**Web Application Project Report**

Table of Contents

[1. Introduction 1](#_Toc171461194)

[Objectives and Goals 1](#_Toc171461195)

[Scope of the Project 1](#_Toc171461196)

[Significance of the Project 2](#_Toc171461197)

[2. Research and Planning 2](#_Toc171461198)

[Research 2](#_Toc171461199)

[Project Plan 2](#_Toc171461200)

[Design Patterns 3](#_Toc171461201)

[3. Choice of Framework and Technologies 3](#_Toc171461202)

[Front-end Framework 3](#_Toc171461203)

[Back-end Technologies 4](#_Toc171461204)

[4. Consumption of Web Services 4](#_Toc171461205)

[Implementation 5](#_Toc171461206)

[User Authentication 6](#_Toc171461207)

[Frontend Integration with Axios 6](#_Toc171461208)

[Handling API Responses 7](#_Toc171461209)

[5. Security Threats and Measures 7](#_Toc171461210)

[Security Threats 7](#_Toc171461211)

[Security Measures 7](#_Toc171461212)

[6. Features 8](#_Toc171461213)

[User Authentication 8](#_Toc171461214)

[Note Management 8](#_Toc171461215)

[Responsive Design 9](#_Toc171461216)

[Real-time Updates 9](#_Toc171461217)

[7. Conclusion 9](#_Toc171461218)

[8. References 10](#_Toc171461219)

# 1. Introduction

As the world transits to information age, good organization and management of notes is significant particularly in the students’ and working class people. This project covers creating of Notes web application based on the MERN stack which includes MongoDB, Express. js, React. js as well as Node. js, a package of technologies aimed at creating solid web applications.

Therefore, the main goal of this project at this stage is to create an application that would be easy to use, safe, and respond to the inputs of a user in the context of note taking adequately. Therefore, employing the opportunities of the MERN stack the application should enable easy and fast access to the notes irrespective of the utilized equipment.

## Objectives and Goals

The main goals of the Notes web application are as follows:

* **User-Friendly Interface**: Develop an intuitive and easy-to-navigate user interface that allows users to create, edit, view, and delete notes effortlessly.
* **Security**: Implement robust security measures, including user authentication, password hashing, and protection against common web vulnerabilities, to ensure the safety and privacy of user data.
* **Responsiveness**: Ensure the application is fully responsive, providing an optimal user experience on desktops, tablets, and mobile devices.
* **Performance**: Optimize the application for performance, ensuring fast load times and smooth interactions.
* **Scalability**: Build a scalable application capable of handling a growing number of users and data without compromising performance.

## Scope of the Project

As will be seen in the following sections, this project entails creation of front-end and back-end parts of the application. The key components and features of the application are:The key components and features of the application are:

* **Front-End Development:** Using React same as it is used in JavaScript frameworks. js, the front-end will offer interactiveness to the users and give interesting visuals. Since, the note is primarily used as means of memo-writing, simple components as forms, lists, and navigation bars will be created to ensure memo organization.
* **Back-End Development:** The back-end, created with Node. js and Express. That is it , js, will serve purposes of calling API’s, business logics and the databases. These are user authentication, CRUD operations of notes, and data validation for the notes.
* **Database Management**: MongoDB will be used to store user data and notes. Mongoose will be utilized for schema definition and data modeling.
* **Security Implementation**: Security features, including JWT-based authentication, bcrypt for password hashing, and input validation, will be integrated to protect user data.
* **Responsive Design**: The application will be designed to be fully responsive, ensuring usability across different devices and screen sizes.

## Significance of the Project

There is stress within today’s society and people & professionals especially are in need of sturdy tools to help them keep track of concepts, duties, and data. By having a suitable notes application, one is able to improve productivity and organization in that there is a central place where all notes are made and managed. This project focus on developing an efficient application, and by using the MERN stack, it can also demonstrate the combining and the possibility of the contemporary web technologies. MERN is famous for using JavaScript on both the client and server sides, which improves the compatibility of the two and makes development easier.

# 2. Research and Planning

## Research

The preliminary stage of the web development project entailed the immersion of the researcher in current trends and practices in web development. The reason behind selecting MERN stack is that they are one of the popular as well as efficient choices for the development of full stack applications. For data management MongoDB was chosen for its high flexibility and integration as the No SQL database. Node.js, coupled with Express.js was chosen because it is asynchronous and based on events which make it easier to create network applications at scale. React.js was chosen based on how it uses components and has optimal rasterization.

Information regarding the usage of the technologies was gathered from various sources like official documents, online tutorial, and forums to study the effectiveness and probable issues that may be faced. Other studies also concerned different types of authentication, data encryption, and security to make an application more secure.

## Project Plan

A detailed project plan was created to guide the development process. The plan included timelines, milestones, and deliverables, ensuring that all aspects of the project were covered systematically. Tools like Gantt charts and Trello boards were used to manage tasks and track progress.

**Project Milestones:**

1. Initial Research and Technology Selection
2. Backend Development (Models, Routes, Controllers)
3. Frontend Development (Components, State Management)
4. Integration and Testing
5. Security Implementation (Authentication, Password Hashing)
6. Deployment and Final Testing
7. Documentation and Presentation Preparation

## Design Patterns

Concerning the structural aspects of the solution, the Model-View-Controller (MVC) pattern was implemented for the application as it would allow for better division of concerns and code maintainability. The MVC pattern divides the application into three interconnected components:The MVC pattern divides the application into three interconnected components:

* **Models:** Model the data, and business rules of the frame. In this case, the models are the user model and the note model, which contain the schema and the methods for relating to MongoDB.
* **Views:** Define the outermost layer of the application and show the data in a convenient for usage form to the user. In this project, views are implemented using the React components that incorporate the presentation and control logic of how the user interacts with the application.
* **Controllers:** Contain all the application’s business logic and control interactions between models and views. When receiving the incoming request, the controllers take care of required manipulations (for example CRUD) and send an adequate response.

# 3. Choice of Framework and Technologies

## Front-end Framework

React.js was chosen as the front-end framework due to its component-based architecture, which promotes reusability and ease of maintenance. React's virtual DOM enhances performance, making it an ideal choice for dynamic and responsive user interfaces.

**Dependencies (as per frontend/package.json)**:

* **React**: For building UI components.
* **React-DOM**: For DOM-specific methods.
* **Axios**: For making HTTP requests to the backend.
* **React Router**: For navigation and routing.

**React Components**:

* **App.js**: The main component rendering the application.
* **NavBar.js**: Component for navigation.
* **NotesList.js**: Component displaying a list of notes.
* **Note.js**: Component for individual note details.
* **Login.js**: Component for user login.
* **Register.js**: Component for user registration.

## Back-end Technologies

Node.js and Express.js were selected for the back-end. Node.js allows for building scalable network applications, while Express.js provides a robust set of features for web and mobile applications.

**Dependencies (as per backend/package.json)**:

* **Express**: For building the API and handling routes.
* **MongoDB**: NoSQL database for storing notes and user information.
* **Mongoose**: For MongoDB object modeling.
* **Bcrypt.js**: For password hashing.
* **JWT (jsonwebtoken)**: For authentication and authorization.

**Express Routes**:

* **/api/notes**: For handling note-related requests.
* **/api/users**: For handling user-related requests (registration, login).

**Models**:

* **User.js**: Defines the user schema and methods for password hashing.
* **Note.js**: Defines the note schema and methods for CRUD operations.

# 4. Consumption of Web Services

The Notes web application is nearly completely based on web services as tools for effective client (frontend)-server (backend) communication. This approach enables users to perform the CRUD operations for the notes and at the same time handle the user authentication and registration. The data is transfer between the application using JSON (JavaScript Object Notation), which is light weighted and easily usable with JavaScript. Described in this part is the usage of web service consumption in the application.

## Implementation

The following RESTful API endpoints were implemented to manage notes and user authentication:

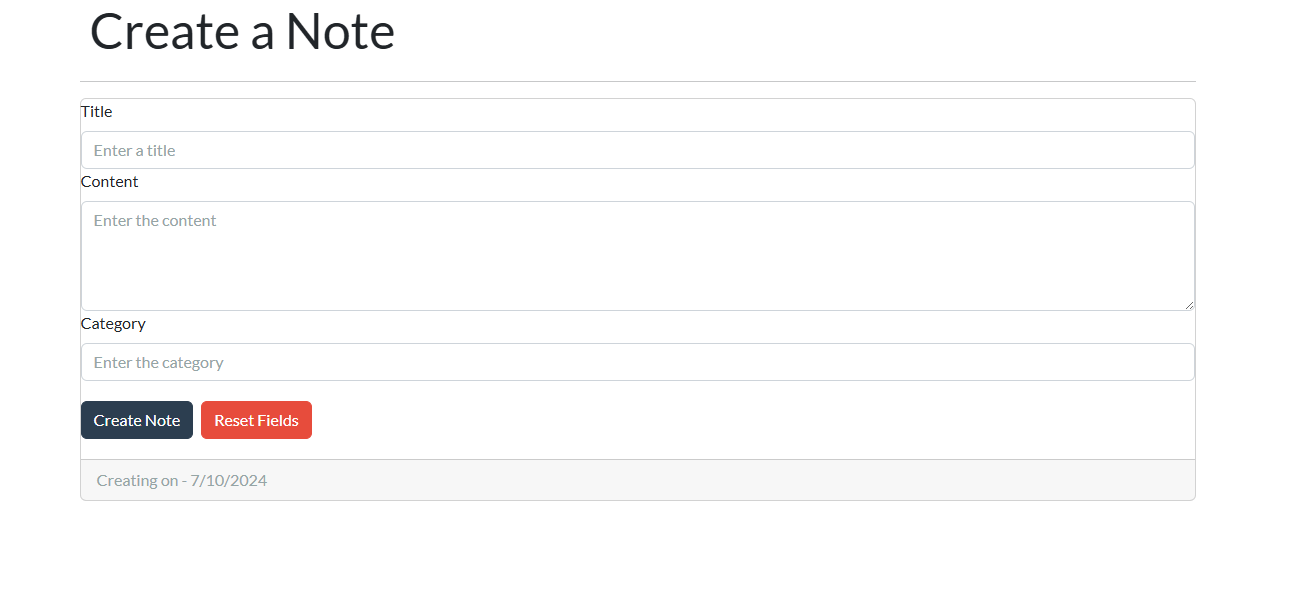


**Notes Management**

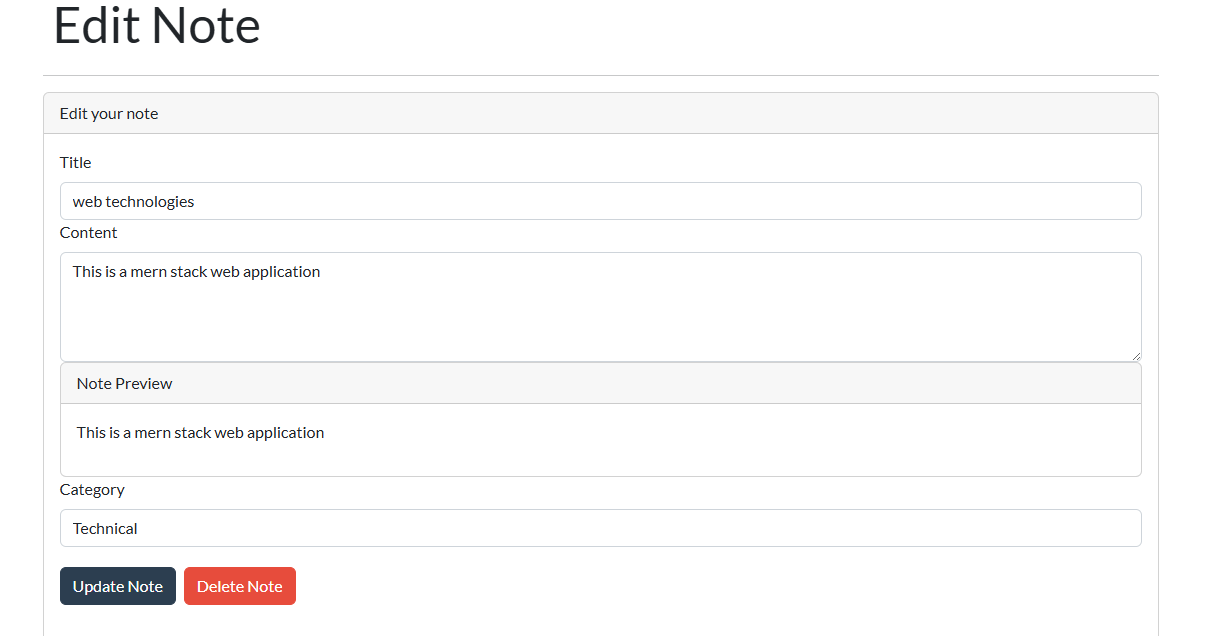
1. **Fetch All Notes**
   * **Endpoint**: GET /api/notes
   * **Description**: Retrieves a list of all notes belonging to the authenticated user.
   * **Implementation**: The backend receives the GET request, verifies the user via JWT, and retrieves the notes from the MongoDB database. The notes are then sent back to the client in JSON format.
   * **Usage**: This endpoint is called when the user accesses the notes dashboard to display all their saved notes.



1. **Create a New Note**
   * **Endpoint**: POST /api/notes
   * **Description**: Creates a new note and saves it to the database.
   * **Implementation**: The client sends a POST request with the note details in the request body. The backend validates the data, saves the new note to MongoDB, and responds with the created note's details.
   * **Usage**: This endpoint is used when the user submits a new note through the user interface.



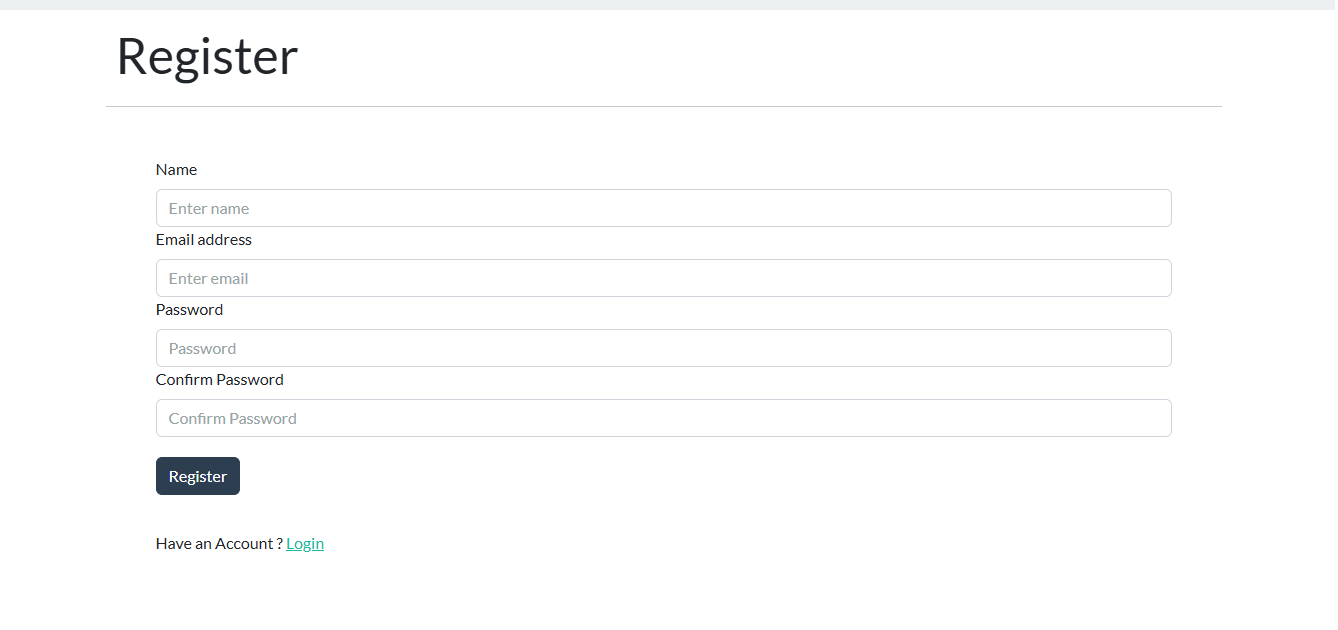
1. **Update an Existing Note**
   * **Endpoint**: PUT /api/notes/:id
   * **Description**: Updates the content of an existing note identified by its ID.
   * **Implementation**: The client sends a PUT request with the updated note data. The backend finds the note by its ID, updates it with the new data, and saves the changes to MongoDB. The updated note is then returned in the response.
   * **Usage**: This endpoint is used when the user edits an existing note and saves the changes.



1. **Delete a Note**
   * **Endpoint**: DELETE /api/notes/:id
   * **Description**: Deletes a note identified by its ID from the database.
   * **Implementation**: The client sends a DELETE request with the note ID. The backend deletes the note from MongoDB and sends a confirmation response.
   * **Usage**: This endpoint is called when the user chooses to delete a specific note.

## User Authentication

1. **Register a New User**
   * **Endpoint**: POST /api/users/register
   * **Description**: Registers a new user by creating a user record in the database.
   * **Implementation**: The client sends a POST request with the user's registration details (e.g., username, password). The backend hashes the password using bcrypt, saves the user record in MongoDB, and responds with the new user's details (excluding the password).
   * **Usage**: This endpoint is used when a new user completes the registration form.

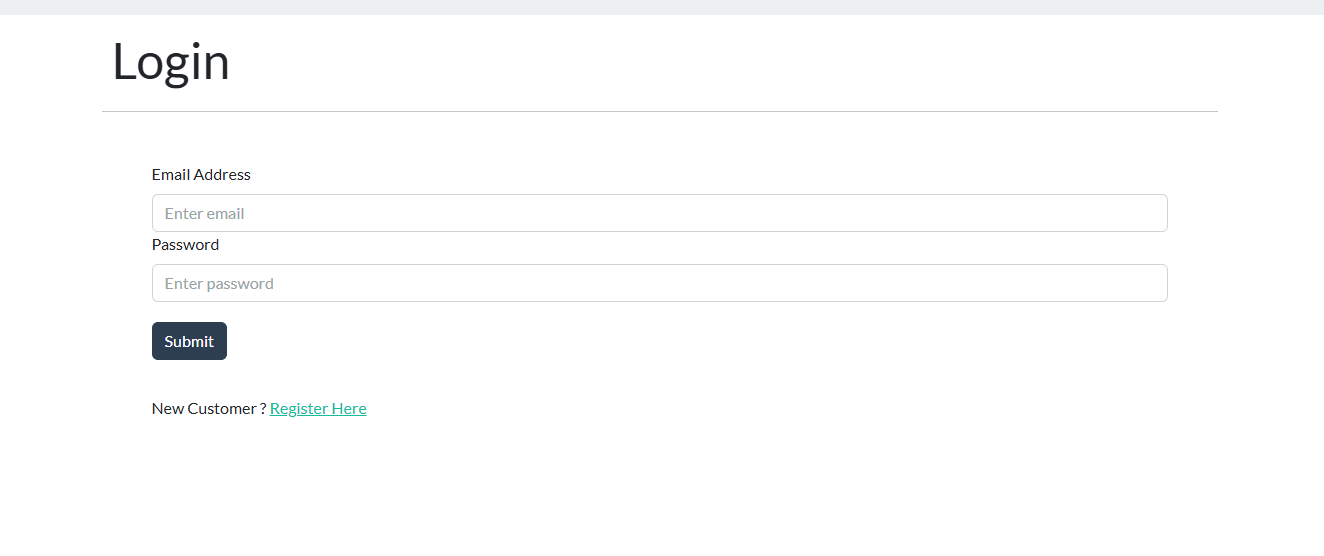


1. **Authenticate a User and Provide a JWT Token**
   * **Endpoint**: POST /api/users/login
   * **Description**: Authenticates the user by verifying their credentials and returns a JWT token for session management.
   * **Implementation**: The client sends a POST request with the user's login credentials. The backend verifies the credentials, and if valid, generates a JWT token and sends it back in the response.
   * **Usage**: This endpoint is used when an existing user logs in to the application. The JWT token is stored on the client side and used for authenticated requests.

## Frontend Integration with Axios

Axios, a popular HTTP client for making requests, is used on the frontend to interact with the backend APIs. Here is how Axios is integrated into the React application for handling the various API requests:

1. Fetching Notes
2. Creating a New Note
3. Updating an Existing Note
4. Deleting a Note
5. User Registration
6. User Login



## Handling API Responses

The responses from the API are processed appropriately to update the application state and provide feedback to the user. For example:

* **Fetching Notes**: The fetched notes are stored in the application state and displayed in the notes list.
* **Creating a Note**: The newly created note is added to the state and displayed in the notes list.
* **Updating a Note**: The updated note is replaced in the state and the UI is refreshed to show the changes.
* **Deleting a Note**: The deleted note is removed from the state and the UI is updated to reflect the deletion.
* **User Registration**: A success message is displayed upon successful registration.
* **User Login**: The JWT token is stored for future authenticated requests, and the user is redirected to the notes dashboard.

# 5. Security Threats and Measures

## Security Threats

The primary security threats considered for this project include:

* **Cross-Site Scripting (XSS)**
* **Cross-Site Request Forgery (CSRF)**
* **SQL Injection**
* **Unauthorized Access**

## Security Measures

To counter these threats, several security measures were implemented:

* **Input Validation**: All user inputs are validated to prevent XSS and SQL Injection attacks.
* **Password Hashing**: Passwords are hashed using Bcrypt.js before being stored in the database, ensuring that even if the database is compromised, passwords remain secure.
* **JWT Authentication**: JSON Web Tokens (JWT) are used for secure user authentication. Tokens are stored securely and validated on each request to ensure authorized access.
* **CSRF Protection**: CSRF tokens are used to protect against cross-site request forgery attacks.
* **Encryption**: Sensitive data is encrypted using industry-standard algorithms.

# 6. Features

The Notes web application is designed with a range of features to enhance user experience and ensure efficient management of notes. These features are implemented with a focus on security, usability, and responsiveness.

## User Authentication

1. **Secure Login and Registration**
   * Such users can be able to register, and login into the site with their personal details provided. During the registration process, basic and rather useful data such as username, e-mail, and password are gathered.
   * **Implementation:** The Express class is being used for the registration and the login the respective EPs. JS and the data is to be stored in MongoDB restrictively.
2. **Password Hashing**
   * User passwords are hashed before being stored in the database, enhancing security.
   * **Implementation**: Bcrypt.js is used to hash passwords before saving them in MongoDB.

## Note Management

1. **Create Notes**
   * Users have the ability to develop new notes that contain title and the main text.
   * **Implementation:** A link is given on the frontend to create a note and this link leads to the form below. The data provided is then again sent as a POST request to the backend where it is stored in MongoDB.
2. **Read Notes**
   * Users can view all their notes in a list format.
   * **Implementation**: The notes are fetched from the backend using a GET request and displayed on the frontend.
3. **Update Notes**
   * Users are allowed to make changes on notes that are already created to add new information.
   * **Implementation:** Every note separately has an ‘edit’ button through which a user is able to edit the title of the note as well as its content. The changes that are made in this component are made using PUT request which is made to the backend and the updated note is then stored in the MongoDB.
4. **Delete Notes**

* Notes that were created by the users and are no longer wanted can be deleted.
* **Implementation:** Every note contains a delete symbol whereby the users can delete the note. The current note is deleted by sending a DELETE request to the backend and removing it from MongoDB. To actually delete the note from the Redux state is updated to match the note list that was given.

## Responsive Design

* The website’s application is compatible with every device, namely computers, tablets, and handheld phones.
* **Implementation**: The frontend is built using React.js and styled-components, allowing for dynamic and responsive layouts.

**Search Functionality**

* Users can search for notes using keywords, making it easy to find specific notes quickly.
* **Implementation**: A search bar is provided on the frontend, allowing users to input keywords. The search functionality filters the notes displayed based on the keywords, providing a quick and efficient way to locate specific notes.

## Real-time Updates

* Changes to notes are reflected in real-time without the need for page reloads, providing a seamless user experience.
* **Implementation**: The application uses WebSockets for real-time communication between the client and server.

# 7. Conclusion

This project clearly depicts how modern web development technologies and company’s best practices are being adopted in developing this web application. With an emphasis on extensive amount of research, as well as the planning of the application, solid and effective security measures were incorporated to tie together a product that was not only secure, but also intuitive to the user. The choice of React.js for the front end and Node.js/Express.js for the back-end impacted the development cycle and created a quality code to be delivered.

# 8. References

auth0.com (2019). *JWT.IO - JSON Web Tokens Introduction*. [online] Jwt.io. Available at: https://jwt.io/introduction/.

Caio, R. and Pereira (2016). *Building APIs with Node.js*.

GeeksforGeeks. (n.d.). *Express.js*. [online] Available at: https://www.geeksforgeeks.org/express-js/.

https://github.com/mongodb/docs-bi-connector/blob/DOCSP-3279/source/index.txt. (n.d.). *Introduction to MongoDB — MongoDB Manual*. [online] Available at: https://www.mongodb.com/docs/manual/introduction/.

Mehra, M., Kumar, M., Maurya, A., Sharma, C. and Shanu (2021). MERN Stack Web Development. *Annals of the Romanian Society for Cell Biology*, [online] 25(6), pp.11756–11761. Available at: http://annalsofrscb.ro/index.php/journal/article/view/7719/5721.

www.w3schools.com. (n.d.). *React Tutorial*. [online] Available at: https://www.w3schools.com/react/.