UOMSRS

STUDENT REQUEST MANAGEMENT SYSTEM

TEAM GTX

Oshanath Rajawasam	190488J
Lasith Rajakarunaratne	190483N
Samadhi Kariyawasam	190159D
Kalana Rubasinghe	190530H
Akash Tharuka	190623V

Content

AVAILABLE PRODUCTS AND THEIR STRENGTHS AND LIMITATIONS

- The front end of the system was designed using HTML, CSS and Javascript.
- Dynamic content manipulation of the front end was done using EJS framework. (Embedded Javascript).
- The back end was implemented using NodeJS and request handling was done with the Express framework, and both used the language JavaScript.
- Data and information transfers between the front end and the back end was done with Fetch API.
- JavaScript was chosen as the back-end development framework rather than PHP, because JavaScript is heavily used in the present and it supports many frameworks that make certain tasks easier, more efficient, and more structured.
- We chose the NoSQL based MongoDB and the Mongoose framework as the DBMS.
- NoSQL based MongoDB is more efficient than SQL based DBMS, which is more suitable for relational data such as transaction systems. Since our system is only slightly relational, NoSQL is the logical choice.
- Source Control – Since all 5 of our team members worked on the project individually from our homes, we used Git and GitHub as a source control platform.
- All our team members used the IDE 'Visual Studio Code' due to its low memory requirements and very good language support.

RESOURCES USED

- **Front End** – CSS, HTML, Javascript, EJS
- **Back End** – NodeJS, ExpressJS
- **DBMS** – MongoDB (NoSQL), Mongoose
- **Source Control** – Git, Github
- **IDE** – Visual Studio Code

DESIGN AND IMPLEMENTATION

FRONTEND

The frontend was developed using EJS, CSS and Vanilla JavaScript. EJS was used to build the layout of the web pages and directly include JavaScript into them. CSS was used to improve the appearance by applying styles and vanilla JavaScript was used to configure the functionality of the web page components.

When applying styles to the pages, we used separate CSS files for each Ejs file. That way we were able to modify any web page without affecting others. Furthermore, for every popup form in the application, we created a single CSS file, which allowed us to maintain a special theme for all of them.

When developing the frontend, browser developing tools, Chrome Inspector in particular, were very useful, especially for styling and error detecting.

Many of the CSS features such as flex boxes, gradients, animations were used to garnish the web pages. In addition to that, frontend and backend validation results were displayed using CSS. This not only decorates the application but also notifies the user to use the accurate inputs.

We needed to maintain a particular format for certain fields and to do that, the Regular Expressions were needed. Instead of studying it from A to Z, to manage our timeline, we used pre-created standard Regular Expressions, provided by the Stack Overflow web site for some fields such as email, password, and phone number.

We used the below mentioned external resources to insert some icons to the application.

https://fonts.googleapis.com/css2?family=Ubuntu:wght@300&display=swap

https://cdnjs.cloudflare.com/ajax/libs/font-awesome/5.13.0/css/all.min.css

https://maxcdn.bootstrapcdn.com/font-awesome/4.4.0/css/font-awesome.min.css

Since this is our first web developing experience, we had to learn it from scratch. In that respect, we mainly referred to “The Net Ninja” YouTube channel. It is a well-structured web development guide, which was extremely helpful for beginners like us to learn the basics. The link to the channel is provided below.

https://www.youtube.com/channel/UCW5YeuERMmlnqo4oq8vwUpg

FORM CONTROLLER

We used a single JavaScript file, FormController.js, to handle all user input in the front end. All the input fields in the app, in all windows, were divided into several categories according to their verification process. Some of these are non-empty, new password etc. All the non-empty type input fields in the app should only be checked if some input exists, and new password type input fields should contain a password text that agrees to the app’s criteria.

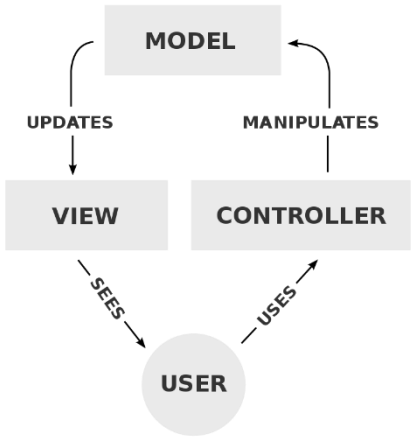
- Nonempty – Some input should be present. They will not be validated.
- Normal – No validation occurs upon these input fields.
- Selected – One of the system-generated inputs should be selected. Custom user input is not valid. (When we are selecting a staff member to send a request)
- Existing password – Should match the password in the account of the specified email.
- Existing email –Input email should already exist in the database.
- New email – Input email should agree to the criteria, must be unique, and must be a usable, accessible email address.
- Index – Should be unique.
- Nonempty radio – At least one radio button from a group should be selected.
- Contact email – Should agree to the criteria.

After all the data has been submitted, if all the validations yield correct results, the finalize() method is called to send all the details to the required locations.

If an input is valid or invalid, we let the user know by displaying an icon and/or a message.

BACKEND

We used the Model - View – Controller architecture to develop the application, which is the most used structure when it comes to web development. It is based on modularizing the whole internal implementation into three main components. The way information is presented to and accepted from the user, can be divided according to the internal representation, and this helps to manipulate the data in the application quite easily.



MODELS

This is the main component and the dynamic data structure of the application. It is independent from the interface and it directly manages the logic and data of the application

- User

  The structure which hold the data and operation belongs to the user object itself.

- Thread

  The structure which holds the data belonging to a conversation between two users on a particular request.

- Message

  The structure which holds the data belonging to a single message of a thread.

VIEW

Any interface or representation which the user uses to input, access or modify data. Although HTML is usually used to design web pages, we used EJS (Embedded JavaScript) view engine to gain below mentioned advantages.

- EJS allows us to use plain JavaScript directly in web pages, which makes manipulating data much easier.
- Unlike plain html, EJS shows errors in a format, which helps us to understand and handle them efficiently.
- EJS caches intermediate JavaScript functions and allows fast executions.

CONTROLLERS

Requests or inputs are accepted by the controllers and converts them to commands for a model or a view to represent or store. When handling requests, async functions were regularly used. Although handling async functions were quite difficuly, they were necessary in order to achieve the desired correct functionality.

- authController

  AuthController handles the requests regarding all the main pages such as login, signup and user profile.

- formController

This handles the requests regarding email and password validation.

- settingsController

  Settings controller handles the requests regarding app settings which are edit profile, change password and delete account.

- threadController

  Thread controller controls the requests regarding threads.

- verificationController

  This controller is used to handle email verification requests.

**Each of these controllers are interconnected with a routes file of its own, which directs frontend request to the backend.

MIDDLEWARE

Archiver – When users download documents that are embedded in a message of a thread, since multiple files can be uploaded, all those files will be archived and be sent to be downloaded.

- Bcrypt

  Encrypts the cookies and salts and hashes the passwords.

- cookieParser

  When a cookie is passed along with a request, this framework decrypts the cookie to authorize the user.

- Multer

  Middleware for file uploads. Takes the files from a form submission and saves them in the server.

- Uuid

  Middleware to generate unique id addresses for files.

FRAMEWORKS

- Express

  This is a NodeJS framework to handle requests easily.

- Mongoose

  This framework connects the NodeJS server to the MongoDB server.

- Node mailer

  A framework used to send mails from NodeJS.

- Nodemon

  A framework used to quickly prototype and test web apps. This restarts the NodeJS server as soon as we make a change in the code and save it.

DATABASE

We used mongo DB, which is a NoSQL database, as the data storage unit of our application. The choice was made due to the below mentioned advantages that made developing the application much easier.

- Since we used JavaScript and Node.js to develop our application, it was more consistent to use a NoSQL database that stores data in BSON format which is short for Binary JavaScript Object Notation. We chose the most widely used and the latest NoSQL database, mongo DB, for our application. It is also well documented.
- NoSQL databases have slightly better performance than SQL, particularly mongo DB. Hence it helped to increase the efficiency of the application
- Querying is easier and faster in MongoDB.

- Mongo DB has a shallow learning curve, which makes it easier to learn the beginner level uses of the database.

SECURITY

We gave some thought about the security of the app and we decided we don't need state of the art security such as encryption of requests because this is a university student request app, and we don't expect forced attacks to come through.

But all the passwords are salted and hashed before they are stored in the database. So that even if the database is exploited, the passwords cannot be decrypted.

Cookies are used for authorization. Once the user logs in, a cookie is sent to the browser that will be expired in 24 hours. When the user logs out, the cookie is destroyed. If the cookie is forcefully changed or the cookie is missing, then the app will be redirected to the login page. It is also "browser-back-button-protected", which means that pressing the "back" button on the browser does not take you back to pages containing sensitive data, because the browser will not save the pages in the cache.

The user's email is the central security point of a user's account. A validation of the user email is necessary for him to log in to the system. And the 'forgot your password?' section of the app also works with the user's email.

CHALLENGES WE FACED AND HOW WE OVERCAME THEM

- Since the project was done from our homes, collaboration was tricky.

  To combine the work done by each member Git and GitHub was used. And Zoom was used for online discussions.

  Separate Git branches were used for specific purposes and was merged back to the Main branch after its development.
- Web development was a topic most of us have never come across before. Domain knowledge in designing DBMS, Web design and network protocols had to be gained in a short time . We followed YouTube tutorials and referred websites to overcome this shortcoming.
- With the heavy academic schedule in the semester it was hard to find time for the project. We had to cut our sleep short and work late for weeks.

CONTRIBUTIONS MADE BY TEAM MEMBERS

- Oshanath Rajawasam – Back-end development
- Kalana Rubasinghe – Back-end development
- Akash Tharuka – Front-end dynamic content creation, CSS and animations
- Lasith Rajakarunaratne – Front-end development (CSS stylist)
- Samadhi Kariyawasam – Front-end development (Static content creation)