UNIVERSITY OF THESSALY

SCHOOL OF ENGINEERING

DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

App for Studying and Cooperating Among ECE Students

Diploma Thesis

Christos Kapotos

Supervisor: Georgios Thanos

June 2023

UNIVERSITY OF THESSALY

SCHOOL OF ENGINEERING

DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

App for Studying and Cooperating Among ECE Students

Diploma Thesis

Christos Kapotos

Supervisor: Georgios Thanos

June 2023

ΠΑΝΕΠΙΣΤΗΜΙΟ ΘΕΣΣΑΛΙΑΣ

ΠΟΛΥΤΕΧΝΙΚΗ ΣΧΟΛΗ

ΤΜΗΜΑ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

Εφαρμογή Διαδικτυακής Μελέτης και Συνεργασίας Φοιτητών του ΤΗΜΜΥ

Διπλωματική Εργασία

Χρήστος Καπότος

Επιβλέπων/πουσα: Γεώργιος Θάνος

Ιούνιος 2023

Approved by the Examination Committee:

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

Member **Eleni Tousidou**
Laboratory Teaching Staff member, Department of Electrical and Computer Engineering, University of Thessaly

Member **Dimitrios Rafailidis**
Associate Professor, 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

Christos Kapotos

Diploma Thesis

App for Studying and Cooperating Among ECE Students

Christos Kapotos

Abstract

Collaboration in engineering universities holds immense value since it promotes the exchange of diverse ideas and perspectives, encouraging students to approach problems from various angles. This diversity of thought enables the development of innovative solutions to complex engineering challenges. By working together, students can leverage their knowledge and experiences to come up with creative solutions. This diploma thesis deals with the implementation of a web application that enables this kind of student collaboration. The development consists of two parts that were worked on parallelly, the frontend and the backend. The former is the client-side application or the user interface that makes the experience as smooth as possible, written in ReactJS, while the latter is the server application that processes the client's requests and sends back the appropriate response, written in NodeJS and ExpressJS. More specifically, the application lets the user sign up/sign in, create or join a subject group, upload/like/comment on a post, like/respond to comments and privately chat with other students.

Keywords:

collaboration, web application, ReactJS, NodeJS, ExpressJS

Διπλωματική Εργασία

Εφαρμογή Διαδικτυακής Μελέτης και Συνεργασίας Φοιτητών του ΤΗΜΜΥ

Χρήστος Καπότος

Περίληψη

Η συνεργασία στα πανεπιστήμια μηχανικής έχει ανεκτίμητη αξία, καθώς προωθεί την ανταλλαγή διαφορετικών ιδεών και οπτικών, ενθαρρύνοντας τους φοιτητές να προσεγγίζουν τα προβλήματα από διάφορες πλευρές. Αυτή η ποικιλία σκέψης επιτρέπει την ανάπτυξη καινοτόμων λύσεων για πολύπλοκες μηχανολογικές προκλήσεις. Συνεργαζόμενοι, οι φοιτητές μπορούν να εκμεταλλευτούν τις γνώσεις και τις εμπειρίες τους για τη δημιουργία έξυπνων λύσεων. Αυτή η διπλωματική εργασία ασχολείται με την υλοποίηση μιας web εφαρμογής που διευκολύνει τέτοιου είδους συνεργασία μεταξύ των φοιτητών. Η ανάπτυξη αποτελείται από δύο μέρη που εκπονήθηκαν παράλληλα, το frontend και το backend. Το πρώτο είναι η client-side εφαρμογή, ή η διεπαφή χρήστη που καθιστά την εμπειρία όσο το δυνατόν ομαλή, γραμμένη σε ReactJS, ενώ το δεύτερο είναι η εφαρμογή του διακομιστή που επεξεργάζεται τα αιτήματα του πελάτη και επιστρέφει την κατάλληλη απάντηση, γραμμένη σε NodeJS και ExpressJS. Πιο συγκεκριμένα, η εφαρμογή επιτρέπει στον χρήστη να εγγραφεί/συνδεθεί, να δημιουργήσει ή να ενταχθεί σε μια ομάδα μαθημάτων, να ανεβάσει/κάνει like/σχολιάσει μια ανάρτηση, να κάνει like/απαντήσει σε σχόλια και να συνομιλήσει ιδιωτικά με άλλους φοιτητές.

Λέξεις-κλειδιά:

συνεργασία, web εφαρμογή, ReactJS, NodeJS, ExpressJS

Table of contents

List of figures

List of tables

Abbreviations

HTML	HyperText Markup Language
CSS	Cascading Style Sheets
JS	JavaScript
JSON	JavaScript Object Notation
JWT	JSON Web Token
HTTP	HyperText Transfer Protocol
API	Application Programming Interface
REST	Representational State Transfer
CORS	Cross-Origin Resource Sharing
URL	Uniform Resource Locator
URI	Uniform Resource Identifier
PHP	Hypertext Preprocessor
SQL	Structured Query Language
DB	Database
JSX	JavaScript XML
XML	Extensible Markup Language
DOM	Document Object Model
AWS	Amazon Web Services
npm	Node Package Manager
npx	Node Package eXecute
ODM	Object Data Modeling
SPA	Single Page Application
UI	User Interface
Git	Global Information Tracker

Chapter 1

Introduction

Working in teams, is proven to produce better results, whether it's in an academic or a professional environment. Universities play a vital role in preparing students build this team spirit that will help them throughout their career, by assigning group projects and assignments. This way, students can learn to build a significantly better result by combining diverse ideas and perspectives, than by working alone. Working in teams can also save valuable time and resources for individuals, but more importantly in a professional workspace.

Technology has a lot to offer when it comes to working in a team. More specifically, social media is a great and convenient way of communicating and collaborating. A social media platform can also serve as a file-sharing tool, or a file-storing mock database.

The idea of implementing a social media application that promotes collaboration draws its origins from said social networking benefits. The fact that there is no such social media platform, dedicated to synchronous collaboration among students of our university, makes this diploma thesis web application all the more relevant.

1.1 Main objective

The primary objective of this thesis is to develop and deploy a web application that functions as a social media platform, facilitating collaboration and knowledge sharing among users. More specifically, this platform will let students join and create groups for each subject and start discussions by uploading posts and commenting on them, or by chatting in private chat rooms. The primary technical objectives revolve around acquiring and expanding knowledge of two web frameworks, including their programming languages and tools.

Additionally, the goals encompass understanding the client-server architecture, developing RESTful APIs, and designing a user-friendly and responsive interface.

1.2 Similar applications

1.2.1 Google Classroom

Google Classroom is a collaboration tool for educators to create assignments, provide feedback, and track student progress. The main similarities with this diploma thesis' application, is the interaction that students can have inside class streams (class groups for this app). The main difference would be voice and video calls, that this application lacks, but would be an important future addition.

1.2.2 Microsoft Teams

Microsoft Teams is also a similar application, with which students and teachers can interact in class groups. The main difference between **Microsoft Teams** and **Google Classroom** is that the latter is more teaching related than the former.

1.3 Thesis Structure

Chapter 1 contains the introduction of this diploma thesis, that presents an analysis of the idea and purpose behind the web application's development, in order to give the reader a brief understanding of what it has to offer. Chapter 2 introduces the reader to the core technologies and concepts that are needed in order to understand the development process. Chapter 3 contains a breakdown of the application's pages (what the user sees, and how to navigate through them) and functionalities, while chapter 4 is a technical breakdown of all the libraries, packages and add-ons that were used in the development process. Chapter 5 contains a brief explanation of how version control and Git work, the URL for the Github repository for the source code, and the and production URLs for the client and server. Lastly, chapter 6 contains a list of possible future additions and performance improvements for the application.

Chapter 2

Technologies and Concepts

2.1 Frontend

The frontend part of the application, or the client-side application refers to the user interface. It is known that the core technologies used to build a user interface are **HTML**, **CSS** and **JavaScript**. HTML creates the structure of the interface, by adding elements to the page, such as text, images, tables etc. Using CSS, the developer can style those elements and make the page more appealing to the user. With JavaScript, the page becomes interactive, for example by adding a click listener to a button that triggers an event when the user clicks it (usually submitting a form, sending a message etc). Frontend JavaScript frameworks like **ReactJS**, **AngularJS**, **VueJS** etc, can give the developer a significantly smoother experience developing an application, by hiding a big part of the code in a black box, and providing some extra tools.

2.2 Backend

The backend, or the server-side application, focuses on processing the clients requests, communicating with the database, and sending the response back to the client. There are various frameworks that help the developer implement this logic. Some of them are: **Spring Boot (Java)**, **PHP**, **ExpressJS (NodeJS/JavaScript)**, **Django (Python)** etc.

2.3 Database

The database is where all the application's data is permanently stored. This way, the client can fetch whatever data is needed through the server, at any time. Databases can be relational and non-relational. Relational databases, like MySQL and PostgreSQL, are based on the relational model, an intuitive, straightforward way of representing data in tables. On the other hand, non-relational databases, like MongoDB, store their data in a non-tabular form. Instead, they might be based on data structures like documents. A document can be highly detailed while containing a range of different types of information in different formats. The application for this thesis uses MongoDB for its databases needs.

2.4 Full Stack Development

In order to create a fully functional, interactive application with permanently stored data, there needs to be a frontend part, a backend part, and a database. Obviously, without the frontend part, there is no user inteface and thus, no application at all. Without the backend, the developer would have to connect the frontend directly to the database, making the application highly vulnerable to users with malicious intent. Also, the frontend would have to run calculations that the server is responsible for, making the application laggy and the experience less smooth.

2.5 Promises in JavaScript

Promises are objects that represent a value that may not be available yet but will be resolved or rejected at some point in the future and is an elegant feature for handling asynchronous operations like API calls.

A promise can be in one of three states:

- **Pending**: The initial state of a promise, representing that the operation is still in progress and the outcome is not determined yet.
- **Fulfilled/Resolved**: The state of a promise when the operation is successfully completed. The promise is resolved with a value, and the associated then() callback is invoked.

- **Rejected:** The state of a promise when the operation encounters an error or fails. The promise is rejected with a reason or an error object, and the associated catch() or onRejected() callback is invoked.

2.6 API

An Application Programming Interface (API) is a collection of functions and protocols designed to facilitate the development of application software. APIs enable communication between different services without requiring knowledge of the internal implementation details of those services. By providing access to an application's resources while abstracting away the underlying implementation, APIs ensure security, control, and maintain the integrity of the system.

2.7 REST (Representational State Transfer) [1]

APIs that follow the REST architecture, generally have to follow six principles.

2.7.1 Uniform Interface

All API requests for the same resource should look the same, no matter where the request comes from. The REST API should ensure that the same piece of data, such as the name or email address of a user, belongs to only one uniform resource identifier (URI). Resources shouldn't be too large but should contain every piece of information that the client might need.

2.7.2 Client-Server Decoupling

In REST API design, client and server applications must be completely independent of each other. The only information the client application should know is the URI of the requested resource; it can't interact with the server application in any other ways. Similarly, a server application shouldn't modify the client application other than passing it to the requested data via HTTP.

2.7.3 Statelessness

REST APIs are stateless, meaning that each request needs to include all the information necessary for processing it. In other words, REST APIs do not require any server-side sessions. Server applications aren't allowed to store any data related to a client request.

2.7.4 Cacheability

When possible, resources should be cacheable on the client or server side. Server responses also need to contain information about whether caching is allowed for the delivered resource. The goal is to improve performance on the client side, while increasing scalability on the server side.

2.7.5 Layered System Architecture

In REST APIs, the calls and responses go through different layers. As a rule of thumb, don't assume that the client and server applications connect directly to each other. There may be a number of different intermediaries in the communication loop. REST APIs need to be designed so that neither the client nor the server can tell whether it communicates with the end application or an intermediary.

2.7.6 Code on Demand (optional)

REST APIs usually send static resources, but in certain cases, responses can also contain executable code (such as Java applets). In these cases, the code should only run on-demand.

2.8 HTTP Response Methods and Codes

The HTTP requests that the client sends to the server, are characterized by certain keywords that describe the action to be done.

The 4 most common HTTP Methods, that were used for this web application are the below[7]:

HTTP Method	Meaning
GET	Retrieve a specific resource
POST	Create a new resource
PATCH	Update a specific resource
DELETE	Delete a specific resource

Table 2.1: Most common HTTP Methods

The HTTP responses that the server sends back, are characterized by certain codes that describe the status of the action (for example success or failure).

The most common HTTP Codes are the below[8]:

Status Code	Meaning
200 (OK)	The request succeeded
201 (CREATED)	The request succeeded, and a new resource was created as a result
202 (ACCEPTED)	The request has been received but not yet acted upon
204 (NO CONTENT)	There is no content to send for this request, but the headers may be useful
301 (MOVED PERMANENTLY)	The URL of the requested resource has been changed permanently
400 (BAD REQUEST)	The server cannot or will not process the request due to a client error
401 (UNAUTHORIZED)	The client must authenticate itself to get the requested response
403 (FORBIDDEN)	The client does not have access rights to the content
404 (NOT FOUND)	The server cannot find the requested resource
500 (INTERNAL SERVER ERROR)	The server has encountered a situation it does not know how to handle

Table 2.2: Most common Status Codes

2.9 JSX

JSX stands for JavaScript Syntax Extension and occasionally referred as JavaScript XML. It is an extension to the JavaScript language syntax, which provides a way to structure component rendering using syntax familiar to many developers commonly used in React. It is similar in appearance to HTML.[9]

The default <App /> component (App.js file) can be seen in the image below, where the similarities between HTML and JSX syntax are clear.

```
import logo from './logo.svg';
import './App.css';

function App() {
  return (
    <div className="App">
      <header className="App-header">
        <img src={logo} className="App-logo" alt="logo" />
        <p>
          Edit <code>src/App.js</code> and save to reload.
        </p>
        <a
          className="App-link"
          href="https://reactjs.org"
          target="_blank"
          rel="noopener noreferrer"
        >
          Learn React
        </a>
      </header>
    </div>
  );
}

export default App;
```

Figure 2.1: Initial App.js File

Of course, in order to build a proper React project, the default contents of the <App /> component need to be erased.

2.10 State

The state is a built-in React object that is used to contain data or information about the component. A component's state can change over time; whenever it changes, the component re-renders. The change in state can happen as a response to user action or system-generated events and these changes determine the behavior of the component and how it will render.[10]

2.11 DOM and Virtual DOM

The HTML DOM is a standard object model and programming interface for HTML. It defines:

- The HTML elements as objects
- The properties of all HTML elements
- The methods to access all HTML elements
- The events for all HTML elements

In other words: The HTML DOM is a standard for how to get, change, add, or delete HTML elements.[11]

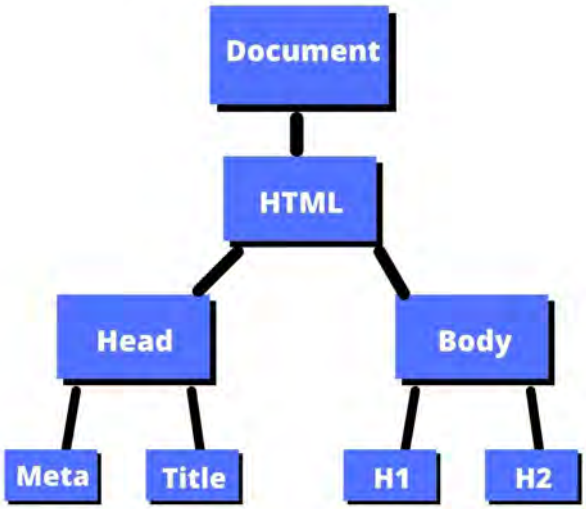

Figure 2.2: DOM Tree visual representation[3]

A virtual DOM is a lightweight JavaScript representation of the Document Object Model (DOM) used in declarative web frameworks such as React, Vue, and Elm. Updating the virtual DOM is comparatively faster than updating the actual DOM, since the update happens only to the component that is rerendered and not the whole DOM tree.

A visualization of a component tree in React, helps with understanding how data is passed though the components.

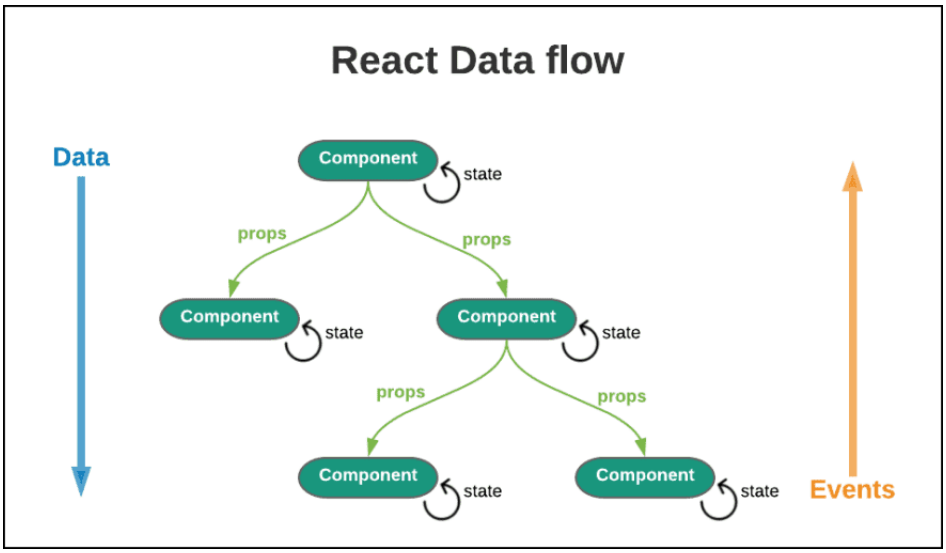

Figure 2.3: React Component Tree visual representation[4]

In React, data is passed down to the children as props (an object that can be destructured inside the child component) and it is an One-Way Data Binding road, meaning that only the parent can pass data to a child, and not the other way around.

Chapter 3

Application functionalities

3.1 Sign Up

Each student can sign up for the application using their university email, which is necessary for obvious reasons.

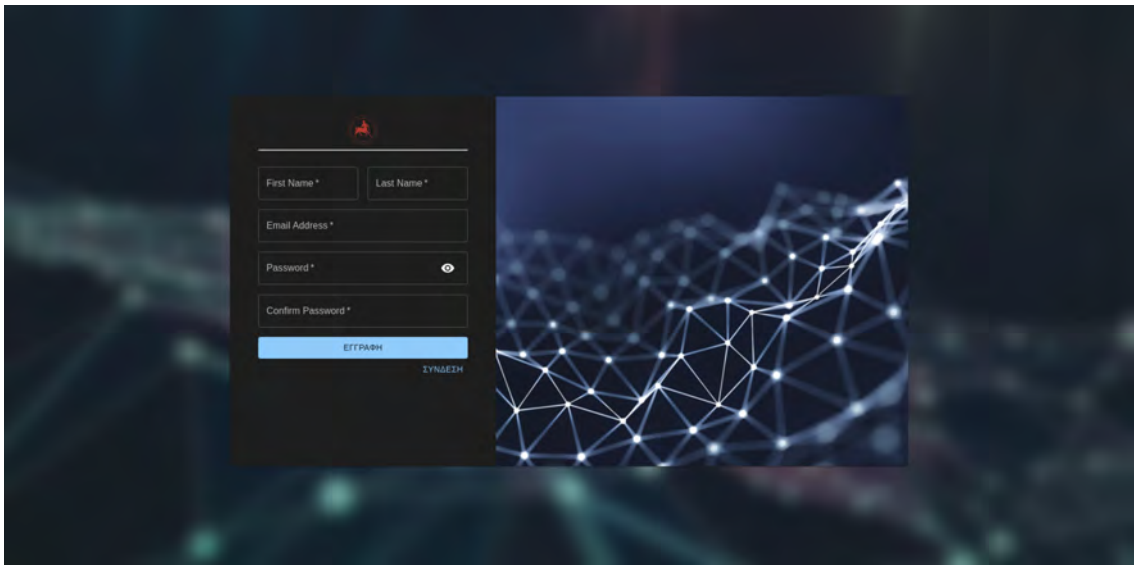

Figure 3.1: Sign Up Form

There are several errors that are checked on the frontend:

- The password should be at least 8 characters long
- The password should contain at least 3 numbers
- The passwords must match

One more error is checked on the backend, and it is whether the provided email address is already being used. If so, the server responds with an error code, and the message below is displayed.

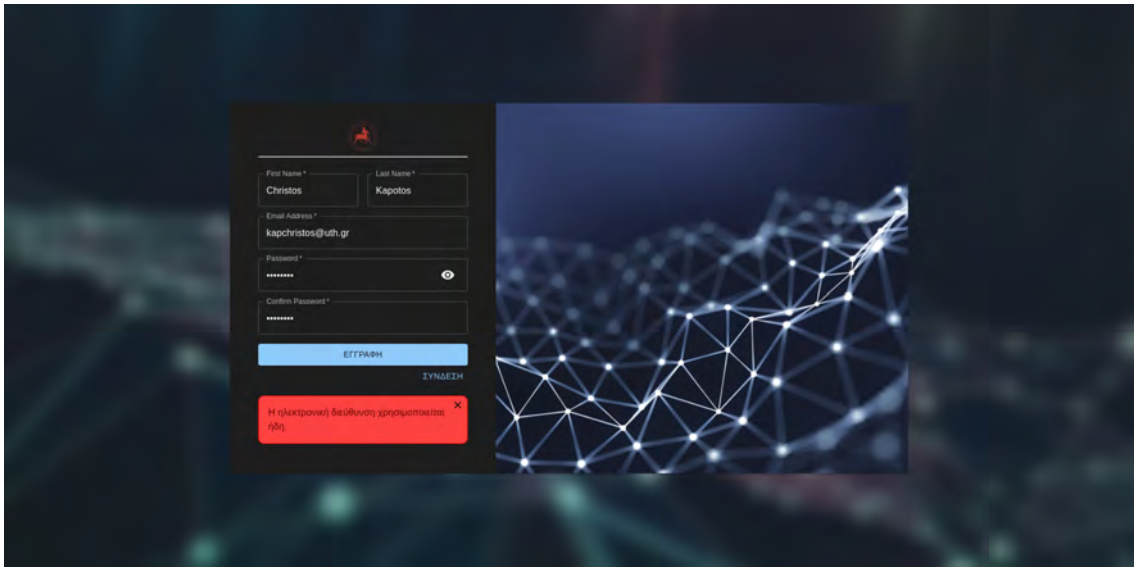

Figure 3.2: ”Email already exists” error

Upon submitting the registration form, a verification email is sent to the provided email address. Clicking the link contained in the email, the user is redirected to the page below.

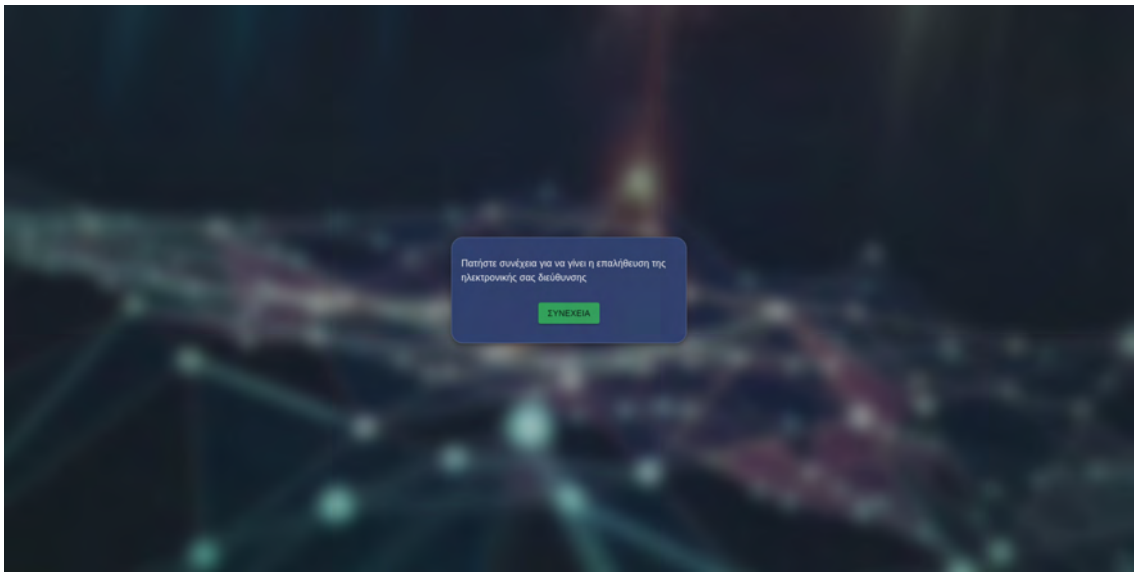

Figure 3.3: Email Verification Page

Obviously, in order to complete the process and ”unlock” their account, the user must click the button. The server responds with a success code if the verification succeeds, otherwise with an error code, and a corresponding message is shown below the button.

3.2 Sign In

To sign in, the user must provide their account's email address and password. The two errors that could occur are if one of them is wrong, and this is obviously checked on the backend. If one of those errors does occur, the server responds with an error code and a corresponding message is shown.

Here's an example of the email address error:

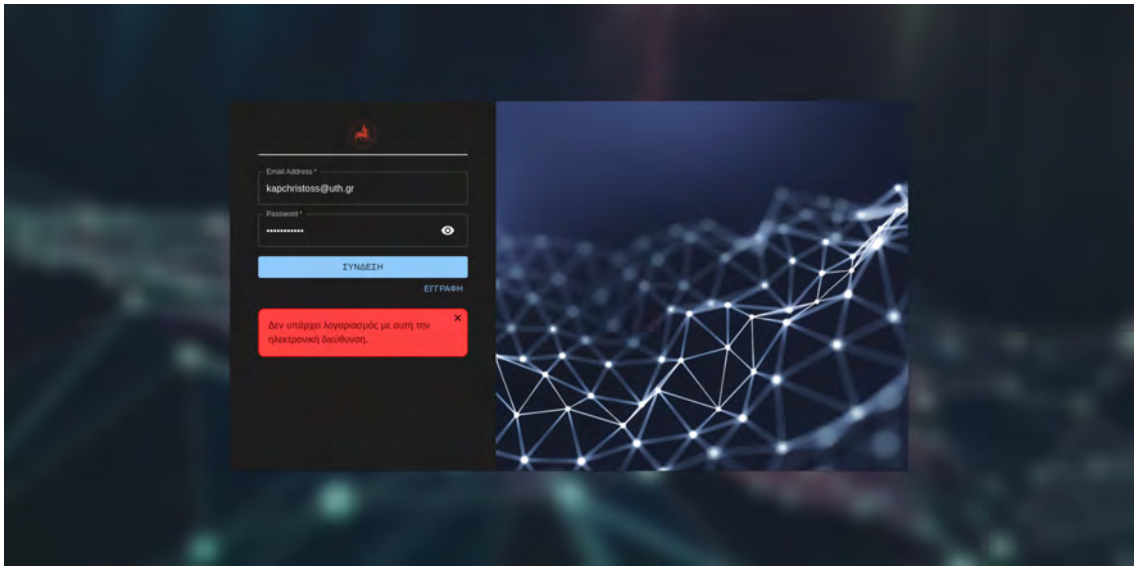

Figure 3.4: "Wrong Email" error

If the sign-in is successful, the user is redirected to the home page.

3.3 Home Page Browsing

The home page consists of a stack that is vertically divided into three sections. The left section is a list displaying the names of the groups which the user is subscribed to. By clicking on a specific group, the application redirects the user to the corresponding group page. The right section of the stack displays previews of the user's private chats. The middle part contains a form for creating a post, and below it are the posts that were posted in the groups that the user is subscribed to. The posts are fetched using pagination, with 5 posts per page. In the initial request, the 5 most recent posts are retrieved. As the user scrolls down, the next request is made to fetch the next 5 posts. This approach allows for faster data retrieval and enables infinite scrolling. This process continues until the server sends the last valid post.

The post creation form can also serve as a group creation form, depending on the activated functionality.

While the frontend is waiting for the posts to be fetched, two ”loading posts” are shown instead, for interactivity reasons. It gives the user a smoother experience, as it lets them know that the request is being processed.

Same goes for the messages box. A loading animation is being displayed, until the chat data is fetched from the server.

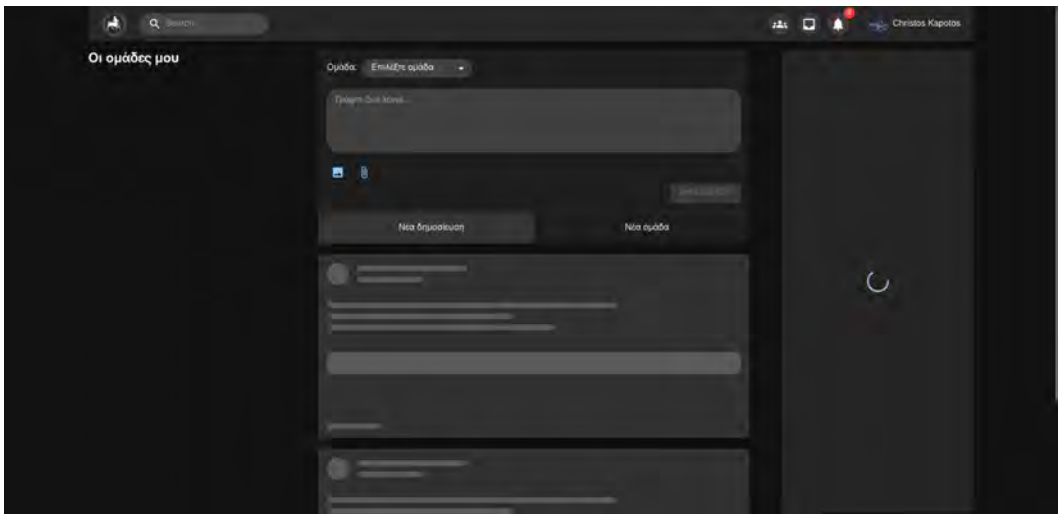

Figure 3.5: Home Page - Loading Data

When everything is fetched, the home page is populated with the corresponding data of each segment.

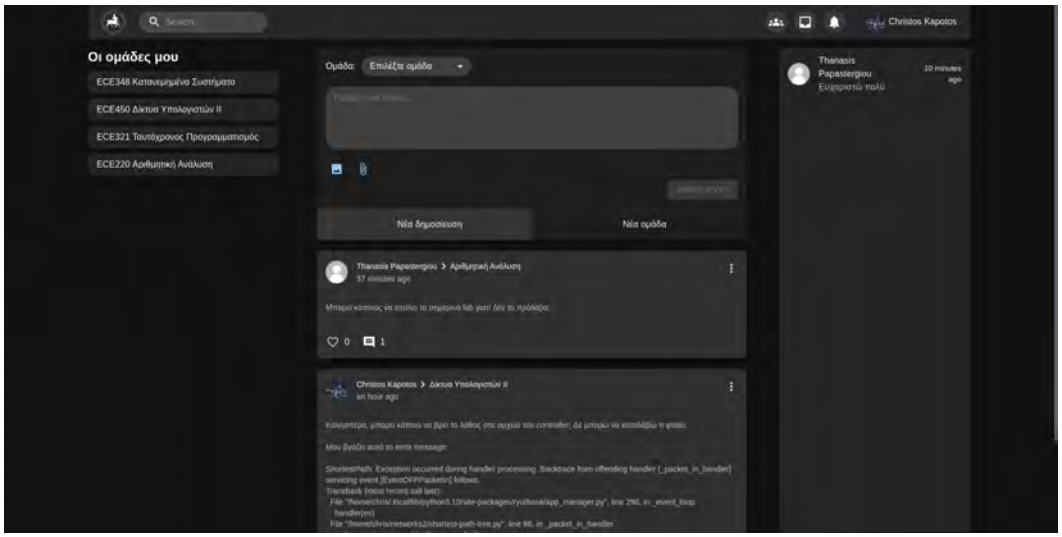

Figure 3.6: Home Page - Fetched Data

3.4 Group Page

On the group page, only the posts that are posted in that group are displayed, along with a list of all users who are registered in the group. The posts are fetched using the same logic as on the home page, and while they are being fetched, the same two loading posts are displayed. If a user is not a member of the group and visits its page, they cannot see the posts, instead a corresponding message is shown. Additionally, there is a search bar at the top of the page where users can search for specific posts using keywords or phrases.

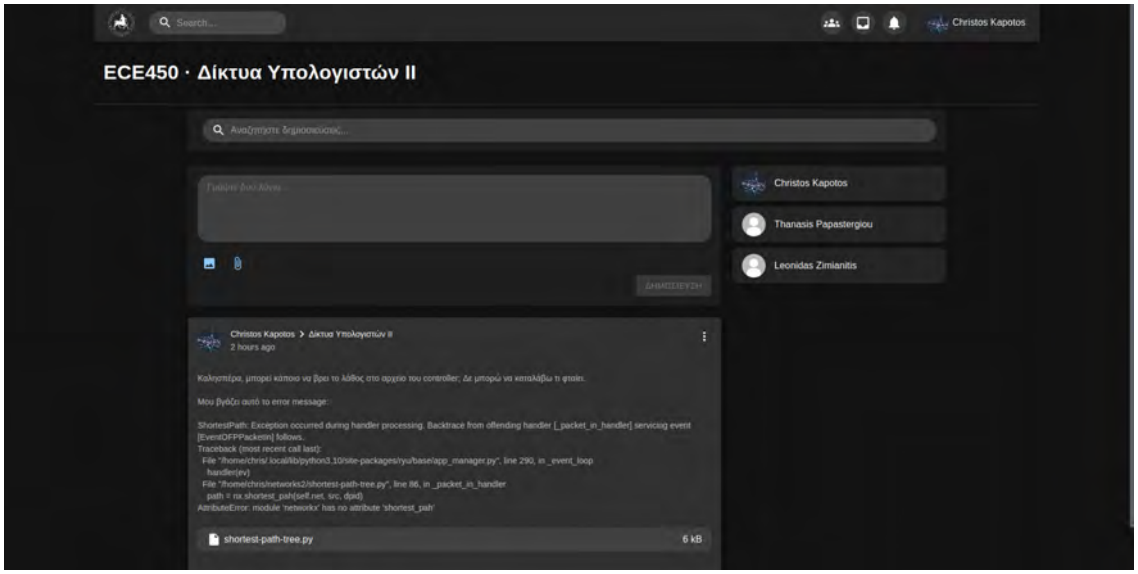

Figure 3.7: Group Page

Using the search functionality, the page does not change its structure, but instead the posts are filtered. For example, if the user searches for the keyword ”error”, only the posts that contain the word ”error” will be fetched. The filter is also applied to the pagination (fetching while scrolling) until the user refreshes or leaves the page.

3.5 Profile page

The user profile page consists of a vertically divided stack into 2 parts. The left part contains the user’s profile picture, name, and their posts, while the right part contains previews of the user’s conversations, which lead to the inbox page, just like on the home page.

If the profile page visited is another user’s, a chat icon is displayed beside their name. Clicking it will redirect the user to the chat room page.

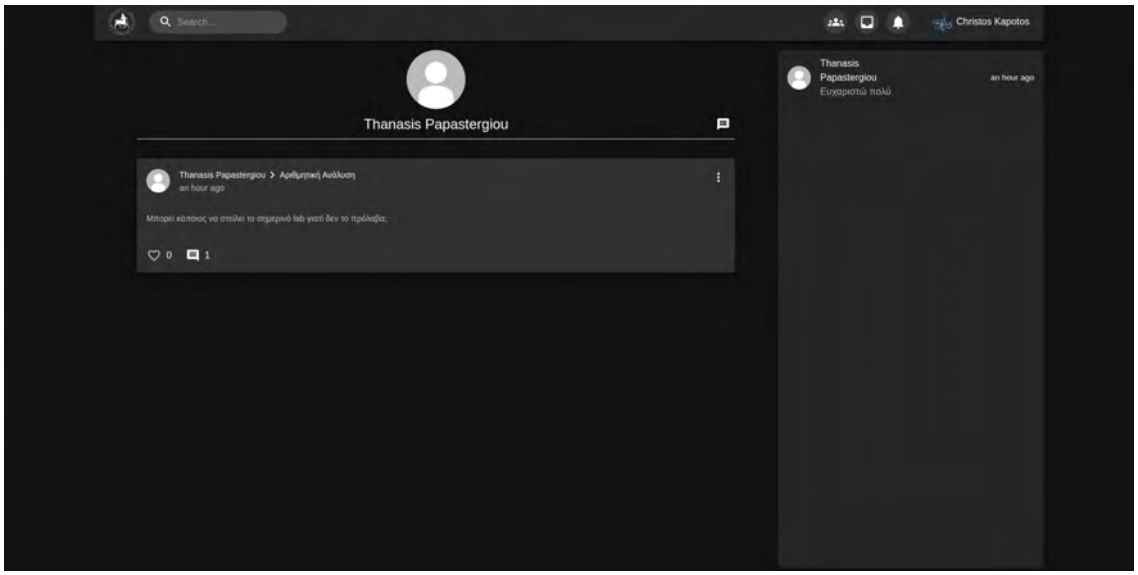

Figure 3.8: Profile Page of other user

On the other hand, if the profile page is the user's, there is no chat icon, but they can edit their profile photo by hovering and clicking the existing one, and selecting the new image.

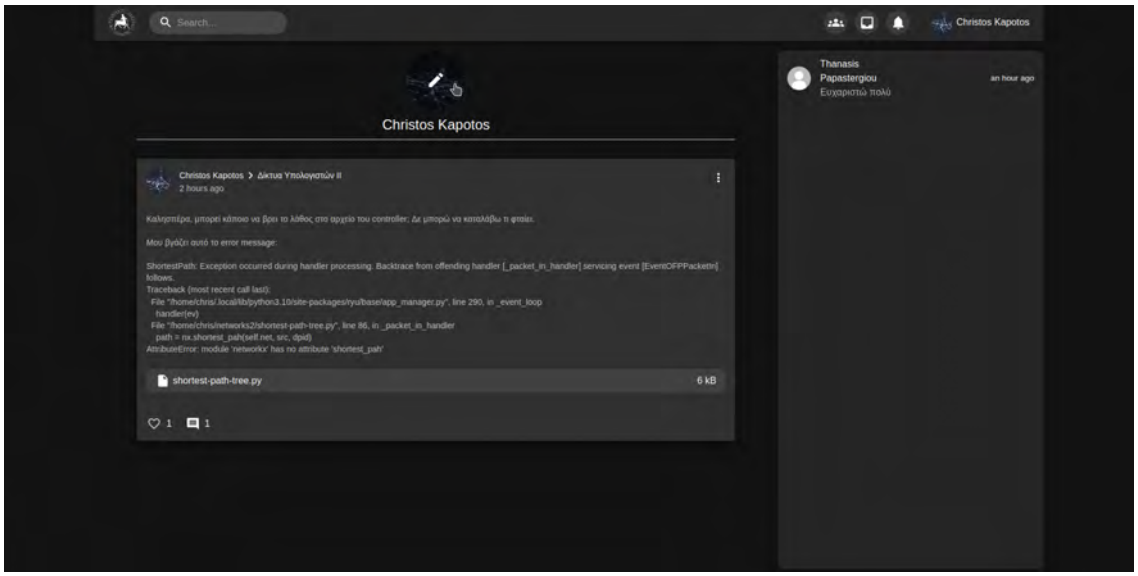

Figure 3.9: User's Profile Page

3.6 Post Page and Posts

The post page is as simple as it gets. It contains a vertically divided stack into 2 parts. The left part contains the post, while the right one is the messages box, that contains the user's

recent chats.

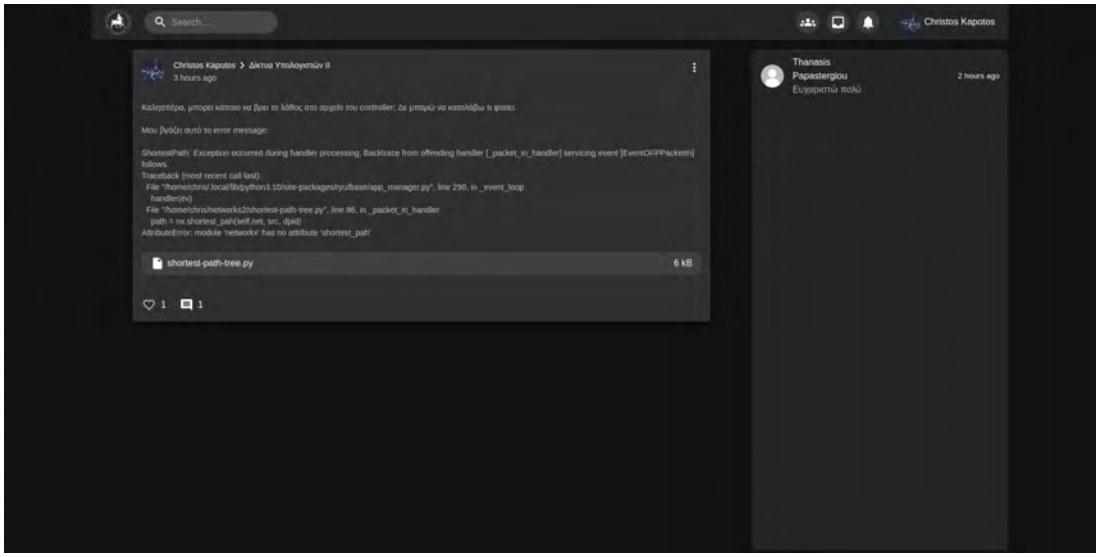

Figure 3.10: Post Page

A post can contain text (required), one or more images, and/or a file that users can download locally. Users can interact with the posts by liking and commenting. Similarly, a comment consists of the same elements as a post, and users can also like and reply to it. However, when it comes to replies, they can also include text, image(s), and a file, but other users can only like them.

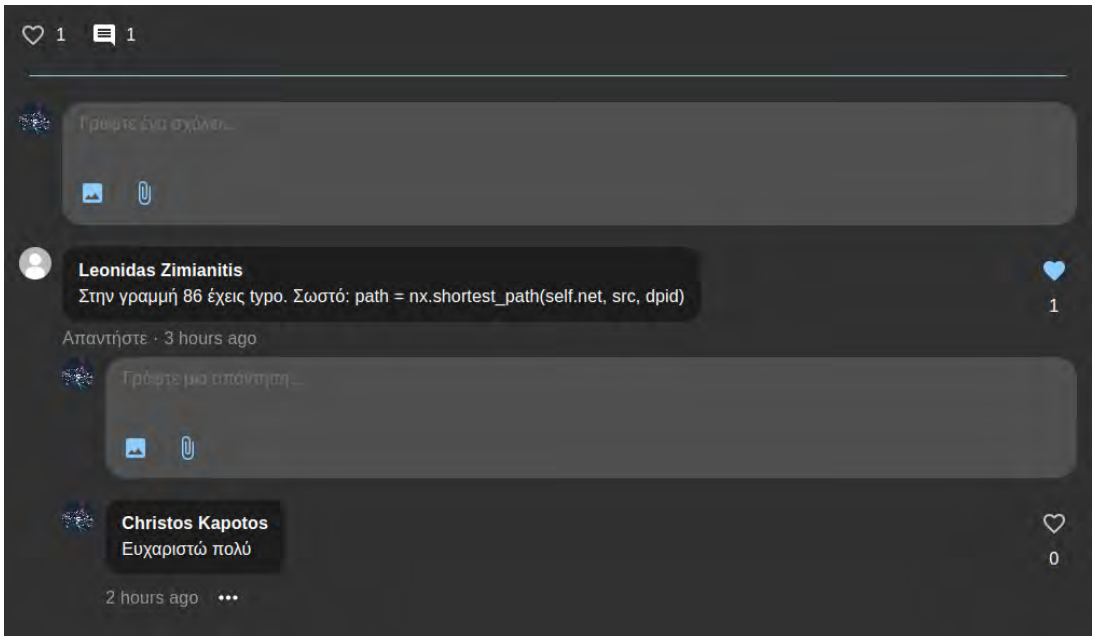

Figure 3.11: Post interaction/comment section

3.7 Inbox Page

The inbox page contains the list of the user's chats sorted from latest to oldest, and right next to the list, the messages of the selected chat room are displayed.

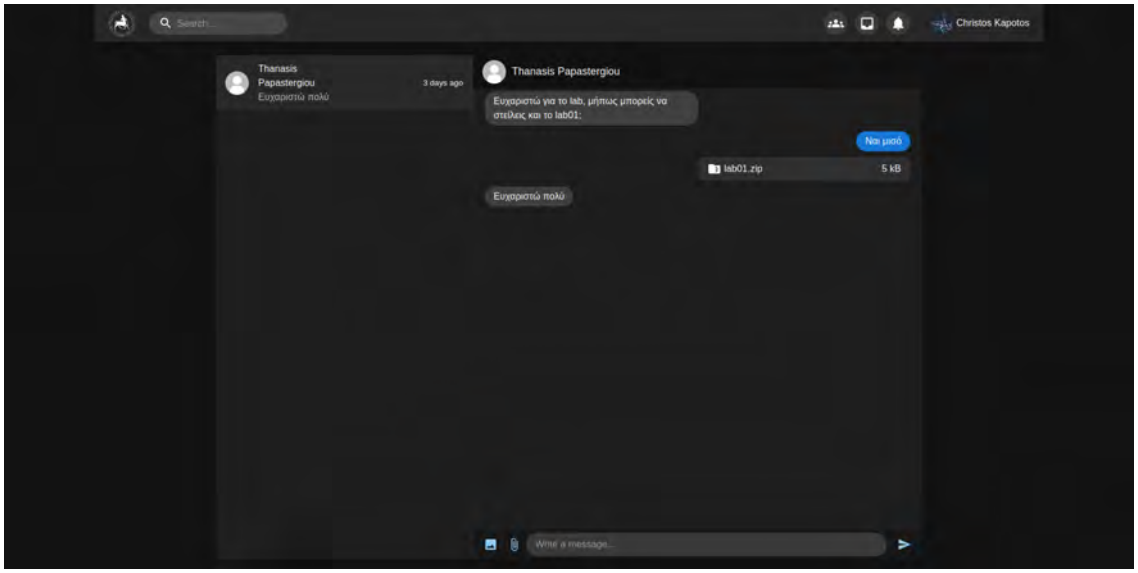

Figure 3.12: Inbox Page

If no chat room is selected, only an inbox icon is shown right in the center of the chat room box.

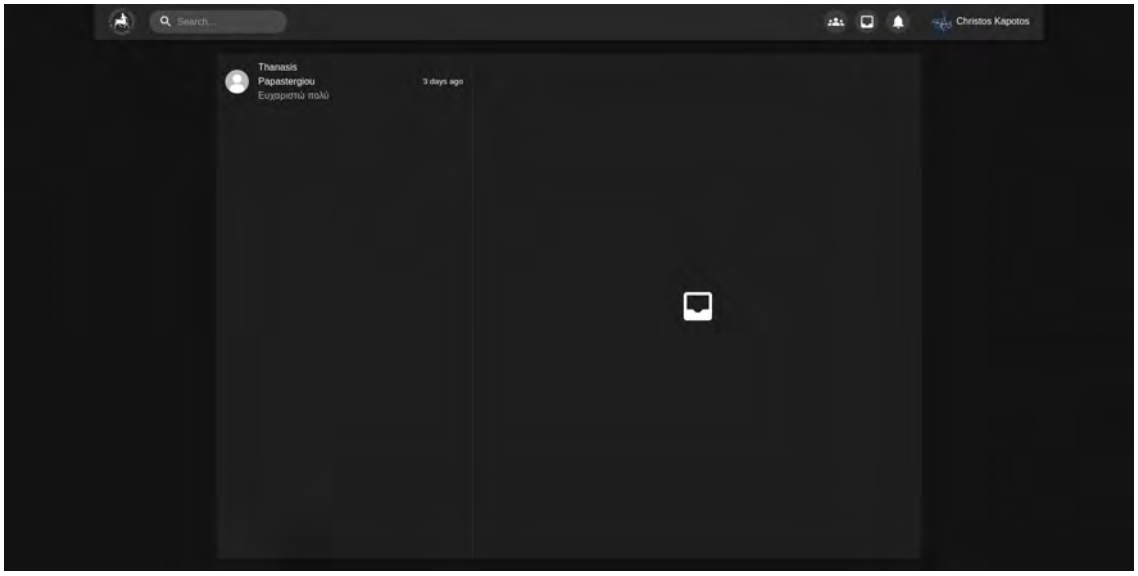

Figure 3.13: Inbox - No chat selected

As can be seen on the page, a message can be either contain an image, a file, or a text message.

Chapter 4

Application Development and Add-ons

As mentioned above, the stack used for this application's development is the MERN stack, which stands for MongoDB, ExpressJS, ReactJS, NodeJS. MongoDB is the platform that hosts our database, and doesn't really have any add-ons.

React was used for the frontend part of the application and Node with Express for the backend. These technologies alone, however, cannot build a fully functional and responsive application, that's why there were numerous add-on packets, that serve different purposes, that were used for the development.

4.1 Database

4.1.1 Setup

Setting up a database with MongoDB is very simple and takes little to no time. After creating the project (by clicking on "new project" and giving it a name), the user only has to configure some settings, like shown below.

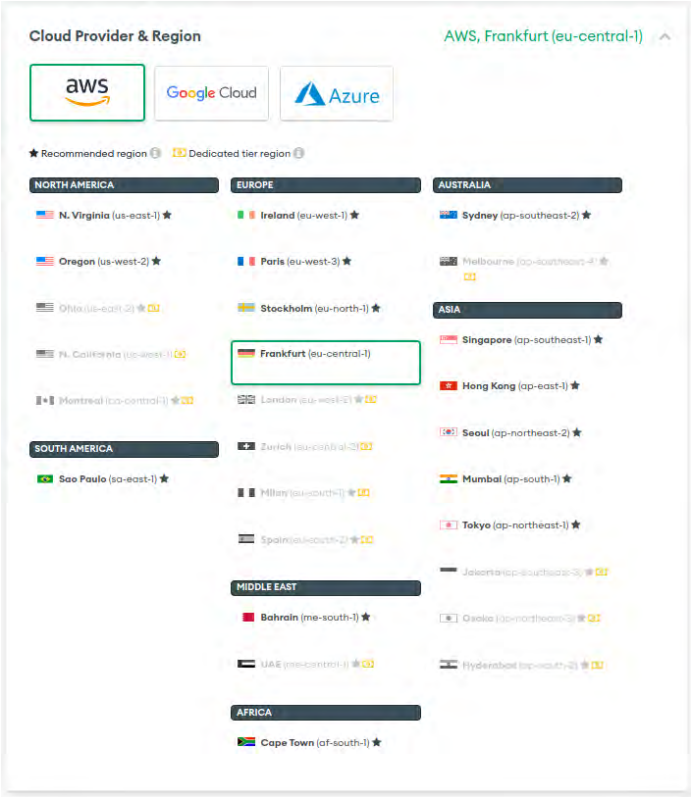

Figure 4.1: Database Setup pt.1

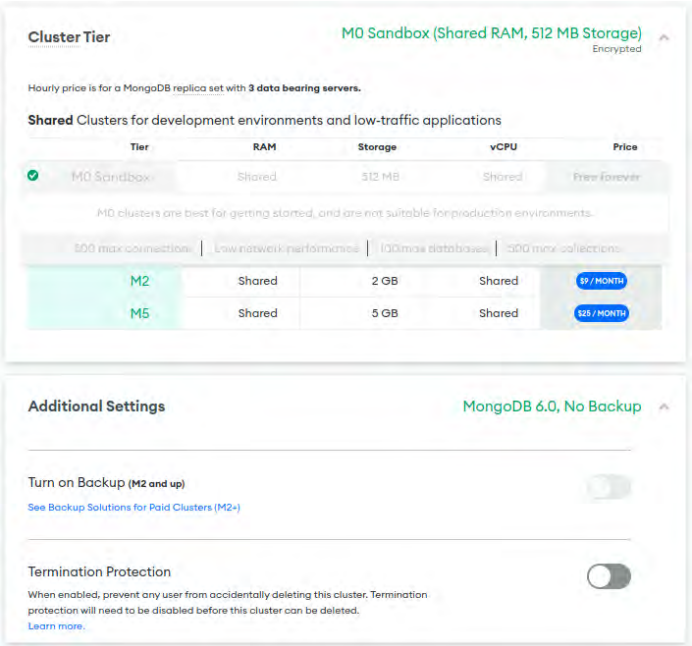

Figure 4.2: Database Setup pt.2

These configurations indicate which cloud provider will be used, and what region our data will be stored in. For this project, AWS and Frankfurt, the default options, were chosen. Other sets of configurations allow the developer to upgrade the cluster tier and expanding the storage size, or enabling backup.

After confirming these options, the developer has to wait several minutes and the database is all set up.

4.1.2 Connecting the server to the database

After the database is set up, MongoDB gives the developer several options for connecting it to the application, or access it with tools. For this project obviously the connection should be to the server application. For that, I copied the given link and pasted it to the Node application, as a parameter to the mongoose.connect() function. Mongoose is the tool that manages the connection and any contact between the server and the database.

```
mongoose.connect(process.env.MONGODB_CONNECTION_STRING, {
  useNewUrlParser: true,
  useUnifiedTopology: true,
});
```

Figure 4.3: MongoDB Database connection string

The reason the string is not visible, is because it is good practice to store valuable data that could be used for malicious purposes, inside a .env file that contains environment variables, and could be rendered invisible by the developer if the source code is uploaded in an online repository.

4.1.3 Collections

Instead of tables, a MongoDB database stores its data in collections. A collection is a list of documents that represent a specific entity that corresponds to the collection. More secifically, this project uses 3 collections: **users**, **groups** and **posts**. A document under the **users** collection would represent the data of a specific user.

4.1.3.1 users

- **_id**: MongoDB's identifier for each object in the database (even if nested inside a document)

- **firstName**: User's first name
- **lastName**: User's last name
- **email**: User's email
- **password**: Hashed string that corresponds to the password, for security reasons
- **verified**: Boolean variable that lets the server know whether the user has verified their email address
- **groups**: Array of the groups' _id's which the user a member of
- **pfp**: String with the base64 data of the user's profile picture
- **messages**: Array of objects that correspond to each of the user's chats
- **notifications**: Array of objects that contain the notification's text, the redirection link, and a boolean variable that becomes true when the user has read the notification
- **__v**: Version key for the user document. Initial value is 0, and is only incremented after operations that could cause a conflict, modifying array positions

4.1.3.2 posts

- **_id**: MongoDB's identifier for each object in the database (even if nested inside a document)
- **userName**: The name (firstName + lastName) of the user that created the post
- **userId**: User's _id
- **groupId**: _id of the group that the post was made in
- **groupName**: Group's name
- **text**: The text of the post
- **images**: Array of base64 images of the post. Default value: null
- **file**: Object with the post's file' information (name, data, size, type). Default τιμή: null

- **postedAt**: The date that the post was created. It has a specific format and is created through a new Date JavaScript object
- **likes**: Array of the _id's of the users that have liked the post
- **comments**: Array of objects that contain information about each comment, like the user's name, the user's _id, likes, text, image, file etc. Also contains an array of the replies, where each element consists of the same fields as a comment object, without a replies array
- **userPfp**: base64 string of the profile picture of the user that posted the comment
- **__v**: Version key for the post document. Initial value is 0, and is only incremented after operations that could cause a conflict, modifying array positions

4.1.3.3 groups

- **_id**: MongoDB's identifier for each object in the database (even if nested inside a document)
- **code**: The code of the subject
- **name**: The name of the subject
- **users**: Array of objects that contain information of every user that is a member of the group (_id, name, profile picture)
- **__v**: Version key for the group document. Initial value is 0, and is only incremented after operations that could cause a conflict, modifying array positions

4.2 Backend

The development of a Node application begins by setting up the environment and all the necessary node modules, by typing *npm init -y* in the command line, inside the project's server folder. This command creates and initializes all the default files needed for the development.

After initializing the application, and creating the starting point of the application (file index.js for this specific project), the rest of the add-ons were downloaded and used.

4.2.1 ExpressJS

Express is a back end web application framework for building RESTful APIs with NodeJS that provides the developer with super useful features, such as **robust routing**, **HTTP helpers (redirection, caching, etc)** etc.

The first thing that was done, even before connecting the server with the database, was initialize it with express, and also passing some important parameters with the body-parser and cors packages. At last, the server starts running, by listening to the port, that was again given through an environment variable.

```
const app = express();
app.use(bodyParser.json({ limit: "30mb", extended: true }));
app.use(bodyParser.urlencoded({ limit: "30mb", extended: true }));
app.use(cors());

app.use("/auth", authRoutes);
app.use("/groups", groupRoutes);
app.use("/posts", postRoutes);
app.use("/profile", profileRoutes);
app.use("/messages", messageRoutes);

mongoose.set("strictQuery", false);

mongoose.connect(process.env.MONGODB_CONNECTION_STRING, {
  useNewUrlParser: true,
  useUnifiedTopology: true,
});

const server = app.listen(process.env.PORT, () =>
  console.log(`Server running on port ${process.env.PORT}`)
);
```

Figure 4.4: Server Initialization

4.2.2 body-parser

body-parser is a Node module which provides middleware for parsing HTTP request bodies. It supports JSON and url-encoded formats and does not support multipart requests. More specifically, it parses a request that was sent to the server in the form of bits and converts it into a format from which the developer can easily extract relevant information that they may need.

4.2.3 cors

Cross-origin resource sharing (CORS) is a mechanism that allows restricted resources on a web page to be requested from another domain outside the domain from which the first resource was served, which means, the developer can specify whether or not the server can accept cross-origin requests (requests from an address other than the destination server address).

4.2.4 JSON Web Token (JWT)

JSON Web Token is a proposed Internet standard for creating data with optional signature and/or optional encryption whose payload holds JSON data that asserts a number of claims. These tokens are signed either using a private secret or a public/private key.

In this specific project, JWT is used to verify a user is correctly logged in, by attaching the token to every request, and the server verfying it with jwt.verify(). This is the function that accepts as parameters the token and a secret key that has to match with the one that is embedded into the JWT that is sent to the user when he logs in. This way, only the server knows the secret key and if the client sends an invalid JWT, the request is ignored.

4.2.5 nodemailer

NodeMailer is a NodeJS module that allows the application to send emails from the back-end. It is a zero dependency module for all NodeJS-compatible applications. The emails sent can be plain text, attachments, or HTML. To set up the transporter (mailer functionality) the developer only needs to enter the sender email address' credentials (Gmail requires the developer to use a custom password that Google generates for this specific purpose) as parameters to the nodemailer.createTransport() function as shown below.

```
export const transporter = nodemailer.createTransport({
  service: "Gmail",
  auth: { user: process.env.SENDER_EMAIL, pass: process.env.SENDER_EMAIL_PASSWORD },
});
```

Figure 4.5: Email Transporter Initialization

4.2.6 Mongoose

Mongoose is an ODM (Object Data Modeling) library that manages the connects between the Node application and the MongoDB database. With mongoose, the developer can specify the structure of each collection (the 3 collection of this project are the users, the groups and the posts).

4.2.7 bcrypt

Bcrypt is a password-hashing library that uses salting. A salt is a random string. By hashing a plain text password plus a salt, the hash algorithm's output is no longer predictable. The same password will no longer yield the same hash. The salt gets automatically included with the hash, so there is no need to store it in a database.[12] In this project, 12 salting rounds are used to hash the user's password.

4.2.8 socket.io

Socket.IO is a library that enables low-latency, bidirectional and event-based communication between a client and a server. In this project, sockets are used to enable chat and notifications functionalities.

In the node app, after the server is initialized and has started listening, a map data structure of the online users is created. This means that for every online user, their id (MongoDB _id identifier) is mapped to their socket id. This socket id will be used to send live messages and notification to them, while they are online. When they disconnect, their entry on the map is removed.

```
global.onlineUsers = new Map();
global.sockets = [];

// console.log(global);

io.on("connection", (socket) => {
  // console.log(socket);
  console.log("user connected");
  global.sockets.push(socket);
  socket.on("add-user", (userId) => {
    global.sockets.find((s) => s.id === socket.id).data.userId = userId;
    onlineUsers.set(userId, socket.id);
    // console.log(socket.id);
    console.log(global.sockets.map((s) => s.id));
    console.log(onlineUsers);
  });
  socket.on("disconnect", () => {
    console.log("user disconnected");
    global.onlineUsers.delete(global.sockets.find((s) => s.id === socket.id).data.userId);
    global.sockets = global.sockets.filter((s) => s.id !== socket.id);
    console.log(global.sockets.map((s) => s.id));
    console.log(onlineUsers);
  });
});
```

Figure 4.6: Online users' map initialization

4.3 Frontend

React is one of the most popular JavaScript libraries for several reasons. Its most significant feature is the construction of Single Page Applications (SPAs). In other words, a React application loads only one web document in the browser upon startup, and from there on, it updates the page content with requests that the user doesn't see happening. This means that for every change in the application's URL (routing), there is no need to fetch a new HTML file, which makes the application faster compared to a non-SPA application.

Creating a React application is as easy a Node application, and can be done simply by typing the *npx create-react-app .* command inside the project's client folder. This initializes the application with the most basic file structure.

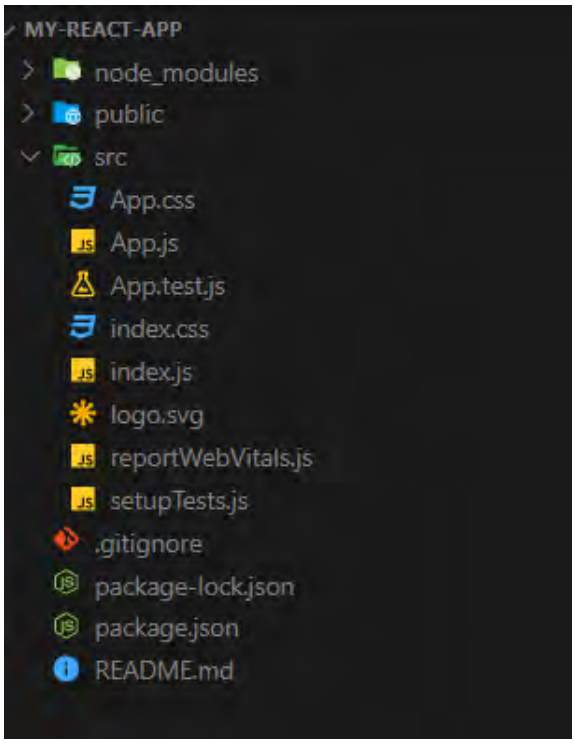

Figure 4.7: React project file structure[5]

The index.js is where the root element of the DOM is created and rendered as a component, which usually exists in the App.js file and is referenced as <App /> thanks to the JSX syntax.

In addition to that, building a full scale React project requires using numerous add-on packages.

4.3.1 react-router-dom

React Router DOM is a React library that enables the implementation of dynamic routing in a web application. Unlike the traditional routing architecture in which the routing is handled in a configuration outside of a running app, React Router DOM facilitates component-based routing according to the needs of the app and platform.[13]

4.3.2 Redux

Redux is a global state management library. These libraries are necessary because without global state, for a component that is a leaf in the DOM tree to have access to a state in the root, it would have to be passed as a prop through all intermediate components. In contrast,

with global state, all components have independent access to the store that stores the state of the entire application.

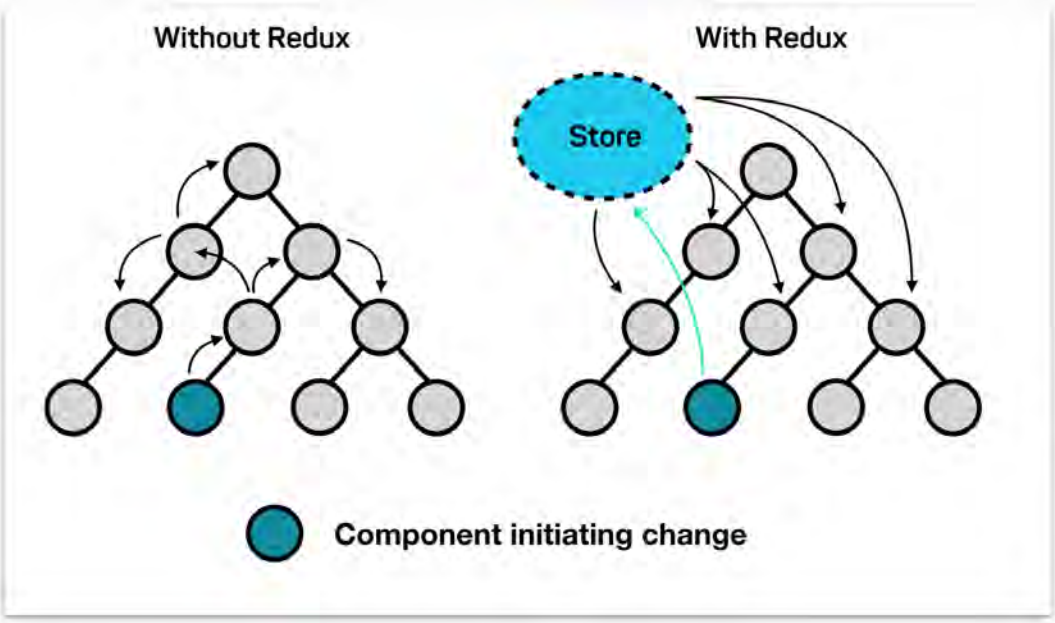

Figure 4.8: Global state visualization

4.3.2.1 Store

In Redux, the store is a central object that holds the application's state. It serves as a single source of truth for the entire application, allowing components to access and update the state in a predictable manner, via the reducers.

4.3.2.2 Actions

In Redux, actions are plain JavaScript objects that describe an intention to change the state of the application. They are the only source of information that gets sent to the Redux store.

An action typically has a type property that describes the type of action being performed. This property is usually defined as a string constant. Additionally, an action can include additional data or payload that provides information necessary to update the state.

4.3.2.3 Reducers

Reducers are pure functions responsible for handling state changes in the application. They define how the application's state should be updated in response to dispatched actions.

A reducer takes two parameters: the current state and an action. It then evaluates the action's type and performs the necessary state transformation based on that action. The reducer should not mutate the existing state; instead, it returns a new state object that reflects the updated data.

Reducers in Redux follow the principle of immutability, meaning they create new copies of the state instead of modifying the original state directly. This ensures that the application's state changes are tracked efficiently and allows for easier debugging and time-travel debugging with Redux DevTools.

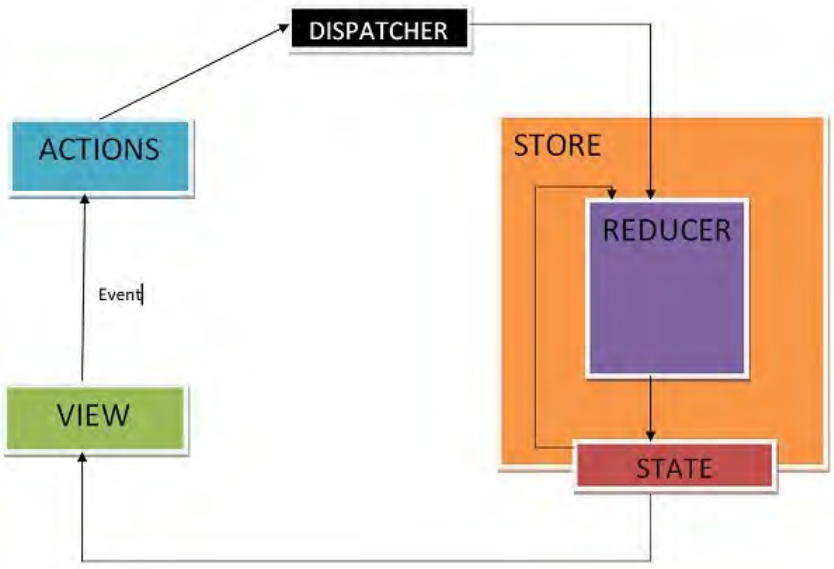

Figure 4.9: Redux state management flow[6]

4.3.3 Axios

Axios is a promise-based HTTP JavaScript library. It has the ability to make HTTP requests from the browser and handle the transformation of request and response data. It offers different ways of making requests such as GET , POST , PUT/PATCH , and DELETE.

4.3.4 Material-UI/MUI

Material-UI is a popular open-source library for building user interfaces in React applications. It provides a set of pre-designed, customizable components and utilities that follow the Material Design guidelines created by Google.

4.3.5 socket.io-client

socket.io-client is the client-side component of Socket.IO, allowing you to establish a connection from the client to a Socket.IO server.

In this diploma thesis project, client sockets are used for receiving notifications and chat messages from the server socket, so that they are updated live on the client side, without needing to refresh the page.

4.3.6 MomentJS

MomentJS is a JavaScript library which helps is parsing, validating, manipulating and displaying date/time in JavaScript.[14]

In this specific project, MomentJS is used for post and chat timestamps like ”1h ago” if, for example, the post was posted 1 hour ago.

4.3.7 React Developer Tools

React Developer Tools is a Chrome DevTools extension for the open-source React JavaScript library. It allows developers to inspect, debug, and profile React components in their applications. It provides valuable insights into the React component hierarchy, state changes, and performance optimizations.[15]

4.3.8 Redux Developer Tools[2]

Redux-Devtools provides a debugging platform for Redux apps. It allows the developer to perform time-travel debugging and live editing. Some of the features in official documentation are as follows:

- It lets the developer inspect every state and action payload.
- It lets the developer go back in time by “cancelling” actions.
- If the developer changes the reducer code, each “staged” action will be re-evaluated.
- If the reducers throw, we can identify the error and also during which action this happened.

- With persistState() store enhancer, the developer can persist debug sessions across page reloads.

Chapter 5

Git, Source Code and Production URLs

5.1 Version Controll and Git

Version control is a system that helps track and manage changes to files and folders over time, and lets the developer backtrack to previous version if needed.

For this project, I used Git, a distributed version control system (DVCS) widely used for tracking changes in source code, that provides efficient branching and merging and has commit-based versioning.

5.2 Source Code

`https://github.com/ChrisKap00/diplomatiki`

5.3 Production URLs

5.3.1 Frontend/Actual URL of the application

`https://ece-thesis-app.netlify.app/`

5.3.2 Backend

`https://thesis-app-5ce575de5da4.herokuapp.com/`

Chapter 6

Future Development

A social media type application like this one, can support countless features to provide the user with a smooth and complete experience. Some of these smart features, such as infinite scrolling, have already been implemented, but there are certainly several others that could be implemented in the future, and are divided into two categories: **Improvements on existing functionalities** and **New functionalities**.

A future plan for the app, that doesn't fall into either of the two categories would be expanding the frontend to a mobile environment, using React Native. That implies that some mobile-exclusive features, like push notifications, can be added.

6.1 Improvements on existing functionalities

- Smarter fetch for each post, that contains a file. This means that initially only the description of the file (name, size, type) is sent to the client, and if the user wants to download it, only then the content is fetched from the server.
- Smarter fetch for each post's comments. Only fetch the comments of a post if the user opens the comment section, and fetch a file only if the user wants to download it.
- Smarter fetch for the user's chats. Only fetch a chat's messages if the user opens the specific chat, and keep fetching messages with infinite scrolling.

6.2 New functionalities

- Active status for users.

- Calls and Video Calls in the chat rooms.
- Code block functionality in posts, so that the users can upload code and display it properly.
- Files block in group pages, so that users can see all the files shared within the group
- Live state update, as soon as another user likes a post or uploads a post, a comment or a reply or even when they join a group, using sockets and thus, erasing the need for refreshing the page to see the updates.

Bibliography

[1] Representational state transfer. https://www.ibm.com/topics/rest-apis. Visit date: 13-05-2023.

[2] Redux dev tools. https://www.tutorialspoint.com/redux/redux_devtools.htm. Visit date: 13-05-2023.

[3] Dom image. https://www.freecodecamp.org/news/what-is-the-dom-document-object-model-meaning-in-javascript/. Visit date: 13-05-2023.

[4] React data flow. https://bosctechlabs.com/react-state-vs-props-introduction-differences/. Visit date: 13-05-2023.

[5] React file structure. https://medium.com/swlh/demystifying-the-folder-structure-of-a-react-app-c60b29d90836. Visit date: 13-05-2023.

[6] Redux flow. https://www.tutorialspoint.com/redux/redux_data_flow.htm. Visit date: 13-05-2023.

[7] Http methods. https://developer.mozilla.org/en-US/docs/Web/HTTP/Methods. Visit date: 13-05-2023.

[8] Http status codes. https://developer.mozilla.org/en-US/docs/Web/HTTP/Status. Visit date: 13-05-2023.

[9] Jsx. https://en.wikipedia.org/wiki/JSX_(JavaScript). Visit date: 13-05-2023.

[10] State. https://www.simplilearn.com/tutorials/reactjs-tutorial/reactjs-state. Visit date: 13-05-2023.

[11] Html dom. https://www.w3schools.com/js/js_htmldom.asp. Visit date: 13-05-2023.

[12] bcrypt. https://heynode.com/blog/2020-04/salt-and-hash-passwords-bcrypt/. Visit date: 13-05-2023.

[13] React router dom. https://blog.logrocket.com/react-router-dom-tutorial-examples/. Visit date: 13-05-2023.

[14] Momentjs. https://www.tutorialspoint.com/momentjs/momentjs_quick_guide.htm. Visit date: 13-05-2023.

[15] React dev tools. https://chrome.google.com/webstore/detail/react-developer-tools/fmkadmapgofadopljbjfkapdkoienihi. Visit date: 13-05-2023.