ΕΛΛΗΝΙΚΗ ΔΗΜΟΚΡΑΤΙΑ

ΠΑΝΕΠΙΣΤΗΜΙΟ ΜΑΚΕΔΟΝΙΑΣ

ΣΧΟΛΗ ΕΠΙΣΤΗΜΩΝ ΠΛΗΡΟΦΟΡΙΑΣ
ΤΜΗΜΑ ΕΦΑΡΜΟΣΜΕΝΗΣ ΠΛΗΡΟΦΟΡΙΚΗΣ

ΕΡΓΑΣΙΑ ΕΞΑΜΗΝΟΥ

Τσορμπάρη Παρασκευή

Η παρούσα εργασία υλοποιήθηκε στα πλαίσια του μαθήματος
“Κατανεμημένα Συστήματα”
7ου εξαμήνου

Διδάσκων: Μαργαρίτης Κωνσταντίνος

1. Εισαγωγή

Το LearnSwift είναι μια σύγχρονη εφαρμογή e-booking που διευκολύνει την σύνδεση μαθητών με καθηγητές και μαθήματα, αποτελώντας μία εύχρηστη και αποτελεσματική πλατφόρμα. Μέσω της εφαρμογής ένας μαθητής μπορεί να δημιουργήσει λογαριασμό και να πραγματοποιήσει κρατήσεις μαθημάτων με τον καθηγητή που επιθυμεί, εξασφαλίζοντας μια προσωποποιημένη μαθησιακή εμπειρία.

Για την ανάπτυξη της εφαρμογής χρησιμοποιήθηκαν προηγμένες τεχνολογίες που διασφαλίζουν την λειτουργία και την αποδοτικότητά της. Συγκεκριμένα ως βάση δεδομένων επιλέχθηκε η PostgreSQL, μια ανοικτού κώδικα σχεσιακή βάση που προσφέρει αξιόπιστη αποθήκευση και οργάνωση μεγάλων ποσοτήτων δεδομένων. Για τη διαχείριση του backend και την υλοποίηση των μικροϋπηρεσιών, αξιοποιήθηκε το Spring Boot. Το Spring Boot αποτελεί ένα ευέλικτο και εύχρηστο framework που διευκολύνει τη γρήγορη δημιουργία web εφαρμογών με Java χρησιμοποιώντας την αρχιτεκτονική REST για την απλοποίηση της επικοινωνίας του backend με το frontend μέσω API endpoints. Η ανάπτυξη του frontend πραγματοποιήθηκε με την χρήση των τεχνολογιών HTML, CSS και JavaScript οι οποίες είναι ευρέως διαδεδομένες για την δημιουργία φιλικών και διαδραστικών διεπαφών χρήστη. Τέλος, χρησιμοποιήθηκε το Docker Compose, το οποίο επιτρέπει την απομόνωση και την αποτελεσματική ρύθμιση των υπηρεσιών σε ξεχωριστά containers. Με το Docker Compose, η ανάπτυξη της εφαρμογής γίνεται πιο ευέλικτη και εύκολη στη διαχείριση, καθώς οι υπηρεσίες εκτελούνται σε απομονωμένα περιβάλλοντα, χωρίς να επηρεάζονται από άλλες εξαρτήσεις. Αυτή η προσέγγιση βελτιώνει τη διαχείριση των πόρων και διευκολύνει τη μεταφορά της εφαρμογής σε διαφορετικά περιβάλλοντα ανάπτυξης.

Στο παρόν έγγραφο γίνεται αναλυτική παρουσίαση της αρχιτεκτονικής του λογισμικού, ξεκινώντας με μια αναφορά στους κύριους καταλόγους και τα αρχεία που συνθέτουν την εφαρμογή. Στη συνέχεια, παρατίθεται μια σειρά από παραδείγματα πηγαίου κώδικα, συνοδευόμενα από επεξηγήσεις για τις βασικές λειτουργίες και τις διαδικασίες που υποστηρίζει η εφαρμογή. Επιπλέον, παρουσιάζονται στιγμιότυπα από τη διεπαφή χρήστη (UI), ώστε να παρουσιαστεί η εμπειρία του χρήστη κατά την αλληλεπίδραση με την εφαρμογή. Στο τέλος του εγγράφου, παρατίθενται ιδέες για μελλοντικές επεκτάσεις και νέες δυνατότητες που θα μπορούσαν να προστεθούν στην εφαρμογή, καθώς και προτάσεις για βελτιώσεις και αναβαθμίσεις που θα μπορούσαν να ενισχύσουν τη λειτουργικότητα και την απόδοσή της.

2. Αρχιτεκτονική λογισμικού

Παρακάτω παρουσιάζεται το δέντρο των καταλόγων και των αρχείων της εφαρμογής.

LearnSwift/
├── LearnSwift/
│ ├── .idea/
│ ├── LearnSwift/
│ ├── src/
│ ├── main/
│ ├── java/
│ ├── com/
│ ├── example/
│ ├── **LearnSwift/**
│ ├── Configurations/
│ ├── InitilizeConfig.java
│ ├── SecurityConfig.java
│ ├── WebConfig.java
│ ├── Controllers/
│ ├── StudentController.java
│ ├── InstructorController.java
│ ├── BookingController.java
│ ├── ReviewController.java
│ ├── CourseController.java
│ ├── Services/
│ ├── StudentService.java
│ ├── InstructorService.java
│ ├── BookingService.java
│ ├── ReviewService.java
│ ├── CourseService.java
│ ├── Model/
│ ├── User.java
│ ├── Booking.java
│ ├── Student.java
│ ├── Instructor.java
│ ├── Review.java
│ ├── Course.java
│ ├── Repositories/
│ ├── StudentRepository.java
│ ├── InstructorRepository.java

```
│   │   │   │   │   │   │   ├── BookingRepository.java
│   │   │   │   │   │   │   ├── ReviewRepository.java
│   │   │   │   │   │   │   ├── CourseRepository.java
│   │   │   │   │   │   ├── LearnSwiftApplication.java
│   │   │   ├── resources/
│   │   │   │   ├── application.properties
│   │   │   │   ├── static/
│   │   │   │   ├── templates/
│   │   ├── Dockerfile
│   │   ├── build.gradle
│   │   ├── build/
│   │   ├── .gradle/
│   │   ├── gradle/
│   │   ├── .gitignore
│   │   ├── .gitattributes
│   │   ├── gradlew
│   │   ├── gradlew.bat
│   │   ├── HELP.md
│   │   ├── settings.gradle
├── frontend/
│   ├── Dockerfile
│   ├── Login_Register/
│   │   ├── Login_Register.html
│   │   ├── Login_Register.css
│   │   ├── Login_Register.js
│   ├── Student_view/
│   │   ├── StudentView.html
│   │   ├── StudentView.css
│   │   ├── StudentView.js
├── docker-compose.yml
```

Με έντονα γράμματα απεικονίζονται οι βασικοί φάκελοι και τα κύρια αρχεία της εφαρμογής. Ειδικότερα, στον φάκελο LearnSwift περιλαμβάνεται όλο το backend με το αρχείο εκκίνησης της εφαρμογής LearnSwiftApplication.java και τους υποφακέλους Configurations, Controllers, Services, Repositories και Model.

- **Configurations:** Ο κατάλογος αυτός αποτελείται από την κλάση InitializeConfig η οποία αρχικοποιεί την βάση δεδομένων με μαθήματα, καθηγητές και μαθητές, την κλάση SecurityConfig για τις ρυθμίσεις ασφαλείας και την κλάση WebConfig η οποία είναι υπεύθυνη για τη διαχείριση των CORS (Cross-Origin Resource Sharing) ρυθμίσεων που επιτρέπουν την

επικοινωνία του frontend με το backend όταν βρίσκονται σε διαφορετικές θύρες.

- **Controllers:** Εδώ βρίσκονται όλοι οι controllers οι οποίοι διαχειρίζονται τα HTTP αιτήματα από το frontend και καθοδηγούν τη λογική του backend. Περιλαμβάνουν endpoints για την εκτέλεση λειτουργιών όπως η ανάκτηση, η δημιουργία, η διαγραφή και η ενημέρωση δεδομένων.
- **Services:** Σε αυτό τον φάκελο υπάρχουν οι κλάσεις με τα services για τη βασική επιχειρησιακή λογική της εφαρμογής. Επικοινωνούν με τα Repositories και υλοποιούν τις διαδικασίες που απαιτούνται από τους Controllers.
- **Repositories:** Τα repositories συνδέονται με οντότητες και αλληλεπιδρούν με τη βάση δεδομένων. Παρέχουν μεθόδους για την αποθήκευση, ανάκτηση, ενημέρωση και διαγραφή δεδομένων, χρησιμοποιώντας το JPA (Java Persistence API).
- **Model:** Αποτελείται από τις οντότητες της εφαρμογής.

Στον φάκελο frontend έχουν τοποθετηθεί δύο υποφακέλοι, ο Login_Register για την αρχική σελίδα της πλατφόρμας και ο Student_View που αφορά την σελίδα περιήγησης του χρήστη για την εξυπηρέτησή του από τις υπηρεσίες της εφαρμογής. Στα αρχεία JavaScript υλοποιείται ο κώδικας που εκτελεί κλήσεις στα endpoints του backend, λαμβάνει τα αποτελέσματα και τα εμφανίζει στα κατάλληλα στοιχεία της διεπαφής, ώστε ο χρήστης να τα δει και να αλληλεπιδράσει με αυτά.

Για το frontend και το backend της πλατφόρμας υπάρχουν αρχεία Dockerfile, τα οποία καθορίζουν τη διαδικασία δημιουργίας των Docker containers για κάθε μέρος της εφαρμογής. Ένα αρχείο Dockerfile περιλαμβάνει τις εντολές για την εγκατάσταση των απαιτούμενων εξαρτήσεων και την εκκίνηση των εφαρμογών μέσα σε ξεχωριστά containers. Επιπλέον, στον ριζικό φάκελο υπάρχει το αρχείο docker-compose.yml, το οποίο επιτρέπει την οργάνωση και τη διαχείριση πολλών containers ταυτόχρονα. Μέσω του docker-compose.yml καθορίζονται οι ρυθμίσεις για το πως οι containers του frontend και backend θα επικοινωνούν μεταξύ τους, καθώς και άλλες παραμέτρους όπως τα ports και τα volumes.

3. Ανάλυση βασικών σημείων κώδικα

3.1 Ρυθμίσεις Docker

Για να επιτευχθεί η επικοινωνία μεταξύ του frontend και του backend μέσω Docker, είναι απαραίτητο να καθοριστούν οι αντίστοιχες θύρες στο αρχείο docker-compose.yml που είναι υπεύθυνο για τον ορισμό και τη διαχείριση πολλαπλών υπηρεσιών Docker. Θέτοντας τη θύρα 8080 για το backend τη θύρα 80 για το frontend και τη θύρα 5432 για τη βάση δεδομένων, η εφαρμογή είναι έτοιμη να λειτουργήσει.

```yaml
backend:
  build:
    context: "C:/Users/evits/Desktop/LearnSwiftProject/LearnSwift/LearnSwift"
    dockerfile: Dockerfile
  ports:
    - "8080:8080"
  depends_on:
    - db
  environment:
    SPRING_DATASOURCE_URL: jdbc:postgresql://db:5432/mydb
    SPRING_DATASOURCE_USERNAME: user
    SPRING_DATASOURCE_PASSWORD: password
  networks:
    - app-network

frontend:
  build:
    context: "C:/Users/evits/Desktop/LearnSwiftProject/frontend"
    dockerfile: Dockerfile
  ports:
    - "80:80"
  environment:
    API_URL: "http://localhost:8080"  # Backend url
  networks:
    - app-network

db:
  image: postgres:13
  environment:
    POSTGRES_DB: mydb
    POSTGRES_USER: user
    POSTGRES_PASSWORD: password
  volumes:
    - postgres-data:/var/lib/postgresql/data
  ports:
    - "5432:5432"
```

Εικόνα 1. Κώδικας από το αρχείο docker-compose.yml

Επιπλέον, στο Dockerfile του frontend έχει οριστεί μια μεταβλητή περιβάλλοντος, με σκοπό την ευελιξία στην αλλαγή της διεύθυνσης που αναφέρεται οποιαδήποτε στιγμή. Η πρόσβαση σε αυτή πραγματοποιείται μέσω του κώδικα, χρησιμοποιώντας την εντολή που περιγράφεται στην Εικόνα 3.

```
# Set the environment variable for the API URL
ENV API_URL=http://localhost:8080
```

Εικόνα 2. Ορισμός περιβαλλοντικής μεταβλητής.

```
//Get the API URL
const apiUrl = window.API_URL || 'http://localhost:8080';
```

Εικόνα 3. Πρόσβαση στην περιβαλλοντική μεταβλητή μέσω του κώδικα.

3.2 Υπηρεσίες εφαρμογής

3.2.1 Σύνδεση / Εγγραφή χρήστη

Η διαδικασία σύνδεσης ή εγγραφής του χρήστη στην πλατφόρμα ξεκινάει με την συμπλήρωση των απαραίτητων πεδίων με τα προσωπικά του στοιχεία ώστε να είναι δυνατή η δημιουργία του αιτήματος από το frontend. Αφού δημιουργηθεί το σώμα της αίτησης, γίνεται κλήση του κατάλληλου API endpoint στον controller του backend με την χρήση του fetch API.

```
const url = form.id === 'registerForm'
    ? `${apiUrl}/api/students/register`
    : `${apiUrl}/api/students/login`;

console.log('API URL:', url);

const body = form.id === 'registerForm' // For register
    ? JSON.stringify({
        username: formData.get('username'),
        email: formData.get('email'),
        password: formData.get('password'),
        role: 'student'
    })
    : new URLSearchParams({ // For login
        email: formData.get('email'),
        password: formData.get('password')
    }).toString();

const headers = form.id === 'registerForm'
    ? { 'Content-Type': 'application/json' }
    : { 'Content-Type': 'application/x-www-form-urlencoded' };



// Make the API call
const response = await fetch(url, {
    method: 'POST',
    headers: headers,
    body: body
});
```

Εικόνα 4. Κώδικας Javascript δημιουργίας αιτήσεων για τις ενέργειες σύνδεσης και εγγραφής του χρήστη.

Στη συνέχεια, ο αντίστοιχος controller επεξεργάζεται την αίτηση, ενεργοποιεί την κατάλληλη υπηρεσία και επιστρέφει στο frontend είτε το αποτέλεσμα της επεξεργασίας είτε ένα εύστοχο μήνυμα λάθους, διασφαλίζοντας την ομαλή και σωστή λειτουργία της εφαρμογής.

```java
@PostMapping("/register")
public ResponseEntity<Student> registerStudent(@RequestBody Student student) {
    //Ensure the role is "student"
    if (!(student.getRole().equals("student"))) {
        return new ResponseEntity<>(HttpStatus.BAD_REQUEST);
    }

    try {
        studentService.registerStudent(student);
        return new ResponseEntity<>(student, HttpStatus.CREATED); //Return student to pass his credentials in the next page after login
    } catch (IllegalArgumentException e) {
        return new ResponseEntity<>( HttpStatus.CONFLICT); // Username already exists
    } catch (Exception e) {
        return new ResponseEntity<>(HttpStatus.INTERNAL_SERVER_ERROR);
    }
}

no usages
@PostMapping("/login")
public ResponseEntity<Student> loginStudent(@RequestParam String email, @RequestParam String password) {
    try {
        Student student = studentService.loginStudent(email, password);
        return new ResponseEntity<>(student, HttpStatus.OK);
    } catch (IllegalArgumentException e) {
        return new ResponseEntity<>(null, HttpStatus.UNAUTHORIZED); // Invalid username or password
    } catch (Exception e) {
        return new ResponseEntity<>(null, HttpStatus.INTERNAL_SERVER_ERROR);
    }
}
```

Εικόνα 5. Κώδικες των controllers για τις ενέργειες σύνδεσης και εγγραφής του χρήστη.

Τέλος, η εκάστοτε υπηρεσία πραγματοποιεί τους κατάλληλους ελέγχους και ενημερώνει την βάση μέσω του ενδεδειγμένου repository. Ειδικότερα, η υπηρεσία εγγραφής ελέγχει αρχικά αν υπάρχει εν λόγω χρήστης σύμφωνα με το email του κι έπειτα αν ο κωδικός πρόσβασης είναι ορθός. Αφού ο χρήστης αυθεντικοποιηθεί, επιστρέφει τα στοιχεία του. Η υπηρεσία εγγραφής διασφαλίζει τη μοναδικότητα του χρήστη, ελέγχοντας αν το username και το email που υποβάλλονται είναι ήδη καταχωρημένα. Στη συνέχεια, προχωρά στην αποθήκευση των στοιχείων του νέου χρήστη στη βάση δεδομένων.

```
1 usage
public void registerStudent(Student student) {
    //check if the student exists according to username
    if (studentRepository.existsByUsername(student.getUsername())) {
        throw new IllegalArgumentException("Username already exists");
    }

    //check if the student exists according to email
    if (studentRepository.existsByEmail(student.getEmail())) {
        throw new IllegalArgumentException("Email already exists");
    }
    student.setPassword(student.getPassword());
    studentRepository.save(student);
}

1 usage
public Student loginStudent(String email, String password) {
    //check if the student exists
    Student student = studentRepository.findByEmail(email)
            .orElseThrow(() -> new IllegalArgumentException("Student not found"));

    if (!(student.getPassword().equals(password))) {
        throw new IllegalArgumentException("Invalid password");
    }

    return student;
}
```

Εικόνα 6. Κώδικες υπηρεσιών σύνδεσης και εγγραφής χρήστη.

Αξίζει να σημειωθεί ότι όλες οι κλήσεις από το frontend ακολουθούν παρόμοια διαδικασία, διαφοροποιημένες μόνο ως προς το endpoint, τα headers και το body που χρησιμοποιούνται σε κάθε περίπτωση. Επιπλέον, την ίδια λογική ακολουθούν και οι controllers στο backend, καθώς κάθε φορά καλούν την αντίστοιχη υπηρεσία που είναι υπεύθυνη για την επεξεργασία του αιτήματος και διαχειρίζονται την εμφάνιση των σφαλμάτων. Για τον λόγο αυτό, παρατίθεται ενδεικτικά ο κώδικας κλήσης της πρώτης υπηρεσίας από το frontend και ο πρώτος controller, ώστε να αναδειχθεί η γενική ροή και ο τρόπος λειτουργίας του συστήματος. Στη συνέχεια θα πραγματοποιηθεί η ανάλυση μόνο των υπηρεσιών.

3.2.2 Αναζήτηση μαθημάτων

Όταν ο χρήστης ζητά να δει τα διαθέσιμα μαθήματα της πλατφόρμας μέσω του frontend, γίνεται αρχικά μια HTTP κλήση στο αντίστοιχο endpoint του Controller των μαθημάτων ο οποίος με την σειρά του καλεί την υπηρεσία εμφάνισης των

μαθημάτων. Η υπηρεσία αυτή αξιοποιεί τη μέθοδο findAll του repository για να ανακτήσει όλα τα αποθηκευμένα μαθήματα από τη βάση δεδομένων.

Εικόνα 7. Κώδικας υπηρεσίας επιστροφής μαθημάτων.

Ωστόσο, οι πληροφορίες που είναι διαθέσιμες από αυτή την κλήση είναι μόνο ο τίτλος του μαθήματος καθώς η διάρκεια και η τιμή του διαφοροποιούνται ανάλογα με τον καθηγητή που το διδάσκει. Για την απόκτηση αυτών των στοιχείων, απαιτείται μια αλληλουχία κλήσεων. Η διαδικασία ξεκινά με την αναζήτηση των καθηγητών που διδάσκουν το επιλεγμένο μάθημα, βασιζόμενη στον τίτλο του. Αν το μάθημα εντοπιστεί, επιστρέφεται στο frontend μια λίστα με τους σχετικούς καθηγητές.

Εικόνα 8. Κώδικας υπηρεσίας επιστροφής καθηγητών που διδάσκουν ένα συγκεκριμένο μάθημα.

Στη συνέχεια, πραγματοποιούνται δύο ξεχωριστές κλήσεις: η μία για την απόκτηση της τιμής του μαθήματος και η άλλη για τη διάρκειά του. Εφόσον επιβεβαιωθεί η ύπαρξη του καθηγητή, τα δεδομένα επιστρέφονται με βάση το ID του επιλεγμένου μαθήματος, παρέχοντας την αντίστοιχη τιμή και διάρκεια.

Εικόνα 9. Κώδικες υπηρεσιών επιστροφής τιμής και διάρκειας μαθήματος.

Η διαδικασία επαναλαμβάνεται και ολοκληρώνεται όταν συγκεντρωθούν όλα τα απαραίτητα στοιχεία για κάθε μάθημα.

3.2.3 Αναζήτηση καθηγητών

Πέρα από την αναζήτηση μαθημάτων, ο χρήστης έχει την δυνατότητα να αναζητήσει καθηγητές. Με παρόμοιο τρόπο πραγματοποιείται η πρόσβαση στους καθηγητές από την βάση δεδομένων.

```
1 usage
public List<Instructor> getAllInstructors() { return instructorRepository.findAll(); }
```

Εικόνα 10. Κώδικας υπηρεσίας επιστροφής καθηγητών.

Έπειτα, με στόχο την εμφάνιση των μαθημάτων που διδάσκει ο κάθε καθηγητής καλείται η παρακάτω υπηρεσία που επιστρέφει όλα τα μαθήματά του, αφού πρώτα ελέγξει για την ύπαρξή του.

```
1 usage
public List<Course> getCoursesByInstructor(String username) {
    //check if the instructor exists
    Instructor instructor = instructorRepository.findByUsername(username)
            .orElseThrow(() -> new IllegalArgumentException("Instructor not found"));

    return instructor.getCourses();
}
```

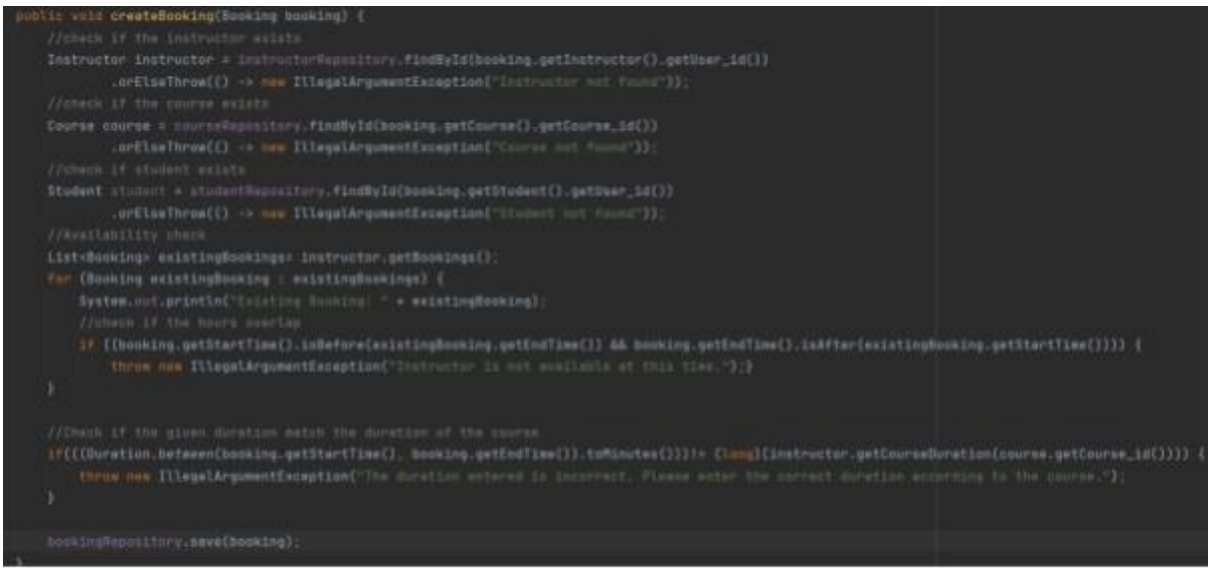
Εικόνα 11. Κώδικας υπηρεσίας επιστροφής μαθημάτων συγκεκριμένου καθηγητή.

Προφανώς, για την πρόσβαση στην τιμή και στην διάρκεια των μαθημάτων καλούνται οι υπηρεσίες που αναφέρθηκαν παραπάνω.

3.2.4 Πραγματοποίηση κράτησης

Η υπηρεσία πραγματοποίησης κράτησης αποτελεί την πιο βασική λειτουργία της εφαρμογής, η οποία συνδυάζει αρκετές οντότητες και απαιτεί πολλαπλούς ελέγχους για να είναι ορθή η επίτευξή της. Ειδικότερα, όταν ένας μαθητής εκδηλώνει ενδιαφέρον για την παρακολούθηση ενός μαθήματος, επιλέγει την ημερομηνία και την ώρα που επιθυμεί, οι οποίες οφείλουν να είναι συμβατές με τη διαθεσιμότητα του αντίστοιχου καθηγητή. Για τον λόγο αυτό, αφού επιβεβαιωθεί η ύπαρξη του καθηγητή, του μαθήματος και του μαθητή, πραγματοποιείται έλεγχος διαθεσιμότητας του καθηγητή για την επιλεγμένη ώρα. Στη συνέχεια, εξετάζεται αν η διάρκεια που επέλεξε ο μαθητής είναι συμβατή με τη διάρκεια που έχει ορίσει ο καθηγητής για το συγκεκριμένο μάθημα. Εφόσον όλοι οι έλεγχοι ολοκληρωθούν επιτυχώς, η βάση

δεδομένων ενημερώνεται με τη νέα κράτηση, ενώ ταυτόχρονα ενημερώνονται αυτόματα οι σχετικοί πίνακες του καθηγητή και του μαθητή.



Εικόνα 12. Κώδικας υπηρεσίας δημιουργίας κράτησης.

3.2.5 Διαγραφή κράτησης

Μια ακόμη απαιτητική διαδικασία είναι η διαγραφή μιας κράτησης, καθώς απαιτείται η σωστή ενημέρωση των σχετικών πινάκων στη βάση δεδομένων. Αρχικά, ενεργοποιείται η υπηρεσία που επιστρέφει τις κρατήσεις του μαθητή, υπό την προϋπόθεση ότι ο μαθητής είναι καταχωρημένος.

```
1 usage
public List<Booking> getBookingsByStudent(String username) {
    //check if the student exists
    Student student = studentRepository.findByUsername(username)
            .orElseThrow(() -> new IllegalArgumentException("Student not found"));

    return student.getBooking_history();
}
```

Εικόνα 13. Κώδικας υπηρεσίας επιστροφής κρατήσεων μαθητή.

Στη συνέχεια, εάν ο μαθητής επιθυμεί να διαγράψει κάποια κράτηση, καλείται η αντίστοιχη υπηρεσία η οποία, αφού επιβεβαιώσει την ύπαρξή του προχωρά στη διαγραφή της κράτησης, ενημερώνοντας αυτόματα τους πίνακες τόσο του καθηγητή όσο και του μαθητή.

```
1 usage
public void deleteBooking(Long bookingId) {
    //check if the booking exists
    Booking booking = bookingRepository.findById(bookingId)
            .orElseThrow(() -> new IllegalArgumentException("Booking not found"));

    bookingRepository.deleteById(bookingId);

}
```

Εικόνα 14. Κώδικας υπηρεσίας διαγραφής κράτησης.

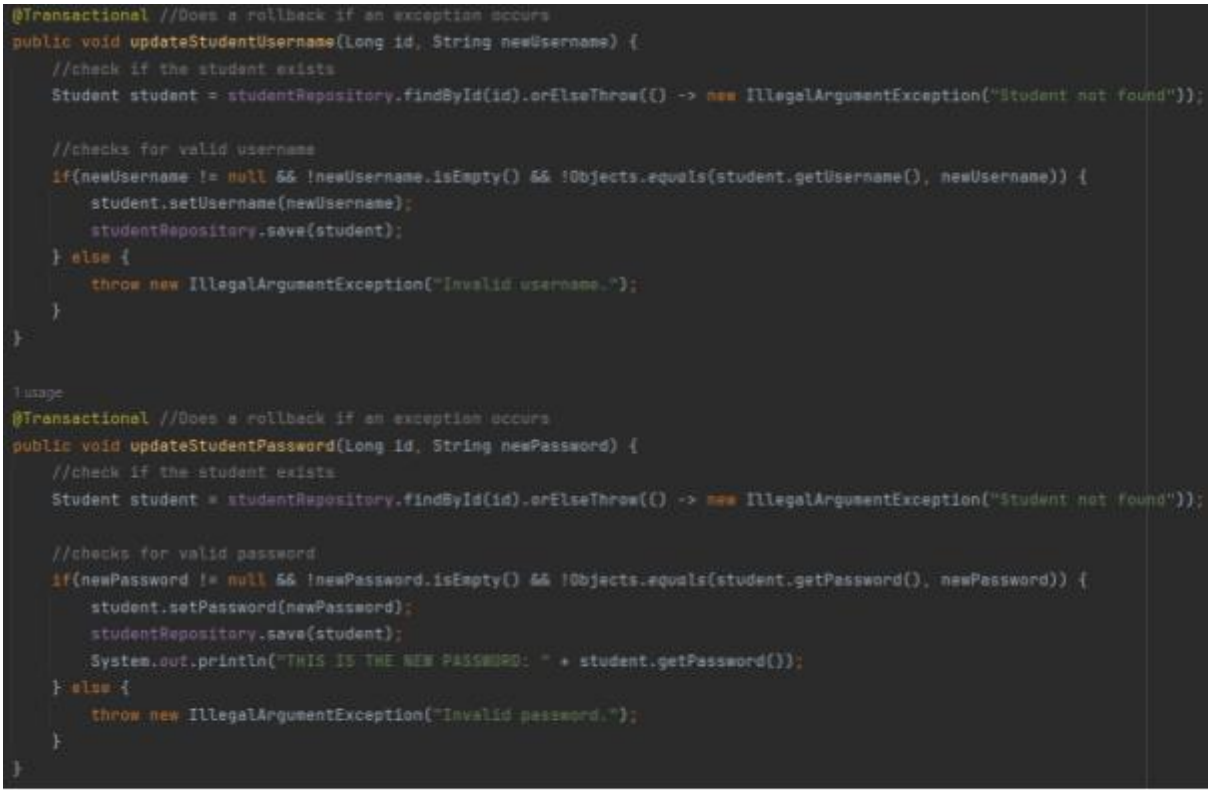
3.2.6 Εμφάνιση προσωπικών στοιχείων μαθητή και επεξεργασία τους

Η τελευταία ενέργεια της πλατφόρμας αφορά την εμφάνιση των προσωπικών στοιχείων του λογαριασμού του μαθητή και τη δυνατότητα επεξεργασίας του username και του password του. Αρχικά, καλείται η υπηρεσία που ανακτά τα στοιχεία του μαθητή βάσει του μοναδικού username, το οποίο πρέπει να υπάρχει στη βάση δεδομένων για να ολοκληρωθεί η διαδικασία.

```
1 usage
public Optional<Student> getStudentByUsername(String username) {
    //check if the student exists
    boolean exists = studentRepository.existsByUsername(username);
    if(!exists) {
        throw new IllegalArgumentException("Student doesn't exists");
    }
    return studentRepository.findByUsername(username);
}
```

Εικόνα 15. Κώδικας υπηρεσίας επιστροφής των στοιχείων του μαθητή.

Στη συνέχεια, εφόσον ο μαθητής επιθυμεί να αλλάξει το username ή το password του, ή και τα δύο, καλούνται οι αντίστοιχες υπηρεσίες. Και οι δύο υπηρεσίες πρώτα ελέγχουν αν ο μαθητής υπάρχει στη βάση δεδομένων και στη συνέχεια διασφαλίζουν ότι τα δεδομένα που δίνονται δεν είναι κενά ή ίδια με τα προηγούμενα, με τη διαφορά ότι η υπηρεσία για το username ελέγχει το νέο όνομα και η υπηρεσία για το password το νέο κωδικό. Εφόσον οι έλεγχοι είναι επιτυχείς, η βάση δεδομένων ενημερώνεται με τα νέα στοιχεία.

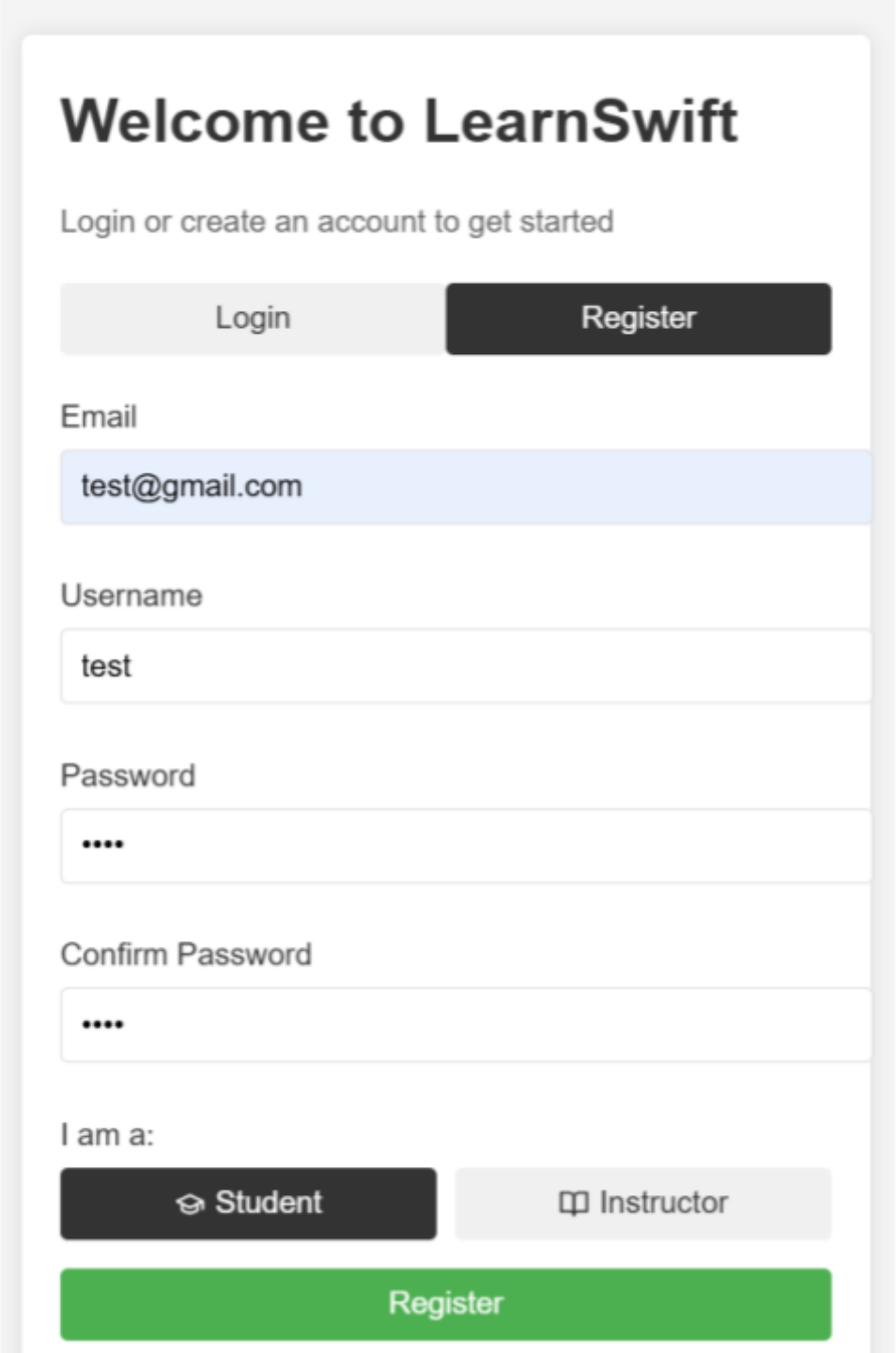


```
@Transactional //Does a rollback if an exception occurs
public void updateStudentUsername(Long id, String newUsername) {
    //check if the student exists
    Student student = studentRepository.findById(id).orElseThrow(() -> new IllegalArgumentException("Student not found"));

    //checks for valid username
    if(newUsername != null && !newUsername.isEmpty() && !Objects.equals(student.getUsername(), newUsername)) {
        student.setUsername(newUsername);
        studentRepository.save(student);
    } else {
        throw new IllegalArgumentException("Invalid username.");
    }
}

1 usage
@Transactional //Does a rollback if an exception occurs
public void updateStudentPassword(Long id, String newPassword) {
    //check if the student exists
    Student student = studentRepository.findById(id).orElseThrow(() -> new IllegalArgumentException("Student not found"));

    //checks for valid password
    if(newPassword != null && !newPassword.isEmpty() && !Objects.equals(student.getPassword(), newPassword)) {
        student.setPassword(newPassword);
        studentRepository.save(student);
        System.out.println("THIS IS THE NEW PASSWORD: " + student.getPassword());
    } else {
        throw new IllegalArgumentException("Invalid password.");
    }
}
```

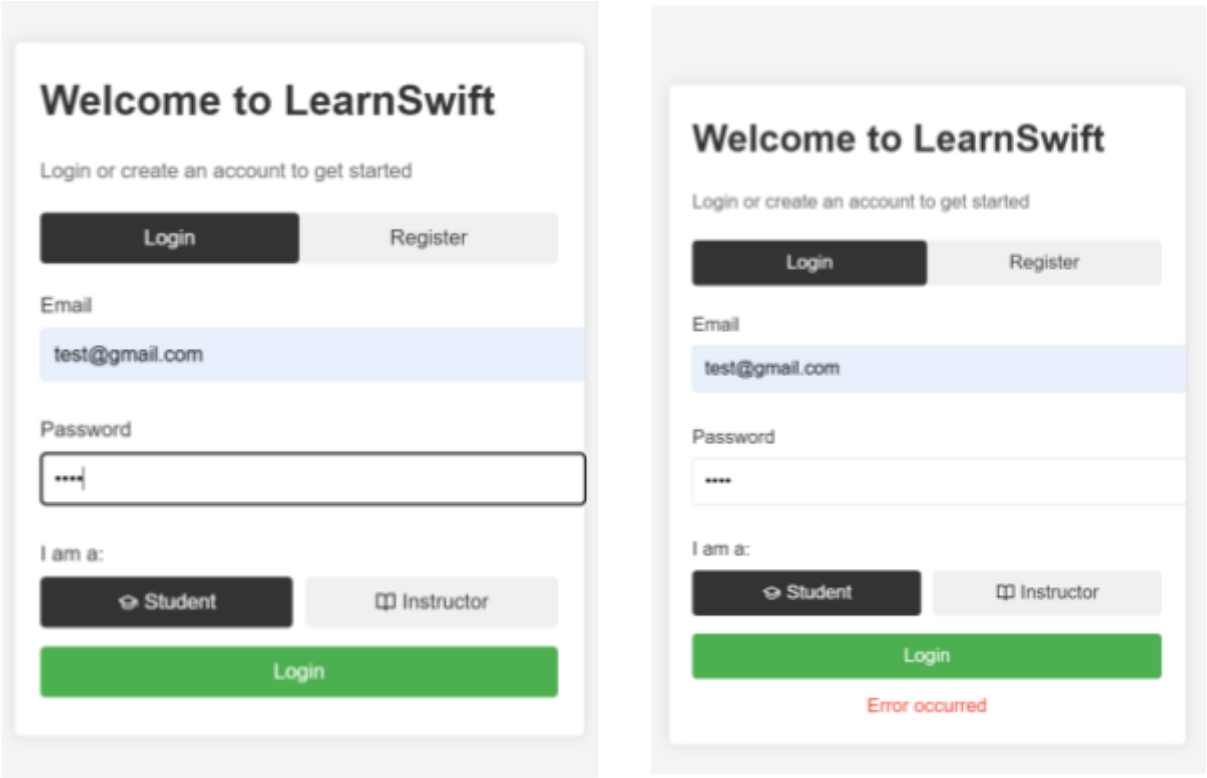
Εικόνα 16. Κώδικες υπηρεσιών επεξεργασίας username και password μαθητή.

4. Διεπαφή χρήστη

Επισκεπτόμενος τη διεύθυνση http://localhost:80/Login_Register/Login_Register.html, ο χρήστης βλέπει την οθόνη σύνδεσης ή εγγραφής. Στη συνέχεια, αφού συμπληρώσει τα απαιτούμενα πεδία, επιλέγει το κουμπί για να δηλώσει αν επιθυμεί να συνδεθεί ως μαθητής ή καθηγητής και εισέρχεται στην εφαρμογή. Σε περίπτωση λανθασμένων στοιχείων εισόδου, εμφανίζεται σφάλμα.

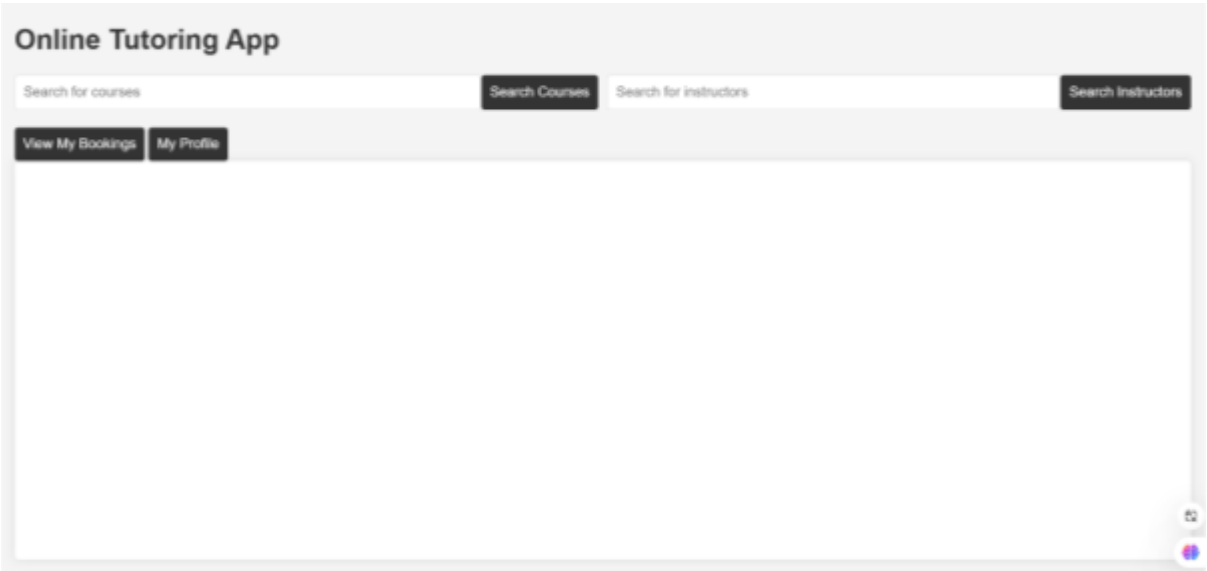


Εικόνα 17. Σελίδα εγγραφής χρήστη.



Εικόνα 18. Σελίδα σύνδεσης χρήστη.

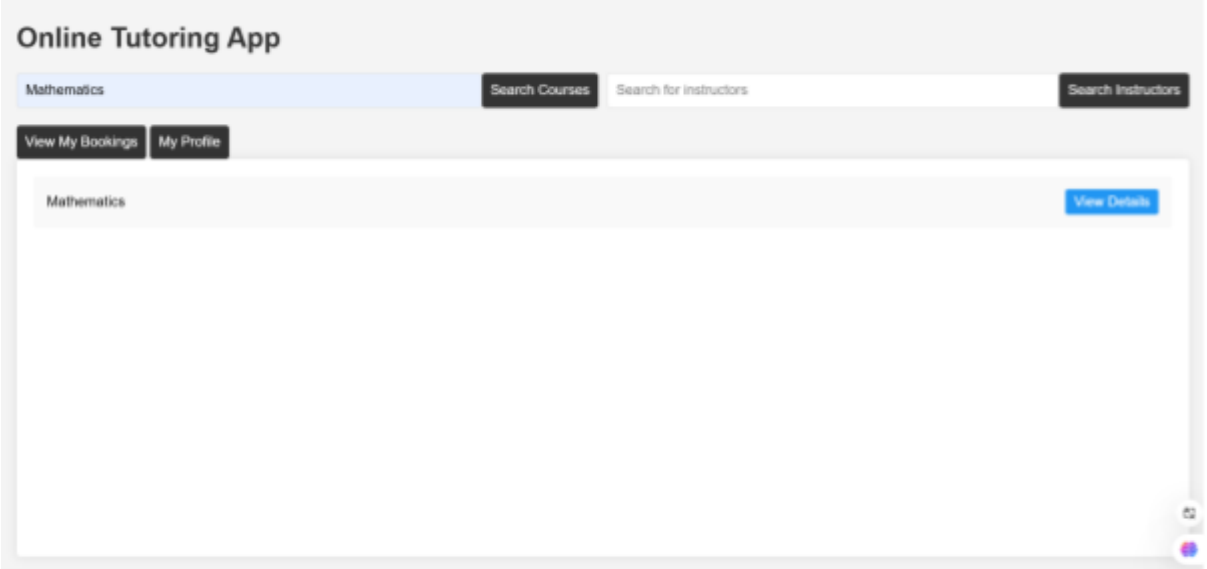
Μετά την επιτυχή του σύνδεση, ο χρήστης μεταβιβάζεται στην κύρια σελίδα της πλατφόρμας η οποία παρέχει τις υπηρεσίες που είναι διαθέσιμες.



Εικόνα 19. Κύρια σελίδα εφαρμογής.

Επιλέγοντας το κουμπί “Search Courses” εμφανίζονται όλα τα διαθέσιμα μαθήματα. Εναλλακτικά ο χρήστης έχει την δυνατότητα να πληκτρολογήσει τον τίτλο συγκεκριμένου μαθήματος και να το αναζητήσει.

Εικόνα 20. Αναζήτηση μαθημάτων.



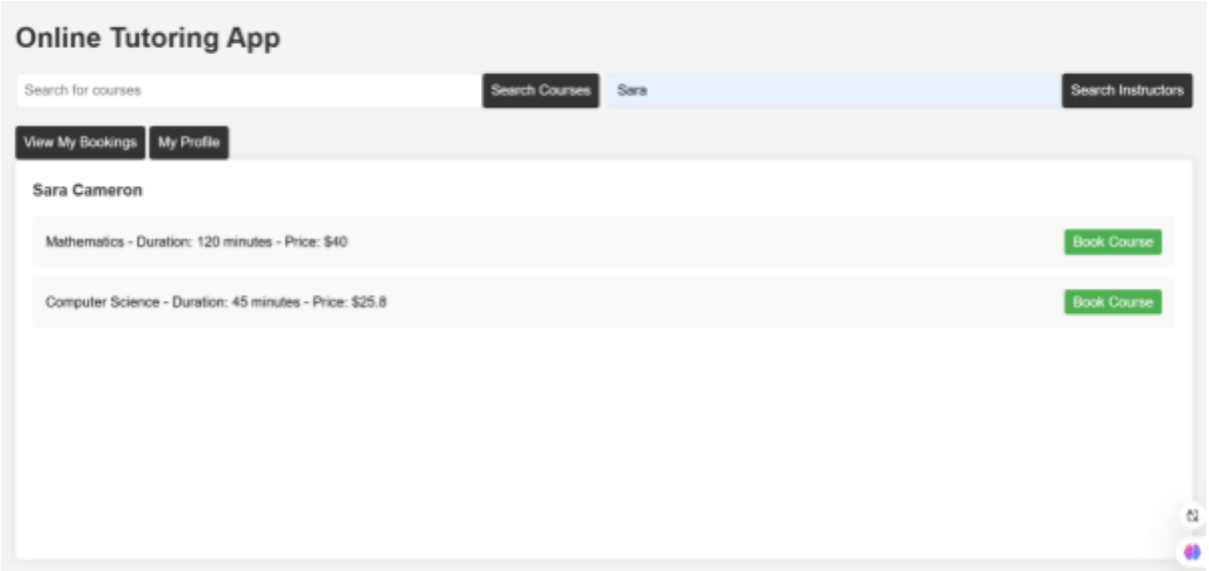
Εικόνα 21. Αναζήτηση συγκεκριμένου μαθήματος.

Με την επιλογή του κουμπιού “View Details,” ο χρήστης αποκτά πρόσβαση στις λεπτομέρειες του επιλεγμένου μαθήματος, συμπεριλαμβανομένων των καθηγητών που το διδάσκουν, καθώς και της διάρκειας και της τιμής που έχει ορίσει κάθε καθηγητής.

Εικόνα 22. Εμφάνιση πληροφοριών μαθήματος.

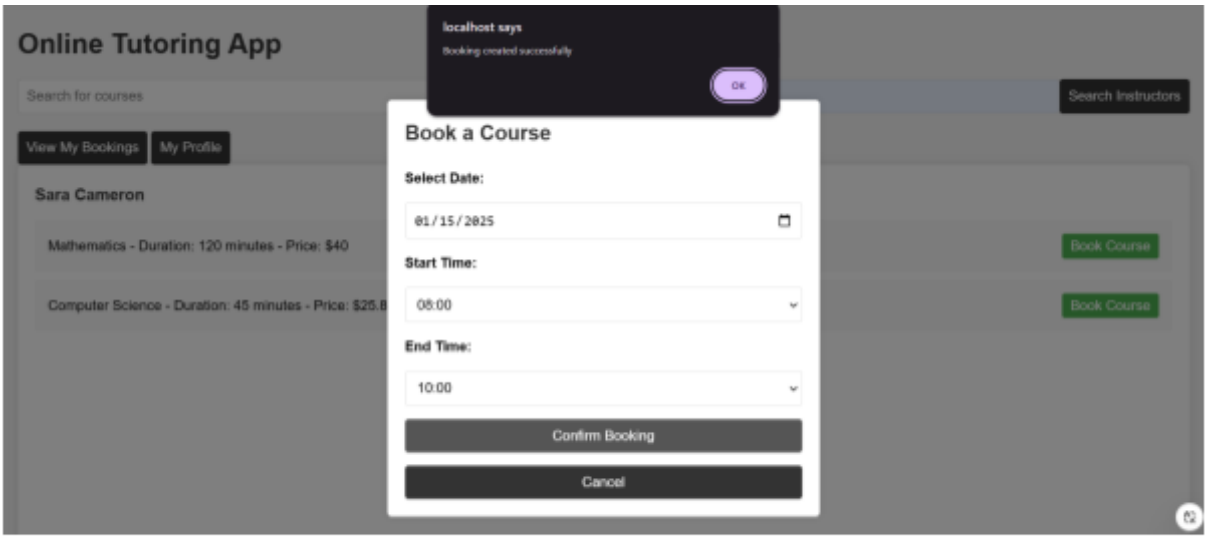
Με το κουμπί “Search Instructors,” ο χρήστης μπορεί να δει τη λίστα με όλους τους εγγεγραμμένους καθηγητές και τα μαθήματα που διδάσκουν. Παράλληλα, παρέχεται η δυνατότητα αναζήτησης ενός συγκεκριμένου καθηγητή.

Εικόνα 23. Αναζήτηση καθηγητών.

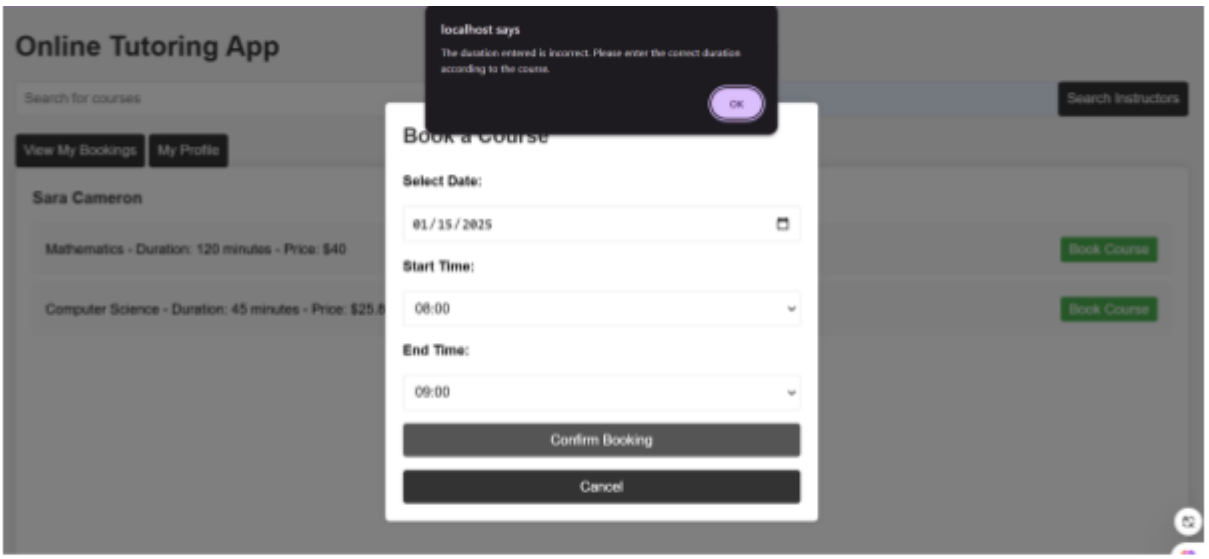


Εικόνα 24. Αναζήτηση συγκεκριμένου καθηγητή.

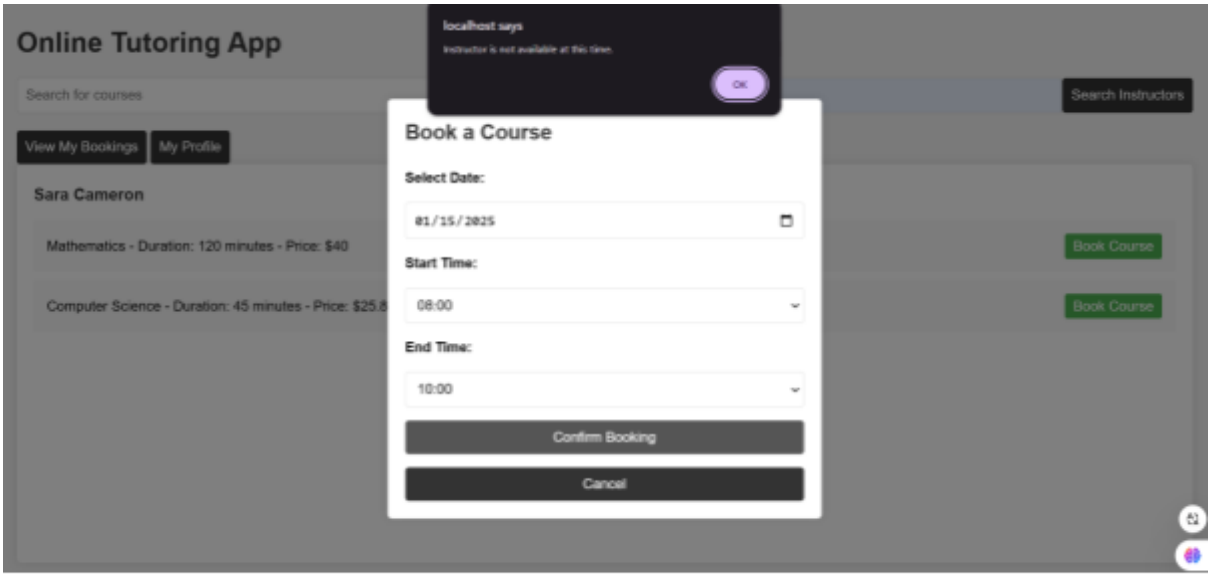
Η κράτηση μαθήματος μπορεί να γίνει με δύο τρόπους: είτε μέσω της διαδρομής “Search Courses” > “View Details” > “Book Course” είτε μέσω της επιλογής “Search Instructors” > “Book Course.” Πατώντας το κουμπί “Book Course,” ο χρήστης μεταφέρεται σε φόρμα όπου μπορεί να επιλέξει την ημερομηνία, την ώρα και τη διάρκεια που επιθυμεί να παρακολουθήσει το μάθημα. Αν η κράτηση ολοκληρωθεί με επιτυχία, εμφανίζεται το μήνυμα “Booking created successfully.” Αν η διάρκεια δεν αντιστοιχεί σε αυτή που έχει ορίσει ο καθηγητής για το μάθημα, εμφανίζεται το μήνυμα “The duration entered is incorrect. Please enter the correct duration according to the course.”. Τέλος, σε περίπτωση που ο καθηγητής δεν είναι διαθέσιμος την επιλεγμένη ώρα, εμφανίζεται το μήνυμα “Instructor isn’t available at this time.”



Εικόνα 25. Παράδειγμα επιτυχούς δημιουργίας κράτησης με την καθηγήτρια “Sara Cameron” για το μάθημα “Mathematics”.



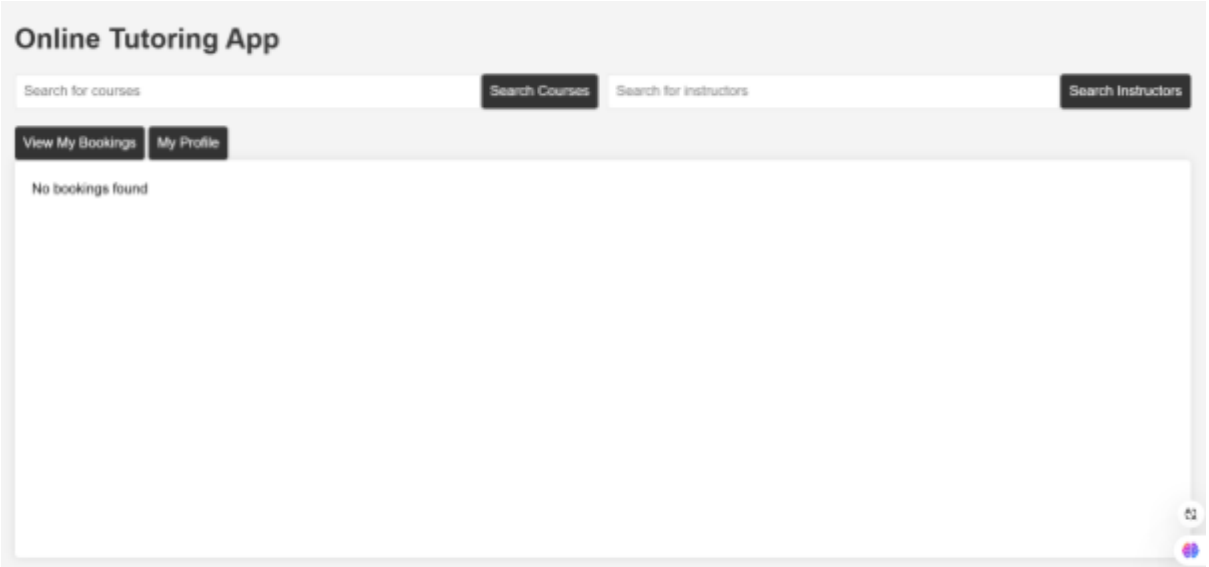
Εικόνα 26. Παράδειγμα αδυναμίας δημιουργίας κράτησης λόγω λανθασμένης διάρκειας. Το μάθημα "Mathematics" της "Sara Cameron" απαιτεί 120 λεπτά ενώ στην φόρμα έχουν δηλωθεί 90 λεπτά.



Εικόνα 27. Παράδειγμα αδυναμίας δημιουργίας κράτησης λόγω μη διαθεσιμότητας του καθηγητή.

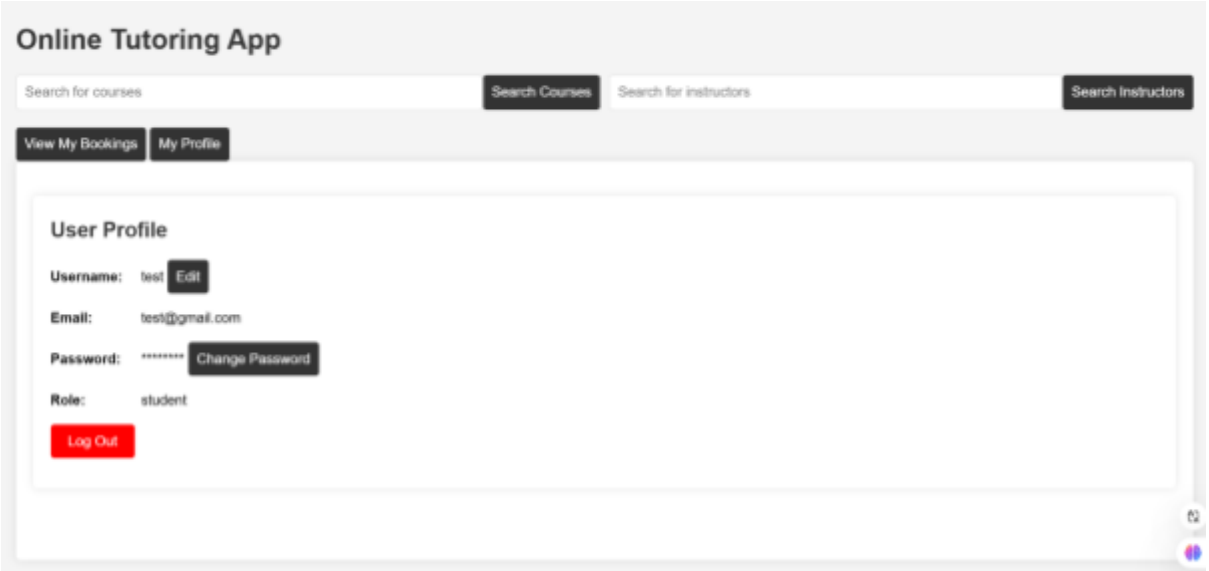
Για να ενημερωθεί ο χρήστης σχετικά με τις κρατήσεις που έχει δημιουργήσει, μπορεί να επιλέξει το κουμπί "View my Bookings." Επιπλέον, από την ίδια σελίδα, έχει τη δυνατότητα να διαγράψει μια κράτηση πατώντας το κουμπί "Delete."

Εικόνα 28. Σελίδα προβολής κρατήσεων.



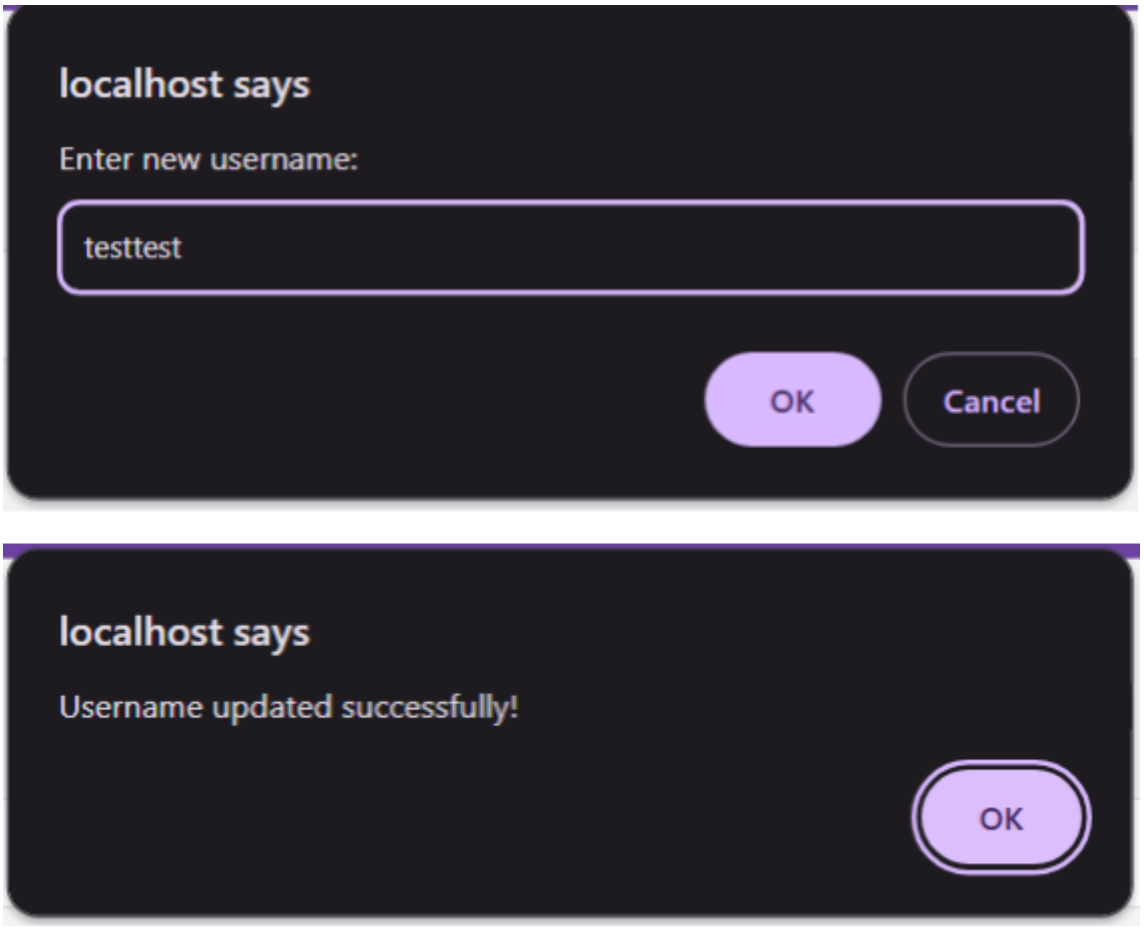
Εικόνα 29. Σελίδα προβολής κρατήσεων μετά την διαγραφή της κράτησης.

Επιλέγοντας το κουμπί “My Profile,” ο χρήστης μπορεί να δει τα στοιχεία του λογαριασμού του. Από εκεί, έχει τη δυνατότητα να αποσυνδεθεί μέσω του κουμπιού “Log Out” ή να επεξεργαστεί το username και το password του πατώντας το κουμπί “Edit” και “Change Password” αντίστοιχα.



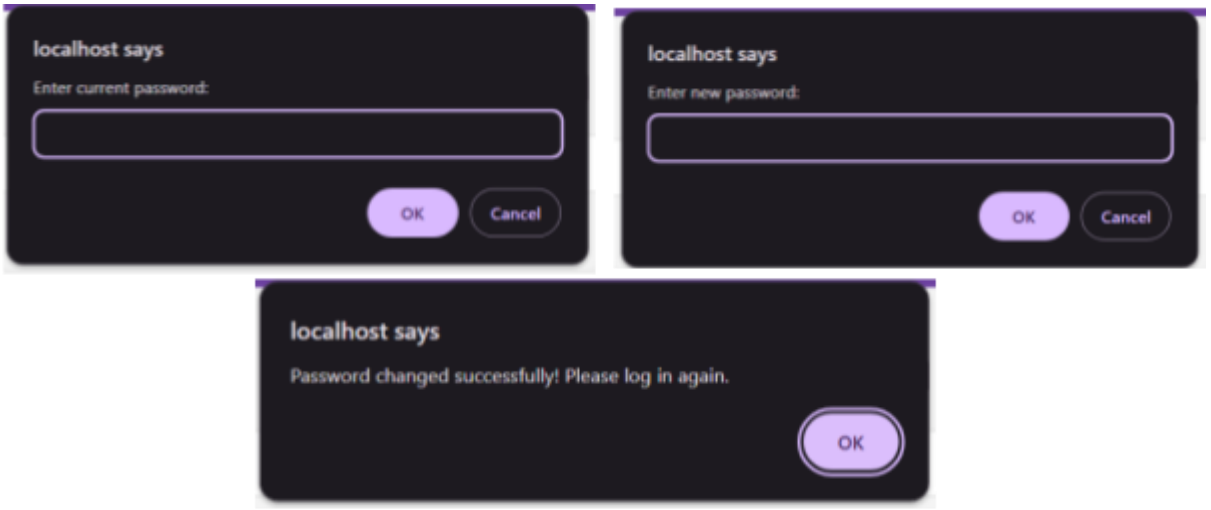
Εικόνα 30. Σελίδα προβολής στοιχείων χρήστη.

Όταν ο χρήστης επιλέξει να τροποποιήσει το username του, αναδύεται ένα παράθυρο όπου μπορεί να εισαγάγει το νέο επιθυμητό username. Εφόσον η αλλαγή ολοκληρωθεί με επιτυχία, το μήνυμα “Username changed successfully.”



Εικόνα 31. Τροποποίηση username.

Για την αλλαγή του κωδικού πρόσβασης ο χρήστης πρέπει πρώτα να εισαγάγει τον τρέχοντα κωδικό και στη συνέχεια τον νέο. Μετά την ολοκλήρωση της αλλαγής, απαιτείται να πραγματοποιήσει εκ νέου σύνδεση στον λογαριασμό του.



Εικόνα 32. Τροποποίηση κωδικού πρόσβασης.

5. Προγραμματισμένες Διορθώσεις και Επεκτάσεις

Η εφαρμογή LearnSwift βρίσκεται στο στάδιο ολοκλήρωσης, με κάποιες διορθώσεις και βελτιώσεις να έχουν ήδη προγραμματιστεί.

- Αρχικά, ο κωδικός πρόσβασης των χρηστών αποθηκεύεται απλώς ως string. Ωστόσο, για να διασφαλιστεί η ασφάλεια της εφαρμογής και η εμπιστοσύνη των χρηστών, μια πιο ενδεδειγμένη πρακτική θα ήταν η κρυπτογράφηση του κωδικού πριν από την αποθήκευσή του στη βάση δεδομένων.
- Επιπλέον, οι απαντήσεις του backend προς το frontend επιστρέφουν ολόκληρες τις οντότητες, γεγονός που προκάλεσε σημαντικές δυσκολίες, ιδιαίτερα στη διαχείριση των σχέσεων μεταξύ τους (ManyToMany, ManyToOne, OneToMany), για την αποφυγή ατέρμονων βρόχων δεδομένων. Για τον λόγο αυτό, αξιοποιήθηκαν annotations του Spring Boot, όπως τα @JsonIgnore, @JsonManagedReference και @JsonBackReference, τα οποία αγνοούν ή διαχειρίζονται κατάλληλα τα πεδία που προκαλούν επαναλήψεις στις αποκρίσεις. Για παράδειγμα, η σχέση ManyToMany μεταξύ των οντοτήτων Instructor και Course είχε ως αποτέλεσμα, όταν γινόταν κλήση για τους καθηγητές, να εμφανίζονται τα μαθήματα που διδάσκουν, στη συνέχεια για τα μαθήματα να εμφανίζονται ξανά οι καθηγητές, και ούτω καθεξής, δημιουργώντας έναν ατέρμονο κύκλο δεδομένων. Μια πιο αποτελεσματική προσέγγιση θα ήταν η χρήση κλάσεων DTO για κάθε οντότητα. Με αυτήν την τεχνική, παρά το γεγονός εμφάνισης διπλότυπου κώδικα, αντί να επιστρέφεται ολόκληρη η οντότητα στο frontend, επιστρέφονται μόνο τα απαραίτητα πεδία

ανά περίπτωση, διευκολύνοντας τη διαχείριση δεδομένων και εξαλείφοντας τον κίνδυνο επαναλαμβανόμενων στοιχείων.

- Κατά την περαιτέρω ανάπτυξη της εφαρμογής, παρατηρήθηκε ένα ζήτημα όταν επιχειρείται η δημιουργία πολλαπλών κρατήσεων για τον ίδιο καθηγητή στο ίδιο μάθημα, την ίδια μέρα αλλά σε διαφορετικές ώρες. Σε αυτή την περίπτωση, εμφανίζεται μήνυμα ότι υπάρχει ήδη κράτηση με το συγκεκριμένο ID, το οποίο όμως δημιουργείται αυτόματα από το Spring Boot για κάθε νέο αντικείμενο. Το φαινόμενο βρίσκεται υπό διερεύνηση, ώστε να διασφαλιστεί ότι η δημιουργία μοναδικών κρατήσεων γίνεται με συνέπεια, σύμφωνα με τη λογική της εφαρμογής.

Τέλος, η εφαρμογή LearnSwift προσφέρει πολλές δυνατότητες για μελλοντικές βελτιώσεις, προκειμένου να ενισχυθεί η εμπειρία του χρήστη και να καλυφθούν νέες ανάγκες. Για παράδειγμα, η προσθήκη αξιολογήσεων καθηγητών θα επιτρέπει στους μαθητές να επιλέγουν ευκολότερα τον κατάλληλο καθηγητή για τα μαθήματά τους. Επίσης, μια μελλοντική επέκταση αφορά την σελίδα για τους χρήστες που συνδέονται ως καθηγητές αντί για μαθητές. Αυτή η σελίδα θα περιλαμβάνει τα στοιχεία του λογαριασμού του καθηγητή, τις κρατήσεις των μαθημάτων του, καθώς και ένα τμήμα διαχείρισης των μαθημάτων του, δίνοντάς του τη δυνατότητα να προσθέτει, να αφαιρεί ή να τροποποιεί τις τιμές και τη διάρκεια των μαθημάτων του. Είναι σημαντικό να τονιστεί ότι όλες αυτές οι λειτουργίες έχουν ήδη υλοποιηθεί στο backend, αλλά το αντίστοιχο frontend δεν έχει αναπτυχθεί ακόμη, ώστε να καταστούν πλήρως λειτουργικές.