



ΤΜΗΜΑ ΕΦΑΡΜΟΣΜΕΝΗΣ ΠΛΗΡΟΦΟΡΙΚΗΣ  
ΠΡΟΓΡΑΜΜΑ ΜΕΤΑΠΤΥΧΙΑΚΩΝ ΣΠΟΥΔΩΝ  
ΕΙΔΙΚΕΥΣΗ ΕΠΙΣΤΗΜΗΣ ΚΑΙ ΤΕΧΝΟΛΟΓΙΑΣ Η/Υ

**«Πρακτικά Ζητήματα στα Δίκτυα και Διαδίκτυα Υπολογιστών»**

***Εισαγωγική προσέγγιση των Microservices και ανάπτυξη  
εφαρμογής μικρής κλίμακας***

Ευγένιος Σωχόπουλος  
Α.Μ. ΜΑΙ19071  
Μάιος 2019

## Περιεχόμενα

Περίληψη.....	2
Εισαγωγή.....	3
Προετοιμασία εφαρμογής .....	5
Δημιουργία 1 <sup>ου</sup> Microservice .....	6
Δημιουργία 2 <sup>ου</sup> Microservice .....	12
Δημιουργία 3 <sup>ου</sup> Microservice και Σύνδεση με τα Άλλα Microservices .....	16
Βιβλιογραφία.....	21

## Κατάλογος Εικόνων

Εικόνα 1. Δημιουργία του βασικού φακέλου του project.....	5
Εικόνα 2. Δημιουργία του φακέλων των τριών services .....	5
Εικόνα 3. Εγκατάσταση των απαραίτητων πακέτων στο books microservice .....	6
Εικόνα 4. Δομή φακέλου του books microservice .....	7
Εικόνα 5. Εκκίνηση λειτουργίας book microservice .....	10
Εικόνα 6. Παράδειγμα δημιουργίας ενός βιβλίου με το Postman .....	11
Εικόνα 7. Ανάκτηση όλων των διαθέσιμων βιβλίων με το Postman .....	11
Εικόνα 8. Δομή φακέλου του customers microservice. ....	12
Εικόνα 9. Παράδειγμα δημιουργίας ενός πελάτη με το Postman.....	15
Εικόνα 10. Δομή φακέλου του orders microservice .....	16
Εικόνα 11. Παράδειγμα δημιουργίας μιας παραγγελίας με το Postman.....	20
Εικόνα 12. Ανάκτηση μιας παραγγελίας με το Postman .....	20

## Περίληψη

Στην παρούσα εργασία πραγματοποιείται μία σύντομη επισκόπηση των microservices και υλοποιείται εφαρμογή μικρής κλίμακας με αυτή την αρχιτεκτονική.

Αρχικά γίνεται μια σύντομη εισαγωγή στα microservices. Αναφέρεται η αναγκαιότητα υιοθέτησής τους στη σύγχρονη βιομηχανία λογισμικού, καθώς και τα οφέλη χρησιμοποίησής τους σε σχέση με τις κλασικές μονολιθικές εφαρμογές. Στη συνέχεια καλύπτεται το κομμάτι της εμπειρίας χρήστη από την κατανάλωση των microservices και η επίλυση του προβλήματος της σύνδεσης και επικοινωνίας των πολλών διαφορετικών microservices μεταξύ τους.

Στο δεύτερο σκέλος της εργασίας, αναλύεται βήμα προς βήμα η ανάπτυξη της εφαρμογής. Η εφαρμογή αυτή παρέχει υπηρεσίες online βιβλιοθήκης, δηλαδή υπάρχει ένας κατάλογος με διαθέσιμα βιβλία, και ο πελάτης μπορεί να πραγματοποιήσει μια παραγγελία ενός βιβλίου. Σε πρώτο στάδιο πραγματοποιείται η επιλογή των διάφορων εργαλείων και frameworks που θα χρησιμοποιηθούν για το στήσιμο της εφαρμογής. Κατόπιν γίνεται εφαρμογή των αρχών αρχιτεκτονικής των microservices που συζητήθηκαν στο πρώτο σκέλος της εργασίας, όπως και η δημιουργία του πρώτου service που θα αποτελέσει το πρότυπο για τη δημιουργία των υπόλοιπων services. Τέλος, αναπτύσσονται και τα 2 υπόλοιπα services και συνδέονται όλα μεταξύ τους, ολοκληρώνοντας τη βασική λειτουργικότητα της εφαρμογής.

## Εισαγωγή

Με την έκρηξη των υπηρεσιών ιστού, καθώς και των κινητών συσκευών και των συσκευών με αισθητήρες, η ζήτηση για νέες υπηρεσίες και κατα συνέπεια και για κυκλοφορία δεδομένων αυξάνεται ραγδαία. Σύμφωνα με το Wireless World Research Forum (WWRF), ο αριθμός των συνδεδεμένων ασύρματων συσκευών αναμένεται να είναι 100 δισεκατομμύρια μέχρι το 2025. Οι υπηρεσίες ιστού, οι συσκευές με αισθητήρες, καθώς και οι τελικοί χρήστες, είναι γενικά διασπαρμένοι σε γεωγραφικά κατανομημένες περιοχές. Αυτό παρακινεί τους ASPs και ISPs να αναπτύξουν τις υπηρεσίες αυτές σε πολλαπλά clouds για κλιμάκωση και ταχύτερη απόκριση στους χρήστες. Η αυξανόμενη χρήση τεχνολογιών εικονικοποίησης βοηθά επίσης τους φορείς παροχής υπηρεσιών ASP και τους ISP να αναπτύξουν τις υπηρεσίες τους μέσω τυπικών υποδομών μεγάλου όγκου για να ικανοποιήσουν τις τεράστιες απαιτήσεις των χρηστών [1].

Στο παρελθόν, οι μεγάλες υπηρεσίες ιστού ήταν μονολιθικό λογισμικό, δηλαδή εφαρμογές που αναπτύσσονταν ως μία ενιαία οντότητα. Οι μονολιθικές εφαρμογές είναι σύνθετες, είναι δύσκολο να κλιμακωθούν, όπως και να αναβαθμιστούν με καινούρια χαρακτηριστικά. Οι εφαρμογές αυτές πλέον αντικαθίστανται από ένα σύνολο ελαφρών υπηρεσιών που ονομάζονται μικροϋπηρεσίες (microservices). Σήμερα, η αρχιτεκτονική των microservices γίνεται όλο και πιο δημοφιλής και υιοθετείται ευρέως από διάφορες μικρές και μεγάλες εταιρείες όπως οι Amazon, Google, Netflix, LinkedIn, SoundCloud και πολλές άλλες. Ο όρος microservices αναφέρεται σε εφαρμογές που αναπτύσσονται ως ένα σύνολο σχετικά μικρών, συνεπών, απομονωμένων και αυτόνομων υπηρεσιών που αναπτύσσονται ανεξάρτητα, με ένα σαφώς καθορισμένο σκοπό [2].

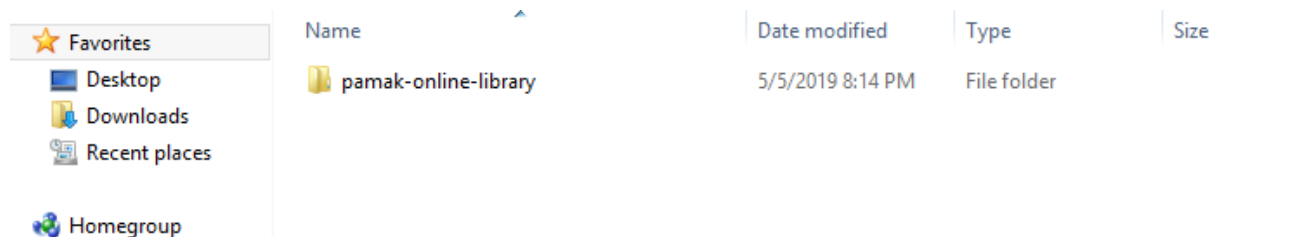
Ένα σημαντικό ζήτημα που υπάρχει σχετικά με την υλοποίηση των microservices, είναι ο τρόπος με τον οποίο αυτά επικοινωνούν μεταξύ τους, διαμορφώνοντας τελικά τις λεγόμενες αλυσίδες λειτουργίας υπηρεσιών (SFCs, Service Function Chains). Ο βέλτιστος σχεδιασμός των SFCs είναι πολύ σημαντικός για την ταχύτερη ανάπτυξη της εφαρμογής, την ελαχιστοποίηση των εξόδων ως προς τους ISPs, αλλά και το επίπεδο της ποιότητας της εμπειρίας που θα απολαμβάνουν οι τελικοί χρήστες.

Στην εφαρμογή που θα αναπτυχθεί στην συγκεκριμένη εργασία θα γίνει προσπάθεια να εφαρμοστούν οι γενικές αρχές που διέπουν την αρχιτεκτονική των microservices όπως αναφέρθηκαν παραπάνω. Η σύνδεση μεταξύ τους θα είναι πολύ απλή, χωρίς να γίνει λεπτομερής

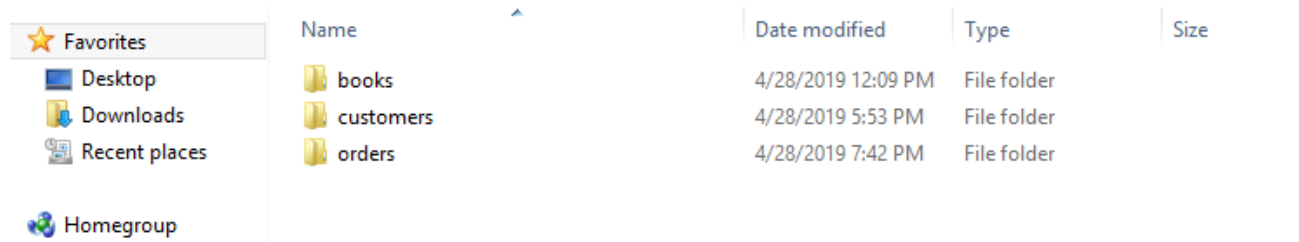
διερεύνηση των βέλτιστων SFCs, όπως θα γινόταν σε μια εφαρμογή που θα έβγαινε στην παραγωγή.

## Προετοιμασία εφαρμογής

Αρχικά θα πρέπει να υπάρχει εγκατεστημένο το NodeJS στον υπολογιστή. Για να γίνει αυτό, κατεβάζεται ο installer από τη διεύθυνση [nodejs.org](https://nodejs.org) και εγκαθίσταται στον υπολογιστή. Στη συνέχεια διαλέγουμε ένα φάκελο σε οποιαδήποτε τοποθεσία στον υπολογιστή, ο οποίος θα αποτελέσει το βασικό directory της εφαρμογής. Οπότε δημιουργείται ένα φάκελος με το όνομα `pamak-online-library` και μέσα σε αυτό το φάκελο δημιουργούνται 3 υποφάκελοι, 1 για κάθε ξεχωριστό microservice: `books`, `customers` και `orders`. Διευκρινίζεται ότι και τα 3 services θα είναι RESTful web services.



Εικόνα 1. Δημιουργία του βασικού φακέλου του project



Εικόνα 2. Δημιουργία του φακέλων των τριών services

Ως text editor θα χρησιμοποιηθεί το Visual Studio Code, το οποίο παρέχει αρκετές δυνατότητες και διαθέσιμα plugins για εύκολο και ευχάριστο προγραμματισμό σε JavaScript.

## Δημιουργία 1<sup>ου</sup> Microservice

Για να μπορεί να λειτουργήσει το web service σε ένα ικανοποιητικό βαθμό, κρίνεται απαραίτητο να εγκατασταθούν ορισμένες βιβλιοθήκες. Για να γίνει αυτό, ανοίγεται ένα command line των Windows και γίνεται πλοήγηση στην τοποθεσία που βρίσκεται το project. Στη συνέχεια εγκαθίστανται τα εξής πακέτα του NodeJS: `express`, `body-parser`, `mongoose`, `morgan` και `nodemon`.

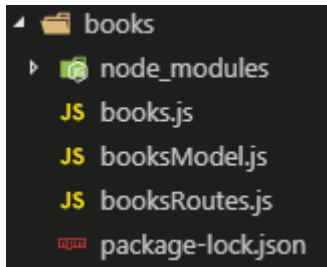


```
C:\Users\Eugenios\Desktop\MTPTXK\Diktya_Diadiktya\Ergasia\pamak-online-library\books>npm install --save express body-parser mongoose morgan nodemon_
```

Εικόνα 3. Εγκατάσταση των απαραίτητων πακέτων στο books microservice

- `express`: είναι ένα web framework του NodeJS, το οποίο καθιστά ευκολότερο τον προγραμματισμό εφαρμογών ιστού σε σχέση με τη χρησιμοποίηση απλής JavaScript στο περιβάλλον του NodeJS.
- `body-parser`: είναι ένα διαλογισμικό επεξεργασίας σώματος δεδομένων αίτησης (request body parsing middleware), το οποίο επιτρέπει την ανάλυση των δεδομένων αίτησης πριν αυτά φτάσουν στο στάδιο επεξεργασίας τους στον server.
- `mongoose`: είναι εργαλείο ανικειμενοστρεφούς μοντελοποίησης για χρήση σε συνδυασμό με μια κατανεμημένη βάση δεδομένων MongoDB.
- `morgan`: είναι ένα διαλογισμικό που επιτρέπει την καταγραφή (logging) των αιτήσεων HTTP.
- `nodemon`: είναι εργαλείο που επιτρέπει σε μια εφαρμογή να κάνει αυτόματη επανεκκίνηση κάθε φορά που εκτελούνται και σώζονται αλλαγές σε αρχεία της.

Αφού εγκατασταθούν τα παραπάνω πακέτα, δημιουργούνται μέσα στο φάκελο `books` τα αρχεία `books.js`, `booksModel.js` και `booksRoutes.js`. Ο φάκελος πλέον θα έχει την εξής μορφή:



Εικόνα 4. Δομή φακέλου του books microservice

Ανοίγεται το αρχείο `books.js`, όπου γίνεται η ενσωμάτωση και χρήση των βιβλιοθηκών που απαιτούνται σε αυτό το σημείο. Επίσης δηλώνεται η συνάρτηση που θέτει το service των books σε λειτουργία αναμονής για εισερχόμενο αίτημα από πελάτη. Οπότε το αρχείο `books.js` θα έχει την εξής μορφή:

```
1. // Load dependencies
2. const express = require("express");
3. const app = express();
4. const morgan = require('morgan');
5. const bodyParser = require("body-parser");
6.
7. // Incoming request data arrive in JSON format
8. app.use(bodyParser.urlencoded({extended: false}));
9. app.use(bodyParser.json());
10.
11. // Log requests before handling them
12. app.use(morgan('dev'));
13.
14. // Books service listens
15. const port = 3000;
16. app.listen(port, () => {
17.   console.log("\n\nBooks service started..");
18. });
```

Στη συνέχεια επεξεργάζεται το αρχείο `booksModel.js`. Φορτώνεται το `mongoose` και δηλώνεται το σχέδιο (Schema) του μοντέλου της βάσης δεδομένων που θέλουμε να έχουμε για το books service. Το κάθε βιβλίο θα χαρακτηρίζεται από τον τίτλο του, το συγγραφέα του, τον αριθμό των σελίδων του και τον εκδοτικό του οίκο.

Το μοντέλο (Model) του βιβλίου δημιουργείται χρησιμοποιώντας το Schema και αποθηκεύεται στο collection 'books' της βάσης δεδομένων. Ακόμα εξάγεται το book Model, ώστε να είναι διαθέσιμο για χρήση από τα υπόλοιπα αρχεία του service και γίνεται η σύνδεση με τη βάση δεδομένων MongoDB.



```

1. // Load dependencies
2. const mongoose = require("mongoose");
3.
4. // Define the Book Schema
5. var BookSchema = mongoose.Schema({
6.   title: {
7.     type: String,
8.     required: true
9.   },
10.  author: {
11.    type: String,
12.    required: true
13.  },
14.  numberPages: {
15.    type: Number,
16.    required: true
17.  },
18.  publisher: {
19.    type: String,
20.    required: true
21.  }
22. });
23.
24. // Create the Book Model
25. const Book = mongoose.model("Book", BookSchema, 'books');
26.
27. // Provide the Book Model to the book service
28. module.exports = Book;
29.
30. // Connect to MongoDB database
31. mongoose.connect("mongodb+srv://username:password@eugenecluster-incta.mongodb.net/books-
    libraryservice", () => {
32.   console.log("Connected to MongoDB Cluster - Books database..");
33. });

```

Το service των βιβλίων ολοκληρώνεται συμπληρώνοντας το αρχείο `booksRoutes.js` με την απαραίτητη πληροφορία. Εισάγονται δηλαδή οι συναρτήσεις που θα πραγματοποιούν την επιθυμητή λειτουργικότητα του service.

Φορτώνεται το πακέτο `express` και ταυτόχρονα εισάγεται και το `Book Model`. Δηλώνονται τέσσερις διαδρομές (routes) για τις τέσσερις διαφορετικές λειτουργίες που θα μπορεί να εκτελέσει το service:

- Η διαδρομή `localhost:3000/books`, η οποία έχει αντιστοιχηθεί με ένα GET request που επιστρέφει όλα τα διαθέσιμα βιβλία προς δανεισμό, αλλά και με ένα POST request με το οποίο δημιουργείται ένα καινούριο βιβλίο στη βάση δεδομένων
- Η διαδρομή `localhost:3000/books/{book_id}`, η οποία έχει αντιστοιχηθεί με ένα GET request που επιστρέφει τις πληροφορίες για 1 διαθέσιμο βιβλίο για δανεισμό δηλώνοντας τον αναγνωριστικό αριθμό του, αλλά και με ένα DELETE request που διαγράφει ένα βιβλίο

από τη βάση δεδομένων.

Οι διαδρομές αυτές γίνονται διαθέσιμες για χρήση και από τα υπόλοιπα αρχεία του service. Ο κώδικας για το αρχείο `booksRoutes.js` θα είναι:

```
1. // Load dependencies
2. const express = require('express');
3. const router = express.Router();
4. const BooksModel = require("./booksModel");
5.
6. // Get all books
7. router.get("/", (req, res, next) => {
8.   BooksModel.find().then((books) => {
9.     res.status(200).json(
10.      books
11.    )
12.   }).catch((err) => {
13.     if (err){
14.       throw err;
15.     }
16.   })
17. });
18.
19. // Get one book by id
20. router.get("/:id", (req, res, next) => {
21.   BooksModel.findById(req.params.id).then((book) => {
22.     if(book){
23.       res.json(book);
24.     } else {
25.       res.sendStatus(404);
26.     }
27.   }).catch((err) => {
28.     if (err){
29.       throw err;
30.     }
31.   })
32. });
33.
34. // Create a new book
35. router.post("/", (req, res, next) => {
36.   var newBook = {
37.     title: req.body.title,
38.     author: req.body.author,
39.     numberPages: req.body.numberPages,
40.     publisher: req.body.publisher
41.   }
42.   var book = new BooksModel(newBook);
43.   book.save().catch((err) => {
44.     if (err){
45.       throw err;
46.     }
47.   });
48.   res.status(201).json({
49.     status: 201,
```

```

50.     message: "New book: " + newBook.title + " added",
51.   });
52. });
53.
54. // Delete a book
55. router.delete("/:id", (req, res, next) => {
56.   BooksModel.findByIdAndDelete(req.params.id).then((book) => {
57.     res.status(200).json({
58.       status: 200,
59.       message: "Book " + book.title + " deleted"
60.     }).catch((err) => {
61.       if (err){
62.         throw err;
63.       }
64.     })
65.   })
66. });
67.
68. module.exports = router;

```

Οι browser υποστηρίζουν μόνο την εκτέλεση των GET requests, οπότε οι λειτουργίες POST και DELETE μπορούν να εκτελεστούν μόνο μέσω γραμμής εντολών ή μέσω ειδικού λογισμικού. Συγκεκριμένα, θα χρησιμοποιηθεί το λογισμικό Postman για όλα τα είδη των requests.

Πριν δημιουργηθούν καινούρια βιβλία, ενημερώνεται κατάλληλα στις παρακάτω γραμμές το αρχείο `books.js` ώστε να μπορεί το service να υποστηρίξει τα routes που δηλώθηκαν πιο πάνω.

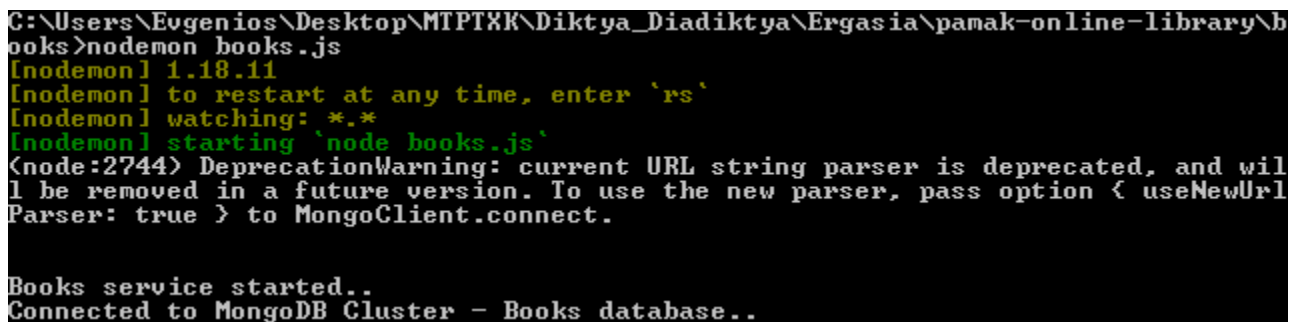
```

6. // Load Book Routes
7. const booksRoutes = require("./booksRoutes");

16. // Routes which are handled by the handlers
17. app.use('/books', booksRoutes);

```

Το service αρχίζει να τρέχει πληκτρολογώντας την εντολή `nodemon books.js`:



```

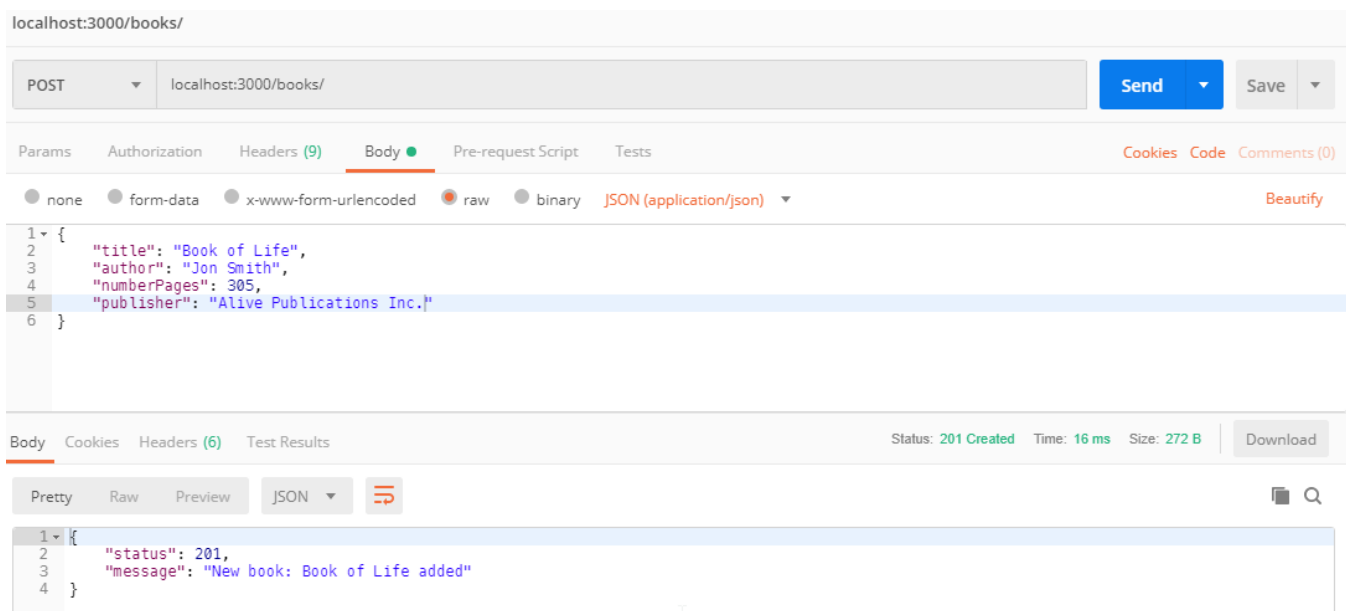
C:\Users\Eugenios\Desktop\MTPTXX\Diptya_Diadiptya\Ergasia\pamak-online-library\books>nodemon books.js
[nodemon] 1.18.11
[nodemon] to restart at any time, enter `rs`
[nodemon] watching: *.*
[nodemon] starting `node books.js`
(node:2744) DeprecationWarning: current URL string parser is deprecated, and will be removed in a future version. To use the new parser, pass option { useNewUrlParser: true } to MongoClient.connect.

Books service started..
Connected to MongoDB Cluster - Books database..

```

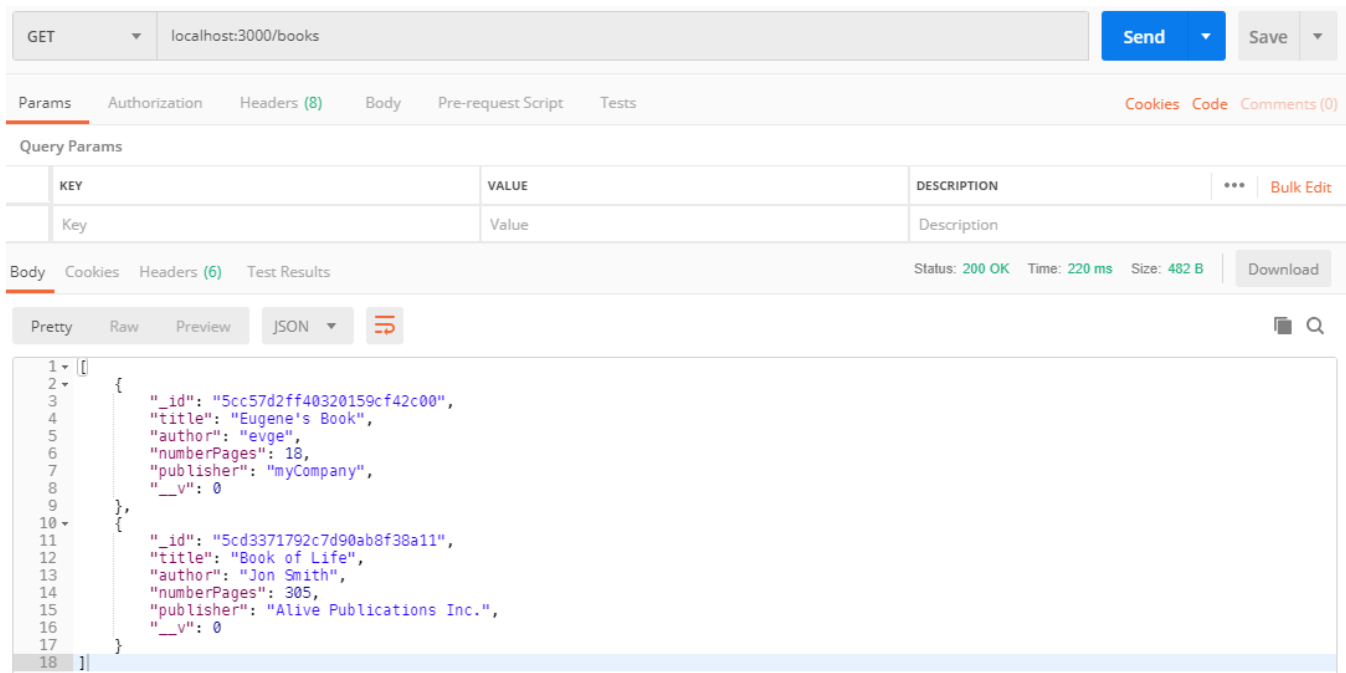
Εικόνα 5. Εκκίνηση λειτουργίας book microservice

Τώρα πλέον μπορούν να πραγματοποιηθούν τα επιθυμητά requests. Χρησιμοποιώντας, το Postman, γίνεται η δημιουργία των δύο πρώτων βιβλίων.



Εικόνα 6. Παράδειγμα δημιουργίας ενός βιβλίου με το Postman

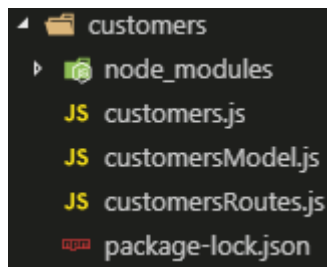
Επίσης μπορεί να κληθεί η λειτουργία για την ανάκτηση όλων των διαθέσιμων βιβλίων.



Εικόνα 7. Ανάκτηση όλων των διαθέσιμων βιβλίων με το Postman

## Δημιουργία 2<sup>ου</sup> Microservice

Για τη δημιουργία του service των πελατών (customers), ακολουθείται παρόμοια διαδικασία με αυτή για τη δημιουργία του books service. Αρχικά εγκαθίστανται και εδώ τα πακέτα express, body-parser, mongoose, morgan και nodemon του NodeJS. Μετά την εγκατάσταση δημιουργούνται μέσα στο φάκελο customers τα αρχεία customers.js, customersModel.js και customersRoutes.js ,με αυτόν να έχει τη μορφή:



Εικόνα 8. Δομή φακέλου του customers microservice.

Στο αρχείο customers.js η ενσωμάτωση των απαραίτητων πακέτων και δηλώνεται η συνάρτηση που θέτει το service των customers σε λειτουργία αναμονής για εισερχόμενο αίτημα από πελάτη. Οπότε το αρχείο customers.js θα έχει την αρχική μορφή:

```
1. // Load dependencies
2. const express = require("express");
3. const app = express();
4. const morgan = require('morgan');
5. const bodyParser = require("body-parser");
6.
7. // Incoming request data arrive in JSON format
8. app.use(bodyParser.urlencoded({extended: false}));
9. app.use(bodyParser.json());
10.
11. // Log requests before handling them
12. app.use(morgan('dev'));
13.
14. // Customers service listens
15. const port = 3050;
16. app.listen(port, () => {
17.   console.log("\n\nCustomers service started..");
18. });
```

Ακολουθεί η δημιουργία μιας καινούριας βάσης δεδομένων για το service των customers αλλά και το customers Model, με παρόμοιο τρόπο όπως και στο books service. Το μόνο που αλλάζει είναι το Schema, όπου εδώ ο κάθε πελάτης θα χαρακτηρίζεται από το όνομά του, την

ηλικία του, τη διεύθυνση και το τηλέφωνό του. Ο κώδικας φαίνεται παρακάτω:

```
1. // Load dependencies
2. const mongoose = require("mongoose");
3.
4. // Define the Customer Schema
5. var CustomerSchema = mongoose.Schema({
6.   name: {
7.     type: String,
8.     required: true
9.   },
10.  age: {
11.    type: Number,
12.    required: true
13.  },
14.  address: {
15.    type: String,
16.    required: true
17.  },
18.  phone: {
19.    type: String,
20.    required: true
21.  }
22. });
23.
24. // Create the Customer Model
25. const Customer = mongoose.model("Customer", CustomerSchema, 'customers');
26.
27. // Provide the Customer Model to the customer service
28. module.exports = Customer;
29.
30. // Connect to MongoDB database
31. mongoose.connect("mongodb+srv://username:password@eugenecluster-incta.mongodb.net/customers-
    libraryservice", () => {
32.   console.log("Connected to MongoDB Cluster - Customers database..");
33. });
```

Τα routes των customers επεξεργάζονται με τον ίδιο τρόπο όπως και στο books service, με διαφορετική όμως σημασιολογία. Δηλώνονται δηλαδή τέσσερις διαδρομές για τις τέσσερις διαφορετικές λειτουργίες που θα μπορεί να εκτελέσει το service:

- Η διαδρομή `localhost:3050/customers`, η οποία έχει αντιστοιχηθεί με ένα GET request που επιστρέφει όλους τους πελάτες της βιβλιοθήκης, αλλά και με ένα POST request με το οποίο δημιουργείται ένας καινούριος πελάτης στη βάση δεδομένων
- Η διαδρομή `localhost:3050/customers/{customer_id}`, η οποία έχει αντιστοιχηθεί με ένα GET request που επιστρέφει τις πληροφορίες για 1 πελάτη δηλώνοντας τον αναγνωριστικό αριθμό του, αλλά και με ένα DELETE request που διαγράφει έναν πελάτη από τη βάση δεδομένων.

Οπότε το αρχείο `customersRoutes.js` θα έχει το παρακάτω περιεχόμενο:

```
1. // Load dependencies
2. const express = require('express');
3. const router = express.Router();
4. const CustomersModel = require("./customersModel");
5.
6. // Get all customers
7. router.get("/", (req, res, next) => {
8.   CustomersModel.find().then((customers) => {
9.     res.status(200).json(
10.      customers
11.    )
12.   }).catch((err) => {
13.     if (err){
14.       throw err;
15.     }
16.   })
17. });
18.
19. // Get one customer by id
20. router.get("/:id", (req, res, next) => {
21.   CustomersModel.findById(req.params.id).then((customer) => {
22.     if(customer){
23.       res.json(customer);
24.     } else {
25.       res.sendStatus(404);
26.     }
27.   }).catch((err) => {
28.     if (err){
29.       throw err;
30.     }
31.   })
32. });
33.
34. // Create a new customer
35. router.post("/", (req, res, next) => {
36.   var newCustomer = {
37.     name: req.body.name,
38.     age: req.body.age,
39.     address: req.body.address,
40.     phone: req.body.phone
41.   }
42.   var customer = new CustomersModel(newCustomer);
43.   customer.save().catch((err) => {
44.     if (err){
45.       throw err;
46.     }
47.   });
48.   res.status(201).json({
49.     status: 201,
50.     message: "New customer: " + newCustomer.name + " added",
51.   });
52. });
53.
54. // Delete a customer
55. router.delete("/:id", (req, res, next) => {
```

```

56.     CustomersModel.findByIdAndDelete(req.params.id).then((customer) => {
57.         res.status(200).json({
58.             status: 200,
59.             message: "Customer " + customer.name + " deleted"
60.         }).catch((err) => {
61.             if (err){
62.                 throw err;
63.             }
64.         })
65.     })
66. });
67.
68. module.exports = router;

```

Κατόπιν συμπληρώνεται το αρχείο `customers.js` με τις γραμμές που του λείπουν για την ενσωμάτωση των routes:

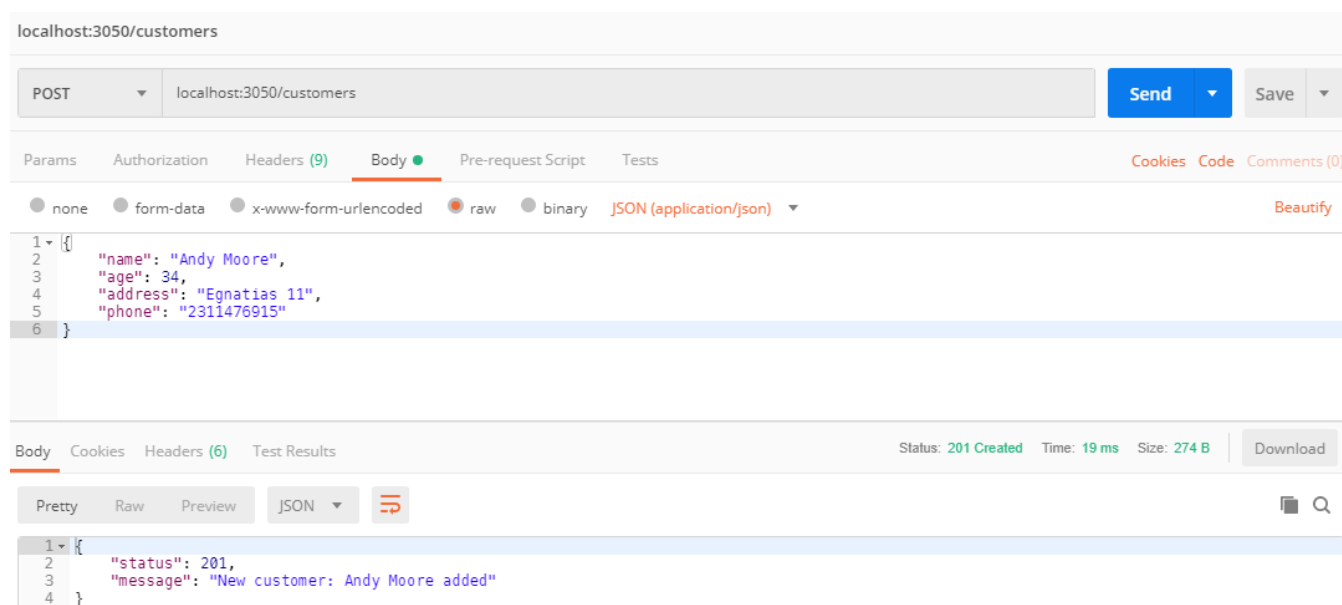
```

6. // Load Customer Routes
7. const customersRoutes = require("./customersRoutes");

16. // Routes which are handled by the handlers
17. app.use('/customers', customersRoutes);

```

Το service αρχίζει να τρέχει πληκτρολογώντας την εντολή `nodemon customers.js` και πλέον μπορούν να δημιουργηθούν τα επιθυμητά requests προς τον server. Δηλαδή χρησιμοποιείται το Postman για τη δημιουργία πελατών αλλά και για όποια άλλη λειτουργία κριθεί απαραίτητο να πραγματοποιηθεί.



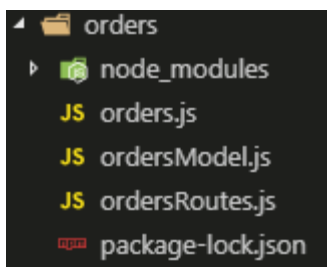
Εικόνα 9. Παράδειγμα δημιουργίας ενός πελάτη με το Postman



## Δημιουργία 3<sup>ου</sup> Microservice και Σύνδεση με τα Άλλα Microservices

Το 3<sup>ο</sup> service είναι αυτό των παραγγελιών (orders). Κάθε χρήστης θα μπορεί να παραγγέλνει ένα βιβλίο. Και εδώ χρειάζεται η εγκατάσταση των πακέτων που εγκαταστάθηκαν στα προηγούμενα service, με μόνη διαφορά την εγκατάσταση ενός επιπλέον πακέτου, του `axios`. Το `axios` είναι μια βιβλιοθήκη βασισμένη σε JavaScript Promises, η οποία επιτρέπει την κλήση web services μέσα στο service που θα χρησιμοποιηθεί.

Μετά την εγκατάσταση δημιουργούνται μέσα στο φάκελο `orders` τα αρχεία `orders.js`, `ordersModel.js` και `ordersRoutes.js`.



Εικόνα 10. Δομή φακέλου του orders microservice

Το αρχείο `orders.js` θα είναι κατ'αναλογία με τα αρχεία `books.js` και `customers.js`. Εδώ θα συμπεριληφθούν και τα routes παρόλο που δεν έχουν οριστεί ακόμα.

```
1. // Load dependencies
2. const express = require("express");
3. const app = express();
4. const morgan = require('morgan');
5. const bodyParser = require("body-parser");
6. // Load Order Routes
7. const ordersRoutes = require("./ordersRoutes");
8.
9. // Incoming request data arrive in JSON format
10. app.use(bodyParser.urlencoded({extended: false}));
11. app.use(bodyParser.json());
12.
13. // Log requests before handling them
14. app.use(morgan('dev'));
15.
16. // Routes which are handled by the handlers
17. app.use('/orders', ordersRoutes);
18.
19. // Orders Service listens
20. const port = 3100;
21. app.listen(port, () => {
22.   console.log("\n\nOrders service started..");
```

```
23. });
```

Σε δεύτερο στάδιο δημιουργείται μια καινούρια βάση δεδομένων για τα orders, αλλά και το orders Model. Στο Schema δηλώνονται οι ιδιότητες του καθενός order, οι οποίες είναι ο αναγνωριστικός αριθμός πελάτη, ο αναγνωριστικός αριθμός βιβλίου, η ημερομηνία παραγγελίας και η ημερομηνία παράδοσης της παραγγελίας.

```
1. // Load dependencies
2. const mongoose = require("mongoose");
3.
4. // Define the Order Schema
5. var OrderSchema = mongoose.Schema({
6.   customerId: {
7.     type: mongoose.SchemaTypes.ObjectId,
8.     required: true
9.   },
10.  bookId: {
11.    type: mongoose.SchemaTypes.ObjectId,
12.    required: true
13.  },
14.  orderDate: {
15.    type: Date,
16.    required: true
17.  },
18.  deliveryDate: {
19.    type: Date,
20.    required: true
21.  }
22. });
23.
24. // Create the Order Model
25. const Order = mongoose.model("Order", OrderSchema, 'orders');
26.
27. // Provide the Order Model to the order service
28. module.exports = Order;
29.
30. // Connect to MongoDB database
31. mongoose.connect("mongodb+srv://username:password@eugenecluster-incta.mongodb.net/orders-
    libraryservice", () => {
32.   console.log("Connected to MongoDB Cluster - Orders database..");
33. });
```

Τα routes των orders είναι δομημένα ως εξής:

- Η διαδρομή localhost:3100/orders, η οποία έχει αντιστοιχηθεί με ένα GET request που επιστρέφει όλες τις παραγγελίες, αλλά και με ένα POST request με το οποίο δημιουργείται μία καινούρια παραγγελία στη βάση δεδομένων
- Η διαδρομή localhost:3100/orders/{order\_id}, έχει αντιστοιχηθεί με ένα GET

request που επιστρέφει τις πληροφορίες για 1 παραγγελία δηλώνοντας τον αναγνωριστικό αριθμό της. Στις πληροφορίες αυτές θα βρίσκεται ο αναγνωριστικός αριθμός του πελάτη και του βιβλίου που παρήγγειλε. Όταν όμως δει κανείς το response body δεν θα καταλάβει ποιον πελάτη και ποιο βιβλίο αφορά η συγκεκριμένη παραγγελία, αφού θα βλέπει μόνο κωδικούς. Οπότε θα ήταν καλό να μετατραπούν αυτοί οι κωδικοί στο όνομα του πελάτη και το όνομα του βιβλίου αντίστοιχα ώστε το response body να είναι πιο ευανάγνωστο. Για να γίνει αυτό θα χρειαστεί να κληθεί το route `localhost:3050/customers/{customer_id}` ώστε να επιστραφεί το όνομα του πελάτη, όπως και το route `localhost:3000/books/{book_id}` ώστε να επιστραφεί το όνομα του βιβλίου. Τα 2 αυτά ονόματα αποθηκεύονται σε ένα JavaScript object το οποίο θα είναι και το response body του route `localhost:3100/orders/{order_id}` όταν γίνεται ένα GET request. Μέσα σε αυτό το route δηλαδή θα γίνεται κλήση 2 άλλων routes από τα 2 υπόλοιπα microservices. Επίσης το route αυτό έχει αντιστοιχηθεί με ένα DELETE request που διαγράφει μία παραγγελία από τη βάση δεδομένων.

Οπότε το αρχείο `ordersRoutes.js` θα είναι δομημένο με τον παρακάτω τρόπο:

```
1. // Load dependencies
2. const express = require('express');
3. const router = express.Router();
4. const OrdersModel = require("./ordersModel");
5. const mongoose = require("mongoose");
6. const axios = require("axios");
7.
8. // Get all orders
9. router.get("/", (req, res, next) => {
10.   OrdersModel.find().then((orders) => {
11.     res.status(200).json(
12.       orders
13.     )
14.   }).catch((err) => {
15.     if (err){
16.       throw err;
17.     }
18.   })
19. });
20.
21. // Get one order by id
22. router.get("/:id", (req, res, next) => {
23.   OrdersModel.findById(req.params.id).then((order) => {
24.     if (order){
25.       axios.get("http://localhost:3050/customers/" + order.customerId).then(response =>
{
```

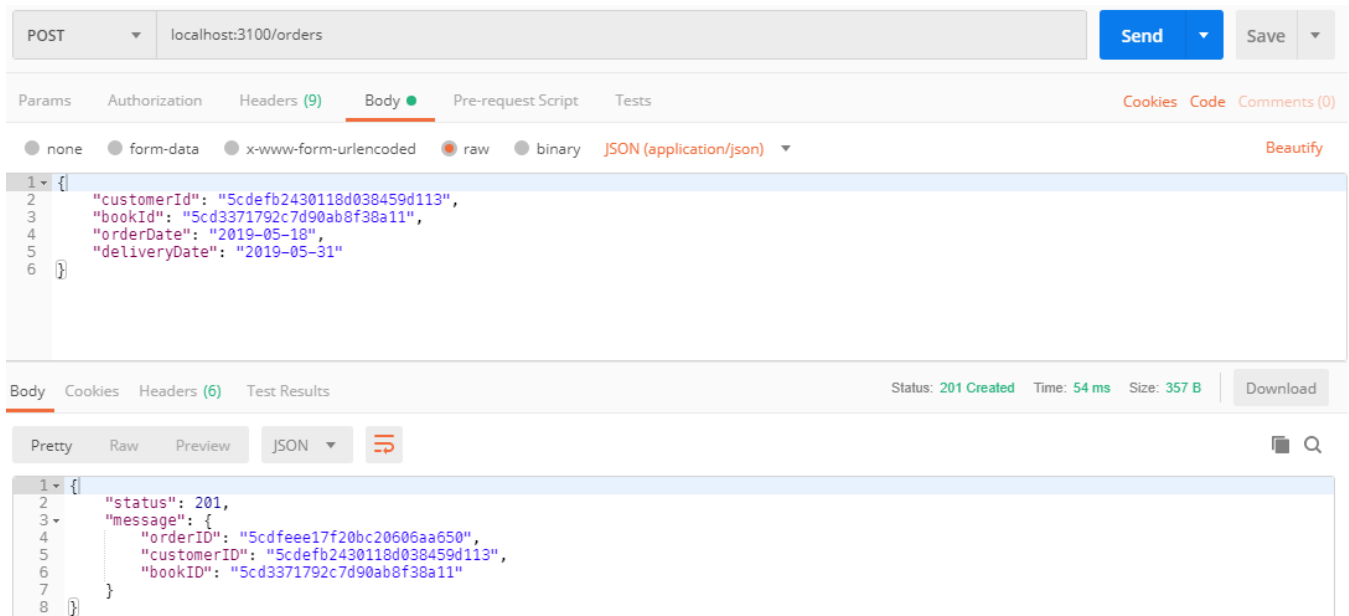
```

26.         var orderResponse = {customerName: response.data.name, bookTitle: ""}
27.         axios.get("http://localhost:3000/books/" + order.bookId).then(response => {
28.             orderResponse.bookTitle = response.data.title
29.             res.send(orderResponse)
30.         })
31.     }).catch(error => {
32.         throw error;
33.     })
34. } else {
35.     res.sendStatus(404);
36. }
37. })
38. });
39.
40. // Create a new order
41. router.post("/", (req, res, next) => {
42.     var newOrder = {
43.         customerId: mongoose.Types.ObjectId(req.body.customerId),
44.         bookId: mongoose.Types.ObjectId(req.body.bookId),
45.         orderDate: req.body.orderDate,
46.         deliveryDate: req.body.deliveryDate
47.     }
48.     var order = new OrdersModel(newOrder);
49.     order.save().catch((err) => {
50.         if (err){
51.             throw err;
52.         }
53.     });
54.     res.status(201).json({
55.         status: 201,
56.         message: {
57.             orderID: order._id,
58.             customerID: newOrder.customerId,
59.             bookID: newOrder.bookId
60.         }
61.     });
62. });
63.
64. // Delete an order
65. router.delete("/:id", (req, res, next) => {
66.     OrdersModel.findByIdAndDelete(req.params.id).then((order) => {
67.         res.status(200).json({
68.             status: 200,
69.             message: "Order deleted"
70.         }).catch((err) => {
71.             if (err){
72.                 throw err;
73.             }
74.         })
75.     })
76. });
77.
78. module.exports = router;

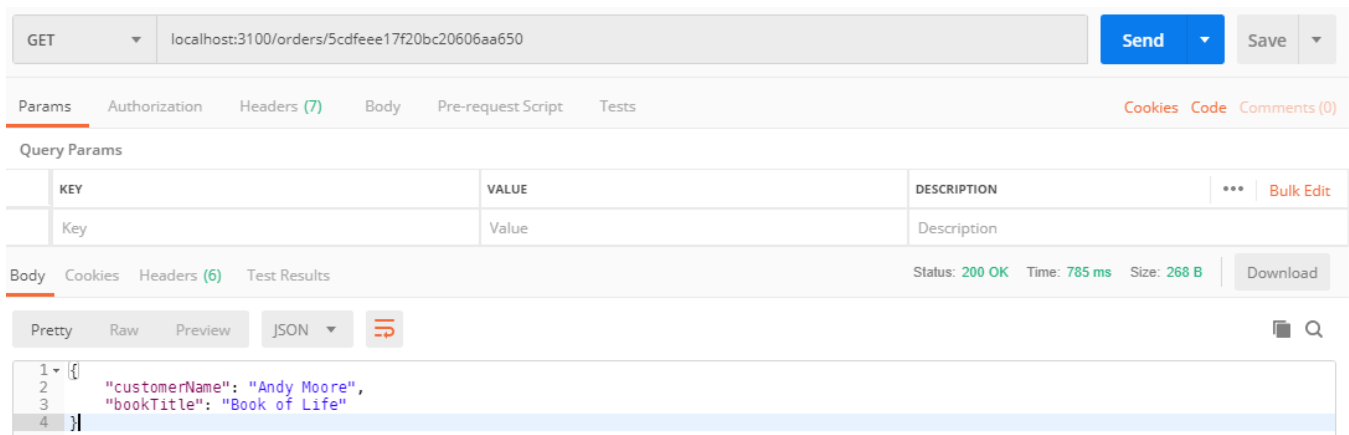
```

To service αρχίζει να τρέχει πληκτρολογώντας την εντολή `nodemon orders.js` στο command line. Πρέπει όμως να τρέχουν παράλληλα και τα 2 άλλα services, οπότε εκκινούνται

και αυτά. Χρησιμοποιώντας το Postman θα γίνει η δημιουργία μιας παραγγελίας, καθώς και η επιστροφή αυτής της παραγγελίας με βάση τον αναγνωριστικό της αριθμό.



Εικόνα 11. Παράδειγμα δημιουργίας μιας παραγγελίας με το Postman



Εικόνα 12. Ανάκτηση μιας παραγγελίας με το Postman

## Βιβλιογραφία

- [1] Bhamare D, Samaka M, Erbad A., Jain R., Gupta L., 2018. *Exploring microservices for enhancing internet QoS*. Trans Emerging Tel Tech.
- [2] Taibi D., Lenarduzzi V., Pahl C., 2017. *Processes, Motivations, and Issues for Migrating to Microservices Architectures: An Empirical Investigation*. IEEE Cloud Computing.