COSC2430 - Web Programming

A Full-Stack Web Application

Ramon Casas i Luque

September 2024

Contents

1 Introduction

This project focuses in developing a full-stack web application aimed to provide an online learning platform. The aim of the platform is to allow users to browse, enroll, and offer courses on various topics. The goal was to create a functional, user-friendly interface with responsive design, ensuring accessibility across different devices. In addition, the backend supports secure user authentication, course management, and dynamic content rendering based on user roles.

Moreover, the project also involved creating a database structure to manage users, courses, and categories efficiently. Instructors can create new courses, while learners can browse and enroll in them based on their interests. The system also includes features like profile management, course details, and browsing to increase user good experience when accessing the site.

As for the group project, I ended up working independently, which was a decision we reached with the only other student in my class. After working for some time together, and seeing that we did not agree on some starting points, I felt that working separately would give us more flexibility and allow each of us to delve deeper into learning full-stack development and web programming. While this project was completed on my own, I think that I have a solid background in working with teams, both in academic settings and in my professional experience. Tools like Git and Microsoft Teams have been part of my tools used during my studies, enabling efficient collaboration on various tasks and projects.

2 Requirement analysis

In this section, I will analyze the project requirements based on the instructions provided for the assessment of the COSC2430 - Web Programming course. The course focused on understanding and developing web applications using client-server models and technologies. The assessment required us to apply this knowledge by building a full-stack application that fulfills specific functionalities.

2.1 User Roles and Functionalities

One of the main requirements for the platform was the integration of four distinct user roles, each with specific permissions and interactions with the system:

- **Admins:** These users have the most control over the system, managing features and monitoring other users. This involves managing instructors, learners, and guests.
- **Instructors:** Instructors use the platform to manage their virtual classrooms. They can create and manage courses, set pricing, upload content, and monitor their interactions with learners. They are also required to pay periodic subscription and commission fees.
- **Learners:** These users are the primary customers of the platform. They can browse and enroll in courses, save favorite instructors, and review their past transactions.
- **Guests:** Guests can browse the platform but are limited in their interaction until they register as learners. They cannot enroll in courses or save favorites.

2.2 UI and Device Responsiveness

The platform needs to be accessible and user-friendly across multiple devices, which requires supporting three types of screen sizes:

- **Desktop Computers:** 1200 pixels or higher in width. These users need to see a fully horizontal navigation menu and multi-column layout.
- **Tablets:** 768 to 1199 pixels width. The navigation menu toggles between visible and hidden states, while the layout collapses into fewer columns or a single-column structure.
- **Smartphones:** 767 pixels or lower in width. The navigation becomes vertical and toggle between hidden and visible, while forms and content rearrange to fit a smaller screen.

In addition to responsive design, the platform needs to be professional and consistent across all pages, with high contrast for readability and ease of navigation. These UI requirements were fully incorporated into my project.

2.3 Core Features and Pages

The core functionality required several important pages, each designed to meet the specific needs of the platform:

- **Home Page:** divided into Header, Footer, and Main Content Area, this page showcases new instructors, new courses, featured instructors, and featured courses.
- **About Us Page:** Provides information about the team members involved in the project. Clicking on a member's photo opens a modal window with more details about that individual.
- **Pricing Page:** Lists the various subscription fees, commission fees, and premium membership options available on the platform.
- **My Account Page:** displays personalized information based on the logged-in user. For guests, it redirects to the login page, which offers options to log in, register, or recover a password.
- **Instructor Profile Pages:** Allows instructors to showcase their courses and provides learners with a detailed view of their characteristics.

2.4 Accessibility and SEO

In addition to creating a user-friendly interface, accessibility and search engine optimization (SEO) were important requirements for the platform. This include using semantic HTML elements, and providing alternative text for images.

3 System design

In this section, I will explain how the various components of the project were structured, including the architecture, tools, and database design. Additionally, I'll cover the implementation of user authentication and session management, which are very important for the functionality of the platform.

3.1 Architecture

The project follows a **Model-View-Controller (MVC)** architecture. Here's how it was implemented:

- **Model**: MongoDB collections were used to store and manage the data for the platform. Each collection represents different entities, such as users, courses, and categories. These models define the structure of the data and handle interactions with the database.
- **View**: EJS (Embedded JavaScript) templates were used for rendering dynamic HTML content. These templates take the data from the models and present it in the appropriate format to the user. I had to use some partials views to have the headers and the footers equal in all the pages throughout the platform.
- **Controller**: I used Express.js to handle the routing and logic. The main file, `server.js`, is responsible for processing HTTP requests. I used `app.get` and `app.post` routes to manage the different actions taken by users on the platform, such as logging in, registering, or adding courses. I also implemented middleware functions were to manage user sessions, authentication, and error handling.

3.2 Technologies Used

To build this project, I used the following technologies and tools we saw during the course sessions this semester:

- **Express.js**: As the web framework for handling server-side logic and routing.
- **MongoDB**: To manage and store the platform's data in different collections.
- **Node.js**: To run the backend server and handle requests.
- **npm*: To manage various packages and dependencies.
- **HTML, CSS, and JavaScript**: For creating the structure, styling, and interactive elements of the frontend.

3.3 Database Design

The database consists of several collections, each serving a specific purpose. These collections are organized as follows:

- **Categories**: Stores the different categories of courses, such as Science, Programming, etc. This allows users to filter and browse courses by category.

- **Courses**: Contains all the information related to a course, such as the course name, description, price, main image, and the instructor's details. This collection is central to the platform's functionality, as it handles the key offerings of the platform.
- **FAQs**: Stores a list of frequently asked questions and their corresponding answers. This helps users quickly find information about the platform.
- **Fees**: Contains information about the various fees, including subscription fees and commission fees for instructors.
- **Members**: Stores details about the team members for the "About Us" section.
- **Users**: Manages all users of the platform, including instructors, learners, and admins. This collection holds user information such as email addresses, passwords (encrypted), and account types (instructor or learner).

The **Courses** and **Users** collections are the most important ones in the platform, as they represent the core functionality of managing courses and user interactions.

3.4 User Authentication and Session Management

For user authentication and session management, I employed:

- **express-session**: Used to manage user sessions, ensuring that users stay logged in as they navigate through the platform.
- **bcrypt**: Used to securely hash user passwords before storing them in the database. This ensures that even if the database is compromised, the passwords remain protected. This hashing is done in the server side and can be seen in the User.js model file.
- **crypto**: Used to generate secure tokens for password recovery. When a user requests to reset their password, a unique token is generated and shown on screen (as simulating sending an email), allowing them to securely reset their password without exposing their information. I also used this built in Node.js module to generate the secret key for the session: `node -e "console.log(require('crypto').randomBytes(32).toString('hex'))"`

3.5 Conclusion

I believe that the system design follows a good structure that combines the different technologies we have seen during the course: the combination of MongoDB for data management, Express.js for routing and logic, and EJS templates for rendering dynamic content have enabled me to build a solid and functional website.

4. Source Code Management

To be able to have an organized code and to scale it without many issues, it is very important to manage the source code consciously. In this section I deal with this topic.

4.1 Version Control

Throughout the project, I used Git for version control. Git allowed me to track changes, manage feature branches, and revert to previous states when needed. I committed changes regularly to make sure that every update was saved and could be reviewed later.

4.2 Package Management

To manage dependencies, I used **npm** (Node Package Manager). This ensured all necessary packages were installed correctly and I could keep track of which packages and tools I was using. Here are the npm packages I installed and used:

- **express**: to handle HTTP requests and responses in the application.
- **mongoose**: to communicate with MongoDB, used for defining schemas and managing database operations.
- **ejs**: as a templating engine for dynamically generating HTML views on the server.
- **express-session**: to handle user session management, enabling functionalities like login and session persistence.
- **bcrypt**: to hash and store user passwords safely.

- **dotenv**: to manage environment variables by loading configuration data (like the database URI) from an `.env` file. In git, this file can be placed in `gitignore` to mantain security.
- **multer**: to deal with file uploads, particularly for profile pictures.
- **slugify**: to generate URL slugs for course and category names.

4.3 Environment Variables

As I said in the previous section, sensitive data like database URIs and session secrets were managed using environment variables stored in a `.env` file. This helped keep credentials and other sensitive information out of the source code, ensuring that they were not exposed publicly.

Here are the environment variables used in the project:

- **MONGODB_URI**: The connection string for the MongoDB database. This avoids exposing private credentials.
- **MONGODB_ATLAS_URL**: the URL path used to redirect the Admin users to the MongoDB Atlas database management.
- **SESSION_SECRET**: secret key used for signing session cookies and enhancing security for user sessions.
- **PROFILE_PICTURES_PATH**: Defines the path where uploaded profile pictures are stored on the server.
- **PROFILE_PICTURES_URL_PATH**: The URL path used to serve profile pictures to the client-side.

To manage these variables effectively, I utilized the **dotenv** package, which loads the environment variables from the `.env` file at runtime, ensuring they are available to the application without hardcoding them directly in the source code. This way, I can change these paths or URIs without having to review the whole code.

4.4 Code Organization

The codebase was structured to separate concerns and improve maintainability:

- **server.js**: entry point of the application, handling routes and middleware setup.
- **routes/**: contains the routing logic for different parts of the application, such as user login, course management, and content browsing.
- **models/**: has the MongoDB models, like `User`, `Course`, `Category`…
- **views/**: contains the EJS templates used to render dynamic HTML pages.
- **public/**: stores static files such as CSS files, JavaScript files, and images.

4.5 Conclusion

In this project, source code management was very important to maintain an organized and secure development process. By utilizing the different tools explained above, the project was able to grow and remain stable. The clear separation of concerns in the codebase increased both scalability and readability.

5 Frontend Development

The frontend of this project was built using HTML, CSS, and JavaScript, with the aim of providing a responsive, user-friendly interface across various devices. In this section, I will discuss how each of these technologies was applied to build the interface, focusing on HTML for structuring, CSS for styling and layout, and JavaScript for interactivity.

5.1 HTML

In this project, I used HTML to design the structure of the user interface. I used semantic elements like `<header>`, `<nav>`, `<main>`, and `<section>` to ensure the content is structured properly. I coded key components such as navigation bars, forms, and content sections with a combination of `div`, `ul`, `li`, and `nav` elements. I used conditionals in EJS templates to adapt content based on user roles, ensuring that specific elements are only shown to the relevant users, as for example instructors or learners.

Some examples of extensive pages in this project are the **Home Page**, the **My Account** page and the **Course Detail** page. These pages provide dynamic content rendering based on user session data. For example, the **My**

Account page displays different sections, such as course management and trial courses, based on whether the user is an instructor or learner. Moreover, it provides a special menu for Admin users.

Additionally, I used partials in EJS to create reusable elements like the header and footer. This approach reduced repetition in the code and allowed me to manage updates across multiple pages by updating just one partial file, which I found very convenient.

Key Features:

- **Conditional Rendering:** For example, instructors have access to course management, while learners see their enrolled courses.
- **Reusable Components:** Header, footer, and form templates were reused across pages using EJS partials.
- **Forms:** forms are very present in all the project for user input, from logging in to adding new courses and editing existing ones. I constructed forms using semantic HTML elements.

5.2 CSS

The CSS for this project focuses on a responsive design that adapts to different screen sizes (desktop, tablet, and smartphone). I used Flexbox (mainly) and CSS Grid (in some cases) to manage layout and alignment across various components.

All styles were included in a single CSS file for this project, which was manageable at this stage. However, I think that as the project grows, I should split the CSS into multiple files for different sections.

I implemented the responsive design using media queries to ensure that navigation bars and forms are styled appropriately for different screen sizes. For example, a horizontal navigation menu on desktops switches to a vertical, collapsible menu on mobile devices. Similarly, forms that are inline on desktop devices stack vertically on mobile devices to provide better readability.

Key Features:

- **Flexbox and Grid:** Used for flexible and responsive layouts across the application.
- **Responsive Design:** Implemented using media queries to adapt the layout to different devices (desktop, tablet, mobile).
- **Hamburger Menu:** For smaller screens, the navigation menu is replaced by a hamburger icon that toggles the visibility of the menu.
- **Color palette:** As we learned in one of the sessions of the course, it is important to use compatible colors throughout a website. I used the Chinese color palette of the website flatuicolors, available here.

5.3 JavaScript

JavaScript is a very important tool to increase the interactivity of the project. It is used to manage form validation, modals, and dynamic page updates based on user input. Client-side form validation is implemented to improve user experience by providing instant feedback (like password matching).

I stored many of the scripts in a dedicated folder in order to be able to reuse them across different EJS pages. A good example is the **password matching feature**, which is reused across different templates (registration and password recovery). Modal management is also coded using JavaScript and reused in more than one page.

Additionally, I used JavasScript in middleware functions in the server handle request validation and authentication, ensuring that only authorized users can access certain pages or perform certain actions. This showed me that JavaScript is used bothe in front and back end scenarios.

Key Features:

- **Client-Side Validation:** For example, checking password matching in registration forms.
- **Reusable Scripts:** Functions like modal management and form validation are reused across different parts of the project.
- **Middleware:** `ensureAuthenticated` and `redirectIfAuthenticated` functions were used to control user access based on session status.

5.4 Conclusion

The combination of HTML, CSS, and JavaScript provides a solid scenario for the frontend development of this project. HTML's semantic elements ensure that the content is structured in a clear and accessible way, which can contribute to styling later and increasing usability. CSS is essential in creating a responsive design that adapts to different screen sizes, while JavaScript adds an extra layer of interactivity, enhancing the user experience through client-side validation and modal management. In this case, testing on various devices and browsers was crucial to ensuring a smooth user experience, and this process helped to identify and fix problems related to responsiveness and functionality.

6 Data management

Effective data management is very important for any full-stack application. In this case I used MongoDB as the database system, together with Mongoose, a MongoDB object data modeling library, to perform data operations. Additionally, I did an account at MongoDB Atlas to host the project's database. This section provides an overview of the database models and relationships, CRUD (create, read, update, delete) operations, and the techniques used to manage security and data integrity.

6.1 MongoDB Atlas

To store and manage the data for this platform, I signed up for a free account on MongoDB Atlas, a cloud database solution. I created a cluster on the AWS free tier, ensuring that the project data is accessible, backed up, and scalable if I need it in the future. This platform allows for collaboration and access across different environments. This service was crucial for ensuring reliable access to the database from any location, reducing the complexity of managing a local database instance. MongoDB Atlas offers built-in tools for monitoring and performance optimization, but I haven't used them yet.

6.2 Data Models

The data models are the key of the database. They define the structure and relationships between different entities in the database. These models are built using Mongoose and coded in `.js` files, which are stored in the `/models` directory.

Each model corresponds to a collection in the **MongoDB** database, representing various entities such as users, courses, categories, and more. Relationships between these models are established using references (`ObjectId`), enabling data retrieval and manipulation. For example, a user may be associated with multiple courses, and each course can belong to different categories.

Here is a list of the key models I used in the platform:

- **User**: stores information about the platform's users, including instructors, learners, and their respective roles. Key attributes include `email`, `phone`, `password`, `accountType` (either 'Instructor' or 'Learner'), and personal details such as `firstName`, `lastName`, `address`, etc. The schema also includes specific fields for instructors, such as `schoolName`, `jobTitle`, and `specialization`. The model also includes references to courses that users instruct (`courses`) and those they are enrolled in (`coursesEnrolled`). The `trialCourses` field tracks courses users have enrolled in under a trial period, complete with expiration dates and active status.
- **Course**: Tstores details about each course, including the `name`, `price`, `description`, and `image`. It references the instructors (using their `ObjectId`) who teach the course, as well as categories (`Category`) to which the course belongs. Courses can also be marked as featured using the `featured` field.
- **Category**: organizes courses into different categories (e.g., Programming, Science). Each course can belong to multiple categories, allowing learners to browse by category.
- **Fee**: represents the platform's subscription and commission fees. This model is used to store fee structures for both instructors and learners.
- **Member**: store information about group members for the "About Us" section of the platform. It includes fields such as name, bio, role, and photo URL.
- **FAQ**: manages frequently asked questions, storing the question and corresponding answer to display on the FAQ page.

6.3 Data Relationships

One of the key features of the platform is the relationship between users, courses, and categories:

- **User and Courses**: Instructors are linked to their courses through the `courses` field, while learners have their `coursesEnrolled` and `trialCourses`.
- **Courses and Categories**: Each course is associated with one or more categories through references. This makes it easy to browse courses by category or filter them based on user preferences.

In order to link references, I userd the `populate()` method was in various parts of the project . For example, when displaying course details, the instructors and categories are populated for easy access to related data.

6.4 CRUD Operations

6.4.1 Creating and Managing Users During registration, a new user is created and saved in the `User` collection. Passwords are hashed using `bcrypt` to ensure security. The `pre('save')` method on the schema ensures that passwords are always stored securely.

```
app.post('/courses/:id/edit-name', ensureAuthenticated, async (req, res) => {
    const { name } = req.body;
    const course = await Course.findById(req.params.id);
    course.name = name;
    await course.save();
});
```

6.4.2 Course Management Instructors can add, update, or delete courses. Each course is linked to the instructor's `User` document through an array of instructor `ObjectId`. Similarly, categories are added and linked to courses. I also implemented functions for editing course attributes such as name, price, and description, ensuring that only the assigned instructor can modify these fields.

```
app.post('/courses/:id/edit-name', ensureAuthenticated, async (req, res) => {
    const { name } = req.body;
    const course = await Course.findById(req.params.id);
    course.name = name;
    await course.save();
});
```

6.4.3 Data Retrieval and Display Data retrieval involved using `find()` and `populate()` for related models. For example, when displaying a user's profile, I used `populate()` to fetch courses the instructor teaches or the learner has enrolled in:

```
User.findById(req.session.user._id).populate('coursesEnrolled.course');
```

This method ensures that I can achieve references to other collections and display complete data without extra database operations.

6.5 Data Security

6.5.1 Password Encryption As I introduced before, user passwords are encrypted using `bcrypt` with a salt value to ensure they are stored securely. Additionally, the `resetPasswordToken` and `resetPasswordExpires` fields allow for secure password recovery, with token expiration to prevent unauthorized access.

6.5.2 Session Management User sessions are managed using `express-session`. A session is created after successful login, allowing users to navigate the platform securely. Sensitive information, such as session secrets and MongoDB connection details, is managed through environment variables.

6.5.3 Environment Configuration Sensitive information like the MongoDB connection URI, session secret, and file paths for profile pictures are stored in the `.env` file. This approach ensures that confidential data is

kept separate from the codebase and can be easily configured across different environments. The MongoDB Atlas credentials are also stored here, ensuring secure access to the cloud-based database.

6.6 Conclusion

When implementing this features I have learned the importance of the role that data management plays in ensuring the platform scalability, security, and usability. With MongoDB and Mongoose, together with the cloud-based MongoDB Atlas, the project implements efficient data operations while maintaining the integrity and security of sensitive user data. From secure password handling to the use of environment variables, the design prioritizes data protection, ensuring a smooth and secure experience for users.

7 Result

The final platform I coded is a fully functional and responsive online learning platform where instructors can manage their courses and descriptions in order to sell them to learners, who can browse, enroll in (with different subscription levels), and trial various courses. Note that I followed all the indications in the assignment instructions and, as a consequence, this first version of the platform does not include an actual system with the course materials, activities, and forums. As requested, it is rather a platform where users can browse courses, see instructors and enroll in those which they are more interested in. Below is a detailed breakdown of the results achieved, accompanied by screenshots that demonstrate the key features and functionality of the system.

7.1 User Experience

The platform has been designed with responsiveness in mind to ensure that it adapts to a variety of devices, from desktop computers to mobile phones.

On larger devices such as desktops, the layout includes a horizontal navigation menu, a multi-column content display, and full-sized images. The following screenshot shows the Home Page layout on a desktop screen:

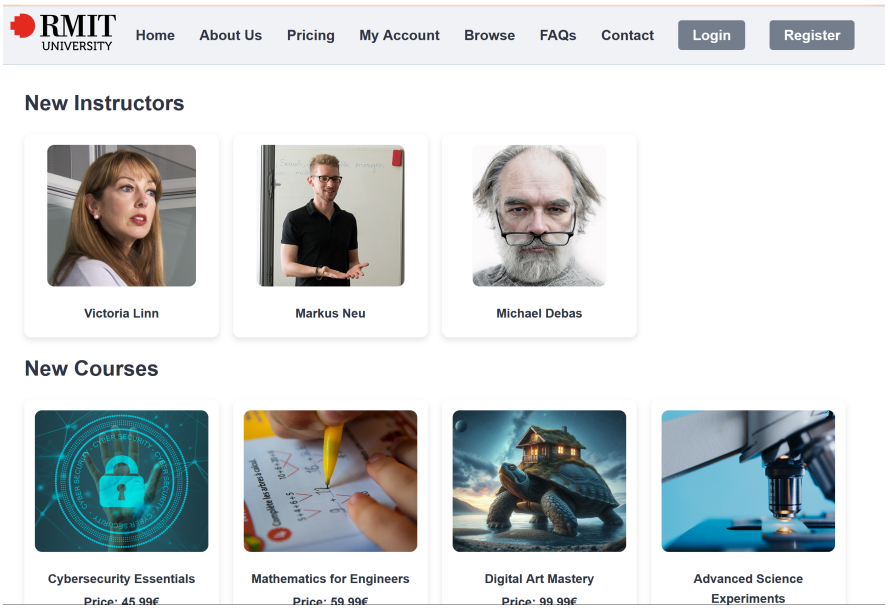

Figure 1: Home Page Layout

When viewed on smaller devices like tablets or smartphones, the layout is adjusted to optimize the available screen space. A hamburger menu is used to simplify the navigation, and content is displayed in a single-column layout:

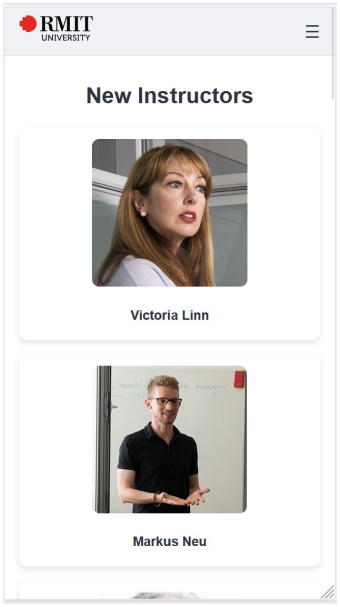

Figure 2: Single Column Layout in Mobile Devices

7.2 User Roles and Access

The platform supports multiple user roles, including Learners, Instructors, Guests, and Admins. The functionality and available features vary based on the user's role.

The admin role can modify any course and has access to the Mongo DB Atlas database from the my-account page. This is a screenshot of an admin user in this page:

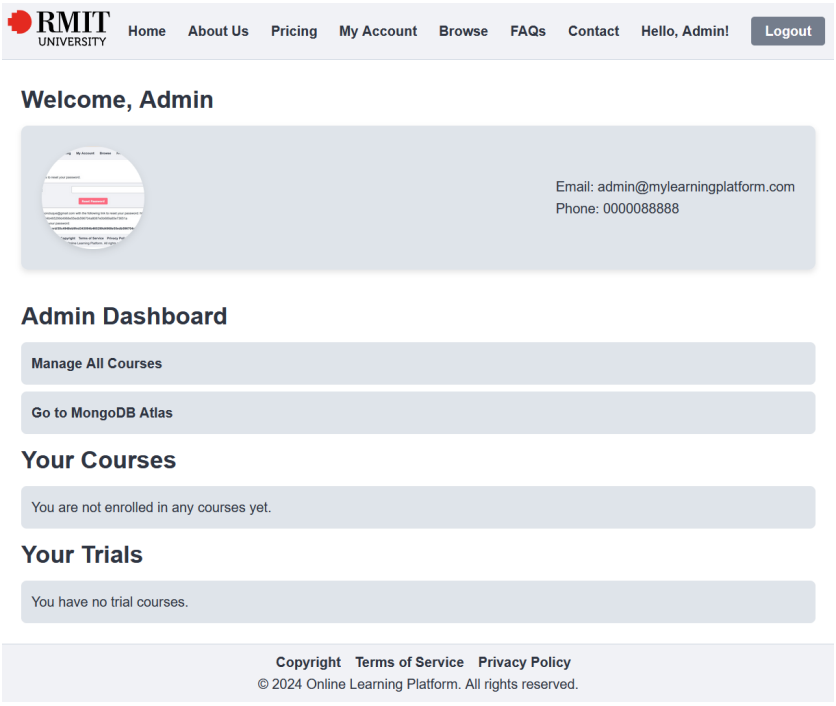

Figure 3: Instructor: My Account View

The credentials to access MongoDB Atlas should be provided to the admin using a different channel to provide

security.

For instructors, the platform provides a dedicated profile page where they can manage their courses. The screenshot below demonstrates how instructors can view and add new courses to their courses:

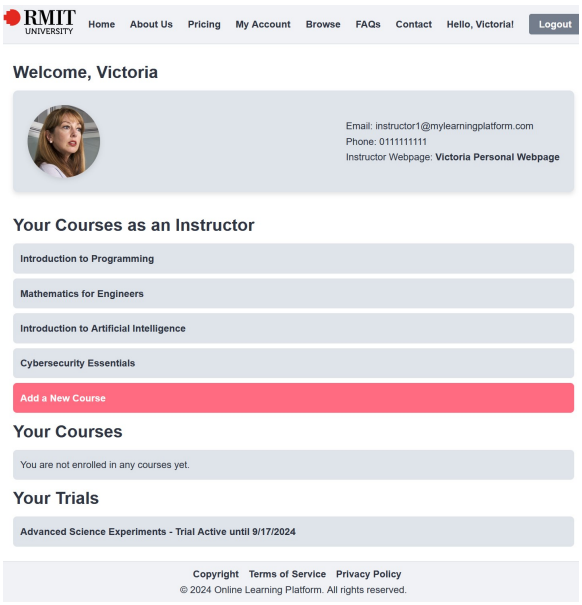

Figure 4: Instructor: My Account View

Learners have access to a profile page where they can view the courses they are enrolled in, as well as any trial courses they have initiated. Below is an example of a Learner's account page:

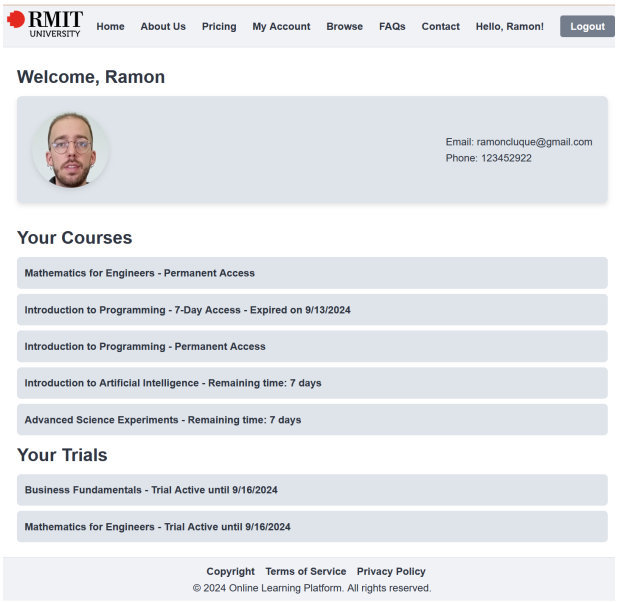

Figure 5: Learner: My Account View

7.3 Course Management

Course management is a key feature for instructors. They can create new courses, edit existing ones, and categorize them for easier browsing. Below is a screenshot showing the course management interface for instructors:

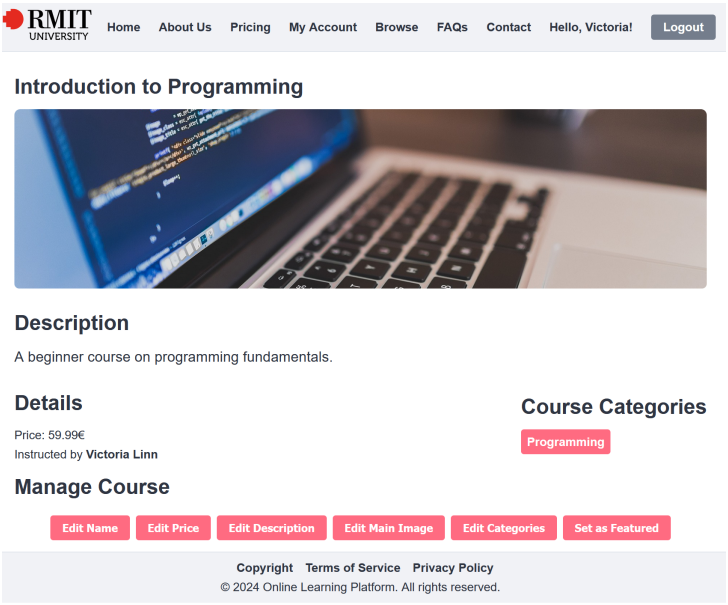

Figure 6: Course Management Interface

Note also that the instructor of the course can edit the parameters of the course, whereas an admin has the ability to modify **any** course.

Learners can browse courses either by name or by category. The following screenshot shows the Browse Courses by Category page:

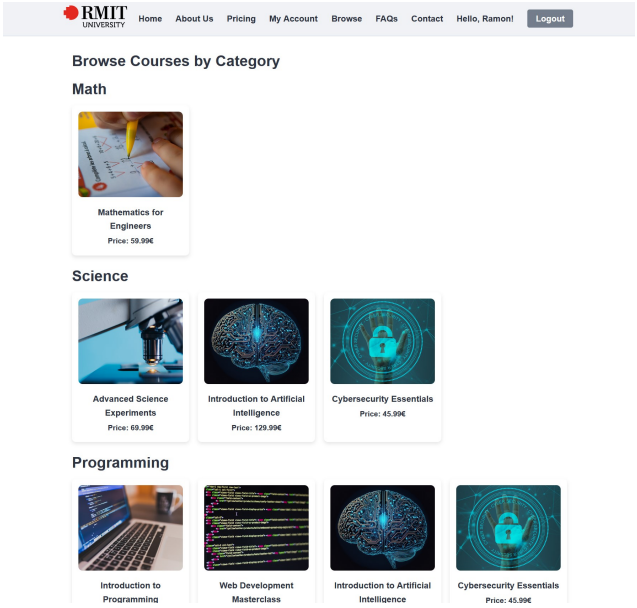

Figure 7: Course Browsing By Category

When a learner selects a course, they are redirected to the Course Detail Page, which provides essential information such as the course name, price, description, and enrollment options. An example of the Course Detail Page is shown below:

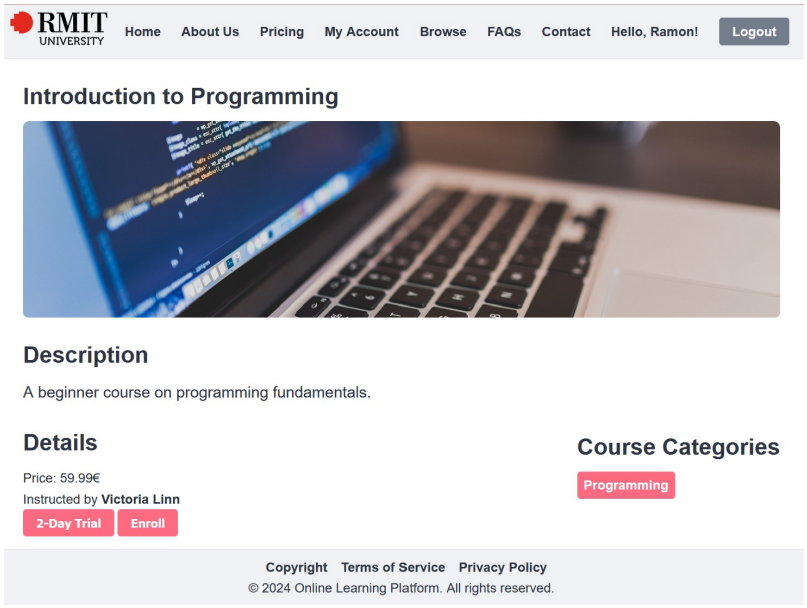

Figure 8: Learner Course Page

7.4. Course Sites and Enrollment System

The **Course Detail** page offers information on each course, tailored to the user's role. As we can observe in the screenshot below, **Price**, **Instructor**, and **Categories** are clearly shown. Based on the user's role (guest, learner, instructor, or admin), different actions and management options are available. A guest would see this:

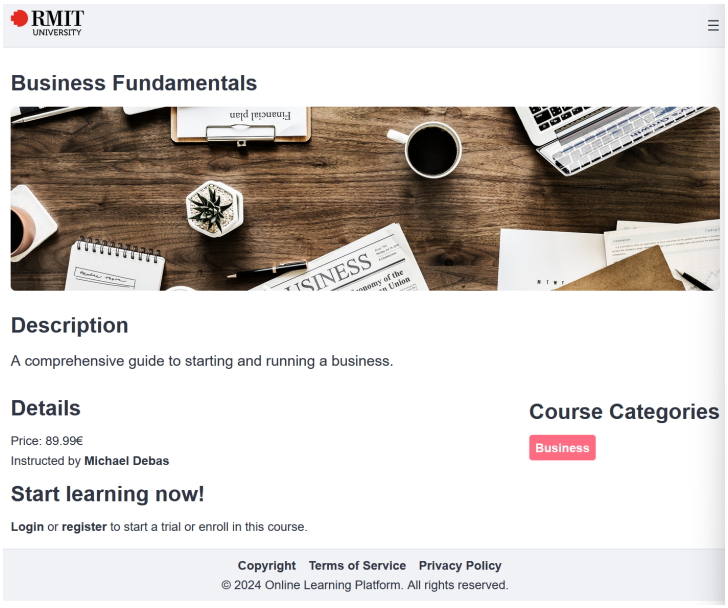

Figure 9: Guest View of a Course Page

Whereas after logging in, a learner has the options to start a trial or enroll into the course:

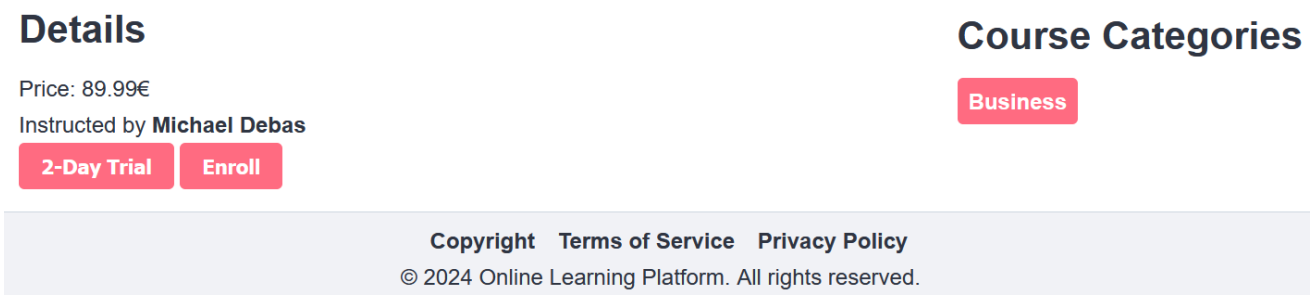

Figure 10: Learner View: Enroll or Trial

This enrollment process is fully developed and covers all cases. For example let's see a user trying to book a trial for a course:

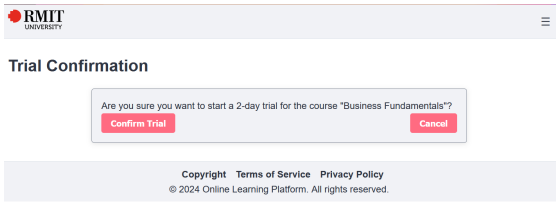

Figure 11: Learner View: Trial Confirmation

If the user has already trialed it, the order will fail:

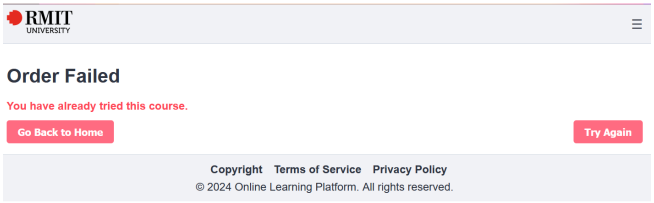

Figure 12: Learner View: Failed Order

7.5 Instructor Sites

The **Instructor Profile** page is where instructors can display their qualifications, specialization, and available courses. As indicated in the assignment instructions, it features a different footer and header, which are personalized according to each instructor. Here we can see an example:

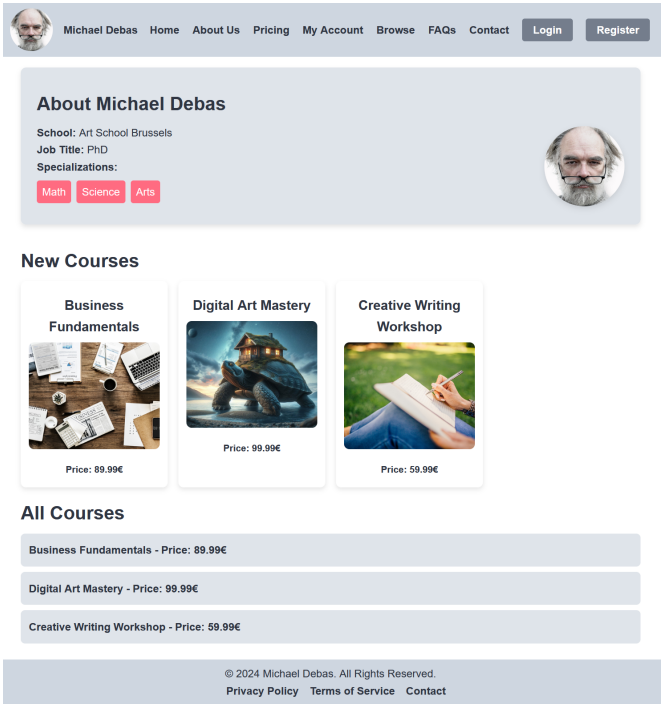

Figure 13: Instructor Profile Page

7.6 Secure Authentication

The platform includes secure authentication features, allowing users to log in, register, and reset their passwords. The password reset process involves generating a unique token for each user, enhancing security.

Here's a screenshot of the **Login Page** where users can log in using their email or phone number:

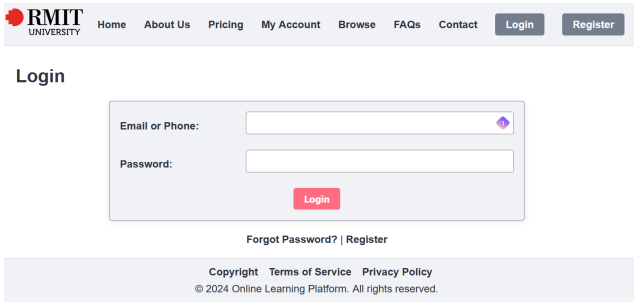

Figure 14: Login Page

Additionally, users who forget their password can initiate the recovery process through the **Password Reset Page**. A reset link is generated and displayed for the user to follow:

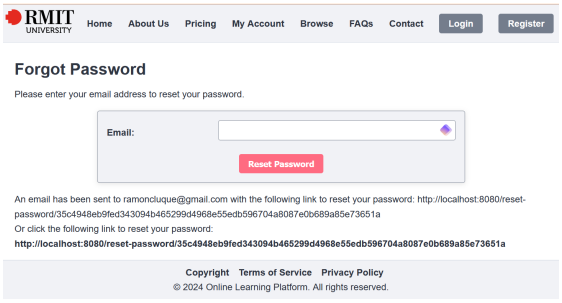

Figure 15: Login Page

7.7 Data Management and Storage The platform uses MongoDB Atlas for data storage, which is hosted on a free tier AWS cluster. Various collections are used to store data, including users, courses, categories, fees, and FAQs. The data is well-structured and allows for efficient querying and manipulation.

While there is no specific UI for the data management system itself, the following screenshots demonstrate how data is displayed to users, such as categories and courses:

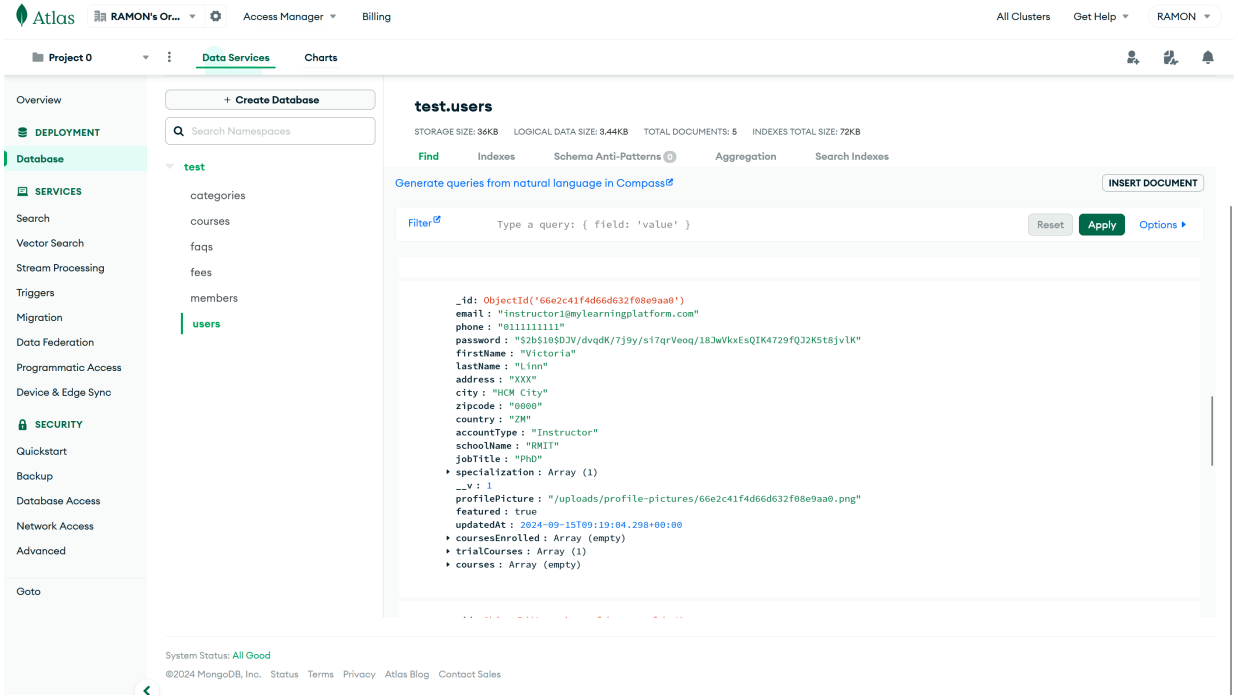

Figure 16: MongoDB Atlas Interface: User Model

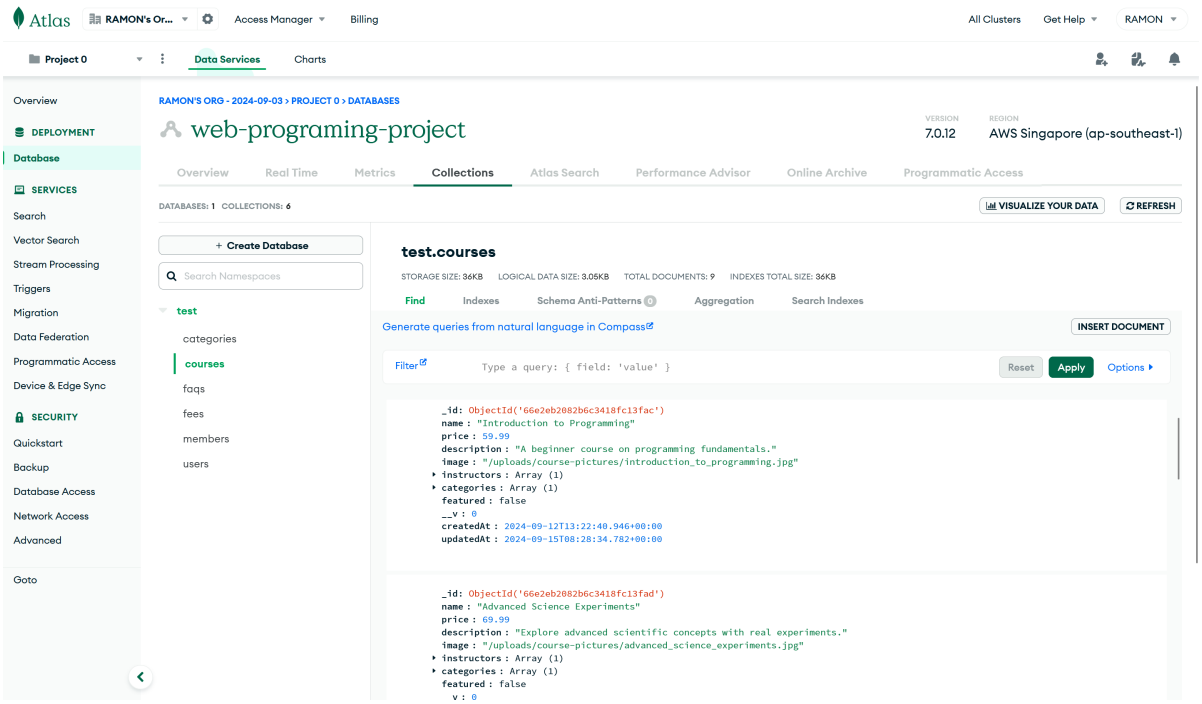

Figure 17: MongoDB Atlas Interface: Course Model

7.8 Generated Content Some of the content used in the project was generated using AI tools and free image sources. Specifically:

- **FAQs**: The FAQs were generated with ChatGPT, providing relevant information for platform users.
- **Pictures of members in the About section**: These were generated with ChatGPT.
- **Course descriptions**: Generated with ChatGPT.
- **Profile pictures of instructors**: Images were sourced from Pixabay.
- **Pictures of courses**: Images used for course banners and descriptions were also sourced from Pixabay.

7.9 Conclusion I believe that the platform achieves the requirements listed in the assignment, which I briefly reviewed at the beginning of the report. The user interface is responsive and user-friendly, adapting succesfully to different device sizes. The multi-role functionality allows for good interaction between instructors and learners. The secure authentication system ensures that user data is well-protected, and the course management features make it easy for instructors to add and update courses. The use of MongoDB Atlas for data management provides scalability and reliability, making the platform a good template for a platform that sells online learning courses. The screenshots provided in this section illustrate some of the main features and functionalities that were developed, ensuring that the platform is ready to be used if needed. The video attached in this assignment is also be useful to see the extent of my work.

8 Final Conclusions

During the development of this online learning platform, I encountered several challenges that required careful consideration and problem-solving. These challenges provided opportunities for learning and improving both my technical skills and understanding of full-stack web development. This section covers the main challenges faced during the project and the lessons learned.

Working Independently

One of the most significant challenges was transitioning from group work to independent work. Initially, I worked with a classmate, but we agreed to part ways after realizing we had different approaches to the project. While this gave me more freedom to implement features the way I wanted, it also meant I had to take full responsibility for

all aspects of the development process, from front-end design to back-end implementation. This situation helped me develop a deeper understanding of full-stack development, as I had to manage every layer of the application.

Secure Authentication

Another key challenge was ensuring secure authentication and session management. I had to make sure user data, especially passwords, were properly encrypted. Learning to use bcrypt for hashing passwords and managing password recovery tokens using crypto was easier than I expected. These methods provided users with secure password storage and recovery processes without exposing sensitive data.

Managing Data with MongoDB and Mongoose

Using MongoDB Atlas as the database solution was new for me. I initially faced difficulties in connecting to the cloud database and properly configuring environment variables. Additionally, designing the database schema with references between models (such as Users, Courses, and Categories) required careful planning to ensure efficient queries and data retrieval. I had to change these models many times during the project.

Working with Mongoose also presented challenges, particularly when it came to managing relationships between collections. I had to familiarize myself with methods like `populate()` to retrieve related data in an efficient manner. This was especially important for displaying instructors' courses and learners' enrolled courses in the profile pages.

Responsive Design

Building a responsive design that works well across desktop, tablet, and mobile devices required significant testing and trial and error. I used media queries and Flexbox and CSS Grid to ensure that the interface was adaptable to different screen sizes. The most challenging part was adjusting navigation elements and forms to maintain usability on smaller devices while keeping the interface clean and intuitive.

Dynamic User Interfaces

Another challenge was implementing dynamic content rendering based on user roles. The platform supports multiple user roles (Admin, Instructor, Learner, Guest), each with different permissions and views. Ensuring that the right information was displayed to the correct user type involved careful use of conditionals in the EJS templates, which use a syntax I had not seen before. Ensuring that each user sees only what they are authorized to manage requires thorough testing and well-organized code.

Final Conclusion

In conclusion, the challenges faced during this project were opportunities for learning a lot about web programming. I think I have enhanced my skills in full-stack development, particularly in areas like authentication, data management, file handling, and responsive design.