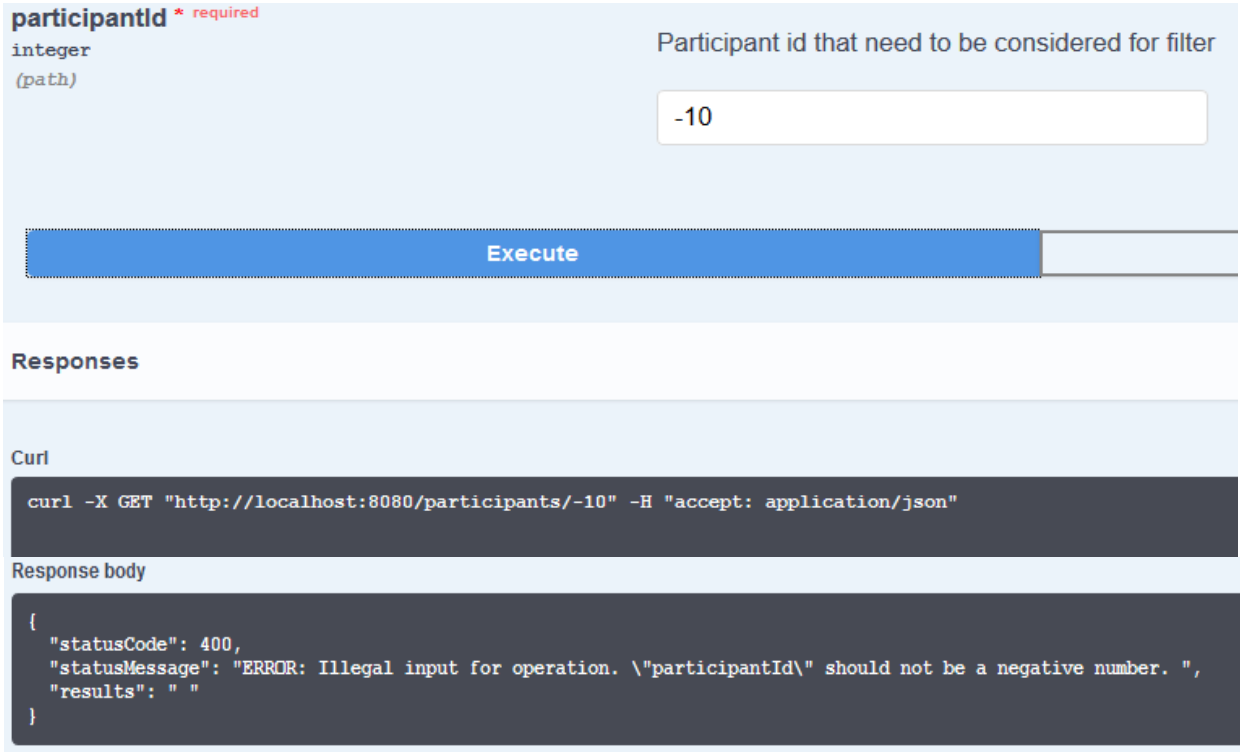
Το μοντέλο αρχιτεκτονικής που ακολουθήθηκε χωρίζει το web service σε controllers και services.

- Στον φάκελο **controllers** περιέχονται οι controllers για κάθε resource. Κάθε controller είναι ένα αρχείο στο οποίο περιέχονται οι μεθόδοι που καλούν το αντίστοιχο service.
- Στον φάκελο **services** περιέχονται τα services για κάθε resource. Κάθε service είναι ένα .js αρχείο στο οποίο περιέχονται οι μεθόδοι που είναι υπεύθυνες για την αλληλεπίδραση με τη βάση.

Παρακάτω δίνεται ένα στιγμιότυπο για το πως είναι η δομή του generated server stub.

## 3.1. Input Validation

Πριν γίνει η κλήση του εκάστοτε service και κατά συνέπεια πριν σταλεί η αντίστοιχη επερώτηση στη βάση δεδομένων, πραγματοποιούνται κάποιοι έλεγχοι στον controller σχετικά με την ορθότητα των παραμέτρων που έχουν δωθεί στο request. Σε περίπτωση που παρατηρηθεί κάποια παράμετρος η οποία δεν υπακούει στους περιορισμούς του αντίστοιχου api call, το request δε συνεχίζει με την αποστολή της επερώτησης στη βάση και το status code παίρνει την τιμή 400. Ακολουθεί ένα παράδειγμα που δώθηκε λάθος participant id.




## 3.2 Simple authentication

Με σκοπό να εξασφαληθεί το security integrity του web service, υλοιποιήθηκε authentication το οποίο απαιτεί τη χρήση ενός APIkey. Προκειμένου να πραγματοποιήθει μια σύνδεση με το API, είναι απαραίτητο ο client να γνωρίζει το APIkey. Αυτό το APIkey πρέπει να περιέχεται στο HTTP Basic Header και είναι απαραίτητο να περιέχεται σε κάθε request που στέλνει ο client. Το APIkey γίνεται export απο το module APIkey.js ( protocol_modules/APIkey.js) το οποίο χρησιμοποιείται στους controllers για τους απαραίτητους ελέγχους. Εφόσον το key που υπάρχει μέσα στο header του εκάστοτε request, είναι valid , τότε το αίτημα γίνεται authenticate και ξεκινάνε οι απαραίτητες ενέργειες για την εξυπηρέτηση του. Στην περίπτωση που το key είτε είναι λανθασμένο είτε δεν

υπάρχει, ένα error status code HTTP 401 θα επιστραφεί στον client με το μήνυμα “Unauthorized” και δεν θα πραγματοποιηθεί κάποια περαιτέρω ενέργεια για την εξυπηρέτηση του αιτήματος. Παρακάτω παρουσιάζονται στιγμιότυπα ενός authorized και ενος unauthorized request μαζί με τις απαντήσεις τους. Τα requests έγιναν με τη χρήση του command line tool curl.

## Authorized Request

**Curl**

```
curl -X GET "http://localhost:8080/protocols/19730" -H "accept: application/xml" -H "authentication: abcdef012345"
```

**Response body**


```
{
  "statusCode": 200,
  "statusMessage": "OK",
  "results": [
    {
      "protocol_id": "19730",
      "protocol_number": "17555",
      "protocol_year": "2008",
      "title": "ΓΙΑ ΠΙΣΤΟΠΟΙΗΤΙΚΟΥ ΧΑΤΖΗΝΙΚΟΛΗ ΝΙΚΟΛΑΟΥ ΧΟΡΗΓΗΓΗ ΤΑΥΤΟΤΗΤΑ",
      "descr": null,
      "created_ts": "1203585402",
      "closed_ts": "1203585402",
      "updated_ts": "0",
      "direction": "1",
      "comments": null,
      "keywords": null,
      "priority_id": "3",
      "status_id": "14",
      "created_dt": 20080221,
      "citizen_request": 0
    }
  ]
}
```


## Unauthorized Request

**Curl**

```
curl -X GET "http://localhost:8080/protocols/19730" -H "accept: application/xml" -H "authentication: sd10dsa10"
```

**Response body**


```
{
  "statusCode": 401,
  "statusMessage": "Unauthorized",
  "results": " "
}
```


Download

## 3.3 Integration with database

Για να επιτύχουμε τη σύνδεση στη βάση έχουμε δημιουργήσει ένα module από το οποίο γίνονται export δύο μεθόδοι(η connect και η connection).
Η connect είναι υπεύθυνη για τη δημουργία της σύνδεσης με τη βάση ενώ η connection λειτουργεί ως “getter" και η μόνη της λειτουργία είναι να επιστρέφει το connection που δημιούργησε η προγούμενη μέθοδος.
Η connect καλείται μία φορά στο index.js κατά την αρχικοποίηση του middleware, ενώ η connection καλείται σε κάθε NameService.js αρχείο, όπου Name το όνομα του εκάστοτε resource στο οποίο δίνεται access από το συγκεκριμένο service.
Για την αλληλεπίδραση με τη βάση δεδομένων, χρειάστηκε να παραμετροποιηθούν οι επερωτήσεις που είχαν δημιουργηθεί για τα ζητούμενα tasks. Αυτό πραγματοποιήθηκε με τη χρήση του πακέτου pg-parameterize και των εξής μεθόδων-συναρτήσεων:
- **toOrdinal** (pg-parameterize)
- **COALESCE** (sql)

Στη συνέχεια στέλνεται η επερώτηση στη βάση με τη χρήση της μεθόδου query η οποία ανήκει στο connection που επιστρέφει η μέθοδος connection.
Ακολουθούν παραδείγματα με την χρήση των παραπάνω συναρτήσεων.

```
 * Returns all departments.
 *
 * deptName String The name of desired department.  (optional)
 * returns List
 **/
exports.getDepartments = function(deptName) {
  return new Promise(function(resolve, reject) {

    const query_string = {
      // give the query a unique name
      name: 'fetch-departments',
      text: 'SELECT dep_id, title as name, phone, email '
      + 'FROM dimos.departments WHERE (COALESCE($1, title) = title);',
      values: [deptName],
    }
```

```
Querytext += ' limit ?';
if (limit != 'undefined' && limit)
  queryValues.push(limit);
else queryValues.push(100);

if (offset != 'undefined' && offset) {
  Querytext += ' OFFSET ?';
  queryValues.push(offset);
}

Querytext = pgParameterize.toOrdinal(Querytext);
```

# 4. Συμπεράσματα-Παρατηρήσεις

Η κατασκευή του web service έγινε πολύ ευκολότερη με τη χρήση του OpenApi χάρη στις δυνάτοτητες που προσφέρει. Κάποια από τα βασικά προνόμια είναι τα εξής:

• Διευκόλυνση της συνεργασίας μεταξύ των developers και εξασφάλιση συνέπειας και αποτελεσματικότητας.
• Εξοικονόμηση χρόνου και αποφυγή λαθών στον κώδικα.
• Έτοιμο περιβάλλον δοκιμών κατά τη διάρκεια της υλοποίησης(Swagger UI).
• Ομαλή μετάβαση από το στάδιο της σχεδίασης στην υλοποίηση με τη χρήση του Swagger Codegen.
• Εύκολη τροποποίηση του κώδικα με τη βοήθεια του specification.
• Επικύρωση των εισερχομένων και εξερχομένων αιτημάτων.

Εν κατακλείδι, το OpenAPI Specification είναι ένα εξαιρετικά αξιόλογο εργαλείο το οποίο θα συνιστούσαμε να λαμβάνεται υπόψην από όλους τους api providers ως μία αξιόπιστη επιλογή σε κάθε περίπτωση.

# 5. Οδηγίες εκκίνησης

**> npm start**

Με την παραπάνω εντολή γίνεται εκκίνηση του server, εκτελείται το index.js αρχείο και αφότου επιτευχθεί η σύνδεση με τη βάση γίνεται η αρχικοποίηση του middleware με την χρήση των OAS tools.
Στη συνέχεια δημιουργούνται τα routes που έχουν οριστεί στο specification και ο sever είναι έτοιμος να δεχτεί requests.

Το παρακάτω link οδηγεί στο Swagger UI Interface το οποίο περιέχει αναλυτικές πληροφορίες για τα api calls που έχουν υλοποιηθεί και μπορεί επίσης ο χρήστης να στείλει requests από εκεί.

http://localhost:8080/docs

# 6. Πηγές και Βιβλιογραφία

## Χρήσιμα links:

SwaggerHub:
https://swagger.io/tools/swaggerhub/

Swagger-codegen:
https://swagger.io/tools/swagger-codegen/
OpenAPI-Specification:
https://github.com/OAI/OpenAPI-Specification/blob/master/versions/3.0.0.md
pg-parameterize:
https://www.npmjs.com/package/pg-parameterize
pgadmin:
https://www.pgadmin.org/
Oas-tools:
https://www.npmjs.com/package/oas-tools