UNIVERSITY OF THESSALY

SCHOOL OF ENGINEERING

DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

WEB APPLICATION FOR PEER-TO-PEER EVALUATION FOR STUDENT HOMEWORK

Diploma Thesis

Nikolaos Mpaltas

Supervisor: Georgios Thanos

March 2024

UNIVERSITY OF THESSALY

SCHOOL OF ENGINEERING

DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

WEB APPLICATION FOR PEER-TO-PEER EVALUATION FOR STUDENT HOMEWORK

Diploma Thesis

Nikolaos Mpaltas

Supervisor: Georgios Thanos

March 2024

ΠΑΝΕΠΙΣΤΗΜΙΟ ΘΕΣΣΑΛΙΑΣ

ΠΟΛΥΤΕΧΝΙΚΗ ΣΧΟΛΗ

ΤΜΗΜΑ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

ΔΙΑΔΙΚΤΥΑΚΗ ΕΦΑΡΜΟΓΗ ΟΜΟΤΙΜΗΣ ΑΞΙΟΛΟΓΗΣΗΣ ΕΦΑΡΜΟΓΩΝ

Διπλωματική Εργασία

Νικόλαος Μπαλτάς

Επιβλέπων/πουσα: Γεώργιος Θάνος

Μάρτιος 2024

Approved by the Examination Committee:

Supervisor **Georgios Thanos**
Laboratory Teaching Staff member, Department of Electrical and Computer Engineering, University of Thessaly

Member **Athanasios Fevgas**
Laboratory Teaching Staff member, Department of Electrical and Computer Engineering, University of Thessaly

Member **Xarikleia Tsalapata**
Laboratory Teaching Staff member, Department of Electrical and Computer Engineering, University of Thessaly

DISCLAIMER ON ACADEMIC ETHICS AND INTELLECTUAL PROPERTY RIGHTS

«Being fully aware of the implications of copyright laws, I expressly state that this diploma thesis, as well as the electronic files and source codes developed or modified in the course of this thesis, are solely the product of my personal work and do not infringe any rights of intellectual property, personality and personal data of third parties, do not contain work / contributions of third parties for which the permission of the authors / beneficiaries is required and are not a product of partial or complete plagiarism, while the sources used are limited to the bibliographic references only and meet the rules of scientific citing. The points where I have used ideas, text, files and / or sources of other authors are clearly mentioned in the text with the appropriate citation and the relevant complete reference is included in the bibliographic references section. I also declare that the results of the work have not been used to obtain another degree. I fully, individually and personally undertake all legal and administrative consequences that may arise in the event that it is proven, in the course of time, that this thesis or part of it does not belong to me because it is a product of plagiarism».

The declarant

Nikolaos Mpaltas

Diploma Thesis

WEB APPLICATION FOR PEER-TO-PEER EVALUATION FOR STUDENT HOMEWORK

Nikolaos Mpaltas

Abstract

This diploma thesis explores the field of educational technology, specifically the design and implementation of a web application to facilitate peer-to-peer evaluation of student homework. The study was motivated by the need to create an efficient and effective platform for lesson management and student collaboration, emphasizing peer-to-peer homework evaluation. The main functionality of the application, along with the features it provides for teachers and students, will be discussed in the functional description. Teachers can create lessons and assign homework tasks specifying if they are to be peer-reviewed, while students can submit their homework and take part in peer reviews. Moving on, the tools used to develop the frontend and backend parts of the application will be mentioned, followed by the technical description of the application. The technical description analyzes the implementation details and the technologies used, including React and Express. These technologies contribute to the application's robustness and streamlined functionality.

Keywords:

Web Application, Peer-to-peer evaluation, React.js, Express.js, Node.js, MySQL

Διπλωματική Εργασία

ΔΙΑΔΙΚΤΥΑΚΗ ΕΦΑΡΜΟΓΗ ΟΜΟΤΙΜΗΣ ΑΞΙΟΛΟΓΗΣΗΣ ΕΦΑΡΜΟΓΩΝ

Νικόλαος Μπαλτάς

Περίληψη

Η παρούσα διπλωματική εργασία διερευνά τον τομέα της εκπαιδευτικής τεχνολογίας, και συγκεκριμένα τον σχεδιασμό και την υλοποίηση μιας διαδικτυακής εφαρμογής για τη διευκόλυνση της ομότιμης αξιολόγησης των εργασιών των μαθητών. Αφορμή για τη μελέτη αποτέλεσε η ανάγκη δημιουργίας μιας αποδοτικής και αποτελεσματικής πλατφόρμας για τη διαχείριση μαθημάτων και τη συνεργασία των μαθητών, με έμφαση στην ομότιμη αξιολόγηση των εργασιών. Η κύρια λειτουργικότητα της εφαρμογής, μαζί με τις δυνατότητες που παρέχει στους εκπαιδευτικούς και τους μαθητές, θα συζητηθούν στη λειτουργική περιγραφή. Οι καθηγητές μπορούν να δημιουργούν μαθήματα και να αναθέτουν εργασίες, διευκρινίζοντας αν πρόκειται να αξιολογηθούν ομότιμα, ενώ οι μαθητές μπορούν να υποβάλλουν τις εργασίες τους και να συμμετέχουν σε αξιολογήσεις. Προχωρώντας, θα αναφερθούν τα εργαλεία που χρησιμοποιήθηκαν για την ανάπτυξη του frontend και του backend της εφαρμογής, ενώ θα ακολουθήσει η τεχνική περιγραφή της εφαρμογής. Η τεχνική περιγραφή αναλύει τις λεπτομέρειες της υλοποίησης και τις τεχνολογίες που χρησιμοποιήθηκαν, συμπεριλαμβανομένων των React και Express. Οι τεχνολογίες αυτές συμβάλλουν στην σταθερότητα και την εξορθολογισμένη λειτουργικότητα της εφαρμογής.

Λέξεις-κλειδιά:

Web εφαρμογή, ομότιμη αξιολόγηση, React.js, Express.js, Node.js, MySQL

Table of contents

List of figures

List of tables

Abbreviations

API	Application Programming Interface
SPA	Single Page Application
REST	Representational State Transfer
ORM	Object Relational Mapping
SQL	Structured Query Language
RDBMS	Relational Database Management System
NPM	Node Package Manager
HTTP	Hypertext Transfer Protocol
JWT	JSON Web Token
URL	Uniform Resource Locator
UI	User Interface
IP	Internet Protocol
DoS	Denial of Service

Chapter 1

Introduction

The field of digital education has been swiftly evolving in the last years, driving transformations not just for students but also for teachers. For teachers, clear benefits include access to resources and the ability to track and monitor student progress. However, there are challenges such as creating effective peer review systems, creating personalized learning journeys, and continuously engaging students. The advantages for students are numerous, including the ability to learn at their own speed, access to a wide range of educational resources, and peer-based learning. However, the abundance of information available can also pose challenges in terms of maintaining active engagement and tracking personal academic progress. One particular challenge in this domain is managing and effectively conducting peer review activities, which can be essential for student development and collaborative learning.

1.1 Main Objective

The focal point of this work is the development and deployment of a dynamic web application, highlighting an approach to resolving distinct pressing issues within the digital education field. Utilizing modern web technologies, including React.js and Express.js, specific solutions are targeted.

1. Streamlining the creation and assignment process of homework activities with provisions for independent tasks or peer-reviewed assignments, relieving teachers from administrative burdens.

2. Facilitating manageable oversight of user roles for administrators, ultimately simplifying management tasks.

3. Providing an intuitive learning platform for students to engage with lessons, participate in peer review, and complete assigned tasks. This directly addresses the challenges of student engagement and self-management in their academic progress.

1.1.1 Similar Applications

While no specific application combines code review and peer feedback into a cohesive tool designed for students, there are several that offer individual aspects of our application's functionality.

Code Review Tools

1. **Gerrit [2] and Collaborator by SmartBear [3]**: Both are excellent examples of professional code review tools. They allow for feedback on code changes and foster collaboration between team members. While they provide robust tools for code review, they are not centered around a student learning environment and do not offer a built-in system for peer-to-peer review assignments.

Peer Review Tools

1. **Instructure's Peer Review Assignments [4]**: This tool allows instructors to create assignments that require students to give feedback on their peer's work. While assignment-based reviews are supported, it does not cater specifically to reviewing code files.

These examples highlight how our application fills a unique niche, bridging the gap between professional code review tools and educational peer review platforms. By offering both code review capabilities and peer feedback functionality in a student-oriented context, our application meets a need that no singular tool currently addresses.

1.1.2 Contribution

The specific contributions of this thesis are:

1. A comprehensive study was conducted to underline and understand the requirements of the application.

2. Based on the requirements study, appropriate development tools were selected.

3. Utilizing the selected tools, a user-friendly frontend was designed and developed.
4. A robust and scalable database was created to manage and store the application's data.
5. The backend logic was designed and deployed, connecting the frontend to the backend and enabling proper data management.
6. A system was designed for administrators to manage user roles effectively.
7. A module was designed for peer review, allowing teachers to create peer-review-based assignments and students to participate effectively.
8. Ensuring the application met all its designed requirements, it was successfully deployed.

1.2 Structure

Chapter 1 of this diploma thesis presents the main idea and purpose for building this web application as well as work related to the subject. Chapter 2 provides an overview of the system's operations, features, and interactions from the user's perspective. Chapter 3 delves into the tools and packages used during the application development process. In Chapter 4, the system's architecture is described in detail, including the system design, database design, and how various components work together to achieve the desired functionality. Lastly, Chapter 5 explores potential advancements, improvements, or adaptations of the system and how these could enhance the functionality and user experience.

Chapter 2

Functional Description

2.1 Introduction

This chapter explores the web application from the perspectives of the three main roles: student, teacher and admin. It is essential to understand the functionality of these roles for reliable app operation and management. We will examine the app's main features for each role and how the user experience was tailored to meet the needs of each role's functionality.

2.2 Main Page / User Sign In

When first navigating to our application, the user encounters a login screen. This screen contains a form that requires the user's credentials. If the provided credentials are incorrect, an error message is displayed showing the reason for the unsuccessful submission. Otherwise, the user is redirected to the main page of his role in our application.

- **Initial Access:** User accesses the web application and is presented with the login screen.
- **Credentials Input:** User enters their unique username/email and password.
- **Form Submission:** User submits the login form, triggering authentication processes.
- **Authentication Check:** System validates entered credentials against database records.
- **Error Handling:** If credentials mismatch, an error message is displayed and the user is prompted to re-enter credentials or retrieve forgotten details.

- **Successful Login:** If credentials match, the user is redirected to their respective dashboard based on their user role.

2.3 User Registration

The registration process is the user's first interaction with out application. This process is initiated by navigating to the sign-up section of the web application. They then must fill out a registration form with various fields, including their username, email and password. If the form is submitted successfully, an email is sent to the user's provided email address. The email contains the confirmation token that will be used for confirming the email and must be used within some minutes. Upon successful email confirmation, the users can then use the confirmed email and password to login in the application. Despite the process having multiple steps, it guarantees that all users who have registered are legitimate and can use their respective functionality.

2.4 Admin functionality

The admin role within the application, has a critical set of functionalities focused on user management. This role has the unique capability to overview all users within the system through a paginated table. This provides a comprehensive view of all registered users, providing an easy and organized way of managing users. Data such as usernames, emails associated roles and other sensitive information is readily available within this table providing a holistic understanding of the user landscape. Beyond viewing user data , the admin holds the authority to assign user roles. They can assign or modify the roles of student, teacher, or admin to any user. This process of role assignment is a critical admin function given its impact on governing user access and permissions across the platform. Every change done by that main user has real time effects, ensuring the system remains updated and serves as a reflection of the current user setup.

2.5 Teacher functionality

2.5.1 Lesson Creation

One of the capabilities of the teacher role is the lesson creation. Within this function, teachers can generate new lessons that reflect their academic approach and accommodate the learning needs of their students. The creation process involves filling out a form with pertinent information, including the lesson title, description and other necessary details. Upon the completion and submission of the lesson creation form, the system dynamically updates, making the new lesson available within the web application.

2.5.2 Management of enrollment requests

Enrollment request handling is another essential element of the teacher's position. Once a lesson is created, students can express an intent to join a specific lesson by sending an enrolment request. Teachers have the ability to monitor these requests through a dedicated paginated table that displays the pending and completed enrolments. Each request can be reviewed and be accepted or declined. Acceptance of a request grants the student access to the lesson while declining a request keeps the students from accessing the lesson content. This feature allows for active management of class sizes, ensuring appropriate ratios for a learning environment. The ability to control enrolments not only keeps the lessons organized but also empowers teachers to curate their teaching focus to a manageable student group.

2.5.3 Homework Creation

Another important feature in the functionality of the teacher role is the ability to create homework assignments under their created lessons. The homework creation process involves completing a form that takes inputs like the homework name, a detailed description, and a submission deadline.

Not peer reviewed

For non-peer-reviewed homework, the process stops at the submission deadline input. The primary focus of this type of homework is for the student to individually prove his grasp of the curriculum without involving their peers in the evaluation process.

Peer Reviewed

For peer-reviewed homework, there are some additional fields in the homework creation form where they can specify the number of peer reviewers and set a review submission deadline. The aim is to foster a collaborative learning environment where students critically assess each other's work and learn from one another through the process.

2.5.4 Homework Question

Review Question Creation

Under peer-reviewed homework, teachers also need to create a series of questions that guide the peer-review process. These questions are critical in shaping the direction of the review and ensuring it covers essential aspects of the homework assignment. The questions can be of three types: number, true/false, or text. Furthermore, each question contains the weight and points it gives for correct answers.

Review Question Editing

Teachers are given the option to edit the homework questions they have created. This provides the ability to adjust the peer review process as needed.

2.5.5 Homework Review Interface

Teachers have access to a comprehensive homework review interface that includes a table to manage homework submissions effectively.

Submissions and Peer Reviews Overview

This table directly lists all student submissions for each homework assignment, providing an organised and efficient way of visual representation. Each row in the table represents an individual student's submission and if expanded, contains not only the submission itself but also the subsequent peer reviews made on that particular submission. This provides teachers with a snapshot of each student's performance and the peer feedback they have received. This structure facilitates easy navigation and quick understanding, thus allowing the teachers to monitor the peer review process effectively against each submission.

Grades and Submission Files

Furthermore, within this interface, teachers are given the ability to export the grading results of all the peer reviews in a specific homework. This final grade file enables the teachers to manage grading records easily and carry out further analysis if required. Also the system allows for the direct download of individual submission files, which can then be reviewed offline or stored for future reference. This comprehensive access to data reinforces the teacher's role in supervising the overall learning process.

2.6 Student functionality

2.6.1 Lesson enrollment request

One of the primary functionalities the students have is viewing the available lessons within the web application. Each displayed lesson contains information such as the lesson's topic, the involved teacher, and a brief description. They also have the ability to send a request to join a lesson that the teacher has to approve. This procedure makes the engagement between the teacher and the student more direct.

2.6.2 See Available Homework

Besides lesson enrollments, students have access to a directory within each lesson that contains all the available homework assignments. Each available homework is listed with details such as the name, a description and the deadline for the submission and the review if the homework is peer-reviewed. This enables students to view all the information related to their homework in one place and to adhere to the deadlines.

2.6.3 Homework Upload

Within each homework, there is a form where students can upload their completed work. Once the submission file is uploaded, it immediately becomes available in the system, and a notification is sent to the user displaying if the upload was successful. Through this, the submission functionality not only allows for an easy upload of homework but also contributes to maintaining the smooth flow of the homework process.

2.6.4 Reviewing Peer's Homework

In the case where the homework is peer-reviewed, students also have the ability to enhance their learning by reviewing their peer's assignments. In our case the system provides an in-built text editor component used to access the uploaded submission files of allocated peer work. This editor ensures smooth access to the content, irrespective of the file type. With these files available, students can then review the work, provide feedback, and answer specific questions that were set by the teacher. During this review process, they can continually reference the original submissions through the editor, enabling a refined and detailed review. This not only supports the communal exchange of knowledge but also encourages students to develop the skill of critically analyzing other's work.

Chapter 3

Development and Deployment Tools

3.1 Github

Github is a development platform that offers distributed version control and source code management functionality in combination with its unique features which enhance the developer experience. Well recognized for its capabilities and increasing popularity, GitHub seamlessly integrates with numerous cloud providers including Azure and Vercel. This streamlined integration supports the direct deployment of applications from a GitHub repository, providing a continuous workflow from code development to production deployment.

3.2 Database

The database is a key component in web applications as this is the place where that data is stored. In our web application, a relational database was used. Relational databases are based on the relational model which is a way of representing data in tables. MySQL [5] is an open-source relational database management system that is compatible with many programming languages. It offers strong transactional support for applications that require performing complex queries and managing large amounts of data and is also compatible with many cloud hosting providers. Furthermore, it offers high scalability, ensuring that with a volume increase in traffic, the performance of the database remains optional.

3.2.1 Setup

For our database, we used the cloud service Database for MySQL - Flexible server. This cloud solution has extra features in comparison with traditional databases, such as multi-region support, scalability, reduced latency, and an uptime guarantee.

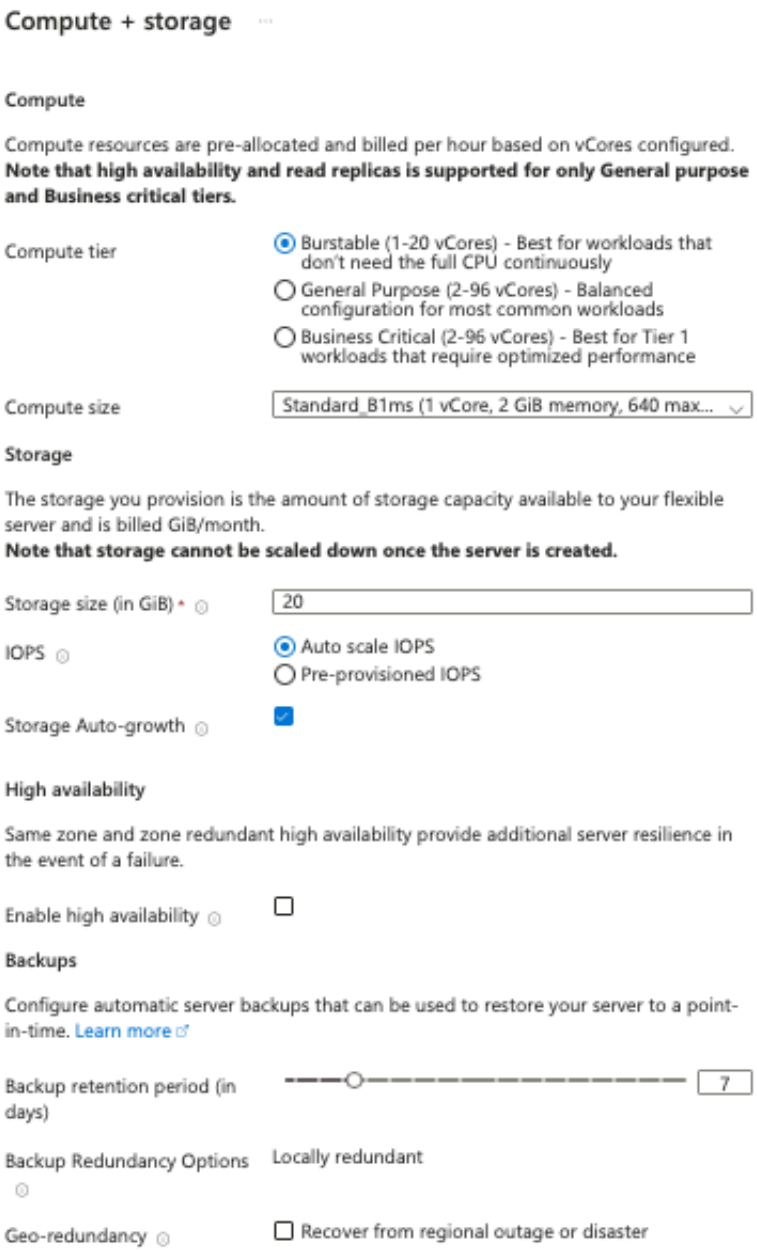

Figure 3.1: Compute and storage configuration

Azure allows you to host an SQL server on their cloud environment and utilize a wide range of management, scalability, and configuration features. Setting up a database in Azure involves the creation of an SQL server instance and requires a unique server name which is

a unique identifier of the database instance. After that, an SQL version must be selected as well as the region in which the instance exists.

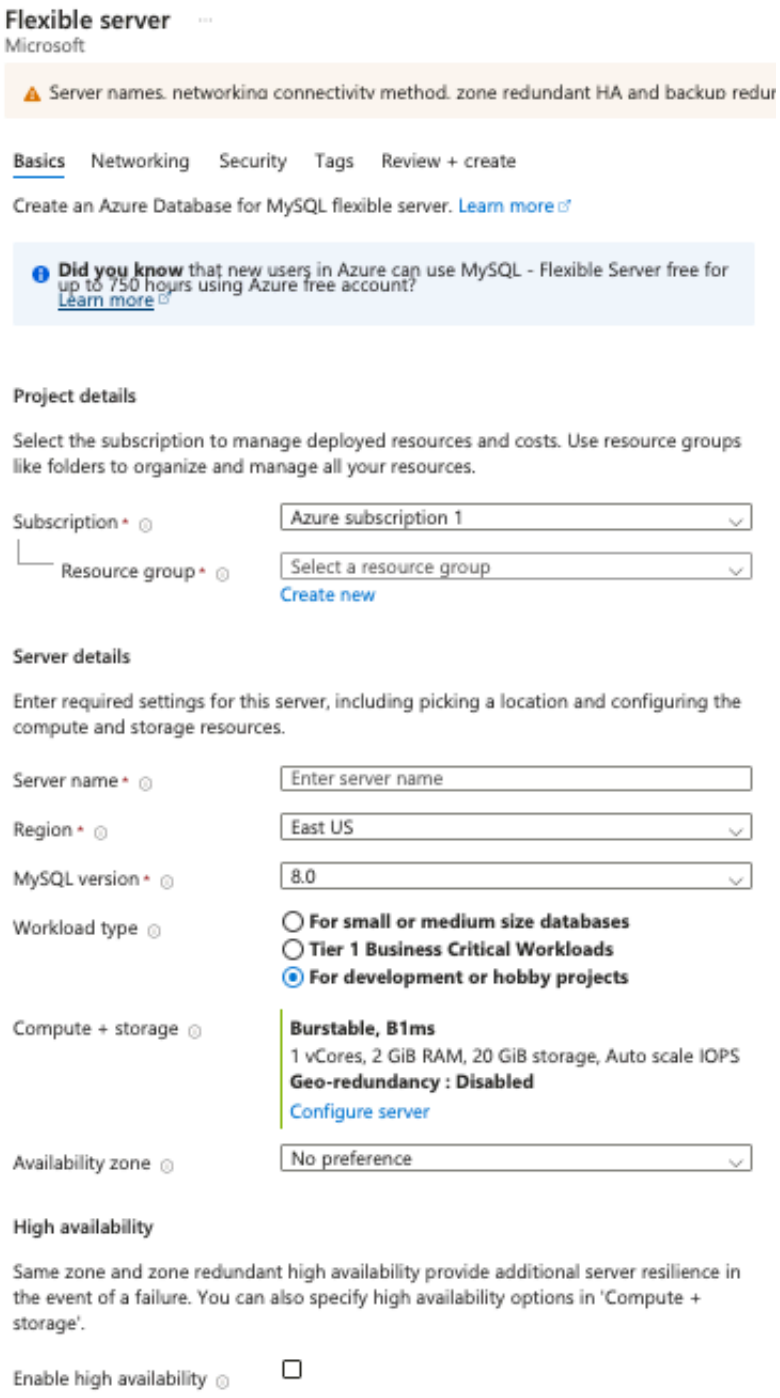

Figure 3.2: Flexible Database for MYSQL configuration

After successfully setting up the SQL server that Azure provides, there are some instance options that can be configured based on the expected workload of the application. For example, the storage and CPU resources of the database can be configured.

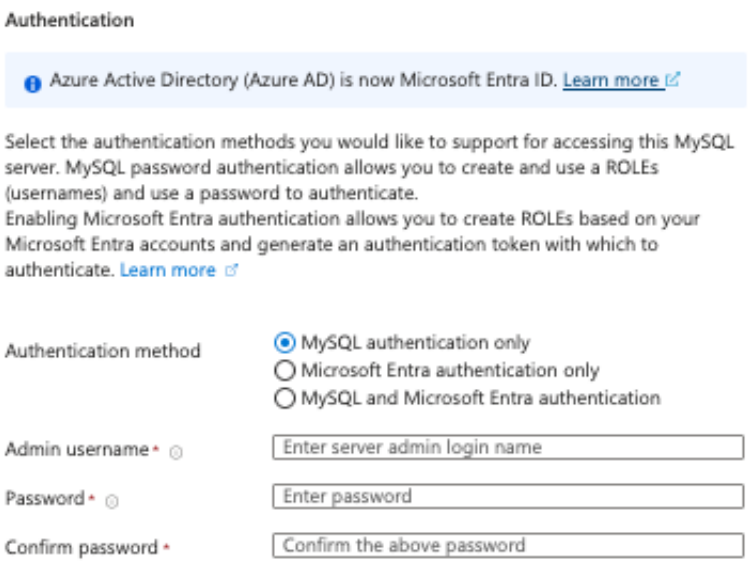

Figure 3.3: Defining the database credentials

Securing the database is crucial, and one of the elements of this security lies in the credentials required to connect to the hosted database. Azure provides two authentication methods SQL Authentication and Azure Active Directory Authentication. The SQL Authentication method was used. This method provides login credentials that can be used to connect to the database instance.

3.2.2 Connecting the database to the Backend

After the successful creation of the database instance, Azure MYSQL- Flexible Server contains many connectivity options are given from the Azure cloud service. In our application, we used a connection string containing the database credentials and the database name, to authenticate the connection between the database and the server. Lastly, Azure provides an SSL certificate which establishes secure and encrypted communication between the server and the database.

```
async function assertDatabaseConnectionOk() {
  console.log("Testing the database connection..");
  try {
    await sequelize.authenticate();
    console.log("Connection has been established successfully.");
  } catch (error) {
    console.error("Unable to connect to the database:", error);
  }
}
```

Figure 3.4: Authentication process using Sequelize ORM

3.3 Backend

This subsection presents a detailed examination of our backend architecture that is primarily built upon Node.js and is further complemented with several npm modules. For the project initialization, we used npm init, a command to create a new Node.js project which creates a package.json file. This file acts as the project manifest, containing metadata about the application and for keeping track of the module dependencies of the project. Moving on to the following subsections we will further the discuss the backend components, tools, and packages that we used in our application.

3.3.1 ExpressJS

Express.js [6] is a minimal, flexible, and un-opinionated Node.js web application framework for building APIs. It helps the development process by providing robust routing capabilities which simplifies the definition of routing parameters and endpoints as well as providing a caching mechanism that temporally stores the duplicate data, leading to decreased load on the server and faster response times. In the initial setup of our application, we made use of several important middleware packages like CORS, BODY-PARSER, RATE-LIMITER, and COOKIE-PARSER to enhance the application handling. With these packages, we were able to create a resilient backend structure that efficiently manages data, processes user requests, and mitigates any performance bottlenecks.

3.3.2 Body-parser

Body-parser [7] is a middleware in Express.js that processes the body of the incoming HTTP requests. It allows parsing the incoming request data to JSON or URL encoded format that can then be processed by the developer to extract the required information.

3.3.3 Rate Limiter

Rate-limiter [8] is a middleware that is responsible for controlling the number of repeated requests made from a specific IP address to a specific API endpoint. It was used as a defence mechanism against brute-force attacks and for preventing the abuse of our API.

3.3.4 Route Compression

Compression middleware [9] helps performance in the application by reducing the size of the server responses and as a result improving the application's speed.

3.3.5 CORS

Cors [10] (Cross-Origin Resource Sharing) is an Express.js middleware that provides a way to overcome the same-origin policy implemented in web browsers which restricts making HTTP request to different domains from the one that the webpage originated from. By using this package rules can be specified for the domains that can access the server and the type of requests they are allowed to make.

3.3.6 Cookie-Parser

Cookie-parser [11] is a middleware that helps manage cookies on client requests. This is achieved by parsing the Cookie from the request headers and populating the req.cookies property with an object representing the cookie object. This simplifies the handling of client-side data stored in cookies while it protects sensitive information from being exposed.

3.3.7 JSON-WEB-TOKEN

JSON Web Token is an open internet standard that defines a way for the secure transmission of information between parties using a JSON object. The tokens can be signed using a

secret or a public/private key. In our application, the JWT is signed using a public/private key, thus ensuring that only the server that holds the private key is the one that signed it. After the user signs in, each subsequent request includes the JWT, allowing the user to access protected routes and endpoints that are permitted with the token.

3.3.8 nodemailer

Nodemailer [12] is a module that allows sending emails from a Node.js web application. It is capable of sending emails with content that can be HTML, plain text and other media attachments. The package supports all major email service providers, making it a good tool for tasks like sending registration confirmation emails, system notifications, and other user engagement emails.

3.3.9 node-cron

The node-cron [13] is a package that allows scheduling tasks in the application at specific intervals or times. Cron jobs are helpful for performing scheduled tasks, such as creating database records and sending out emails or notifications.

3.3.10 bcrypt

Bcrypt [14] is a cryptography library that helps in securing user passwords. Instead of storing the password in clear text which poses a security risk, bcrypt helps hash the password. What makes bcrypt special is its built-in salt and adaptability. Salting is the addition of random data to a password before hashing it. The result of a hashed password that has passed through several salting rounds is that even if these hashed passwords get stolen, the attackers cannot reverse-engineer the original user's password.

3.3.11 @azure/communication-email

The @azure/communication-email [15] is part of the Microsoft Azure platform and provides real-time communication functionality in the application. It enables sending emails on a global scale without the need of an infrastructure setup and it provides a secure and scalable way to send transactional or marketing emails.

3.3.12 Sequelize ORM

Sequelize [16] is a promise-based ORM (Object-Relational Mapping) for Node.js. It supports a variety of SQL datbases including MySQL, PostgreSQL, SQLite while providing a high-level abstraction for database manipulation, enabling developers to interact with the database by using javascript instead of writing raw SQL queries. In our application, we used Sequelize as an interface to interact with our database. It gave us the ability to perform CRUD operations (CREATE, READ, UPDATE, DELETE) and to define tables as models and their relationships using JavaScript.

3.4 Frontend

React.js [17] is an open-source front-end JavaScript library built for creating UIs based on components. Its key operational process revolves around the use of a virtual DOM, a copy of the actual DOM. Here's how it functions:

- **Realizing Updates:** Whenever there's a state change in the components, React.js first applies these updates to the virtual DOM, which is faster due to it being a lightweight copy of the actual DOM.

- **Tree Reconciliation:** After the virtual DOM has been updated, React.js performs a 'diffing' operation. It compares the updated virtual DOM with the previous snapshot and figures out the exact changes.

- **Selective Rendering:** With the changes isolated, React.js efficiently updates only those parts of the actual DOM that correspond to these changes. This strategic selectivity significantly boosts React's performance by avoiding unnecessary updates to the actual DOM.

This mechanism allows React.js to deliver high efficiency, contributing to a smooth user experience.

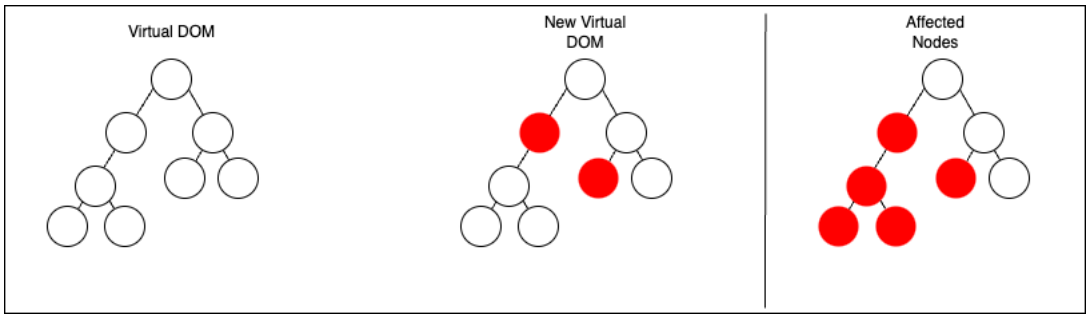

Figure 3.5: React's Update and Render mechanism

3.4.1 react-router-dom

The React-Router-Dom [18] is a React.js library that enables the implementation of routes and navigation on the client side. It accomplishes that without querying the server for new pages, thus providing a smoother user experience.

3.4.2 react-hook-form

React-Hook-Form [19] is an efficient and lightweight form validation library for React.js. Its key features include form state management, a simpler form validation, and also better performance by initiating fewer component re-renders. Apart from that it gives the ability to work with both controlled and uncontrolled form states using a single API. The integration of this library ensures a smooth, feature-rich form handling experience that is performance optimized.

3.4.3 Axios

Axios [20] is an HTTP client with a unified interface for all kinds of HTTP requests (GET, PUT, POST, PATCH, DELETE). It is used for client and server-side web applications. It operates on Promises and provides functionality for handling asynchronous requests.

3.4.4 jwt-decode

JWT-decode [21] is a library that helps decode JWT tokens in order to read the content of the payload without validating. It does not handle JWT verification and validation, so it should only be used in the case where the JWT source can be trusted.

3.4.5 Mantine-UI

Mantine [22] is an open-source React.js component library for building responsive UIs. It provides a variety of responsive core components that are highly customizable and can fit to any design system. In addition to components, it offers a collection of utility hooks for state and UI management for more advanced usage.

3.4.6 monaco-editor-react

The Monaco-editor-react [23] is a wrapper of Microsoft's Monaco Editor that powers Visual Studio Code. It provides features including syntax-highlighting,error detection and auto-completion. With this library, developers can easily integrate a robust code editor in their applications, enhancing usability.

3.4.7 @tanstack-query

Tanstack query [24] is a data synchronization library designed for React.js. It excels at fetching, caching , synchronizing and updating the server state within a React.js application. This library provides some import features like automatic caching and background updating. When a component fetches data using this library, the data is automatically cached and as a result the subsequent requests for the same data are retrieved from cache boosting the performance of the application. It also automates the tracking of loading and error states removing the need for manual management. Garbage collection is another important feature of this library, when a component unmounts the data it fetched is tagged for garbage collection. However the data isn't removed immediately but instead it employs a retry policy, giving the data a certain grace period before it is collected. As a result unnecessary refetching of data if the component remounts shortly after unmounting is avoided. In summary it greatly enhances React.js application functionality, performance and resource management promoting more efficient and reliable application development.

Chapter 4

Technical Description

4.1 Introduction

The main goal of this chapter is to present the building blocks that drive our applications's functionality. We will discuss about the REST architecture and the properties it has and also about the design of the frontend and backend parts of our application. Lastly we will analyze how each part of the application communicates with the other parts and about their interaction with the MySQL database.

4.2 REST Architecture [1]

Representational State Transfer is an architecture that is widely used in web applications. In this architecture, the data and functionality are considered as resources and can be accessed using URLs. It uses HTTP requests to perform CRUD operation and every client to server request must contain all the details needed to process the request. The stateless nature of REST ensures that each request can be understood independently of others, contributing to the REST API's scalability and reliability.

4.2.1 Uniform Interface

All the requests from the API for the same resource should look the same, no matter the request's origin. It should ensure that given a set of data, like a specific attribute from a model, belongs to a unique uniform resource identifier.

4.2.2 Client-Server Decoupling

In the REST API design, the client and server must be independent. The only information that the client should know is the URI of the requested resource, as it cant interact with the server in any other way. Likewise, the server shouldn't modify the client, and should only pass the data to the response of the HTTP request.

4.2.3 Statelessness

In REST Architecture the session is not stored in the server between requests. Each request from the client contains all the information available to process the request, and the session is kept entirely on the server side.

4.2.4 Cacheability

Responses sent to the client can be cached, thus improving performance by reducing the latency between the request and response. These responses must be defined as cache-able explicitly to prevent the client from making the same request twice if there are no changes and from reusing stale data.

4.2.5 Layered System Architecture

In REST APIs, requests and responses go through different layers without the client having any prior knowledge of the number of intermediaries between the client and server. This kind of APIs need to be designed so neither the client or server can tell whether it communicates with the other end or an intermediary.

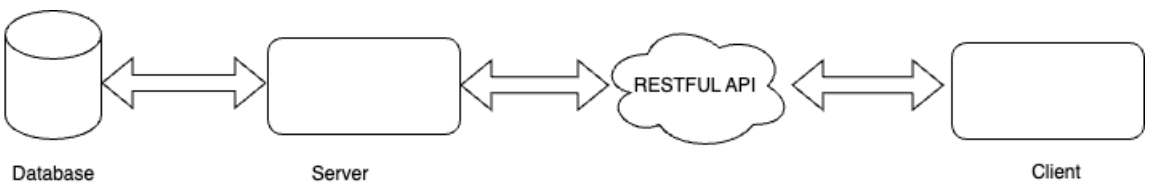

Figure 4.1: REST Architecture.

4.3 Frontend Design

Our application is a SPA, built using the React Vite framework. SPAs provide a seamless user experience by dynamically rewriting the current web page without performing additional reloads. Vite enhances our development process with its fast and lightweight build tooling.

4.3.1 User Interface Design

The user interface consists of several key components and screens:

Authentication Pages:

In our APP, authentication pages such as Sign-In, User Registration, Email Confirmation and Forgot Password exist. The Registration page allows users to register to the platform, guiding them towards the Email Confirmation step and thus ensuring the validity of their email. The Sign-In screen is the initial point of our application, providing validated users entry to the application. Additionally, the Forgot Password page given the options to users to retrieve their forgotten credentials.

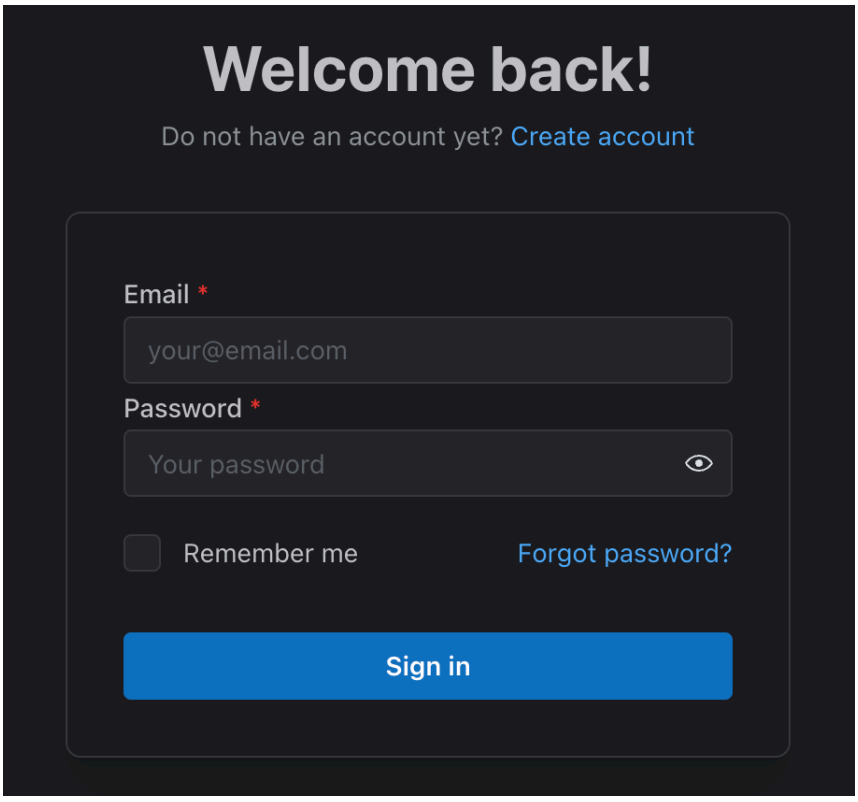

Figure 4.2: Login Page

Layouts:

Our application incorporates a card-based and a Header-Navbar-Footer-Aside layout to make the best use of the screen space and readability. These various layouts provide flexibility in how information is presented, optimizing readability and usability.

Application Shell Component:

The application shell is a layout component that implements the Header-Navbar-Footer-Aside layout component. This key component includes the application's primary navigation elements, in addition to housing the various application views that are dynamically loaded and switched.

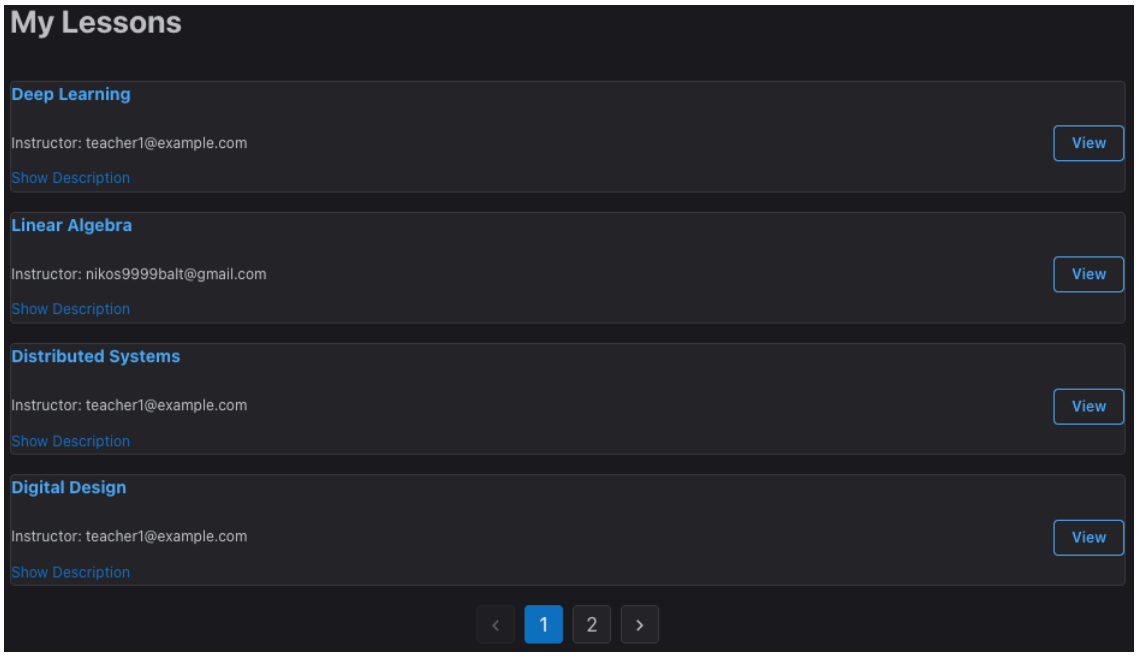

Figure 4.3: Card Layout: Displaying Lessons

4.3.2 User Experience

In order to mimic the native app navigation and provide a smoother user experience, a characteristic feature of SPAs a solid routing system was implemented that enables conditional and state-driven rendering of various components.

Protected Routes

To maintain certain parts of our application available only to authenticated users, we have implemented protected routes. This adds an extra layer of security to our application by redirecting the user that tried to access a protected part of the application to the login page.

Role Based Authorization

The routing configuration is designed to adapt based on the role of the connected user. Certain parts of the application are role-protected and are visible only to the users who possess specific permissions.

Nested Routes

There are situations requiring a hierarchical UI representation for the better organization of the routing system. These indicate sub-views beneath a parent route, illustrating the URL schema's clear and logical structure. In our application, nested routes are used to show the hierarchy and the relation between lessons, a lesson, homework, and specific homework.

- **Lessons** This is the root in the context of the application. It displays a list of all lessons available to the user.
- **Lesson** When a user clicks on a specific lesson from the list, they are routed to a detailed view of that Lesson that is under Lessons in the hierarchy.
- **Homework** Inside a specific lesson, there can be multiple homework, so at this level a list with all the available homework is displayed.
- **Homework** Clicking on a specific homework from the Homework list takes the user to a detailed view of that specific Homework. This is the deepest level of this hierarchy.

Figure 4.4: Nested Routes URL.

4.3.3 State Management

For the state management of our application two state management methods were combined.

- **Context API:** React's built-in Context API was used in order to share specific authentication-related data across all levels of the application. This allows components to react to context changes and thus ensuring that the components can react to updates in the authentication state. This was mainly used at the sign-in and sign-out process.
- **Tanstack Query** This library was used for synchronizing server state in the application. It provides several state variables through its hooks, `useQuery`, `useMutation` that give control and insights over the data fetching process, including loading state, error state and success state. This helps to couple the asynchronous data fetch to the react component life-cycle.

4.3.4 Communication with Backend

The SPA has been created to interact with the backend through RESTful API calls. This ensures a persistent and functional synchrony between the frontend and backend of the application. The API interactions are authorized with a JWT token-based mechanism to ensure security and session management for each user.

4.3.5 Error Handling

- **Notifications:** The application integrates a comprehensive notification system for handling general errors and unexpected exceptions. This mechanism allows real-time anomalies that might occur during asynchronous operations. Upon the event of an error, a notification is dispatched to the UI that alerts the user about the issue. This interactive and immediate error feedback enhances the end user's experience, and as a result strengthens the overall application resilience.
- **Form Validation and Errors** Complementing the error notification system, this system is used as a preventative error handling solution targeted at the form data input. Upon form submission, the system validates user inputs against some predetermined rules. Failure to meet these rules prompts the system to display a concise error message

at the form input or the complete form. This process offers instantaneous feedback, guiding the end user towards correction and ultimately leading to a successful form submission.

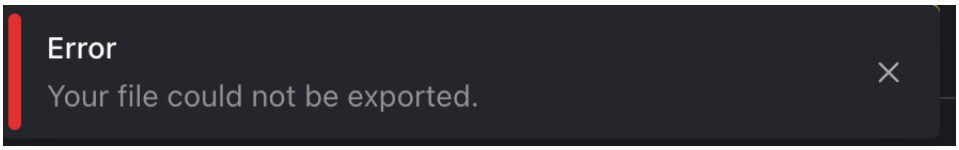

Figure 4.5: Error displayed in a notification.

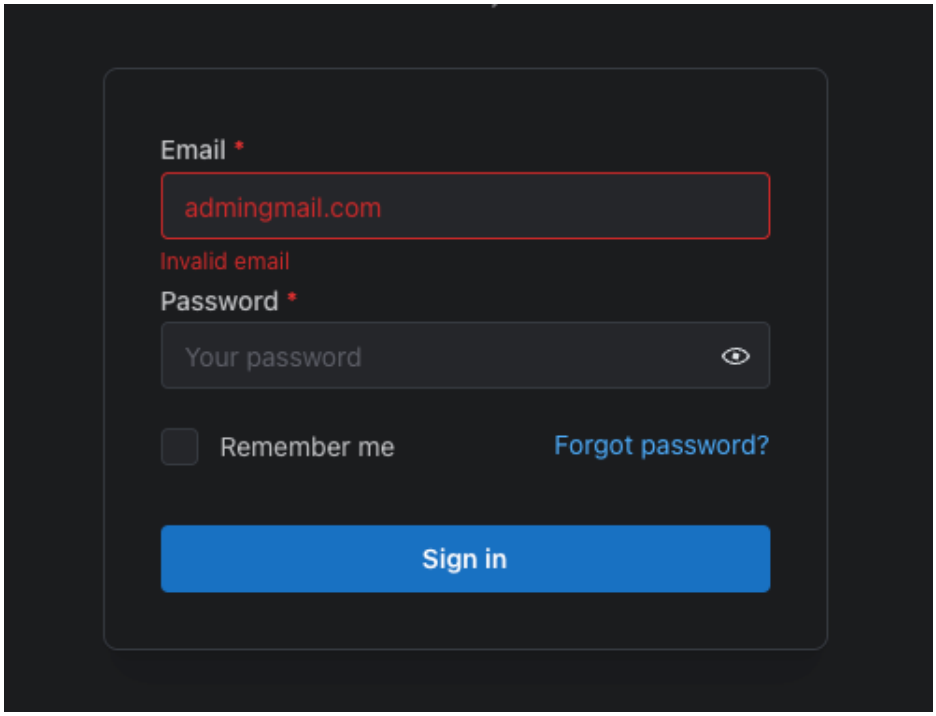

Figure 4.6: Form validation error on invalid email address.

4.4 Backend Design

4.4.1 Routing

Our backend application employs RESTful conventions, structuring endpoints around the primary resources to maintain clarity and logical grouping of associated functions. The major HTTP methods are leveraged for acting upon the resources and facilitating intuitive interaction for the client.

- **/auth** This route is used for user authentication. It ensures secure access to our application by verifying user credentials and managing user sessions. The HTTP methods associated with this route plays a pivotal role in user sign-up, login, email confirmation, forgot password and logout functionality.

- **/lesson** The lesson route is responsible for handling lesson related data. It can include operations like fetching lesson details, creating new lessons, updating existing ones or managing associated lesson components.

- **/enrollment** This route deals with user enrolment in lessons. It enables users to enroll in lessons and check the enrolment status of users.

- **/homework** This route manages homework related functions. It includes retrieving, updating or removing homework tasks as well as calculating the grade for users in the homework.

- **/homeworkQuestion** This route is used to manage questions for each homework assignment. It can create new questions, retrieve them for a particular homework assignment, update existing questions or delete them.

- **/homeworkUpload** The "/homeworkUpload" route handles the uploading of homework by the user. It supports file upload and manages submissions for the homework assignments.

4.4.2 Middleware

Middleware functions are a crucial part of the backend design. These are available at different levels in our application, with the ability to manage their invocation as needed. The primary, high-level middleware brings functionality to most routes and is placed at the start of the application, while the route-specific middleware provides functionality to specific routes.

- **Body-parser Middleware:** This middleware is added at the highest, application level, and it acts as the first interaction point of the server with an incoming network request. Body-parser parses the incoming request body and populates the req.body object, ensuring all subsequent middleware or route handlers can readily access incoming request data.

- **Rate-limiter Middleware:** Also applied at the top level, rate-limiter helps us guard against potential abuse or inadvertent server overload. It monitors and restricts the frequency of requests a client can make within a set time interval, providing a crucial defense mechanism against denial-of-service (DoS) attacks or brute-force login attempts.

- **CORS Middleware:** Like the rate-limiter and body-parser, this middleware is implemented at the application level. CORS (Cross-Origin Resource Sharing) middleware manages cross-origin requests coming from different domains, maintaining control over API interaction rules and policies.

- **Cookie-parser Middleware:** Implemented at the application level, the cookie-parser middleware is crucial in managing client-side session data stored in cookies. It parses the Cookie header from the incoming request and converts it into an easy-to-use JavaScript object, facilitating subsequent session checks or security assessments.

- **Authentication and Authorization Middleware:** Differing from the rest, the custom authentication and authorization middleware is applied at the route level. It uses the data parsed by the cookie-parser middleware to identify users and verify their permissions before they can access specific routes, providing a secure mechanism for our server's resources.

4.4.3 Controllers

The controllers are responsible for dealing with incoming client requests, performing necessary checks and invoking appropriate logic from the Models, which are the components that manage business logic and interact with the database.

Data Sanitation Another aspect of the controllers is the security measure of data sanitation. When user-related data is retrieved, the output is sanitized before sending the response to the client to avoid potential data leaks.

Error Handling If a user's request encounters an issue with our controllers, the server sends back a response with a custom error message. This notification informs the user of the

Table 4.1: Most common response status codes

STATUS CODE	MEANING
200	OK: The request was successful
201	Created: The request was successful, and a resource was created.
204	No Content: The request was successful, but there's no content to include in the response body.
400	Bad Request: The server could not understand the request.
401	Unauthorized: The request has not been applied because it lacks valid authentication credentials for the target resource.
403	Forbidden: The client does not have permission to access the requested resource.
404	Not Found: The requested resource could not be found.
500	Internal Server Error: The server encountered an error.

error and points to its likely location while maintaining system security by not disclosing sensitive information.

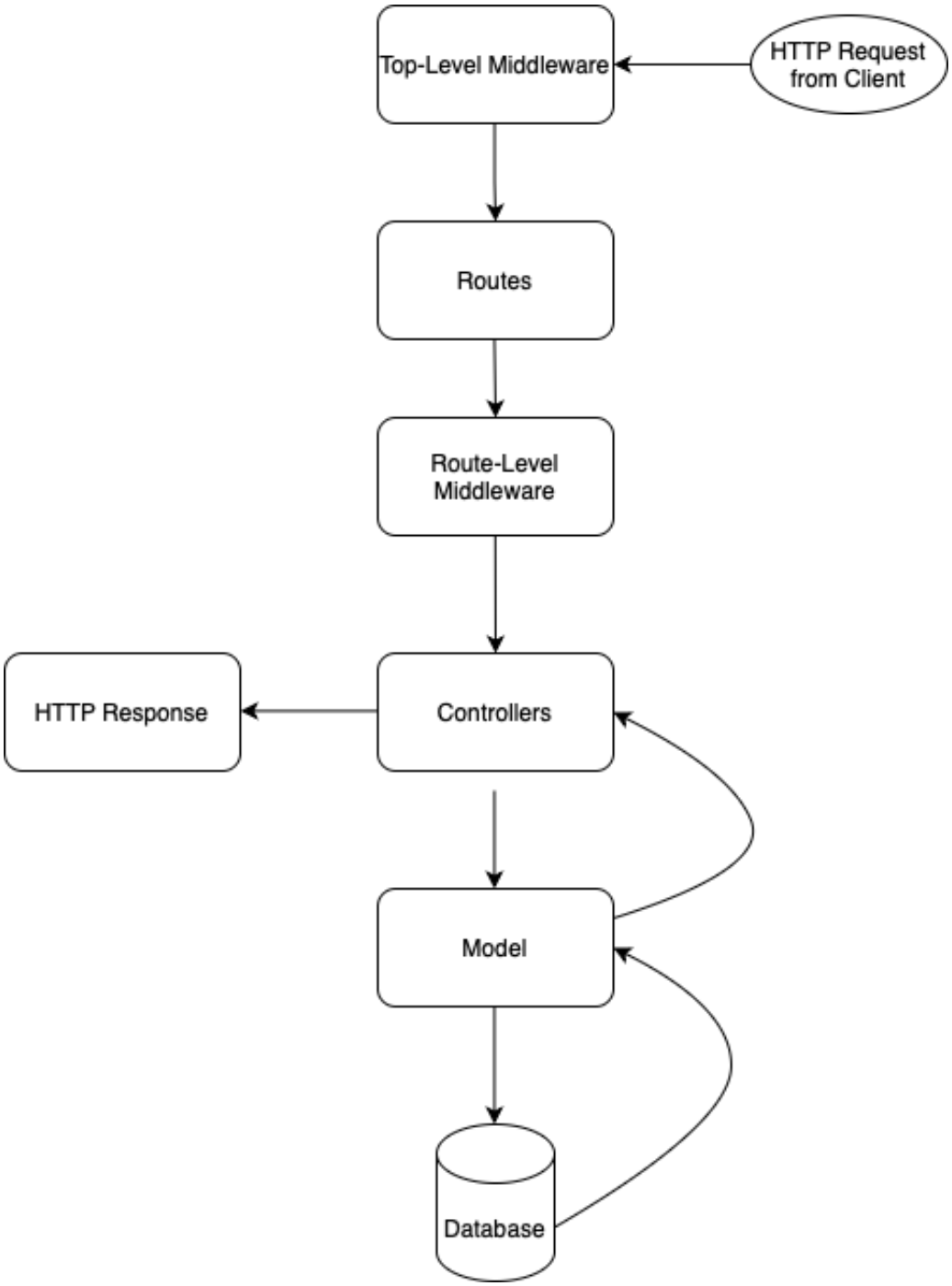

Figure 4.7: Lifecycle of a Client Request in a Model-Controller Design

4.4.4 Models

For the needs of our application, we implemented the following models:

Role Model The Role model defines the various roles a user can have within the application. Each role corresponds to a certain set of access within the system.

- **roleName** This attribute represents the name of the role.

User Model The User model manages user related data, such as their username, email, passwords and various tokens for email confirmation, resetting password, and refreshing session.

- **username:** User's username.
- **email:** User's email
- **emailConfirmationToken:** The token for the email confirmation process.
- **isConfirmed:** Boolean variable that shows if the user's email is confirmed.
- **hashedPassword:** This field holds the password's hashed value.
- **passwordResetToken:** The token for the forgot password process.
- **passwordResetExpiresAt:** The process for password reset can have a limited time validity. This field stores the expiration timestamp for the passwordResetToken to ensure it's not used after this timestamp.
- **refreshToken:** A refresh token is generated and used in JWT token-based authentication mechanism. When an access token expires, a refresh token can be used to request a new access token without requiring the user to log in again.

Lesson The Lesson model contains information details of the lesson

- **name:** Lesson's name.
- **description:** Lesson's description.
- **creatorId/UserId:** A foreign key linked to the User model that represents who created the lesson.

Enrollment The Enrollment model has foreign keys between Lesson and User and contains information about the enrollment status of a user in a lesson

- **UserId:** A field that serves as a foreign key referencing the ID in the User table. It associates each Enrollment record with a particular user.

- **LessonId:** A field that serves as a foreign key referencing the ID in the Lesson table. It allows to identify in which lesson the user is enrolled.
- **status:** This field contains important context about the user's current state in relation to the enrolled lesson. It can be pending, accepted and rejected.

Homework Contains information about a homework task including its name, description, key dates, and details about peer review requirements.

- **name:** Homework's name.
- **description:** Homework's description.
- **presentationDate:** Homework's presentation date.
- **submissionDeadline:** Homework's submission deadline.
- **isPeerReviewed:** A boolean variable that signifies if the homework is to be peer-reviewed.
- **reviewDeadline:** Homework's peer review deadline.
- **reviewerCount:** Homework's number of reviewers per homework submission.
- **peerReviewsAssigned:** A boolean variable that shows whether the peer review tasks have been assigned.

Homework Submission Manages homework submissions from users, including submission date and status.

- **UserId:** A field that serves as a foreign key referencing the ID in the User table. It links each Homework Submission entry to the user who submitted the homework and is also used to track if a user has submitted a particular homework.
- **HomeworkId:** A field that serves as a foreign key referencing the ID in the Homework table. It allows each submission to be associated with a specific homework.
- **submissionDate** The date the homework was submitted.
- **isSubmitted** A boolean variable that shows if the homework has been submitted by the user.

PeerReview Contains information relating to peer reviews of homework including associated user, homework, and submission.

- **UserId/ReviewerId:** A field that serves as a foreign key referencing the ID in the User table. It connects each PeerReview to a specific user that has to perform a review. This allows tracking of the users that are taking place in the peer review process.
- **HomeworkId:** A field that serves as a foreign key referencing the ID in the Homework table. It associates each PeerReview with a particular homework. This enables identifying which homework assignment the review is referring to.
- **HomeworkSubmissionId:** A field that serves as a foreign key referencing the ID in the Homework table. It ties each PeerReview to a specific homework submission that has to bee reviewed. With this, you can see which specific submission the review is connected to.

Homework Upload Manages uploaded files for homework submissions including the file name, path, and associated submission.

- **HomeworkSubmissionId:** This is a foreign key associated with the HomeworkSubmission model. It links each HomeworkUpload to a specific homework submission. This makes it possible to track which file upload is associated with which homework submission.
- **fileName:** This field contains the name of the uploaded file.
- **filePath:** The path to the uploaded file's location.

Homework Question This model represents the questions created under a homework that is to be peer reviewed.

- **HomeworkId:** A field that serves as a foreign key referencing the ID in the Homework table. It ties each HomeworkQuestion entry to a specific homework.
- **questionText:** Question's text.
- **questionType:** Question's type. It can be a Number ,True/False or TextArea.

- **questionAnswer:** Question's answer.
- **points:** The max points that this question gives if answered correctly.
- **weight:** Question's weight.

User Answer Stores users' answers to homework questions, the types of questions, and related peer review.

- **HomeworkQuestionId:** A field that serves as a foreign key referencing the ID in the Homework Question table. It ties each User Answer to a particular homework question, allowing to see which question the answer is related to.
- **PeerReviewId:** A field that serves as a foreign key referencing the ID in the Peer Review table. It allows seeing which review the answer is connected to.
- **questionType:** The type of the question the user gave an answer for.
- **answerText:** User's answer to the question.

UserRole The UserRole model represents an intermediate table between the User model and the Role model. It defines a many-to-many relationship, meaning that a user can have multiple roles and a role can be assigned to multiple users.

- **UserId:** A field that serves as a foreign key referencing the ID in the User table.
- **RoleId:** A field that serves as a foreign key referencing the ID in the Role table.

4.5 Backend Deployment

For the deployment of this application's backend Azure App Service [25] was used. It allows developers to manage and deploy scalable web applications in the cloud. Key features of this service include automatic scaling, continuous deployment, and various environment setups for testing and staging. The repository that hosted the code was integrated with the Azure App Service environment thus allowing for a streamlined deployment process.

(Backend URL: `https://peer-review-app-testing.azurewebsites.net/`).

4.6 Frontend Deployment

Vercel [26] is a cloud platform designed to provide an efficient way to build, deploy, and scale applications. In this application, Vercel was used to manage the frontend deployment due to its simple deployment that was accomplished by linking the Github repository containing the code to Vercel. This direct link enables continuous deployment. Vercel not only simplifies the deployment process but also ensures an up-to-date live version of the application is available at all times.

(Frontend URL: `https://thesis-frontend-kappa.vercel.app/`).

Chapter 5

Conclusions

As we reach the closing stages of this dissertation, it is pertinent to reflect on the exploration and the insights gained in the process. This study has delved into the challenges and opportunities of digital education, particularly in streamlining the collaborative learning process involved in peer review of code assignments for students.

5.1 Summary and Conclusions

Upon conclusion of this thesis, we reflect on the study's exploration of digital education. Specifically, we discuss the development of a tool to address the challenges of peer-reviewing code assignments for students. This work contributes to an application that combines the benefits of code review tools and peer review processes. Existing tools like Gerrit, Collaborator by SmartBear, and Instructure, while helpful, do not cater to this unique need. Our application promotes efficient assignment management and structured peer review procedures. In essence, this thesis presents a solution that fills a gap in tech, projecting a fundamental transformation in handling peer reviews and code assignments in digital education.

5.2 Future Improvements

Building upon the conclusions of this thesis, it is clear that our application provides a beneficial tool in the current landscape of digital education. However, as with any solution, there is room for future development.

1. **Integration with Github:** To streamline the homework submission process, future

application versions could enable uploads directly from a GitHub repository, replacing the current zip file submission system. Students and teachers can benefit from the version control and collaborative features of Github.

2. **In-app Communication:** Implementing a chat functionality within the application could enhance communication between teachers and students. It would provide a platform for discussion, real-time question-and-answer sessions, and immediate feedback.

3. **User Profiles** Introducing the feature of creating user profiles would personalize the experience, enabling users to track their assignments, reviews, and progress within one place.

Bibliography

[1] Rest api. https://www.ibm.com/topics/rest-apis. Access date: 27-1-2024.

[2] Gerrit. https://www.gerritcodereview.com/. Access date: 27-1-2024.

[3] Collaborator by smartbear. https://smartbear.com/product/collaborator/. Access date: 27-1-2024.

[4] Instructure. https://www.instructure.com/resources/blog/engaging-your-students-through-use-peer-review. Access date: 27-1-2024.

[5] Hugh E. Williams Saied M.M. Tahaghoghi. *Learning MySQL, 2nd Edition*. O'Reilly Media, Inc, 1 edition, 2006.

[6] Express.js. https://expressjs.com/. Access date: 27-1-2024.

[7] Body-parser. https://expressjs.com/en/resources/middleware/body-parser.html. Access date: 27-1-2024.

[8] Express-rate-limit. https://www.npmjs.com/package/express-rate-limit. Access date: 27-1-2024.

[9] Express-compression. https://expressjs.com/en/resources/middleware/compression.html. Access date: 27-1-2024.

[10] Cors. https://expressjs.com/en/resources/middleware/cors.html. Access date: 27-1-2024.

[11] Cookie-parser. https://expressjs.com/en/resources/middleware/cookie-parser.html. Access date: 27-1-2024.

[12] Nodemailer. https://nodemailer.com/. Access date: 27-1-2024.

[13] node-cron. https://www.npmjs.com/package/node-cron. Access date: 27-1-2024.

[14] Bcrypt. https://en.wikipedia.org/wiki/Bcrypt. Access date: 27-1-2024.

[15] Azure communication email. https://learn.microsoft.com/en-us/azure/communication-services/quickstarts/email/create-email-communication-resource. Access date: 27-1-2024.

[16] Sequelize orm. https://sequelize.org/. Access date: 27-1-2024.

[17] Eve Porcello Alex Banks. *Learning React, 2nd Edition*. O'Reilly Media, Inc, 2 edition, 2020.

[18] react-router-dom. https://reactrouter.com/en/main. Access date: 27-1-2024.

[19] react-hook-form. https://react-hook-form.com/docs. Access date: 27-1-2024.

[20] Axios. https://axios-http.com/docs/intro. Access date: 27-1-2024.

[21] Jwt-decode. https://www.npmjs.com/package/jwt-decode. Access date: 27-1-2024.

[22] Mantine ui. https://ui.mantine.dev. Access date: 27-1-2024.

[23] Monaco-editor-react. https://www.npmjs.com/package/@monaco-editor/react. Access date: 27-1-2024.

[24] Tanstack-query. https://tanstack.com/query/latest/docs/framework/react/overview. Access date: 27-1-2024.

[25] Azure app service. https://learn.microsoft.com/en-us/azure/app-service/quickstart-nodejs. Access date: 27-1-2024.

[26] Deploy to vercel. https://vercel.com/guides/deploying-react-with-vercel. Access date: 27-1-2024.