|  |  |
| --- | --- |
| **Image result for ÏÎ±Î½ÎµÏÎ¹ÏÏÎ®Î¼Î¹Î¿ Î¸ÎµÏÏÎ±Î»Î¯Î±Ï** | **Πανεπιστήμιο Θεσσαλίας**  **Πρόγραμμα Σπουδών**  **Τμήματος Μηχανικών Πληροφορικής Τ.Ε. Λάρισας** |

**REST API για διαχείριση**

**Parking**

**Πτυχιακή Εργασία**

**Ρασκόπουλος Κωνσταντίνος (ΑΜ: 4416050)**

**Επιβλέπων: *< Φώτιος Κόκκορας>***

**ΛΑΡΙΣΑ 2023**

*Εγώ ο Ρασκόπουλος Κωνσταντίνος, δηλώνω υπεύθυνα ότι η παρούσα Πτυχιακή Εργασία με τίτλο* «*REST API για Διαχείριση Parking*» *είναι δική μου και βεβαιώνω ότι:*

* *Σε όσες περιπτώσεις έχω συμβουλευτεί δημοσιευμένη εργασία τρίτων, αυτό επισημαίνεται με σχετική αναφορά στα επίμαχα σημεία.*
* *Σε όσες περιπτώσεις μεταφέρω λόγια τρίτων, αυτό επισημαίνεται με σχετική αναφορά στα επίμαχα σημεία. Με εξαίρεση τέτοιες περιπτώσεις, το υπόλοιπο κείμενο της πτυχιακής αποτελεί δική μου δουλειά.*
* *Αναφέρω ρητά όλες τις πηγές βοήθειας που χρησιμοποίησα.*
* *Σε περιπτώσεις που τμήματα της παρούσας πτυχιακής έγιναν από κοινού με τρίτους, αναφέρω ρητά ποια είναι η δική μου συνεισφορά και ποια των τρίτων.*
* *Γνωρίζω πως η λογοκλοπή αποτελεί σοβαρότατο παράπτωμα και είμαι ενήμερος(-η) για την επέλευση των νόμιμων συνεπειών»*

Ρασκόπουλος Κωνσταντίνος

*Ο φοιτητής εντάχθηκε αυτοδίκαια στο Πανεπιστήμιο Θεσσαλίας, σύμφωνα με την παρ. 1 του άρθρου 6 του Ν.4589/2019 (ΦΕΚ 13/Α'/29.01.2019). Η εκπαιδευτική λειτουργία του ανωτέρου προγράμματος σπουδών συνεχίζεται μεταβατικά σύμφωνα με την παρ. 2 του άρθρου 6 του Ν.4589/2019 (ΦΕΚ 13/Α'/29.01.2019).*

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή

**Τόπος**:

**Ημερομηνία**:

**ΕΠΙΤΡΟΠΗ ΑΞΙΟΛΟΓΗΣΗΣ**



**Περίληψη**

Το REST API που αναπτύχθηκε για τη Διαχείριση Parking αποτελεί ένα ευέλικτο και αποτελεσματικό σύστημα που επιτρέπει τη διαχείριση διαφόρων πτυχών σχετικά με το Parking. Το API παρέχει δυνατότητες για τον καθορισμό και την παρακολούθηση παρκινγκ, ζωνών και θέσεων.

Αρχικά, το API επιτρέπει τη δήλωση νέων παρκινγκ, με τη δυνατότητα καθορισμού ονόματος για το Parking. Για κάθε ζώνη μπορούν να καθοριστούν όνομα, τύπος και ποσότητα θέσεων. Επιπλέον, οι χρήστες μπορούν να παρακολουθούν την κατάσταση των θέσεων, προσδιορίζοντας το όνομα, τον τύπο και το αν είναι κατειλημμένες. Η δυνατότητα καθορισμού κλιμακούμενου τιμοκαταλόγου για κάθε ζώνη επιτρέπει την ευέλικτη διαχείριση των τιμών σε διάφορες περιοχές του Parking.

Το API επίσης παρέχει τη δυνατότητα ελέγχου της συνολικής χωρητικότητας του Parking σε ποσοστιαία βάση, επιτρέποντας τη γρήγορη αξιολόγηση της χρήσης του Parking. Το σύστημα προσφέρει εύκολη ενσωμάτωση με άλλες εφαρμογές και πλατφόρμες, επιτρέποντας την αυτοματοποιημένη διαχείριση και παρακολούθηση του Parking. Η αρχιτεκτονική του API είναι σχεδιασμένη με γνώμονα την απλότητα και την ευελιξία, καθιστώντας το κατάλληλο για διάφορες εφαρμογές και σενάρια χρήσης στον τομέα της διαχείρισης Parking. Τέλος δίνεται ένα αρχείο Excel με την διαδικασία υλοποίησης κώδικα.

Περιεχόμενα

[1 Εισαγωγή 7](#_Toc155554618)

[2 Web εφαρμογή 8](#_Toc155554619)

[2.1 Front – End 8](#_Toc155554620)

[2.2 Database 9](#_Toc155554621)

[2.3 Back – End 10](#_Toc155554622)

[3 REST API 12](#_Toc155554623)

[4 JSON 15](#_Toc155554624)

[5 Spring Boot 16](#_Toc155554625)

[6 Λειτουργικότητα Parking REST API 24](#_Toc155554626)

[6.1 Postman 24](#_Toc155554627)

[6.2 Post, Get και PUT 25](#_Toc155554628)

[7 Λεπτομέρειες και βιβλιοθήκες 40](#_Toc155554629)

[7.1 lombock dependency 41](#_Toc155554630)

[7.2 Testing 41](#_Toc155554631)

[8 SCRUM AGILE 45](#_Toc155554632)

[9 Συμπεράσματα 46](#_Toc155554633)

[Βιβλιογραφία 46](#_Toc155554634)

## 1 Εισαγωγή

Στην εποχή της ψηφιοποίησης και της εκρηκτικής ανάπτυξης του διαδικτύου, τα REST APIs (Representational State Transfer Application Programming Interfaces) αναδεικνύονται ως ισχυρό εργαλείο για την ανάπτυξη δυναμικών web εφαρμογών. Τα REST APIs επιτρέπουν την αποτελεσματική επικοινωνία μεταξύ διαφόρων συστημάτων και προσφέρουν ευελιξία στον τρόπο που διαχειριζόμαστε τα δεδομένα και τις λειτουργίες μέσω του διαδικτύου.

Τα REST APIs συμβάλλουν στη δημιουργία εξαιρετικά προσαρμοστικών web εφαρμογών, επιτρέποντας τη διαμόρφωση και τον έλεγχο του τρόπου που αλληλεπιδρούν οι χρήστες με τις υπηρεσίες. Η δημιουργία custom λειτουργιών γίνεται πλέον πραγματικότητα, καθώς τα REST APIs επιτρέπουν την εύκολη προσθήκη, αναβάθμιση και προσαρμογή λειτουργιών, χωρίς την ανάγκη για σημαντικές αλλαγές στην υποκείμενη υποδομή.

Η δυνατότητα προσαρμογής των web εφαρμογών επιτρέπει στους προγραμματιστές να δημιουργήσουν προσαρμοσμένες λύσεις που ανταποκρίνονται συγκεκριμένα στις ανάγκες των χρηστών. Αυτό επιτρέπει τη δημιουργία εντυπωσιακών και εξειδικευμένων λειτουργιών, προσφέροντας μια εκτεταμένη γκάμα δυνατοτήτων για τη βελτίωση της χρήσης και της εμπειρίας του χρήστη.

Σε αντίθεση με τις παραδοσιακές μεθόδους ανάπτυξης λογισμικού, τα REST APIs επιτρέπουν τη δημιουργία εφαρμογών που λειτουργούν απροβλημάτιστα σε πολλαπλές πλατφόρμες και συσκευές. Η δυνατότητα ανοιχτού κώδικα επίσης διευκολύνει τη συνεργασία μεταξύ διαφορετικών ομάδων προγραμματιστών, ενισχύοντας την κοινότητα και την ανάπτυξη καινοτόμων λύσεων.

Παρακάτω εξετάζονται οι τομείς και οι τρόποι με τους οποίους οι Web εφαρμογές έχουν ενσωματωθεί στην καθημερινή ζωή του ανθρώπου, πώς επικοινωνούν αυτές οι εφαρμογές και με ποια εργαλεία μπορεί να αναπτυχθεί ένα τέτοιο σύστημα.

## 2 Web εφαρμογή

Web εφαρμογή θεωρούμε μια ιστοσελίδα που λειτουργεί μέσω του παγκόσμιου ιστού (Διαδίκτυο) και προσφέρει διαδραστική εμπειρία στον χρήστη, επιτρέποντας την πρόσβαση, την αναζήτηση, και την αλληλεπίδραση με περιεχόμενο ή υπηρεσίες που Παρέχονται μέσω του παγκόσμιου ιστού. Υπάρχουν πολλοί τρόποι και πολλά εργαλεία με τα οποία μπορεί να γίνει ανάπτυξη μίας web εφαρμογής, ακολουθώντας ένα μοτίβο **Front – End**, **Back – End**, **Database**.

### 2.1 Front – End

Το Front-End αντιπροσωπεύει το "μπροστινό μέρος" μιας ιστοσελίδας ή μιας εφαρμογής που βλέπει ο χρήστης. Είναι αυτό που συναντάτε όταν επισκέπτεστε μια ιστοσελίδα, όπως το στυλ, τα χρώματα, οι εικόνες, και τα κουμπιά που αλληλεπιδρούν με εσάς.

Ένα απλό παράδειγμα Front-End μπορεί να είναι μια ιστοσελίδα ειδήσεων. Όταν επισκέπτεστε αυτήν τη σελίδα, βλέπετε τα άρθρα, τις εικόνες, και το μενού περιήγησης. Όλα αυτά τα στοιχεία ανήκουν στο Front-End, καθώς απευθύνονται απευθείας στον χρήστη και καθορίζουν την αισθητική και την εμπειρία του χρήστη.

Το Front-End επιτρέπει στον χρήστη να διαδραματίζει με την εφαρμογή, να περιηγείται σε διάφορες σελίδες, και να αλληλεπιδρά με το περιεχόμενο. Όλες οι γραφικές και διαδραστικές συστατικές που βλέπετε στην οθόνη σας ανήκουν στο Front-End.

Υπάρχουν πολλά εργαλεία και τρόποι ανάπτυξης Front-End. Η επιλογή εξαρτάται από τις ανάγκες σας, την εμπειρία σας, και τον τρόπο που θέλετε να δημιουργήσετε το Front-End. Εδώ είναι μερικά δημοφιλή εργαλεία και τρόποι:

1. HTML, CSS, JavaScript: Είναι οι βασικές γλώσσες για το Front-End ανάπτυξη. HTML χρησιμοποιείται για τη δημιουργία της δομής της σελίδας, το CSS για το στυλ και την εμφάνιση, ενώ το JavaScript για τη διαδραστικότητα.

2. React: Μια βιβλιοθήκη JavaScript για τη δημιουργία των χρήστικών διεπαφών. Αναπτύσσεται από το Facebook και είναι εξαιρετικά δημοφιλές για τη δημιουργία ενιαίων και αποδοτικών χρηστικών διεπαφών.

3. Angular: Ένα πλήρες πλαίσιο ανάπτυξης, αναπτύσσεται από την Google. Παρέχει ένα πλήρες σετ εργαλείων για την ανάπτυξη δυναμικών ενισχυμένων εφαρμογών.

4. Bootstrap: Ένα πλαίσιο CSS που παρέχει προκαθορισμένα στυλ και στοιχεία σχεδιασμού για γρήγορη και αποτελεσματική ανάπτυξη.

Αυτά είναι μόνο μερικά από τα πολλά εργαλεία που μπορείτε να χρησιμοποιήσετε. Η επιλογή εξαρτάται από τις ανάγκες του έργου σας και την προσωπική σας προτίμηση. Η ποικιλία των τεχνολογιών Front-End προσφέρει απεριόριστες δυνατότητες προσαρμογής και επιλογής για τη δημιουργία ισχυρών ιστοσελίδων και εφαρμογών. Κάθε τεχνολογία έχει τα πλεονεκτήματά της, και η επιλογή εξαρτάται από τις ανάγκες και τους στόχους του σχεδιαστή/προγραμματιστή.

### 2.2 Database

Μια βάση δεδομένων χρησιμοποιείται σε μια ιστοσελίδα ή web εφαρμογή για να αποθηκεύει, ανακτά, ενημερώνει και διαχειρίζεται τα δεδομένα που Χρησιμοποιούνται από την εφαρμογή. Αυτό είναι σημαντικό γιατί οι ιστοσελίδες και οι εφαρμογές χρειάζονται τρόπους για να αποθηκεύουν και να ανακτούν δεδομένα για να παρέχουν εξατομικευμένες και διαδραστικές εμπειρίες στους χρήστες.

Συγκεκριμένα, η βάση δεδομένων συνδέεται με το back-end της ιστοσελίδας ή της εφαρμογής. Κατά την υλοποίηση, οι προγραμματιστές χρησιμοποιούν γλώσσες προγραμματισμού και εργαλεία που επιτρέπουν τη σύνδεση του back-end με τη βάση δεδομένων. Αυτό τους επιτρέπει να ανακτούν και να αποθηκεύουν δεδομένα ανάλογα με τις ανάγκες της εφαρμογής.

Παράδειγμα:

Έστω ότι έχουμε μια ιστοσελίδα καταστήματος. Η βάση δεδομένων μπορεί να περιλαμβάνει πίνακες όπως "Προϊόντα", "Πελάτες", και "Παραγγελίες". Όταν ένας πελάτης πραγματοποιεί μια παραγγελία, τα δεδομένα της παραγγελίας αποθηκεύονται στον πίνακα "Παραγγελίες". Κατόπιν, όταν ο πελάτης συνδέεται στον λογαριασμό του, η ιστοσελίδα χρησιμοποιεί τη βάση δεδομένων για να ανακτήσει τις προηγούμενες παραγγελίες του από τον πίνακα "Παραγγελίες" και να του προσφέρει εξατομικευμένες πληροφορίες.

Κάποια παραδείγματα εργαλείων βάσεων δεδομένων είναι :

1. MySQL:

   - Χαρακτηριστικά:

   - Είναι μια δωρεάν και ανοικτού κώδικα σχέση βάσης δεδομένων.

   - Χρησιμοποιείται συχνά σε μικρομεσαίες εφαρμογές και ιστοσελίδες.

   - Υποστηρίζεται από μια ενεργή κοινότητα και έχει ευρεία διανομή.

   - Χρήση:

   - Κατάλληλο για μικρότερα έργα ή εφαρμογές που δεν απαιτούν πολύπλοκες λειτουργίες.

2. SQL Workbench (Συνοδευτικό Εργαλείο):

   - Χαρακτηριστικά:

   - Δεν είναι βάση δεδομένων αλλά εργαλείο διαχείρισης βάσεων δεδομένων.

   - Χρησιμοποιείται για τον σχεδιασμό, τη διαχείριση και την επικοινωνία με διάφορες βάσεις δεδομένων.

   - Χρήση:

   - Κυρίως χρησιμοποιείται ως εργαλείο ανάπτυξης και διαχείρισης για διάφορες βάσεις δεδομένων.

3. Oracle:

   - Χαρακτηριστικά:

   - Είναι μια εμπορική βάση δεδομένων με πλούσιες λειτουργίες και υψηλή απόδοση.

   - Χρησιμοποιείται σε μεγάλες επιχειρήσεις και εφαρμογές με σημαντικές απαιτήσεις ασφάλειας και επεξεργασίας δεδομένων.

   - Χρήση:

   - Κατάλληλο για εφαρμογές μεγάλης κλίμακας που απαιτούν προηγμένες λειτουργίες και αξιοπιστία.

Επιλέγετε τη βάση δεδομένων ανάλογα με τις απαιτήσεις του έργου. Το MySQL είναι κατάλληλο για μικρότερες εφαρμογές, το SQL Workbench ως εργαλείο διαχείρισης, ενώ η Oracle είναι κατάλληλη για μεγάλες και πολύπλοκες εφαρμογές.

### 2.3 Back – End

Το "back-end" αναφέρεται στον πίσω μέρος μιας ιστοσελίδας ή μιας εφαρμογής, όπου εκτελούνται οι διάφορες λειτουργίες και γίνεται η διαχείριση των δεδομένων. Είναι υπεύθυνο για την επεξεργασία των αιτημάτων των χρηστών, την αλληλεπίδραση με τη βάση δεδομένων, την επιχειρησιακή λογική και άλλες λειτουργίες που εξασφαλίζουν τη σωστή λειτουργία της εφαρμογής.

Ορισμένα δημοφιλή πλαίσια εργασίας (frameworks) για back-end ανάπτυξη είναι τα παρακάτω:

1. Django:

- Περιγραφή:

- Είναι ένα πλαίσιο ανάπτυξης που χρησιμοποιείται κυρίως για τη γλώσσα προγραμματισμού Python.

- Παράδειγμα:

- Ένα blog σύστημα όπου οι χρήστες μπορούν να δημοσιεύουν άρθρα, να σχολιάζουν, και το Django να διαχειρίζεται την αποθήκευση, την ανάκτηση και την εμφάνιση των δεδομένων.

2. Express.js (για Node.js):

- Περιγραφή:

- Είναι ένα ελαφρύ και ευέλικτο πλαίσιο για τη δημιουργία back-end εφαρμογών με χρήση της γλώσσας JavaScript.

- Παράδειγμα:

- Ένα απλό σύστημα διαχείρισης εργασιών, όπου οι χρήστες μπορούν να προσθέτουν, να ενημερώνουν και να διαγράφουν εργασίες.

3. Ruby on Rails:

- Περιγραφή:

- Είναι ένα πλαίσιο ανάπτυξης για τη γλώσσα προγραμματισμού Ruby, γνωστό για την αρχιτεκτονική του "convention over configuration".

- Παράδειγμα:

- Ένα ηλεκτρονικό κατάστημα όπου οι διαχειριστές μπορούν να προσθέτουν προϊόντα, να διαχειρίζονται τις παραγγελίες και το Ruby on Rails να εξυπηρετεί τα δεδομένα.

4. Spring Boot :

- Περιγραφή: Το Spring Boot είναι ένα πλαίσιο ανάπτυξης για τη γλώσσα προγραμματισμού **Java**. Είναι εξαιρετικά δημοφιλές για τη δημιουργία ευέλικτων και αποδοτικών back-end εφαρμογών. Τα βασικά χαρακτηριστικά του Spring Boot περιλαμβάνουν:

1. Ευκολία Έναρξης (Opinionated Defaults):

   - Παρέχει προεπιλεγμένες ρυθμίσεις και δομές καταλόγων για γρήγορη έναρξη ενός έργου.

2. Αυτοδιαμόρφωση (Auto-Configuration):

   - Αυτόματη διαμόρφωση της εφαρμογής βάσει των περιβαλλοντικών συνθηκών, χωρίς την ανάγκη για πολύπλοκες ρυθμίσεις.

3. Ενσωματωμένος Εξυπηρετητής (Embedded Server):

   - Περιλαμβάνει έναν ενσωματωμένο εξυπηρετητή (π.χ. Tomcat), εξοικονομώντας την ανάγκη για εξωτερικούς εξυπηρετητές.

4. Διαχείριση Εξαρτημάτων (Dependency Management):

   - Ευκολία στη διαχείριση των εξαρτημάτων της εφαρμογής με το εργαλείο του Apache Maven.

5. Πλούσιο Οικοσύστημα (Rich Ecosystem):

   - Παρέχει πληθώρα επεκτάσεων και εργαλείων για την ανάπτυξη, τη διαχείριση και τη συντήρηση των εφαρμογών.

Παράδειγμα:

Ένα σύστημα διαχείρισης χρηστών όπου οι χρήστες μπορούν να εγγραφούν, να συνδεθούν, να διαχειρίζονται το προφίλ τους, και το Spring Boot να χειρίζεται την επικοινωνία με τη βάση δεδομένων και τη λογική επιχείρησης.

Αυτά τα πλαίσια εργασίας βοηθούν τους προγραμματιστές να οργανώσουν τον κώδικα τους, να διαχειριστούν τις διαδικασίες δρομολόγησης (routing), να συνδέσουν την εφαρμογή τους με βάσεις δεδομένων, και να ανταποκρίνονται στα αιτήματα των χρηστών.

## 3 REST API

Ένα REST API (Representational State Transfer Application Programming Interface) είναι ένας τύπος διεπαφής προγραμματισμού εφαρμογών που ακολουθεί τις αρχές του αρχιτεκτονικού στυλ REST. Το REST είναι ένα σύνολο απλών κανόνων για τη σχεδίαση διακριτικών, ευέλικτων και επεκτάσιμων συστημάτων.

Τα βασικά χαρακτηριστικά ενός REST API περιλαμβάνουν:

1. Πόροι (Resources): Οι πόροι είναι αντικείμενα ή υπηρεσίες που διατίθενται μέσω του API. Κάθε πόρος έχει μια μοναδική ταυτότητα (URI).

2. Αναπαράσταση Κατάστασης (State Representation): Οι πόροι αναπαρίστανται με καθορισμένο τρόπο, συχνά με χρήση διαδεδομένων μορφών αναπαράστασης όπως JSON ή XML.

3. Οperations (Λειτουργίες): Οι επιτρεπτές λειτουργίες που μπορούν να εκτελεστούν στους πόρους περιγράφονται μέσω των HTTP μεθόδων, όπως GET, POST, PUT και DELETE.

4. Αυτονομία (Statelessness): Κάθε αίτηση από τον πελάτη πρέπει να περιέχει όλες τις πληροφορίες που απαιτούνται για την κατανόηση και εκτέλεση του αιτήματος. Καμία πληροφορία κατάστασης δεν αποθηκεύεται στον διακομιστή μεταξύ αιτήσεων.

5. Υπερμεταφορά (Uniform Interface): Παρέχει μια ενιαία και καθολική διεπαφή για την αλληλεπίδραση με τους πόρους.

**

*Εικόνα 1. REST API παράδειγμα*

Ένα παράδειγμα απλού REST API μπορεί να είναι ένα σύστημα διαχείρισης εργασιών (To-Do List). Σε αυτό το σενάριο, οι πόροι θα είναι οι εργασίες, και η διαδικασία διαχείρισης των εργασιών θα γίνεται μέσω HTTP αιτήσεων. Ας δούμε πώς θα μπορούσε να λειτουργεί:

1. Πόρος Εργασίες (Tasks Resource):

- URI: `/tasks`

- Λειτουργίες:

- `**GET**`: Επιστρέφει τη λίστα με όλες τις εργασίες.

- `**POST**`: Δημιουργεί μια νέα εργασία.

2. Πόρος Συγκεκριμένης Εργασίας (Individual Task Resource):

- URI: `/tasks/{taskID}`

- Λειτουργίες:

- `**GET**`: Επιστρέφει τις λεπτομέρειες μιας συγκεκριμένης εργασίας.

- `**PUT**`: Ενημερώνει τις πληροφορίες μιας εργασίας.

- `**DELETE**`: Διαγράφει μια εργασία.

Στο παράδειγμα αυτό, οι πόροι είναι οι εργασίες, και μπορούμε να αλληλεπιδρούμε μαζί τους χρησιμοποιώντας τις παραπάνω HTTP μεθόδους. Για παράδειγμα, με ένα `GET /tasks`, μπορούμε να λάβουμε τη λίστα με όλες τις εργασίες, ενώ με ένα `POST /tasks` μπορούμε να δημιουργήσουμε μια νέα εργασία.

Η αναπαράσταση των εργασιών μπορεί να είναι σε μορφή **JSON** ή άλλης κατάλληλης μορφής δεδομένων. Αυτό είναι ένα απλό παράδειγμα πώς ένα REST API μπορεί να χρησιμοποιηθεί για τη διαχείριση εργασιών σε μια εφαρμογή.

## 4 JSON

Το JSON (JavaScript Object Notation) είναι ένα ελαφρύ διαδικτυακό πρότυπο μορφοποίησης δεδομένων. Είναι εύκολο στην κατανόηση από τους ανθρώπους και εύχρηστο για τις υπολογιστικές συσκευές. Χρησιμοποιείται ευρέως στον προγραμματισμό και στον διαδικτυακό κόσμο για τη μεταφορά δεδομένων.

Οι βασικές ιδιότητες του JSON περιλαμβάνουν:

1. Αναγνωρισιμότητα ανθρώπινου:

Οι ανθρώποι μπορούν εύκολα να διαβάσουν και να γράψουν JSON. Η σύνταξη του μοιάζει πολύ με τη σύνταξη των αντικειμένων σε πολλές γλώσσες προγραμματισμού.

2. Διαχωρισμός Δεδομένων:

Το JSON διαχωρίζει τα δεδομένα σε ζεύγη "κλειδί-τιμής". Αυτό δίνει τη δυνατότητα να οργανώνονται δομημένα τα δεδομένα και να είναι εανάγνωστα.

3. Διαθεσιμότητα σε Διάφορες Γλώσσες:

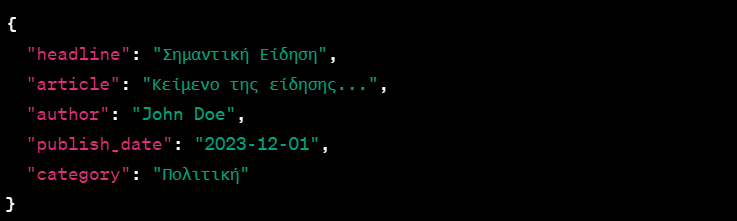
Οι περισσότερες γλώσσες προγραμματισμού έχουν υποστήριξη για τη μετατροπή JSON σε δομές δεδομένων και αντίστροφα.

4. Ανεξαρτησία Πλατφόρμας:

Το JSON είναι ανεξάρτητο από την πλατφόρμα, δηλαδή μπορεί να χρησιμοποιηθεί ανεξαρτήτως του λειτουργικού συστήματος ή της γλώσσας προγραμματισμού.

Ας φανταστούμε ένα REST API για μια εφαρμογή ειδήσεων. Καταλαβαίνουμε τις ειδήσεις μέσω ενός απλού JSON αντικειμένου.

Ένα απλό JSON αντικείμενο με μια είδηση θα μπορούσε να έχει την ακόλουθη μορφή:



*Εικόνα 2. JSON παράδειγμα*

Στο παραπάνω παράδειγμα, τα κλειδιά ("headline", "article", "author", κλπ.) αντιπροσωπεύουν πληροφορίες σχετικά με την είδηση. Ένα REST API μπορεί να εκθέσει ένα endpoint (σημείο πρόσβασης) όπου οι εφαρμογές μπορούν να αιτούνται δεδομένα σε αυτήν τη μορφή, και το JSON θα μεταδίδεται μεταξύ του Front-End (π.χ., μια ιστοσελίδα ειδήσεων) και του Back-End (π.χ., ένα σύστημα διαχείρισης ειδήσεων).

Αυτό το JSON αντικείμενο μεταφέρει τις βασικές πληροφορίες για μια είδηση με τα κλειδιά "headline", "article", κλπ., και μπορεί να χρησιμοποιηθεί από το Front-End για να παρουσιάσει την είδηση στον χρήστη.

## 5 Spring Boot

Σε αυτό το κεφάλαιο θα γνωρίσουμε την βασική τεχνολογία που χρησιμοποιείται στο συγκεκριμένο project όπως θα δούμε και παρακάτω, σε μεγαλύτερο βάθος το framework **Spring Boot.**

Το Spring Boot είναι μία επέκταση του πλαισίου (framework) Spring για την ανάπτυξη εφαρμογών Java. Έχει σχεδιαστεί για να διευκολύνει την ανάπτυξη γρήγορων και ευέλικτων εφαρμογών, κυρίως εστιάζοντας στην παροχή προεπιλεγμένων ρυθμίσεων και συνδυάζοντας τις βέλτιστες πρακτικές για τον προγραμματισμό σε ένα πλαίσιο.

Ορισμένα κύρια χαρακτηριστικά του Spring Boot περιλαμβάνουν:

1. Ευκολία Παραμετροποίησης (Opinionated Configuration): Παρέχει προεπιλεγμένες ρυθμίσεις για τα περισσότερα πράγματα, μειώνοντας την ανάγκη για χειροκίνητη παραμετροποίηση και διευκολύνοντας την έναρξη.

2. Ενσωματωμένος Web Server: Περιλαμβάνει ενσωματωμένους web servers όπως το Tomcat, πράγμα που σημαίνει ότι δεν απαιτείται η εγκατάσταση ή η παραμετροποίηση ενός εξωτερικού web server.

3. Μείωση της Πολυπλοκότητας: Προσφέρει απλότητα στην ανάπτυξη με λιγότερο κώδικα και περισσότερη συμβατότητα με τις βέλτιστες πρακτικές.

4. Δυνατότητες Ενσωματωμένης Διαχείρισης: Παρέχει ενσωματωμένες λειτουργίες διαχείρισης και παρακολούθησης όπως το Spring Boot Actuator.

5. Ολοκληρωμένη Υποστήριξη για Προτεινόμενες Βιβλιοθήκες: Υποστηρίζει και ενθαρρύνει τη χρήση προτεινόμενων βιβλιοθηκών όπως το Spring Data, το Spring Security κ.ά.

Το Spring Boot συνδυάζει την δύναμη του Spring Framework με την απλότητα και τη γρήγορη ανάπτυξη που απαιτείται για την ανάπτυξη σύγχρονων εφαρμογών.

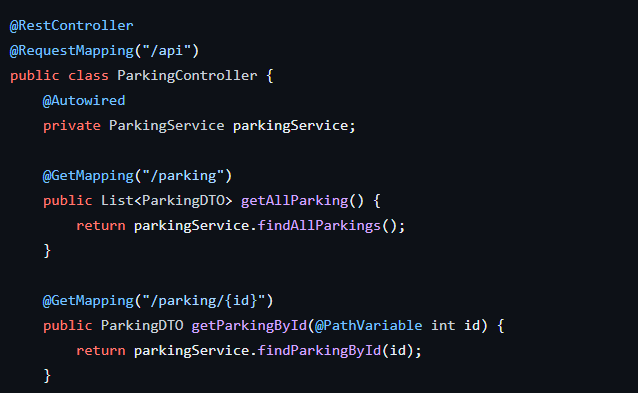
Οι παρακάτω κατηγορίες αντιπροσωπεύουν μια τυπική αρχιτεκτονική Spring Boot εφαρμογής, βασισμένη στα πρότυπα σχεδιασμού της Java EE (Enterprise Edition). Συνδυάζονται για να δημιουργήσουν μια συνεκτική, ευέλικτη και εύκολα συντηρήσιμη εφαρμογή.

Το Java EE (Enterprise Edition) είναι ένα πλαίσιο ανάπτυξης λογισμικού που παρέχει ένα ολοκληρωμένο σετ προδιαγραφών και υπηρεσιών για την ανάπτυξη και την εκτέλεση επιχειρησιακών εφαρμογών Java. Σχεδιάστηκε για να υποστηρίζει μεγάλη κλίμακα, ασφαλείς, και αξιόπιστες επιχειρησιακές εφαρμογές.

1. Controller:

- Περιγραφή: Οι ελεγκτές (controllers) στο Spring Boot είναι υπεύθυνοι για τη διαχείριση των HTTP αιτημάτων που λαμβάνει η εφαρμογή.

- Χρησιμότητα: Επεξεργάζονται τα αιτήματα που έρχονται από τους χρήστες, ενεργούν σαν επικοινωνιακά κομμάτια μεταξύ της εφαρμογής και του κόσμου.



*Εικόνα 3. Ένα κομμάτι του controller μέσα από το project*

Ο παραπάνω κώδικας ανήκει σε έναν Spring Boot controller και χειρίζεται αιτήσεις (requests) που αφορούν πληροφορίες σχετικές με Parking. Ας αναλύσουμε τι κάνει ο κώδικας:

1. **@RestController**: Αναφέρει ότι η κλάση αυτή είναι ένα RESTful controller, που σημαίνει ότι θα χειρίζεται αιτήσεις HTTP και θα επιστρέφει απαντήσεις σε μορφή JSON (ή άλλα προτύπου αντικειμένων).

2. **@RequestMapping("/api"):** Καθορίζει τη βασική διαδρομή (path) για όλα τα αιτήματα που χειρίζεται ο controller.

3. **@Autowired private ParkingService parkingService**: Χρησιμοποιεί τον μηχανισμό dependency injection του Spring για να ενσωματώσει ένα bean της κλάσης `ParkingService`. Ουσιαστικά, ο controller θα χρησιμοποιεί τις υπηρεσίες που προσφέρει ο `ParkingService`.

4. **@GetMapping("/parking"):** Χειρίζεται HTTP GET αιτήματα στο `/api/parking` και επιστρέφει μια λίστα από αντικείμενα `ParkingDTO`. Πιθανώς, αυτή η μέθοδος επιστρέφει όλα τα διαθέσιμα Parking.

5. **@GetMapping("/parking/{id}"):** Αυτή η μέθοδος χειρίζεται HTTP GET αιτήματα στο `/api/parking/{id}`, όπου `{id}` αναπαριστά έναν αριθμό (ID). Επιστρέφει ένα αντικείμενο `ParkingDTO` που αντιστοιχεί στο συγκεκριμένο ID.

Συνοπτικά, ο controller αυτός παρέχει δυο ενέργειες: να επιστρέψει όλα τα Parking και να επιστρέψει πληροφορίες για ένα συγκεκριμένο Parking βασιζόμενο στο ID.

2. DTO (Data Transfer Object):

- Περιγραφή: Τα αντικείμενα μεταφοράς δεδομένων (DTO) χρησιμοποιούνται για τη μεταφορά δεδομένων ανάμεσα σε διάφορα σημεία της εφαρμογής.

- Χρησιμότητα: Βοηθούν στη διατήρηση της καθαρότητας της αρχιτεκτονικής, αποφεύγοντας τη μεταφορά άχρηστων πληροφοριών.



*Εικόνα 4. Ένα κομμάτι DTO μέσα από το project*

Ο παραπάνω κώδικας αναφέρεται σε ένα αντικείμενο (DTO - Data Transfer Object) που ονομάζεται `ParkingDTO`. Ας δούμε τι κάνει αυτή η κλάση:

1. @Data, @AllArgsConstructor, @NoArgsConstructor, @Builder: Αυτές οι ετικέτες προέρχονται από το Lombok, μια βιβλιοθήκη που επιτρέπει την αυτοματοποιημένη δημιουργία μεθόδων όπως το constructor, τα getters, setters, κ.ά. Η `@Data` συνοψίζει πολλές από αυτές τις λειτουργίες.

2. private int parkingId; Μια μεταβλητή που αντιπροσωπεύει το ID του παρκινγκ.

3. private String name; Μια μεταβλητή που αντιπροσωπεύει το όνομα του παρκινγκ.

4. private List<ParkingZoneDTO> parkingZoneDTOList; Μια λίστα από αντικείμενα `ParkingZoneDTO`, αντιπροσωπεύοντας τις ζώνες Parking μέσα στο συγκεκριμένο παρκινγκ.

5. private float parkingCapacity; Ένας αριθμός που αντιπροσωπεύει τη χωρητικότητα του παρκινγκ ως ποσοστό.

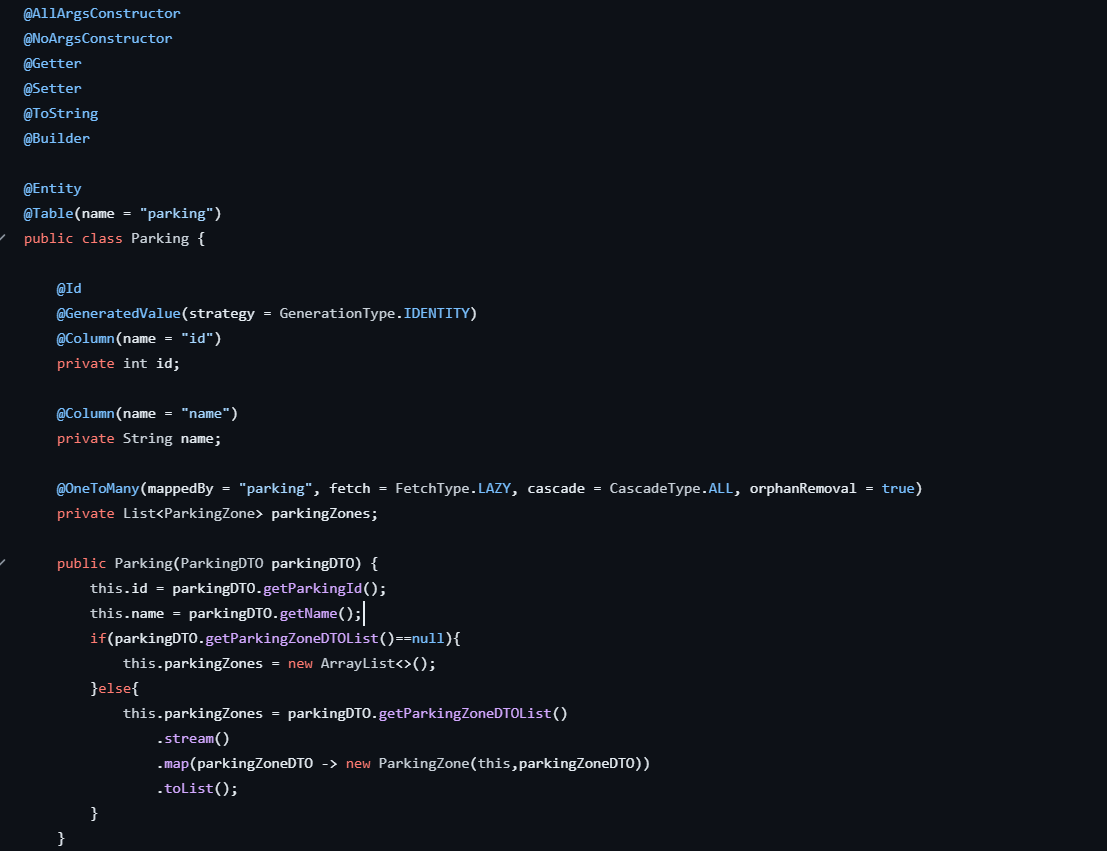
6. public ParkingDTO(Parking parking): Ένας ειδικός constructor που δέχεται ένα αντικείμενο `Parking` και το μετατρέπει σε ένα αντικείμενο `ParkingDTO`. Αυτός ο constructor χρησιμοποιείται για τη μετατροπή ενός αντικειμένου τύπου `Parking` σε `ParkingDTO`.

Συνοπτικά, η κλάση `ParkingDTO` λειτουργεί ως ένα αντικείμενο που μεταφέρει δεδομένα σχετικά με το Parking. Χρησιμοποιείται για να αποσυσχετίσει τα δεδομένα από την κλάση `Parking` σε μια πιο γενική μορφή που μπορεί να χρησιμοποιηθεί, για παράδειγμα, στη μεταφορά δεδομένων μεταξύ του back-end και του front-end σε μια web εφαρμογή.

3. Entity:

- Περιγραφή: Οι οντότητες αντιπροσωπεύουν τα δεδομένα της βάσης δεδομένων και αντιστοιχούν σε πίνακες στη βάση.

- Χρησιμότητα: Παρέχουν ένα αντικείμενο Java για την αλληλεπίδραση με τη βάση δεδομένων και αποτελούν τον πυρήνα της αντιστοίχισης αντικειμένου-σχήματος βάσης δεδομένων (ORM).



*Εικόνα 5. Ένα κομμάτι του Entity μέσα από το project*

Η κλάση `Parking` είναι μια Java entity που αντιπροσωπεύει ένα Parking.

1. @Entity: Αναφέρει ότι αυτή η κλάση αντιπροσωπεύει μια οντότητα που μπορεί να αποθηκευτεί σε μια βάση δεδομένων.

2. @Table(name = "parking"):\*Καθορίζει το όνομα του πίνακα στη βάση δεδομένων στον οποίο αντιστοιχεί αυτή η οντότητα.

3. @Id, @GeneratedValue, @Column: Χαρακτηριστικά που ορίζουν το πεδίο `id` ως πρωτεύον κλειδί με αυξανόμενη τιμή, καθώς και το όνομα της στήλης στη βάση δεδομένων.

4. @OneToMany(mappedBy = "parking", fetch = FetchType.LAZY, cascade = CascadeType.ALL, orphanRemoval = true): Ορίζει μια σχέση "ένα προς πολλά" (One-to-Many) με την κλάση `ParkingZone`. Το `mappedBy` καθορίζει το όνομα του πεδίου στην κλάση `ParkingZone` που χρησιμοποιείται για τον αντίστροφο συσχετισμό.

5. public Parking(ParkingDTO parkingDTO): Ένας ειδικός constructor που δέχεται ένα αντικείμενο `ParkingDTO` και το μετατρέπει σε ένα αντικείμενο `Parking`. Αυτός ο constructor χρησιμοποιείται για τη μετατροπή ενός αντικειμένου τύπου `ParkingDTO` σε `Parking`. Στο τέλος του constructor, επιλύονται οι σχέσεις και δημιουργούνται αντικείμενα `ParkingZone` βάσει των πληροφοριών που παρέχονται από το `ParkingDTO`.

Συνοπτικά, η κλάση `Parking` αποτελεί την αντικειμενική αναπαράσταση ενός Parking στο σύστημα, με σχέση με τις ζώνες Parking (`ParkingZone`) και διαθέτει μεθόδους για τη μετατροπή δεδομένων από και προς το αντίστοιχο DTO.

4. Repository:

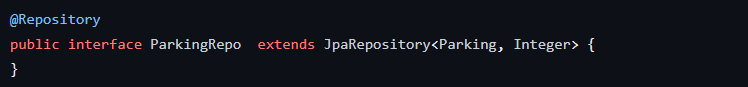
- Περιγραφή: Τα αποθετήρια (repositories) παρέχουν μια ευκολία στην αλληλεπίδραση με τη βάση δεδομένων, προσφέροντας μεθόδους για την ανάκτηση, την αποθήκευση και τη διαγραφή δεδομένων.

- Χρησιμότητα: Απλοποιούν τις λειτουργίες πρόσβασης/ενημέρωσης της βάσης δεδομένων.

5. JPA (Java Persistence API):

- Περιγραφή: Το JPA είναι ένα API που παρέχει ένα πρότυπο για τον προγραμματιστή προκειμένου να διαχειρίζεται τα δεδομένα σε μια βάση δεδομένων με Java.

- Χρησιμότητα: Συνδέει τα αντικείμενα Java με τη βάση δεδομένων, παρέχοντας έναν τρόπο ORM για την αντιστοίχιση των Java αντικειμένων στα δεδομένα της βάσης.



*Εικόνα 6. Ένα κομμάτι του Repository μέσα από το project*

Η αντιστοίχιση `@Repository` σε ένα διεπαφικό υποκείμενο (interface) σημαίνει ότι το συγκεκριμένο διεπαφικό υποκείμενο είναι μια συστατική του Spring Data και χρησιμοποιείται για την αλληλεπίδραση με μια βάση δεδομένων. Συγκεκριμένα, η δήλωση `ParkingRepo` κληρονομεί από την `JpaRepository<Parking, Integer>`, η οποία είναι μια γενική διεπαφή που παρέχει βασικές λειτουργίες ανάγνωσης και εγγραφής για ένα οντότητα `Parking` με πρωτεύον κλειδί τύπου `Integer`.

- `@Repository`: Επισημαίνει την κλάση ως ένα bean του Spring, υπεύθυνο για την πρόσβαση σε δεδομένα. Αυτό είναι συνήθως δηλωμένο πάνω σε ένα Interface που κληρονομεί από την `JpaRepository`.

- `JpaRepository<Parking, Integer>`: Είναι μια διεπαφή που παρέχει πολλές λειτουργίες αλληλεπίδρασης με τη βάση δεδομένων. Το `Parking` αναφέρει τον τύπο της οντότητας που χειρίζεται (σε αυτή την περίπτωση, η οντότητα `Parking`), και το `Integer` αναφέρεται στον τύπο του πρωτεύοντος κλειδιού.

Με αυτόν τον τρόπο, το Spring Data παρέχει προκαθορισμένες μεθόδους για τις βασικές λειτουργίες CRUD (Create, Read, Update, Delete) πάνω στην οντότητα `Parking`.

6. Service:

- Περιγραφή: Τα σέρβις (services) περιέχουν την επιχειρησιακή λογική της εφαρμογής και παρέχουν υπηρεσίες στους ελεγκτές για την εκτέλεση συγκεκριμένων λειτουργιών.

- Χρησιμότητα: Χωρίζουν τη λογική της εφαρμογής ανάμεσα σε διάφορα στρώματα, βοηθώντας στη συντηρησιμότητα και τη διαχείριση της πολυπλοκότητας.



*Εικόνα 7. Ένα κομμάτι του Parking Service μέσα από το project*

Η κλάση `ParkingServiceImpl` φέρει τη σημείωση `@Service`, η οποία υποδηλώνει ότι αυτή η κλάση είναι ένας σέρβις του Spring. Συνοπτικά, ένας σέρβις στο Spring χρησιμοποιείται για να παρέχει επιχειρηματική λογική ή λειτουργίες υπηρεσιών.

- `@Autowired private ParkingRepo parkingRepo`: Αυτή η σημείωση χρησιμοποιείται για να δηλώσει την εξάρτηση του σέρβις από το `ParkingRepo`, το οποίο είναι ένα αντικείμενο διεπαφής που χρησιμοποιείται για την αλληλεπίδραση με τη βάση δεδομένων για τον πίνακα `Parking`.

- `@Autowired private ParkingZoneRepo parkingZoneRepo;` και `@Autowired private ParkingSpotRepo parkingSpotRepo;`: Αντίστοιχα, αυτές οι σημειώσεις δηλώνουν τις εξαρτήσεις από τα αντίστοιχα αντικείμενα διεπαφής που χρησιμοποιούνται για τους πίνακες `ParkingZone` και `ParkingSpot`.

- `public List<ParkingDTO> findAllParkings()`: Είναι μια μέθοδος που επιστρέφει μια λίστα `ParkingDTO`, η οποία προέρχεται από το αποτέλεσμα της λειτουργίας `findAll()` του `ParkingRepo`. Κάθε αντικείμενο `Parking` που ανακτάται από τη βάση δεδομένων μετατρέπεται σε ένα αντίστοιχο `ParkingDTO`.

- `public ParkingDTO findParkingById(int id)`: Είναι μια μέθοδος που αναζητά ένα αντικείμενο `Parking` με βάση το πρωτεύον κλειδί. Χρησιμοποιεί τη μέθοδο `findById()` του `ParkingRepo`. Εάν βρεθεί το αντικείμενο, μετατρέπεται σε ένα `ParkingDTO`, αλλιώς εκτοξεύεται μια εξαίρεση `ParkingException`.

Κατά βάση, ο `ParkingServiceImpl` λειτουργεί ως ενδιάμεσο επίπεδο μεταξύ του ελεγκτή (`ParkingController`) και της βάσης δεδομένων (`ParkingRepo`), παρέχοντας λειτουργίες επεξεργασίας δεδομένων για τον πίνακα `Parking`.

## 6 Λειτουργικότητα Parking REST API

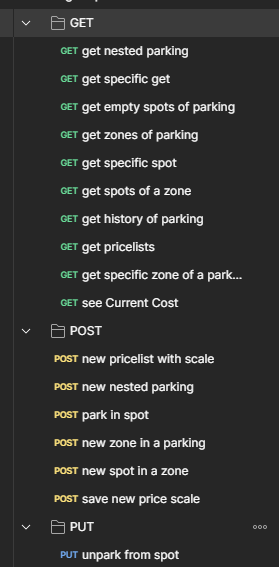
Σε αυτό το κεφάλαιο θα αναλύσουμε την λειτουργικότητα του API με βάση όσα έχουν αναφερθεί παραπάνω.

### 6.1 Postman

Το εργαλείο που θα χρησιμοποιήσουμε για να δούμε τα αποτελέσματα του API ονομάζεται Postman. Το Postman είναι ένα εργαλείο ανάπτυξης (API development tool) που χρησιμοποιείται για τον έλεγχο, τη δοκιμή και την ανάπτυξη των διεπαφών προγραμματισμού εφαρμογών (APIs). Ορισμένες από τις βασικές λειτουργίες του Postman περιλαμβάνουν:

1. Δημιουργία Αιτήσεων (Requests): Μπορείς να δημιουργήσεις διάφορους τύπους αιτήσεων HTTP όπως **GET, POST, PUT, DELETE** κ.ά. Κάθε αίτηση περιλαμβάνει το **URL** του στόχου, τις κεφαλίδες, το σώμα του αιτήματος και πολλά άλλα χαρακτηριστικά.

2. Οργάνωση Αιτήσεων σε Συλλογές (Collections): Μπορείς να οργανώνεις τις αιτήσεις σου σε συλλογές, πράγμα που διευκολύνει τον διαχωρισμό των λειτουργικοτήτων.



*Εικόνα 8. Ta Collection που έχω οργανώσει στο Postman για testing*

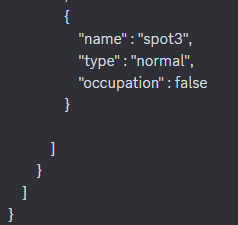
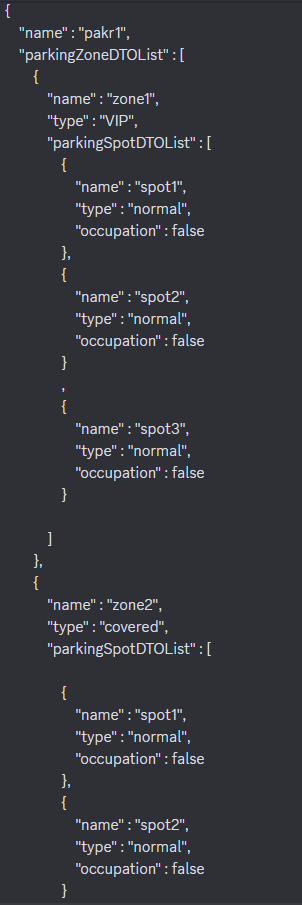
Κατά βάση, το Postman είναι ένα πολύ ισχυρό εργαλείο που βοηθά τους προγραμματιστές και τους αναπτυσσόμενους να διαχειρίζονται, να ελέγχουν και να αναπτύσσουν τις διεπαφές προγραμματισμού εφαρμογών με μεγαλύτερη ευκολία.

### 6.2 Post, Get και PUT

Σε αυτήν την ενότητα θα αναλύσουμε τις κλείσεις Post – Get που καλούνται από το API και θα δούμε τα αποτελέσματα που μας δίνει, ξεκινώντας από την δημιουργία μέχρι και τους τιμοκαταλόγους με την σειρά που χρειάζονται οι κάθε ενέργειες για να δημιουργήσουμε το ¨εικονικό¨ Parking.

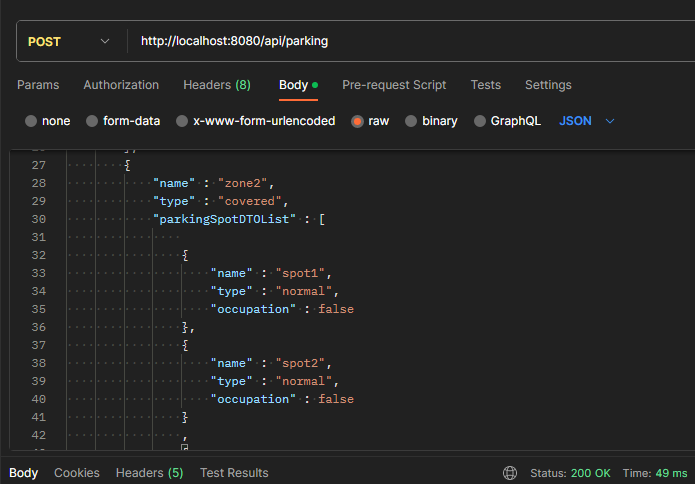
#### 6.2.1 Post-Get nested parking

Πρώτη μας κίνηση είναι να κάνουμε POST ένα νέο Parking. Για περισσότερη ευκολία, στον κώδικα έχω φτιάξει τη χαρτογράφηση του DTO να μπορεί να γίνει με ένα nested (εμφολευμένο) JSON. Οπότε δημιουργούμε για παράδειγμα ένα Parking που θα έχει 2 ζώνες και από 3 σημεία να παρκάρεις η κάθε ζώνη.



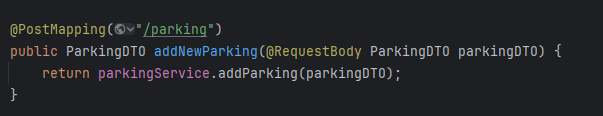
*Εικόνα 9-10. JSON NESTED CALL*

Tο παραπάνω JSON είναι ένα εμφωλευμένο (nested) JSON αντικείμενο. Το JSON αυτό περιγράφει μια δομή Parking με δύο ζώνες (parking zones), καθεμία από τις οποίες περιέχει πληροφορίες για το όνομά της, τον τύπο της, και μια λίστα από θέσεις στάθμευσης (parking spots). Με αυτό το JSON call μας έρχεται σαν αποτέλεσμα 200 ΟΚ, που αυτό σημαίνει ότι όλα πήγαν σωστά στην εκτέλεση του POST

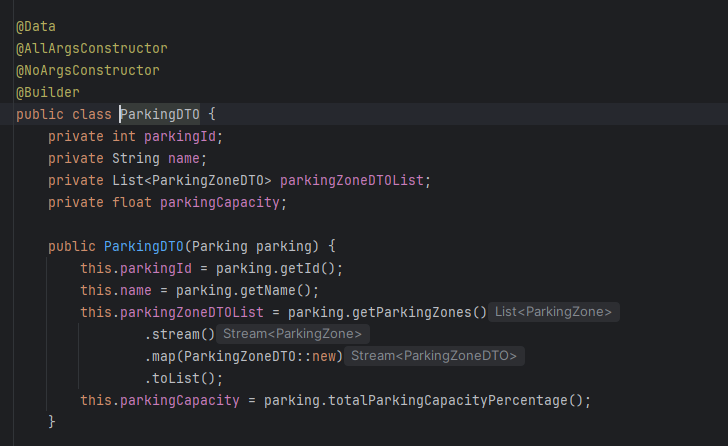


*Εικόνα 11. Αποτέλεσμα Postman*

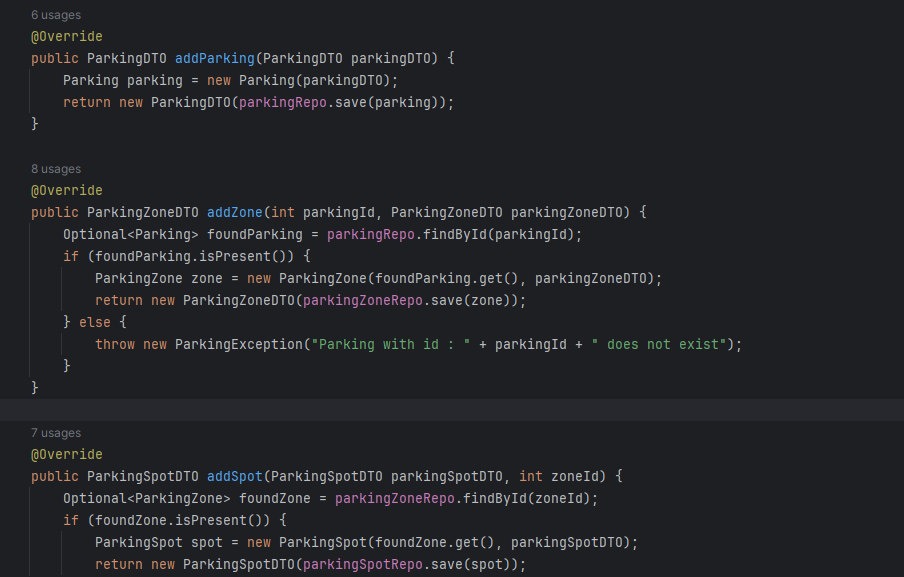
Παρακάτω παραθέτω τον κώδικα που εκτελείται από τον Controller:



*Εικόνα 12. addNewParking μέθοδος στον Controller (Java)*

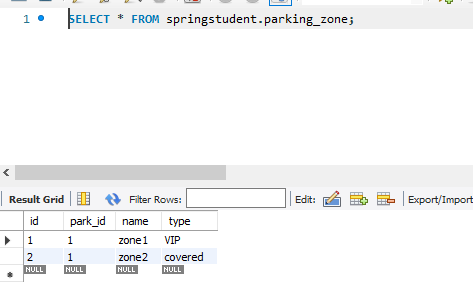
**

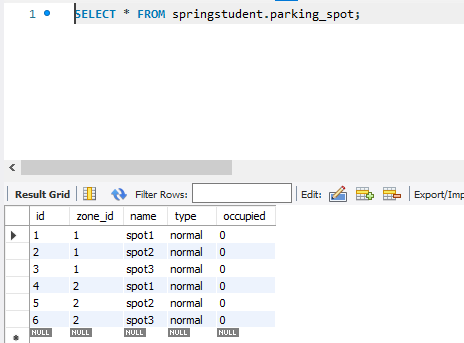
*Εικόνα 13. ParkingDTO (Java)*

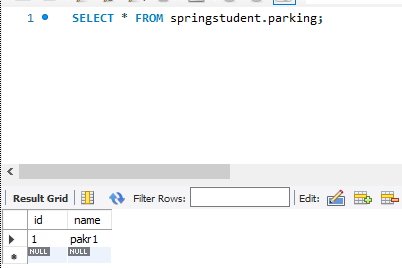
**

*Εικόνα 14. Το Service του Parking (Java)*

Παρακάτω θα δούμε τα αποτελέσματα στην βάση δεδομένων.

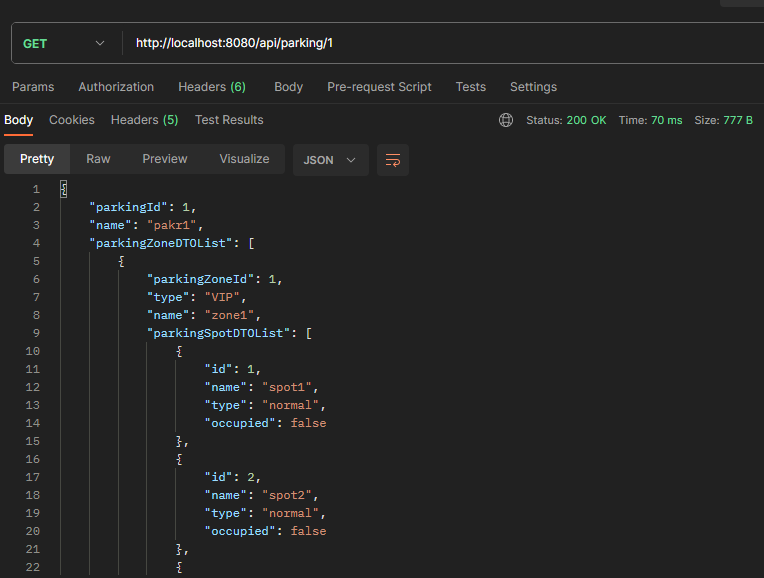
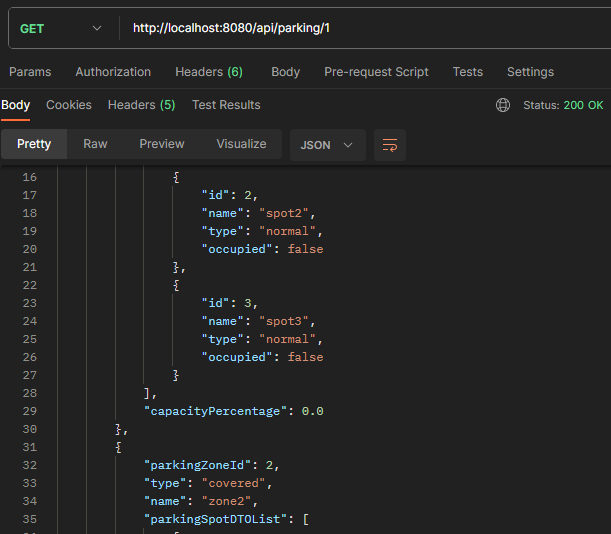
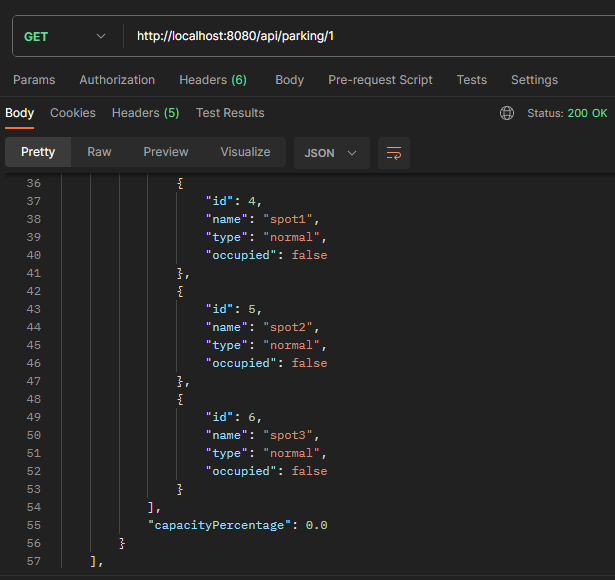






*Εικόνα 15-16-17. Αποτελέσματα στην Βάση Δεδομένων (MySQL)*

Έπειτα από την εντολή Post, τώρα θα δοκιμάσουμε την Get περιμένοντας να μας φέρει σαν αποτέλεσμα το nested JSON με τα δεδομένα του Parking μας.

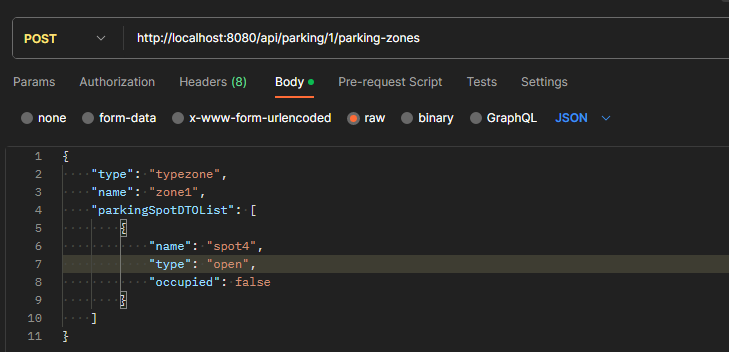


*Εικόνα 18-19-20. Αποτελέσματα κλήσης GET (Postman)*

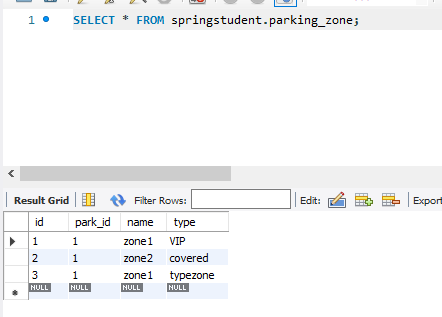
Με αυτόν τον τρόπο μπορούμε να δούμε τι περιεχόμενα έχει το parking.

#### 6.2.2 Post-Get zones και spots

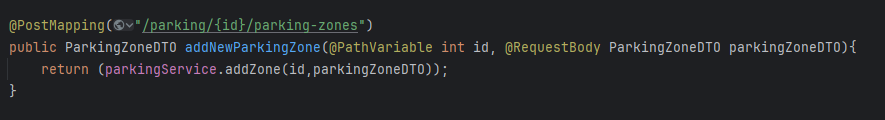
Παρακάτω θα δούμε πώς μπορώ να προσθέσω παραπάνω ζώνες ή και spots στο ήδη υπάρχων Parking. Θα προσθέσω μία νέα ζώνη με όνομα zone1 που θα έχει 1 spot4.



*Εικόνα 21. Αποτελέσματα κλήσης POST (Postman)*

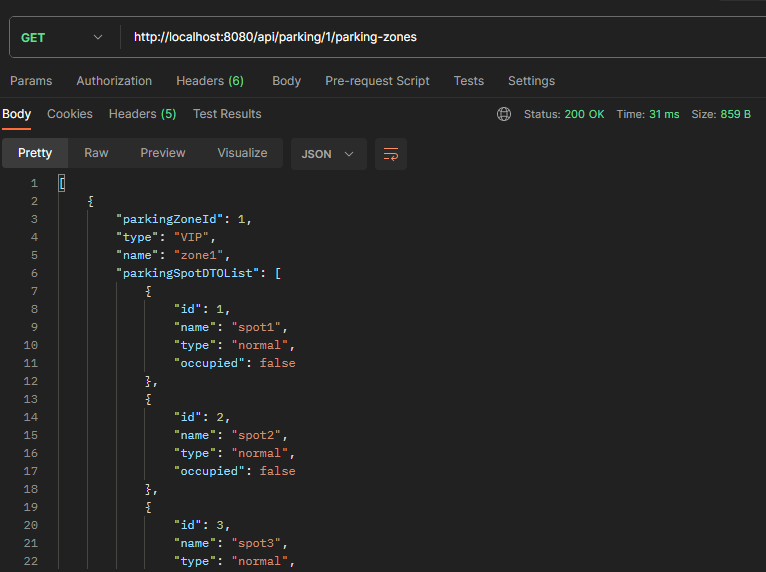
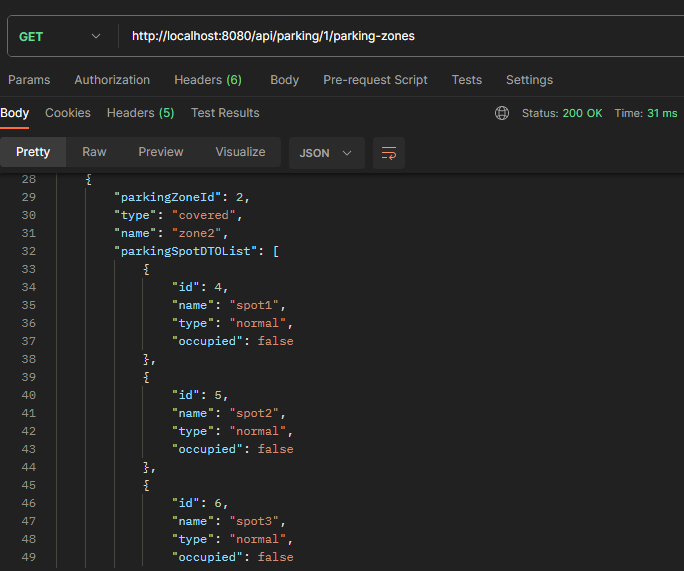
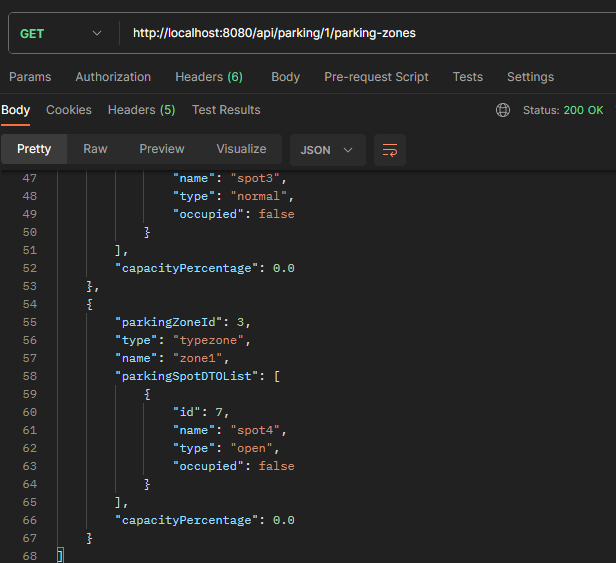


*Εικόνα 22. Το αποτέλεσμα στην βάση δεδομένων (MySQL)*



*Εικόνα 23. Ο Controller και η μέθοδος που κάλεσε (Java)*

Όπως και στο προηγούμενο παράδειγμα, θα δοκιμάσουμε και την κλήση get για να δούμε αναλυτικά τι έχουμε δημιουργήσει μέσα στο Parking

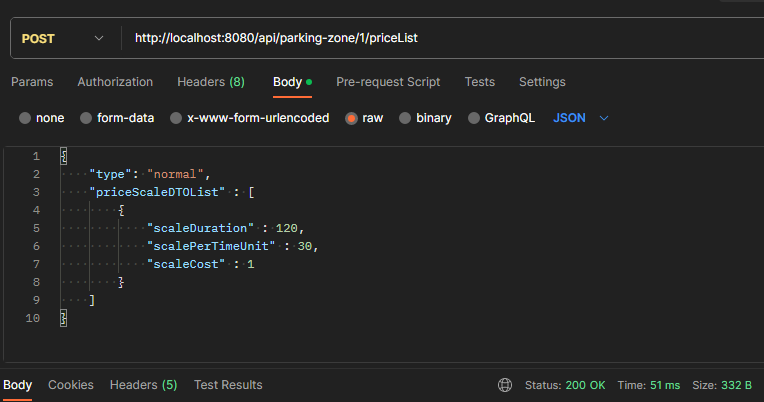


*Εικόνα 24-25-26. Αποτέλεσμα του Postman*

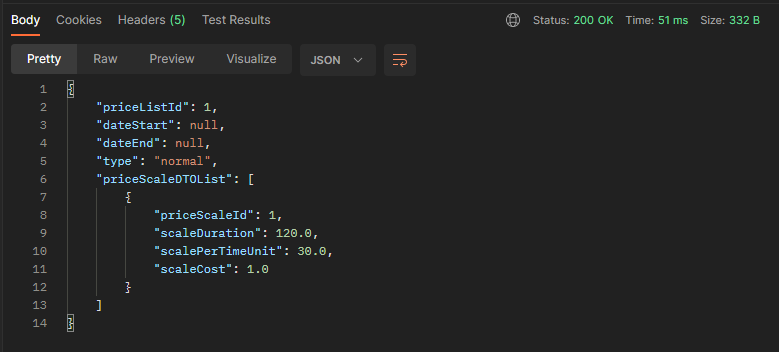
Όπως βλέπουμε στο παραπάνω αποτέλεσμα του Postman, μας επιστρέφει το αποτέλεσμα όπως το περιμέναμε, με την νέα ζώνη και το spot.

#### 6.2.3 Post-Get pricelist και pricescales

Παρακάτω θα δούμε πώς μπορούμε να δημιουργήσουμε έναν τιμοκατάλογο και να τον κάνουμε κλιμακωτό.



*Εικόνα 27. Κλήση για δημιουργία pricelist με pricescale*



*Εικόνα 28. Αποτέλεσμα του Postman*

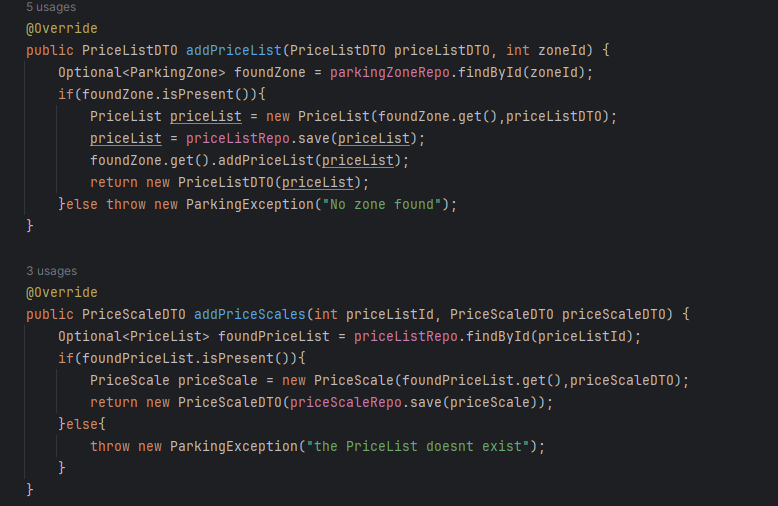
Όπως βλέπουμε δημιουργείτε κανονικά ο τιμοκατάλογος μαζί με την κλιμάκωση του. Αναλυτικότερα : ScaleDuration – πόσο χρόνο θα κρατήσει αυτή η κλιμάκωση

ScalePerTimeUnit – ανα πόσο θα χρεώνει μέσα σε αυτόν τον χρόνο

ScaleCost – πόσο θα χρεώνει.

Με λίγα λόγια ορίσαμε μία κλιμάκωση η οποία μας λέει (σε λεπτά) ότι για τα πρώτα 120 λεπτά που θα είναι σταθμευμένο ένα όχημα σε αυτήν την ζώνη, θα χρεώνεται ανά 30 λεπτά, 1 ευρώ.

Στις παρακάτω εικόνες απεικονίζεται ο κώδικας που κάνει αυτές τις ενέργειες



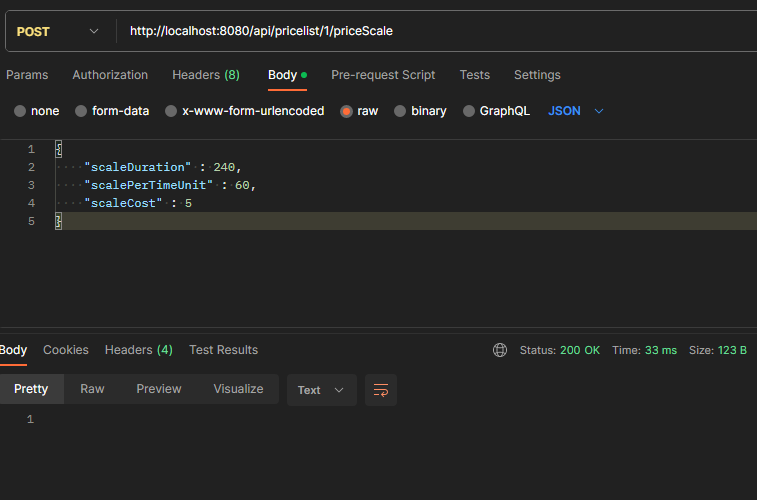
*Εικόνα 29. PriceList Service (Java)*



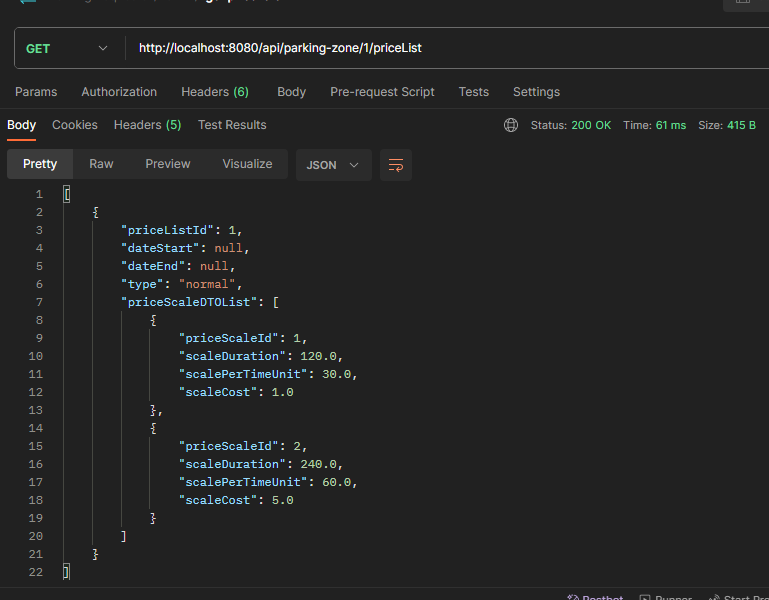
*Εικόνα 30. PriceList Controller (Java)*

#### 6.2.4 Post-Get Pricescale σε ήδη υπάρχον τιμοκατάλογο

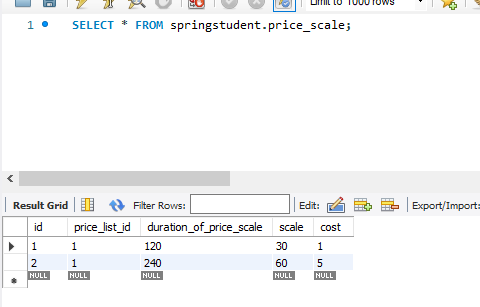
Σε αυτήν την ενότητα θα δούμε πως μπορούμε να αναθέσουμε και άλλες κλιμακώσεις σε κάποιον τιμοκατάλογο.



*Εικόνα 31. POST εξτρά κλιμάκωση στον τιμοκατάλογο (Postman)*



*Εικόνα 32. GET τον τιμοκατάλογο για να δούμε τις κλιμακώσεις που ορίσαμε (Postman)*



*Εικόνα 33. Το αποτέλεσμα στην βάση δεδομένων (MySQL)*

Όπως μπορούμε να διακρίνουμε στις παραπάνω εικόνες, προστέθηκε η εξτρά κλιμάκωση στον τιμοκατάλογο που του είπαμε (id=1). Όπως και στην πρώτη έτσι και σε αυτήν ακολουθείτε η ίδια λογική:

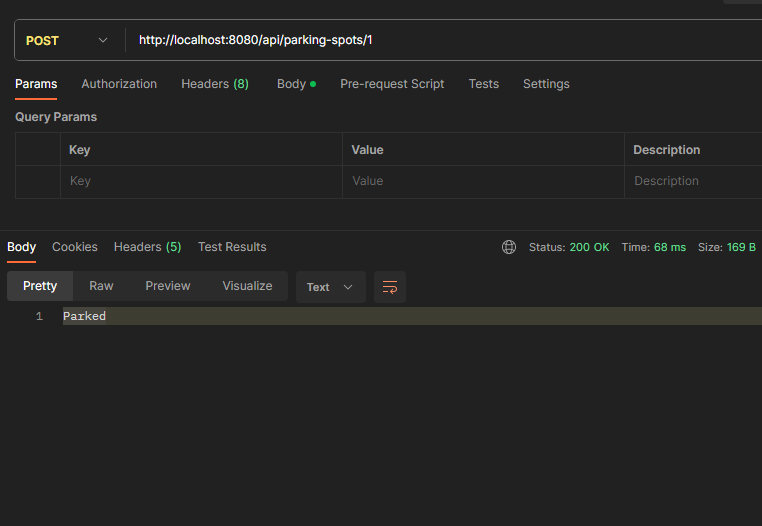
ScaleDuration – πόσο χρόνο θα κρατήσει αυτή η κλιμάκωση

ScalePerTimeUnit – ανα πόσο θα χρεώνει μέσα σε αυτόν τον χρόνο

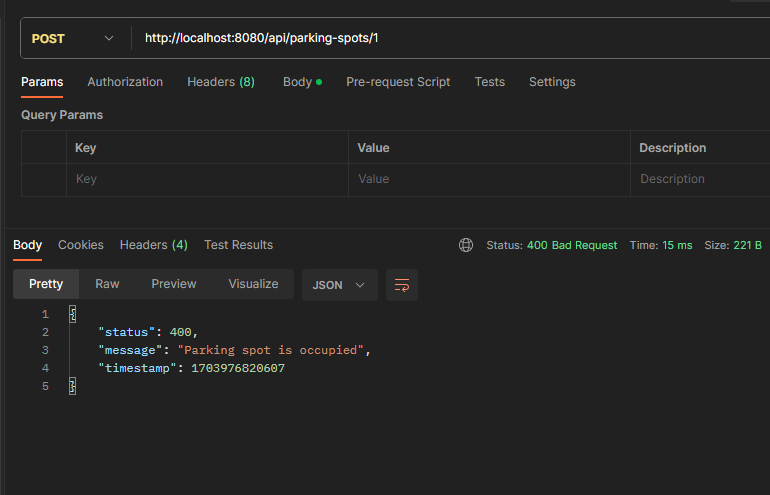
ScaleCost – πόσο θα χρεώνει.

#### 6.2.5 Post park-in-spot

Εφόσον έχουμε δημιουργήσει το Parking και έχουμε ορίσει τιμοκαταλόγους στις ζώνες, τότε μπορούμε να κάνουμε μια εικονική στάθμευση.

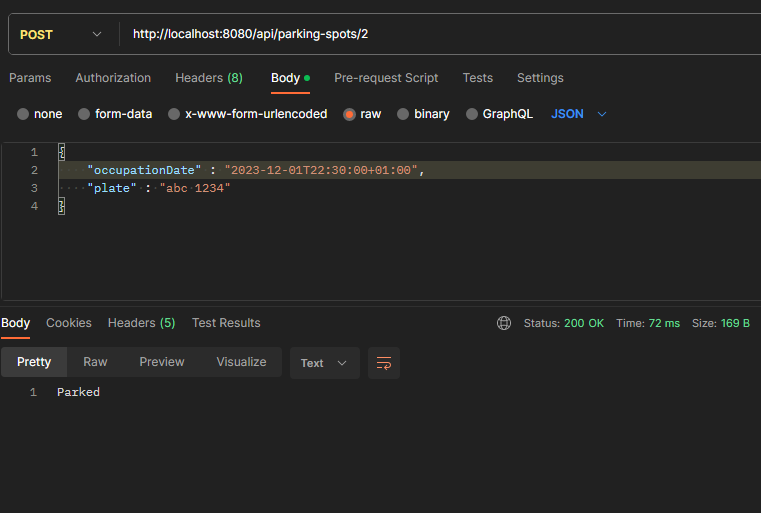


*Εικόνα 34. Το αποτέλεσμα της ενέργειας στάθμευσης (Postman)*

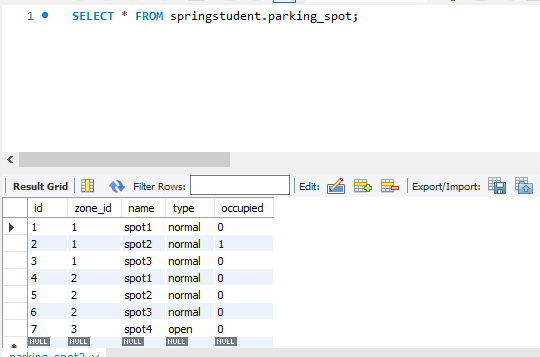


*Εικόνα 35. Το αποτέλεσμα της ενέργειας στάθμευσης στο ίδιο spot (Postman)*

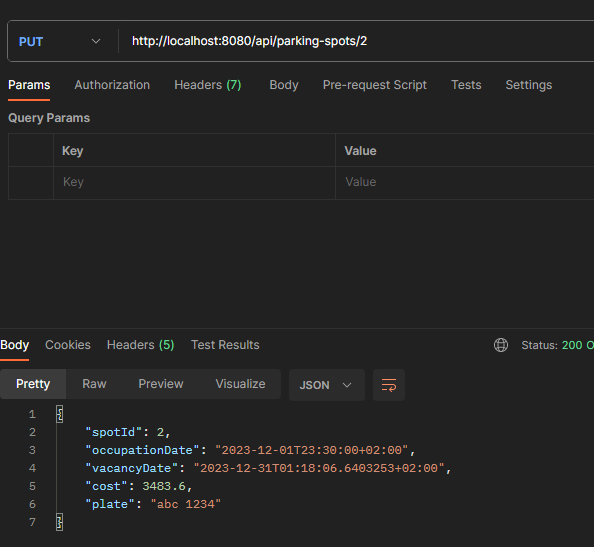
Όπως μπορούμε να διακρίνουμε στην εικόνα 35, δεν επιτρέπετε η στάθμευση σε σημείο που είναι ήδη σταθμευμένο κάποιο όχημα. Για να μπορεί να σταθμεύσει κάποιο άλλο όχημα σε εκείνη την θέση, θα πρέπει πρώτα να αποχωρίσει το όχημα που βρίσκεται ήδη εκεί.



*Εικόνα 36. Νέα στάθμευση οχήματος στην θέση 2 (Postman)*

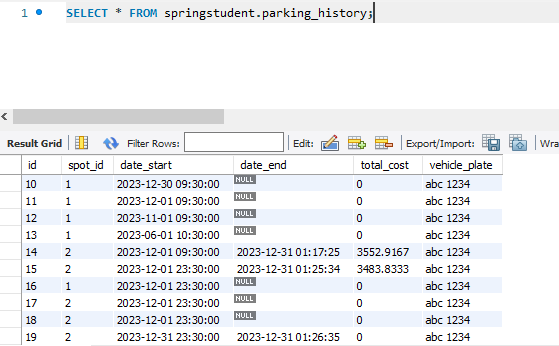
**

*Εικόνα 37. Η ανανέωση στην βάση δεδομένων για το occupied 1=true (MySQL)*

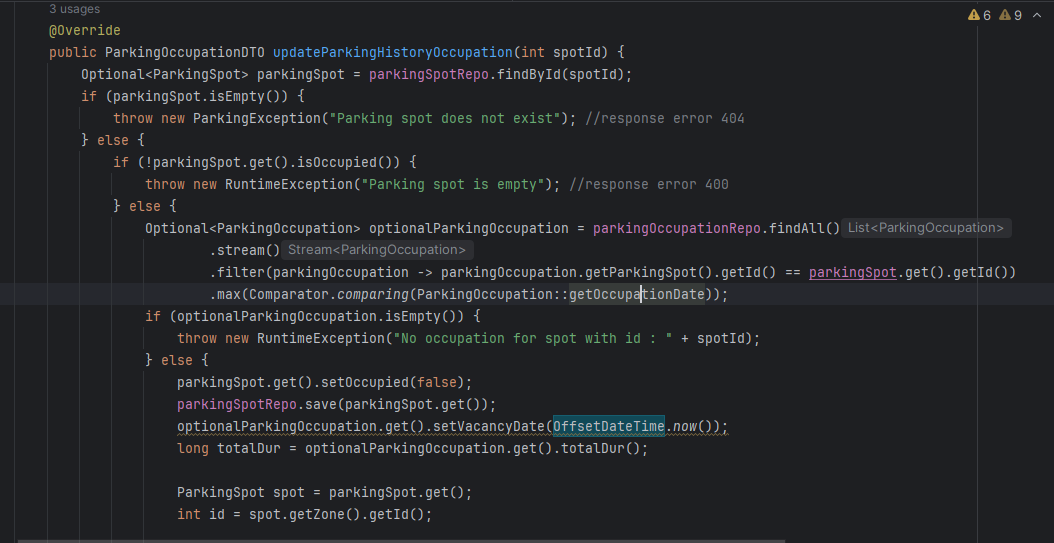


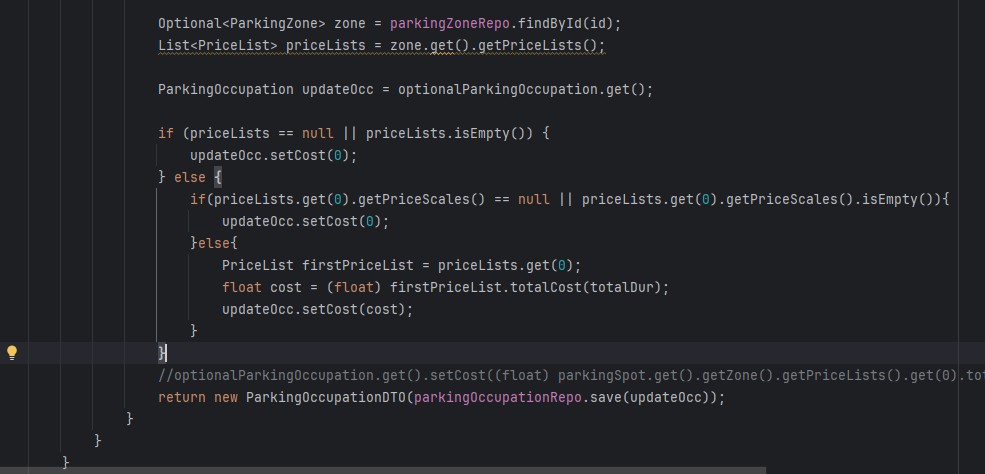
*Εικόνα 38. Αποχώρηση του οχήματος από την θέση 2 και χρέωση ανάλογη (Postman)*

Στις παραπάνω εικόνες χρησιμοποιήσαμε για ημερομηνία 01/12/2023 και 23:30 για ημερομηνία στάθμευσης και 31/12/2023 και 01:18 για ώρα αποχώρησης, με βάση τον κλιμακωτό τιμοκατάλογο το όχημα με πινακίδα abc 1234 θα χρεωθεί 3483.6 ευρώ (οι χρόνοι όπως διακρίνουμε είναι εντελώς πλασματικοί και ορίζονται από τον κώδικα με offsetDateTime).



*Εικόνα 39. Βάση δεδομένων για το parkingHistory (Postman)*

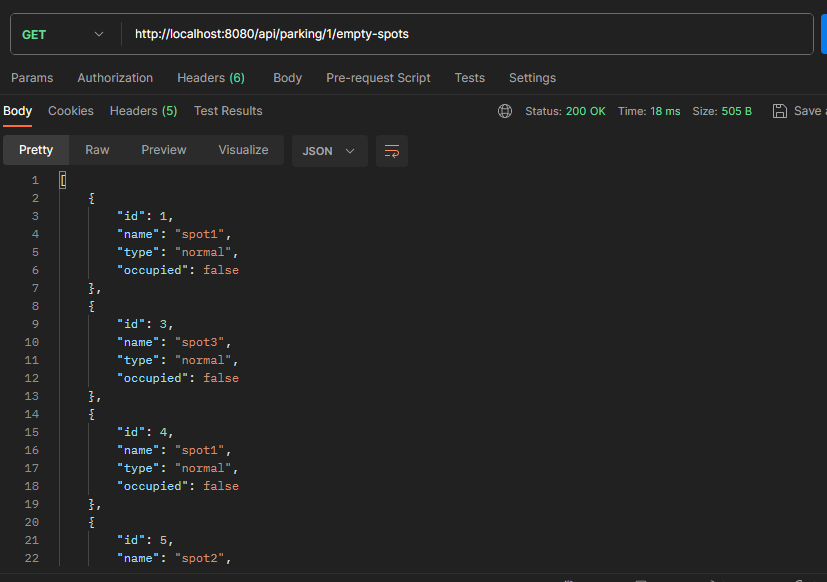




*Εικόνα 40-41. Μέθοδος του ParkingOccupation (Java)*

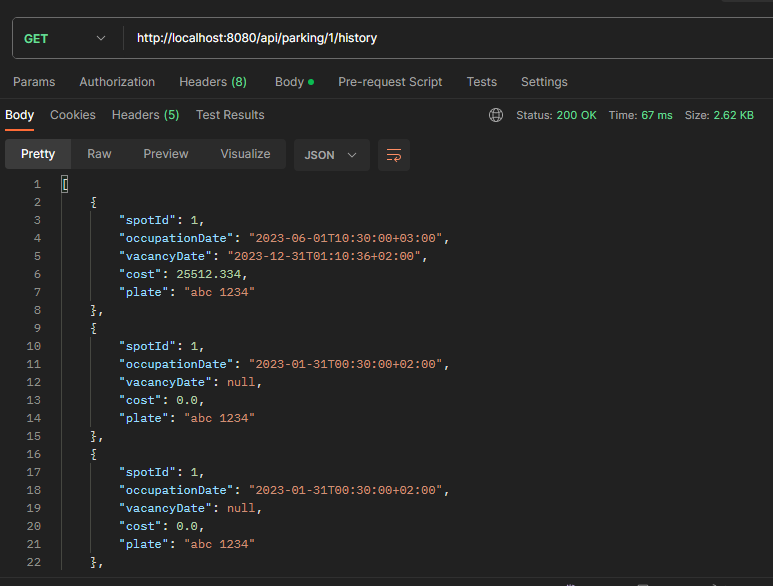
#### 6.2.6 Πρόσθετες λειτουργίες

1. GetEmptySpotsOfParking: Εμφανίζει όλα τα μη κατειλημμένα σημεία του Parking



*Εικόνα 42. Τα άδεια Spot (Postman)*

2. GetHistoryOfParking: Εμφανίζει όλο το ιστορικό του Parking

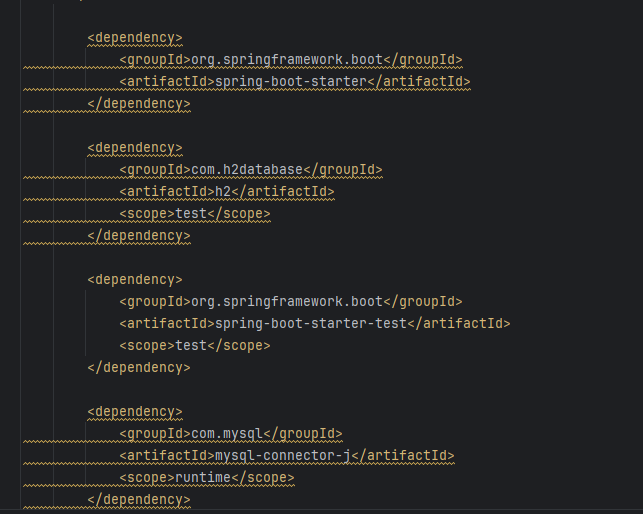


*Εικόνα 43.Όλο το ιστορικό σταθμεύσεων του Parking*

## 7 Λεπτομέρειες και βιβλιοθήκες

Η ανάπτυξη του κώδικα, πέρα από το SpringBoot framework χρησιμοποιεί και κάποιες βοηθητικές βιβλιοθήκες, μέσο του Maven.

Στο πλαίσιο του Apache Maven, η έννοια της "βιβλιοθήκης" συνδέεται συνήθως με την έννοια της εξάρτησης (dependency). Όταν αναφερόμαστε σε βιβλιοθήκες στο πλαίσιο του Maven, εννοούμε συνήθως τις εξαρτήσεις (dependencies) του έργου μας, δηλαδή τις βιβλιοθήκες που το έργο χρησιμοποιεί για να λειτουργήσει σωστά.



*Εικόνα 44.Παράδειγμα dependencies από το Project*

### 7.1 lombock dependency

Το Lombok είναι ένα εργαλείο λογισμικού για τη γλώσσα προγραμματισμού Java που βοηθά στη μείωση της ποσότητας κώδικα που πρέπει να γραφεί για την ανάπτυξη Java εφαρμογών. Χρησιμοποιείται για τη δημιουργία και τη διαχείριση των μεθόδων που απαιτούνται για την εκτέλεση τυπικών εργασιών στον κώδικα Java, όπως η δημιουργία getters, setters, constructors, και άλλα.

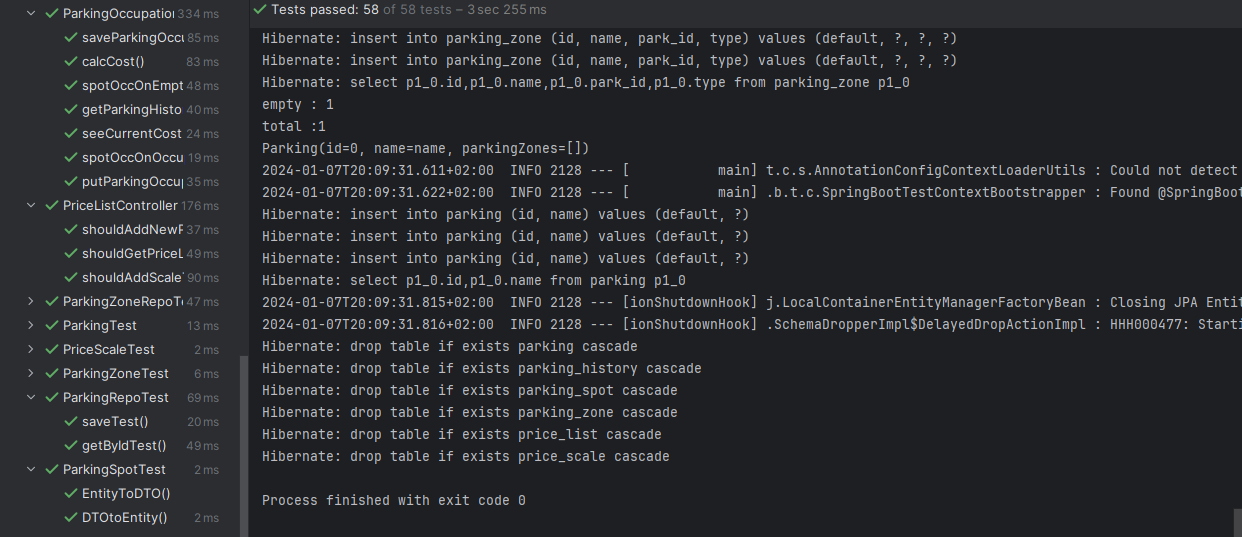
Με το Lombok, μπορούμε να προσθέσουμε ανατομικές επισημάνσεις (annotations) στον κώδικα, όπως το @Data, @Getter, @Setter και το Lombok θα δημιουργήσει αυτόματα τον αντίστοιχο κώδικα, χωρίς την ανάγκη να τα γράφουμε χειροκίνητα. Αυτό διευκολύνει τον προγραμματιστή και βοηθάει στην εξοικονόμηση χρόνου και προσπάθειας.



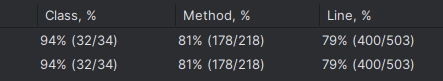
*Εικόνα 45.Παράδειγμα Lombock μέσα από το Project*

### 7.2 Testing

Στο Project έχει αναπτυχθεί επίσης και μια σειρά από testings, καλύπτοντας ένα ποσοστό λειτουργικότητας του Project 79% - 94%



*Εικόνα 46. Κάποια Test που τρέχουν.*



*Εικόνα 47. Το Coverage σε ποσοστό όλου του Project*

#### 7.2.1 Mock Tests

Τα mock tests (επίσης γνωστά ως mock exams ή mock assessments) αναφέρονται σε προσομοιωμένα δοκιμαστικά ή αξιολογήσεις. Συνήθως, τα mock tests αναπαριστούν μια πραγματική εξέταση, δοκιμασία ή αξιολόγηση και χρησιμοποιούνται για εκπαίδευση, προετοιμασία και αξιολόγηση των δεξιοτήτων ενός ατόμου.

Τα mock tests έχουν τα εξής χαρακτηριστικά:

1. Προσομοίωση των Πραγματικών Συνθηκών: Προσπαθούν να αντικατοπτρίσουν τις πραγματικές συνθήκες μιας εξέτασης ή αξιολόγησης, εξοικονομώντας χρόνο και προσπάθεια στην προετοιμασία.

2. Αξιολόγηση Επιδόσεων: Χρησιμοποιούνται για να αξιολογήσουν τις γνώσεις, τις δεξιότητες και την προετοιμασία του ατόμου πριν από την πραγματική εξέταση.



*Εικόνα 48. Παράδειγμα Mock Test μέσα από το Project*

#### 7.2.2 Unit Test

Το unit test είναι ένα είδος δοκιμής λογισμικού που ελέγχει τη σωστή λειτουργία ενός μικρού μέρους (ή "μονάδας") του λογισμικού, όπως μια συνάρτηση, μια μέθοδος ή ένα τμήμα κώδικα. Το κύριο όφελος των unit tests είναι η διασφάλιση ότι τα μικρά κομμάτια του λογισμικού λειτουργούν όπως αναμένεται, ακόμη και όταν προσθέτουμε νέο κώδικα ή προβαίνουμε σε αλλαγές.

Κάποια βασικά χαρακτηριστικά των unit tests περιλαμβάνουν:

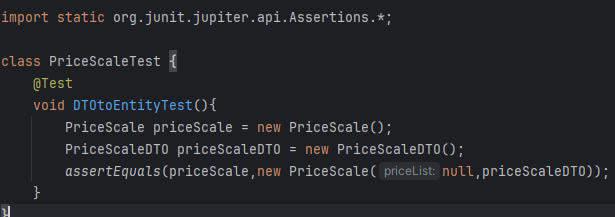
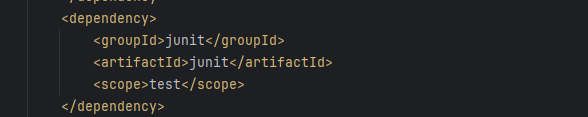
1. Αυτοματοποίηση: Τα unit tests εκτελούνται αυτόματα χωρίς την ανάγκη για χειροκίνητη παρέμβαση. Αυτό το χαρακτηριστικό επιτρέπει τη συχνή εκτέλεση των δοκιμών και εξασφαλίζει τη σταθερότητα του κώδικα.

2. Απομόνωση: Τα unit tests συνήθως εκτελούνται σε απομονωμένο περιβάλλον, ώστε να είναι δυνατή η έλεγχος της μονάδας κώδικα χωρίς την επίδραση άλλων μονάδων.

3. Επαναληψιμότητα: Τα unit tests πρέπει να είναι επαναληπτικά και να παράγουν τα ίδια αποτελέσματα κάθε φορά που εκτελούνται. Αυτό εξασφαλίζει την αξιοπιστία των δοκιμών.

4. Ταχύτητα εκτέλεσης: Τα unit tests πρέπει να εκτελούνται γρήγορα, επιτρέποντας έτσι τη συχνή εκτέλεση κατά τη διάρκεια της ανάπτυξης.

Τα unit tests είναι ένα σημαντικό εργαλείο στη διαδικασία ανάπτυξης λογισμικού, καθώς βοηθούν στη διασφάλιση της ποιότητας του κώδικα και στην αντιμετώπιση προβλημάτων πριν αυτά εντοπιστούν σε πιο προχωρημένα στάδια της ανάπτυξης ή κατά τη διάρκεια της λειτουργίας.

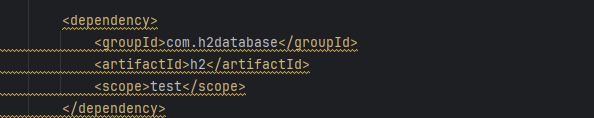


*Εικόνα 49-50. Dependency και παράδειγμα Unit Test μέσα από το Project*

#### 7.2.3 Integration Test με H2 Database

Τα integration tests είναι είδος δοκιμής λογισμικού που ελέγχει την αλληλεπίδραση μεταξύ διάφορων μονάδων ή συστημάτων. Όταν αναφερόμαστε σε "Integration Test με H2 Database," αναφερόμαστε σε δοκιμές που ελέγχουν πώς το σύστημα σας αλληλεπιδρά με μια H2 database όταν εκτελεί πραγματικές διεργασίες.

Το H2 είναι μια ελαφριά, αρθρωτή βάση δεδομένων που συχνά χρησιμοποιείται για integration testing. Οι βάσεις δεδομένων στο H2 μπορούν να δημιουργηθούν εντός της μνήμης κατά τη διάρκεια της εκτέλεσης των δοκιμών, και αυτό επιτρέπει την προσομοίωση πραγματικών συνθηκών βάσης δεδομένων χωρίς τον κίνδυνο επηρεασμού των πραγματικών δεδομένων.





*Εικόνα 51-52. Dependency και παράδειγμα Integration Test με H2 DB μέσα από το Project*

## 8 SCRUM AGILE

Το Scrum είναι ένα πλαίσιο εργασίας στο πλαίσιο της Agile μεθοδολογίας ανάπτυξης λογισμικού. Η Agile επικεντρώνεται στην ευελιξία, τη συνεργασία και την προσαρμοστικότητα στη διαδικασία ανάπτυξης λογισμικού, ενώ το Scrum παρέχει μια συγκεκριμένη δομή για την οργάνωση και τη διαχείριση έργων.

Τα βασικά χαρακτηριστικά του Scrum περιλαμβάνουν:

1. Επαναλαμβανόμενες Ενότητες (Sprints): Το έργο χωρίζεται σε μικρές ενότητες, γνωστές ως Sprints, με συγκεκριμένο χρονικό πλαίσιο (συνήθως 2-4 εβδομάδες).

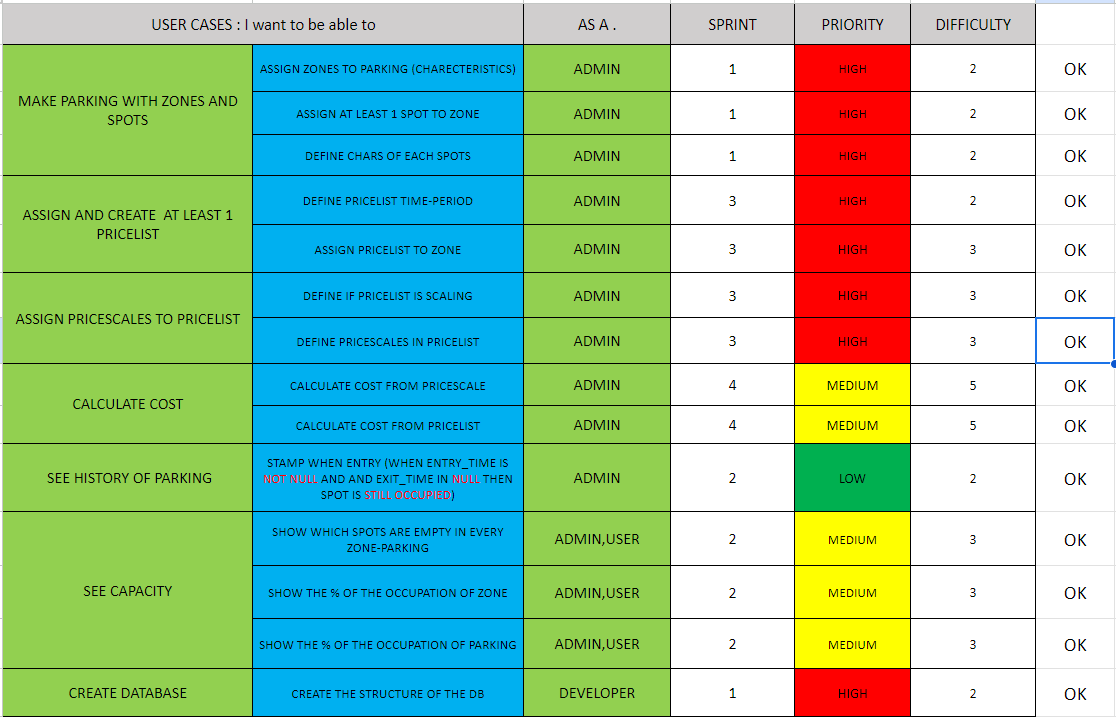
2. Προϊόν Αυξανόμενης Λειτουργικότητας: Σε κάθε Sprint, προστίθενται νέες λειτουργίες στο λογισμικό, δημιουργώντας μια αυξανόμενη έκδοση του προϊόντος.

3. Συντελεστές Ομάδας (Scrum Team): Η ομάδα Scrum περιλαμβάνει μέλη με διάφορες δεξιότητες (developers, testers, designers κ.ά.) που εργάζονται από κοινού για τον στόχο του Sprint.

4. Συντονιστής (Scrum Master): Ο Scrum Master υποστηρίζει την ομάδα Scrum και διευκολύνει την εφαρμογή των αρχών του Scrum.

5. Συντονιστής Προϊόντος (Product Owner): Ο Product Owner εκπροσωπεί τις ανάγκες του πελάτη και καθορίζει τις προτεραιότητες των λειτουργιών.

To Project αναπτύχθηκε ακολουθώντας μεθοδολογία SCRUM και θα παραδοθεί το αντίστοιχο Excel αρχείο με τα αναλυτικά User Cases και Sprints που γίναν κατά την διάρκεια ανάπτυξης



*Εικόνα 53. Κάποια από τα περιεχόμενα του Excel*

## 9 Συμπεράσματα

Με το πέρας της εργασίας αυτής, προέκυψαν τα ακόλουθα συμπεράσματα. Καταρχάς, η υλοποίηση ενός πλήρως λειτουργικού REST API με τη χρήση τεχνολογιών όπως το Spring Boot αποδεικνύεται εφικτή και ευέλικτη, προσφέροντας πληθώρα δυνατοτήτων για τη δημιουργία προσαρμοσμένων λύσεων.

Η χρήση του Postman κατέστησε δυνατό τον εύκολο έλεγχο και τη δοκιμή του API, επιτρέποντας την αποτελεσματική ανάπτυξη και την ασφαλή λειτουργία του. Οι διάφορες φάσεις δοκιμής, συμπεριλαμβανομένων των ολοκληρωμένων και των δοκιμών απόδοσης, συνέβαλαν στην αξιοποίηση της ποιότητας του κώδικα και τη βελτίωση της συνολικής απόδοσης του συστήματος.

Η υλοποίηση της μεθοδολογίας SCRUM AGILE αποτελεί σημαντικό στοιχείο για τη διαχείριση του έργου, επιτρέποντας την ευελιξία και τη συνεχή προσαρμογή. Η συνεχής ανάδραση και ο κύκλος παράδοσης (sprint) συνέβαλαν στην επίτευξη των στόχων εγκαίρως και με υψηλή ποιότητα.

Συνολικά, η ενσωμάτωση των παραπάνω στοιχείων καθιστά το έργο αυτό ένα παράδειγμα ανάπτυξης ένος ευέλικτου, ασφαλούς και λειτουργικού REST API, ακολουθώντας σύγχρονες βέλτιστες πρακτικές και διαδικασίες ανάπτυξης λογισμικού.

## Βιβλιογραφία

1. <https://docs.spring.io/spring-boot/docs/current/reference/htmlsingle/> -Ανάπτυξη REST API με Spring Boot: - Επίσημη τεκμηρίωση Spring Boot: [Spring Boot Documentation]
2. <https://learning.postman.com/docs/getting-started/introduction/> Postman και Δοκιμές API: - Επίσημη τεκμηρίωση Postman: [Postman Learning Center]
3. <https://martinfowler.com/bliki/UnitTest.html> Δοκιμές και Μονάδες: - "Test Driven Development: By Example" από Kent Beck

1. <https://projectlombok.org/> Επίσημη Ιστοσελίδα Lombok: [Project Lombok]

<https://projectlombok.org/features/all> Επίσημη Τεκμηρίωση Lombok: [Lombok Documentation]

1. <https://www.json.org/> [JSON.org] Η επίσημη ιστοσελίδα του JSON παρέχει το πρότυπο JSON και πολλές πληροφορίες σχετικά με τη μορφή JSON
2. https://dev.mysql.com/doc/ MySQL - [MySQL Documentation] - https://www.mysqltutorial.org/ [MySQL Tutorial]
3. <https://www.udemy.com/course/spring-hibernate-tutorial/> - Udemy Course για Spring Boot