A

Project Report

On

Notes Manager using Node.js

SUBMITTED BY -

Rahul Shivaji Vikhe
Roll No.: 65

UNDER THE GUIDANANCE

Dr. Shubhangi Potdar

SAVITRIBAI PHULE PUNE UNIVERSITY
MASTER IN COMPUTER APPLICATION

Dr. Vithalrao Vikhe Patil Foundation's
INSTITUTE OF BUSINESS MANAGEMENT AND RURAL DEVELOPMENT (IBMRD)
AHMEDNAGAR

For the year
2023-2024

DR. VITHALRAO VIKHE PATIL FOUNDATION'S

Institute of Business Manage & Rural Development

DATE: / /2024

Certificate

THIS IS TO CERTIFY THAT,

MR. / MS. RAHUL SHIVAJI VIKHE _____,

IS A STUDENT OF THIS INSTITUTE. HE / SHE HAS SUCESSFULLY COMPLETED THE PROJECT ON " Notes Manager using Node.js " _____ AS PARTIAL FULFILLMENT OF THE COURSE **'MASTER OF COMPUTER APPLICATION'** (MCA), SEM I, AFFILIATED TO **SAVITRIBAI PHULE PUNE UNIVERSITY, PUNE** FOR THE ACADEMIC YEAR 2023-2024.

SEAT NO: - _____

Dr. Sanjay Dharmadhikari

Project Guide | Examiner | Director, IBMRD

DATE : / /2024

PLACE : Ahmednagar

Acknowledgement

At the outset, we express our gratitude for the support that has enabled the successful completion of this project. While our names appear on the cover of this book, numerous individuals have played a crucial role in various capacities during the development of this project. We acknowledge and thank each one of them for their indispensable assistance and support.

I wish to express my heartfelt gratitude to all the people who have played a crucial role in this project. Without their active cooperation, the preparation of this project could not have been completed within the specified time limit.

I would like to record our deep sense of gratitude to our project guide for providing constant motivation and invaluable help throughout the project work. Special thanks to **Dr. Sanjay Dharmadhikari (Director)** and **Dr. Shubhangi Potdar Mam (H.O.D)** of the Institute of Business Management and Rural Development for their valuable suggestions and advice throughout the MCA course. Additionally, we extend our thanks to other faculty members for their cooperation during our course.

Finally, we express our appreciation to our friends for their cooperation, which played a crucial role in the successful completion of this project.

Mr. Rahul Shivaji Vikhe

Chapter 1: Introduction

Abstract

The "Notes Manager using Node.js" project is a comprehensive solution designed to facilitate efficient note-taking and management for users. Leveraging a modern tech stack including Node.js, MongoDB, HTML, CSS, Express, and Passport.js, this application empowers users to capture, organize, and retrieve information seamlessly.

Note-taking is a fundamental process for capturing vital information from various sources or events. Our application streamlines this process, offering features such as writing, jotting, labeling, and outlining. Users can customize their notes, leveraging diverse editing options.

In a landscape where numerous note-taking applications exist, each with unique features and functionalities, our project stands out for its user-centric design and robust capabilities. By harnessing the power of Node.js and MongoDB, we ensure scalability, reliability, and performance, while HTML and CSS provide a polished and intuitive user interface.

In summary, the "Notes Manager using Node.js" project represents a sophisticated yet user-friendly approach to note-taking, providing a seamless experience for users to organize and access their digital information effectively.

1.1 Existing System:

Prior to the development of the "Notes Manager using Node.js" project, traditional note-taking systems relied on manual methods and disparate tools. Users typically resorted to pen and paper or basic digital documents to capture and organize their notes. However, these approaches often lacked cohesion and efficiency, leading to challenges in information retrieval and organization.

In this existing system, users faced difficulties in managing their growing volume of notes, as manual processes made tasks such as categorization, searching, and editing cumbersome and time-consuming. Furthermore, the absence of a centralized platform hindered collaboration and sharing among users, limiting the accessibility and usefulness of the notes.

Moreover, the lack of integration with modern technologies like cloud storage and multimedia attachments restricted the richness and versatility of note-taking experiences. Users were unable to seamlessly incorporate images, audio, or video into their notes, limiting their ability to capture diverse types of information effectively.

Overall, the existing note-taking system suffered from inefficiencies in organization, collaboration, and multimedia integration, resulting in a suboptimal user experience for both individual users and collaborative groups.

1.2 **Need for the System:**

The introduction of the "Notes Manager using Node.js" system addresses several critical needs and considerations, significantly improving the efficiency and effectiveness of note-taking processes. The need for this system arises from the following factors:

Enhanced Efficiency and Accuracy: The existing manual note-taking processes are prone to errors and inefficiencies, leading to challenges in organizing, searching, and managing notes effectively. By implementing an automated system, users can streamline note-taking tasks such as writing, editing, categorizing, and searching, thereby enhancing overall efficiency and accuracy.

Centralized Information Management: The new system establishes a centralized database that consolidates all notes, user accounts, and associated metadata. This centralized approach simplifies data management, reduces redundancy, and ensures consistency across the platform. Users benefit from seamless access to their notes and associated information, promoting better organization and retrieval.

Improved User Experience: Unlike the traditional manual system, which lacks an intuitive interface, the new system leverages modern frontend technologies such as HTML and CSS to provide a user-friendly interface. This design enhances the overall user experience by facilitating easy navigation, interaction, and customization of notes. Users can access and manage their notes with greater ease and efficiency.

Access Control and Security: The implementation of user authentication and authorization mechanisms, such as Passport.js, ensures secure access control to the system. User roles, including administrators, are defined to manage access rights and permissions effectively. This feature safeguards the integrity and confidentiality of users' notes, mitigating risks associated with unauthorized access and data breaches.

Overall, the "Notes Manager using Node.js" system addresses the pressing needs for efficient, centralized, user-friendly, and secure note-taking solutions, providing users with a seamless and effective platform to capture, organize, and manage their digital information.

1.2 Scope of Work:

Requirements Analysis:

Conduct in-depth consultations with stakeholders and end-users to understand the specific needs and challenges faced by the library. Gather insights into the functionalities required for efficient library management, considering aspects such as book cataloguing, user registration, resource tracking, and reporting.

System Design:

Develop a comprehensive system design that encompasses architectural considerations, database schema design, and user interface design. Ensure that the system design adheres to industry best practices and standards, with a focus on scalability, reliability, and performance.

Backend Development with Node.js:

Implement the backend logic using Node.js, leveraging the Express framework for building web applications.Develop modules for core functionalities such as note creation, retrieval, updating, and deletion. Integrate with MongoDB to establish a secure and efficient database backend for storing and managing notes and user data.

Frontend Development with HTML/CSS:

Design and develop a user-friendly and responsive frontend interface using HTML and CSS. Create an intuitive and visually appealing interface for administrators and users to interact with the system. Ensure cross-device compatibility to provide a seamless experience across desktop and mobile platforms.

Build Automation with npm:

Utilize npm as the package manager for managing project dependencies and automating build processes. Configure the build pipeline to streamline development and deployment workflows, ensuring efficiency and consistency.

Testing:

Conduct thorough testing across multiple levels, including unit testing, integration testing, and user acceptance testing. Identify and address any bugs or issues to ensure the reliability and stability of the system.

1.3 Operating Environment

Hardware Requirements:

Client-side requirements:

- RAM: 512mb

Developer Side Requirements:

- Processor: Core 2 duo or more.
- RAM: 2gb

Software Requirements:

Client-Side Requirements:

- Operating System: Any
- Browser: Any (Firefox recommended)

Developer side requirements:

- Operating System: Windows 10
- Software: Firefox, Visual Studio Code, Git, Docker, NPM

1.4 Detailed Description of Technology Used

1. Nodejs

Node.js is a powerful open-source, cross-platform JavaScript runtime environment built on Chrome's V8 JavaScript engine. It enables developers to execute JavaScript code outside of a web browser, allowing for server-side scripting and the development of scalable network applications. Here's a detailed description of Node.js:

Asynchronous and Event-Driven: Node.js employs an event-driven, non-blocking I/O model, which makes it lightweight and efficient for handling concurrent operations. This architecture allows Node.js to handle multiple connections simultaneously without the need for threading, resulting in high scalability and responsiveness.

JavaScript Runtime: Node.js allows developers to write server-side code in JavaScript, leveraging their existing skills and libraries from the front-end development ecosystem. This unified language approach simplifies the development process and promotes code reuse across different parts of the application.

Package Management with npm: Node.js comes with npm (Node Package Manager), the largest ecosystem of open-source libraries and tools for JavaScript. npm facilitates package installation, version management, and dependency resolution, enabling developers to integrate third-party modules seamlessly into their projects.

Cross-Platform Compatibility: Node.js is designed to run on various operating systems, including Windows, macOS, and Linux, ensuring flexibility and portability across different environments. This cross-platform compatibility allows developers to write code once and deploy it across multiple platforms without modifications.

Scalability and Performance: Node.js is well-suited for building real-time, data-intensive applications that require high concurrency and low-latency responses. Its non-blocking I/O model and event-driven architecture enable efficient utilization of hardware resources, resulting in enhanced scalability and performance for applications handling a large volume of simultaneous connections.

Rich Ecosystem: Node.js has a vibrant and active community, supported by a vast ecosystem of modules, frameworks, and tools for various use cases. From web frameworks like Express.js for building web servers to utilities for task automation and testing, Node.js offers a wide range of resources to streamline development workflows and enhance productivity.

Overall, Node.js is a versatile and powerful runtime environment for building server-side applications, offering developers the flexibility, performance, and scalability needed to tackle diverse project requirements effectively.

2. HTML (Hypertext Markup Language)

HTML, or Hypertext Markup Language, is the standard markup language used to create and structure web pages and web applications. It provides the foundation for presenting content on the World Wide Web and defines the structure and layout of web documents. Here's a detailed description of HTML:

Semantic Structure: HTML employs a hierarchical structure of elements to represent content on web pages. These elements, such as , <p>, <header>, <footer>, and <nav>, provide semantic meaning to different parts of the document, facilitating accessibility, search engine optimization (SEO), and readability.

Tag-based Syntax: HTML documents consist of a series of tags enclosed in angle brackets (< >). Tags define the structure and content of the document, including headings, paragraphs, lists, images, links, forms, and more. Each tag may contain attributes that provide additional information or behavior, such as the href attribute for hyperlinks or the src attribute for image sources.

Cross-Browser Compatibility: HTML is supported by all modern web browsers, ensuring cross-browser compatibility for web pages and applications. Developers can write HTML code that renders consistently across different browsers and devices, providing a seamless user experience for visitors.

Integration with CSS and JavaScript: HTML works in conjunction with Cascading Style Sheets (CSS) and JavaScript to enhance the presentation and interactivity of web pages. CSS is used to style HTML elements, defining properties such as colors, fonts, layouts, and animations. JavaScript enables dynamic behavior and client-side interactions, allowing developers to create interactive features and respond to user actions.

Accessibility and SEO: HTML's semantic structure enhances accessibility by providing meaningful labels and landmarks for assistive technologies like screen readers. Properly structured HTML documents improve search engine rankings by enabling search engines to understand the content and context of web pages, leading to better visibility and discoverability.

Responsive Design: HTML supports responsive web design principles, allowing developers to create layouts that adapt to different screen sizes and devices. By using techniques such as media queries and flexible layouts, HTML enables the creation of mobile-friendly and responsive web pages that provide optimal viewing experiences across desktops, tablets, and smartphones.

Overall, HTML serves as the backbone of web development, providing the structural framework for creating accessible, cross-browser compatible, and responsive web pages and applications. Its simplicity, versatility, and interoperability with other web technologies make it an essential skill for developers building modern web experiences.

3. CSS (Cascading Style Sheets):

CSS, or Cascading Style Sheets, is a stylesheet language used to control the presentation and layout of HTML documents. It allows developers to define the visual appearance of web pages, including styles such as colors, fonts, spacing, layout, and animations. Here's a detailed description of CSS:

Styling HTML Elements: CSS enables developers to apply styles to HTML elements, specifying attributes such as colors, fonts, backgrounds, borders, and margins. By using selectors and declarations, developers can target specific elements or groups of elements and define their visual properties.

Separation of Concerns: CSS promotes the separation of content (HTML) from presentation (CSS) and behavior (JavaScript), facilitating modular and maintainable code. This separation allows developers to update the visual appearance of web pages without modifying the underlying HTML structure, enhancing code reusability and scalability.

Selector Specificity and Inheritance: CSS uses selectors to target HTML elements for styling, with varying levels of specificity. Selectors can target elements based on their type, class, ID, attributes, or relationships with other elements. Additionally, CSS properties can inherit values from parent elements, providing a hierarchical approach to styling.

Responsive Design: CSS supports responsive web design techniques, allowing developers to create layouts that adapt to different screen sizes and devices. Media queries enable conditional styling based on factors such as viewport width, height, and device orientation, enabling the creation of mobile-friendly and responsive web pages.

Flexbox and Grid Layout: CSS introduces layout models such as Flexbox and CSS Grid, which provide powerful tools for creating complex and flexible layouts. Flexbox enables one-dimensional layouts, allowing elements to be aligned and distributed within a container along a single axis. CSS Grid provides a two-dimensional grid system, enabling precise control over row and column layout.

Animation and Transitions: CSS allows developers to create animations and transitions to add interactivity and visual effects to web pages. Animation properties such as animation, transition, and keyframes enable the creation of smooth transitions, transformations, and motion effects, enhancing the user experience.

Overall, CSS plays a crucial role in web development by providing the means to control the visual presentation of HTML documents. Its flexibility, modularity, and support for responsive design make it an essential tool for creating modern and visually appealing web interfaces.

4. MongoBD

MongoDB is a widely used, open-source NoSQL database management system that provides high performance, scalability, and flexibility for storing and managing unstructured or semi-structured data. Here's a detailed description of MongoDB:

Document-Oriented Database: MongoDB is a document-oriented database, which means it stores data in flexible, JSON-like documents instead of traditional row-and-column tables used in relational databases. Each document can have its own structure, allowing for the storage of heterogeneous data types within the same collection.

Schemaless Design: Unlike relational databases, MongoDB does not enforce a rigid schema for data storage. This schemaless design allows developers to store and modify data dynamically, without the need for predefined schemas or migrations. It provides flexibility in adapting to evolving data requirements and simplifies the development process.

High Scalability: MongoDB is designed for horizontal scalability, allowing for distributed data storage and high availability across multiple nodes. It supports sharding, a technique for distributing data across multiple servers, to handle large volumes of data and high traffic loads. This scalability enables MongoDB to meet the needs of rapidly growing applications and environments.

High Performance: MongoDB offers high-performance read and write operations through features such as in-memory caching, asynchronous I/O, and native indexing. It leverages a memory-mapped storage engine and optimized query execution to deliver low-latency responses and efficient data retrieval. Additionally, MongoDB's flexible data model minimizes the need for complex joins and transactions, further enhancing performance.

Rich Query Language: MongoDB provides a powerful query language with support for a wide range of query operators, aggregation pipelines, and geospatial queries. Developers can perform complex queries, aggregations, and data manipulations directly within the database, reducing the need for additional processing in application code.

Replication and Fault Tolerance: MongoDB supports replica sets, a form of data replication that ensures data durability and fault tolerance. Replica sets consist of multiple nodes that maintain copies of the same data, allowing for automatic failover and recovery in the event of node failures or network issues. This architecture provides data redundancy and high availability for mission-critical applications.

Overall, MongoDB is a versatile and feature-rich database management system that offers flexibility, scalability, and performance for a wide range of use cases, from small-scale applications to large-scale enterprise deployments. Its document-oriented approach, combined with robust scalability and fault tolerance features, makes it a popular choice for modern web and mobile applications, big data analytics, and real-time data processing.

5. Express

Express.js is a minimal and flexible Node.js web application framework that provides a robust set of features for building web applications and APIs. It simplifies the process of developing server-side applications by providing a lightweight, unopinionated framework with powerful middleware capabilities. Here's a detailed description of Express.js:

Middleware Architecture: Express.js is built around the concept of middleware, which are functions that have access to the request and response objects in the application's request-response cycle. Middleware functions can perform tasks such as parsing request bodies, handling sessions, authenticating users, and serving static files. This modular architecture enables developers to compose complex applications by chaining together middleware functions in a flexible and reusable manner.

Routing: Express.js provides a simple and intuitive routing system that allows developers to define routes for handling different HTTP *requests (GET, POST, PUT, DELETE, etc.)* and URL patterns. Routes can be defined using HTTP methods and URL patterns, making it easy to create RESTful APIs and web services. Express.js supports route parameters, query string parameters, and route handlers, enabling developers to build dynamic and flexible APIs.

Template Engines: Express.js supports a variety of template engines, including popular options like Pug (formerly Jade), EJS (Embedded JavaScript), and Handlebars. Template engines allow developers to generate HTML dynamically by rendering data and logic within templates. Express.js integrates seamlessly with template engines, providing built-in support for rendering views and passing data to templates.

Error Handling: Express.js provides built-in error handling middleware that can catch and handle errors occurring during the request-response cycle. Developers can define custom error-handling middleware to handle specific types of errors, such as validation errors, database errors, or server errors. Express.js also supports asynchronous error handling, allowing errors to be propagated through the middleware chain and handled asynchronously.

Static File Serving: Express.js includes middleware for serving static files, such as HTML, CSS, JavaScript, images, and other assets. Developers can specify a directory containing static files, and Express.js will serve those files directly to clients without requiring additional routing or configuration. This feature simplifies the process of serving static content and improves application performance by offloading file serving to the web server.

Overall, Express.js is a powerful and versatile framework for building web applications and APIs in Node.js. Its minimalist design, middleware architecture, and robust feature set make it a popular choice for developers looking to create scalable and maintainable server-side applications. Whether building simple RESTful APIs or complex web applications, Express.js provides the tools and flexibility needed to bring ideas to life quickly and efficiently.

6. npm (Node Package Manager)

npm, short for Node Package Manager, is the default package manager for Node.js. It is a command-line tool that allows developers to discover, install, and manage dependencies for Node.js projects. Here's a detailed description of npm:

Dependency Management: npm simplifies the process of managing dependencies for Node.js projects by providing a centralized repository of packages and libraries. Developers can use npm to install packages from the npm registry, which contains thousands of open-source modules and libraries for various use cases.

Package Installation: npm allows developers to install packages locally within a project or globally on their system. Local installations are typically used for project-specific dependencies, while global installations are useful for command-line tools and utilities that are intended to be accessible from anywhere on the system.

Package.json Configuration: npm uses a file named package.json to store metadata and configuration for Node.js projects. The package.json file contains information such as project name, version, description, author, dependencies, scripts, and more. Developers can use npm commands to initialize a new package.json file, add dependencies, and manage project settings.

Semantic Versioning: npm follows semantic versioning (SemVer) conventions for versioning packages and dependencies. Each package version consists of three numbers separated by periods (e.g., 1.2.3), representing major, minor, and patch versions, respectively. Developers can specify version ranges and constraints in the package.json file to control which versions of dependencies are installed.

Dependency Resolution: npm automatically resolves and installs dependencies for Node.js projects based on the dependencies listed in the package.json file. When installing packages, npm analyzes the dependency tree to ensure that all dependencies are installed correctly and that version conflicts are resolved according to SemVer rules.

Custom Registries and Scoped Packages: npm supports custom registries and scoped packages, allowing organizations to host private packages and repositories internally. Scoped packages are namespaced under a specific organization or user, providing a way to manage and distribute packages within a controlled environment.

Overall, npm is an essential tool for Node.js developers, providing a convenient and reliable way to manage dependencies, automate tasks, and streamline development workflows. Its vast ecosystem of packages, robust dependency resolution, and support for versioning and lifecycle management make it indispensable for building modern Node.js applications and libraries.

7. Passport.js

Passport.js is a popular authentication middleware for Node.js applications. It provides a simple and modular approach to authentication by abstracting away the complexities of authentication mechanisms and providing a unified interface for integrating various authentication strategies. Here's a detailed description of Passport.js:

Modular and Extensible: Passport.js follows a modular architecture, allowing developers to integrate authentication strategies (also known as "providers") as separate modules. Authentication strategies represent different methods of authenticating users, such as username/password, social logins (e.g., OAuth, OpenID), JWT (JSON Web Tokens), and more. Developers can choose the appropriate strategy(s) for their application's authentication requirements and easily plug them into their application.

Authentication Flow: Passport.js simplifies the authentication flow by providing a consistent interface for handling authentication requests and responses. It abstracts away the low-level details of authentication mechanisms, such as token generation, session management, and user verification, allowing developers to focus on application logic rather than authentication implementation.

Middleware Integration: Passport.js integrates seamlessly with popular Node.js web frameworks such as Express.js. It provides middleware functions that can be easily integrated into Express.js middleware pipelines to authenticate incoming requests. Passport.js middleware can be added to specific routes or globally to authenticate all requests, providing fine-grained control over authentication logic.

Community and Ecosystem: Passport.js has a vibrant and active community with a wide range of authentication strategies and plugins available. Developers can leverage existing Passport.js strategies or create custom strategies to support authentication with various third-party providers, services, and identity providers. The extensive ecosystem of Passport.js plugins and extensions simplifies integration with external authentication systems and enhances the functionality of Passport.js.

Security: Passport.js follows best practices for authentication security, including protection against common security vulnerabilities such as CSRF (Cross-Site Request Forgery), session fixation, and brute-force attacks. It provides mechanisms for securely storing and transmitting user credentials, protecting sensitive information, and preventing unauthorized access to protected resources.

Overall, Passport.js is a flexible and versatile authentication middleware for Node.js applications, offering a streamlined approach to implementing authentication functionality. Whether building simple web applications or complex API servers, Passport.js simplifies the authentication process and provides a solid foundation for securing Node.js applications against unauthorized access.

8. Amazon Elastic Compute Cloud (EC2):

Amazon Elastic Compute Cloud (EC2) is a web service provided by Amazon Web Services (AWS) that offers resizable compute capacity in the cloud. It allows users to launch and manage virtual servers, known as instances, to run applications and workloads in a scalable and cost-effective manner. Here's a detailed description of Amazon EC2:

Virtual Servers (Instances): Amazon EC2 enables users to create and launch virtual servers, known as instances, within the AWS cloud. Instances are available in various configurations, including different instance types (e.g., general-purpose, compute-optimized, memory-optimized) and sizes (e.g., small, medium, large), allowing users to choose the resources that best suit their application requirements.

Scalability: Amazon EC2 provides auto-scaling capabilities, allowing users to automatically scale the number of instances based on demand. Auto-scaling helps maintain performance and availability by dynamically adjusting capacity to handle fluctuations in traffic and workload. Users can define scaling policies based on metrics such as CPU utilization, network traffic, or custom application metrics.

Elasticity: EC2 offers elasticity, allowing users to easily increase or decrease compute capacity as needed. Users can launch instances on-demand, provision additional capacity for temporary workloads, or terminate instances to reduce costs when resources are no longer needed. EC2 instances can be resized, stopped, started, and terminated programmatically via APIs or through the AWS Management Console.

Customization: Amazon EC2 provides flexibility and customization options for configuring instances to meet specific application requirements. Users can choose from a wide range of operating systems, including Amazon Linux, Ubuntu, Windows Server, and others.

Security: Amazon EC2 offers various security features to protect instances and data in the cloud. Users can leverage security groups to control inbound and outbound traffic, define access rules, and enforce network segmentation. EC2 instances can be launched within Virtual Private Clouds (VPCs), providing isolation and control over network settings, routing, and access control.

Integration with AWS Services: Amazon EC2 integrates seamlessly with other AWS services, enabling users to build scalable and resilient architectures. Users can leverage services such as Amazon Elastic Block Store (EBS) for persistent block storage, Amazon Simple Storage Service (S3) for object storage, and AWS Identity and Access Management (IAM) for user authentication and authorization.

Overall, Amazon EC2 is a versatile and scalable compute service that provides on-demand access to virtual servers in the cloud.

9. Docker

Docker is a popular platform for developing, shipping, and running applications in containers. Containers are lightweight, portable, and self-sufficient units that encapsulate application code, runtime, system tools, libraries, and dependencies. Docker provides tools and services to build, deploy, and manage containers efficiently. Here's a detailed description of Docker:

Containerization: Docker enables containerization, a technology that allows developers to package applications and their dependencies into containers. Containers provide an isolated environment for running applications, ensuring consistency and reproducibility across different environments, such as development, testing, and production.

Docker Engine: Docker Engine is the core component of Docker that provides the runtime environment for containers. It consists of the Docker daemon (dockerd) and the Docker command-line interface (CLI). The Docker daemon runs as a background service on the host system and manages container lifecycle, image management, networking, and storage operations. The Docker CLI provides commands for interacting with Docker, such as building images, running containers, and managing Docker resources.

Docker Images: Docker images are read-only templates used to create containers. Images contain a filesystem snapshot with the application code, dependencies, libraries, and runtime environment needed to run the application. Developers can create custom Docker images using Dockerfiles, which are text files that specify the steps for building the image. Docker images can be shared and distributed via Docker Hub, a public registry for Docker images, or private repositories.

Resource Efficiency: Docker containers are lightweight and efficient, with minimal overhead compared to traditional virtual machines (VMs). Containers share the host system's kernel and resources, reducing memory and CPU overhead and enabling higher resource utilization density. Docker's efficient resource management makes it suitable for running multiple containers on a single host, optimizing resource usage and infrastructure costs.

DevOps and Continuous Integration/Continuous Deployment (CI/CD): Docker plays a crucial role in modern DevOps practices by enabling automation, collaboration, and consistency in the software development lifecycle. Docker containers are integral to CI/CD pipelines, allowing developers to build, test, and deploy applications rapidly and reliably. Docker images serve as immutable artifacts that can be versioned, tested, and deployed across different environments with confidence.

Overall, Docker revolutionizes software development and deployment by providing a standardized and efficient way to package, distribute, and run applications in containers. Its versatility, portability, and scalability make it a foundational technology for building modern, cloud-native applications and accelerating the pace of innovation in the software industry.

Chapter 2: Proposed System

2.1 The proposed system

The proposed "Notes Manager System" aims to revolutionize traditional note-taking practices by introducing a comprehensive and efficient solution that addresses the diverse needs of users. At the core of this system lies a commitment to automation, streamlining essential processes such as note creation, organization, and sharing. By automating these tasks, the system ensures accuracy and reliability, while reducing the burden of manual effort on users.

Key Features:

Automation and Efficiency:
The system automates note creation, organization, and management processes, reducing manual effort and minimizing errors. Advanced algorithms facilitate intelligent note categorization, tagging, and indexing, enhancing organizational efficiency.

User-Friendly Interface: The system boasts a user-friendly interface with intuitive navigation and clear visual elements, enhancing user experience. Responsive design ensures seamless interaction across various devices, catering to the preferences of modern users.

Role-Based Access Control:
Role-based access control ensures secure access to the system, with exclusive privileges granted to administrators.

Enhanced Note Management:
Robust note management functionalities empower users to organize, search, and retrieve notes effortlessly. Real-time updates on note status and revisions provide users with accurate and up-to-date information.

Reporting and Analytics:
Built-in reporting and analytics tools offer valuable insights into note usage patterns and trends. Customizable reports enable administrators to make informed decisions and optimize resource allocation.

Security and Privacy:
Stringent security measures, including secure user authentication and encrypted communication, safeguard sensitive note data. Data privacy and confidentiality are prioritized, ensuring the protection of user information and notes.

The proposed "Notes Manager System" represents a significant advancement in note-taking technology, offering users a modern, efficient, and user-centric solution.

2.2 Objectives of the System

Automation: Streamline note creation, organization, and sharing processes to increase operational efficiency and reduce manual effort for users.

User-Friendly Interface: Develop an intuitive and responsive interface that provides a seamless experience for users, allowing easy navigation and interaction across different devices.

Access Control: Implement role-based access control to regulate access to notes and ensure secure authentication for users, maintaining data privacy and confidentiality.

Efficient Note Management: Provide robust note management functionalities, including categorization, tagging, and search capabilities, to enable users to organize and retrieve notes efficiently.

Scalability: Design the system to accommodate the growth of the user base and note collection, ensuring scalability and performance as the application usage increases.
+
Security Measures: Implement stringent security measures, such as secure user authentication and encrypted communication, to protect sensitive note data and prevent unauthorized access or data breaches.

Enhanced User Experience: Prioritize user experience by focusing on usability, accessibility, and responsiveness, ensuring a positive and engaging experience for users throughout their interaction with the application.

Adherence to Standards: Ensure compliance with industry standards in software development, data management, and security practices to maintain the integrity, reliability, and trustworthiness of the application

2.3 User Requirements

Understanding the needs and expectations of users is crucial for the success of the "Library Management System." The user requirements are outlined to ensure the system meets the specific needs of administrator

Admin:

User-Friendly Interface: Administrators require an intuitive and easy-to-use interface for efficient navigation and task execution.

Role-Based Access: Differentiated access levels for administrators, with exclusive privileges for the "admin" user role.

Notes Management: Efficient tools for creating, updating, and tracking the note's collection, including real-time status updates.

Reporting: Comprehensive reporting and analytics feature to track notes usage, trends, and generate actionable insights.

Security: Robust security measures, including secure authentication, to safeguard sensitive library data.

Users:

Registration and Authentication: Users should be able to register for an account on the Notes App platform. Upon registration, users should be able to log in securely using their credentials, such as username/email and password.

Note Creation: Users should have the ability to create new notes within the application.
The note creation process should be straightforward and user-friendly, allowing users to input text, add attachments, and categorize notes as needed.

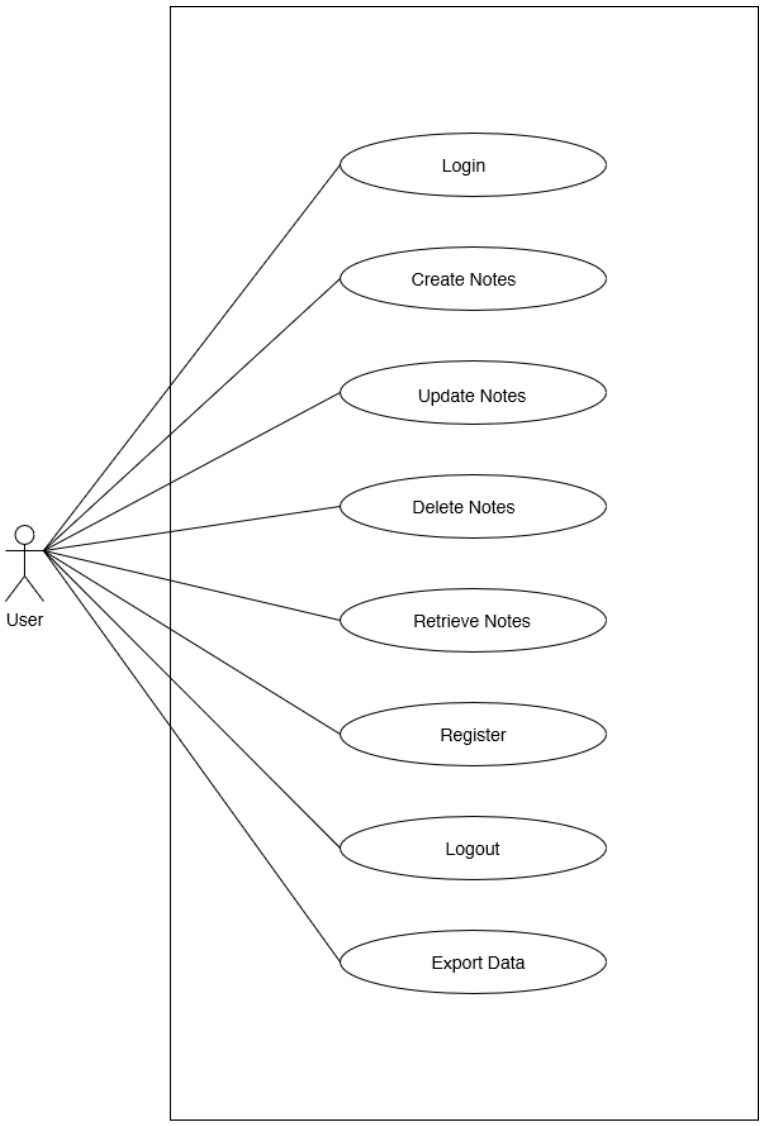
Note Management: Users should be able to view a list of their existing notes upon logging into the application.
Users should have the ability to update or modify existing notes, including editing text, and changing note attributes.

Note Retrieval: Users should be able to easily search for specific notes within the application. The application should provide efficient retrieval mechanisms, such as keyword search, filtering by tags or categories, and sorting options. Users should have access to their notes from any device or platform, ensuring seamless retrieval of information wherever they are.

Chapter 3: Analysis & design

3.1 Use Case Diagram

The use case diagram for the user interaction with the Notes App illustrates the various functionalities and interactions between the user and the system. The primary actor in this diagram is the "User," representing individuals who interact with the application to perform various tasks. Here's a description of the use case diagram elements:

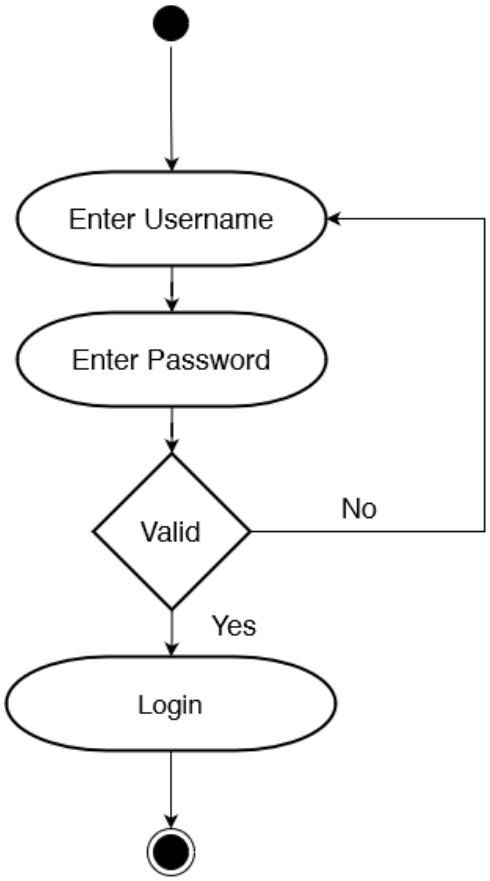


In this use case diagram:

The "User" use case includes various actions that an user can perform, Create Notes, User Login/Register, Update Notes, Delete notes, View Notes, etc

3.2 Activity Diagram - User Login

The following activity diagram illustrates the process of a user logging into the "Library Management System":



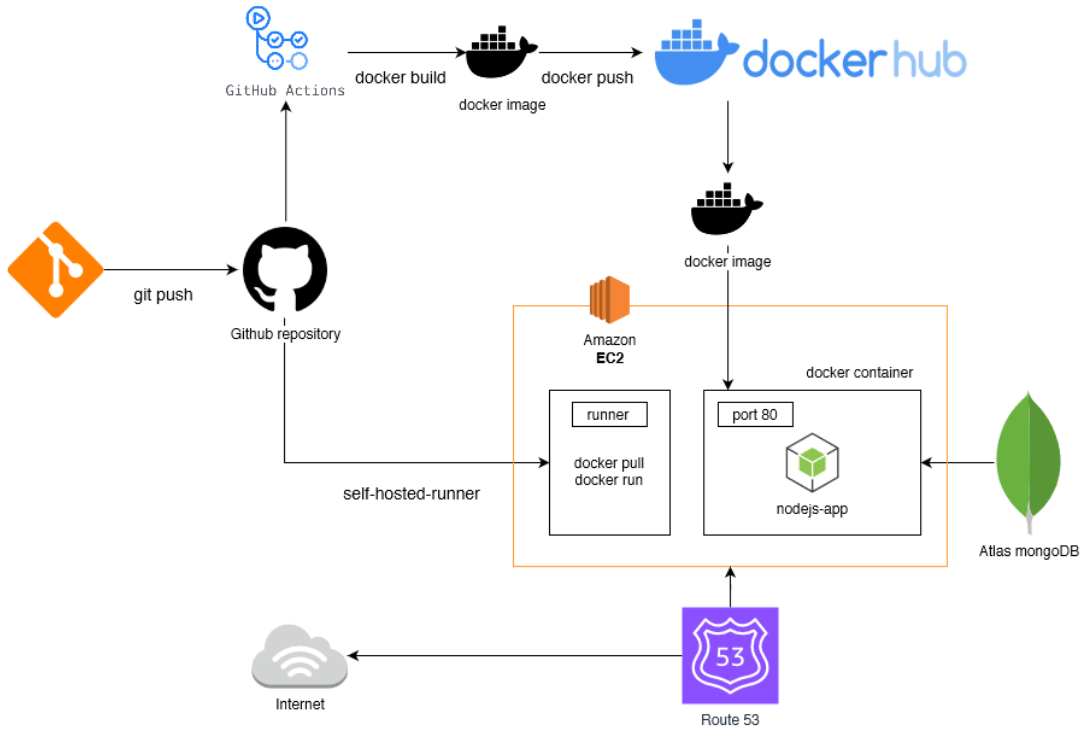
In this activity diagram:

The process begins with the user selecting the "Login" option.
The system displays a "Login" form where the user enters their credentials.
The system validates the entered credentials, based on whether they are valid or not.
If the credentials are valid, the system logs in the user.
If the credentials are not valid, the system prompts the user to re-enter their credentials.

This activity diagram provides a visual representation of the steps involved in the user login process, including credential validation.

3.3 Deployment and CICD - Diagram

This architecture diagram illustrates the deployment process for a Node.js application on an AWS EC2 instance using Docker containers with Continuous Integration and Continuous Deployment (CI/CD) implemented through GitHub Actions.

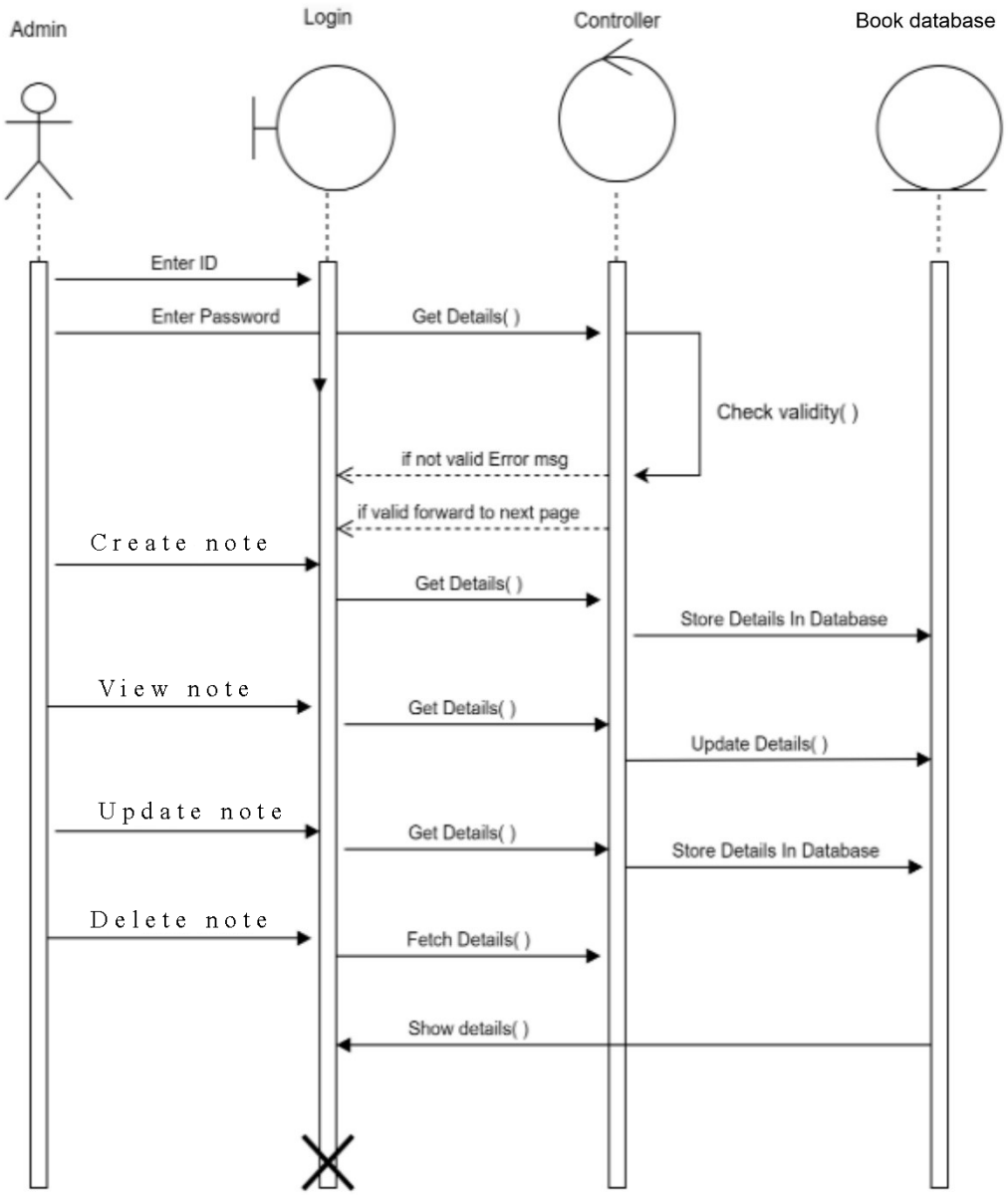


Visit for detailed deployment process

https://medium.com/@rahulvikhe25/deployment-of-node-js-application-on-ec2-with-ci-cd-using-github-actions-and-docker-container-46c3efa2905f

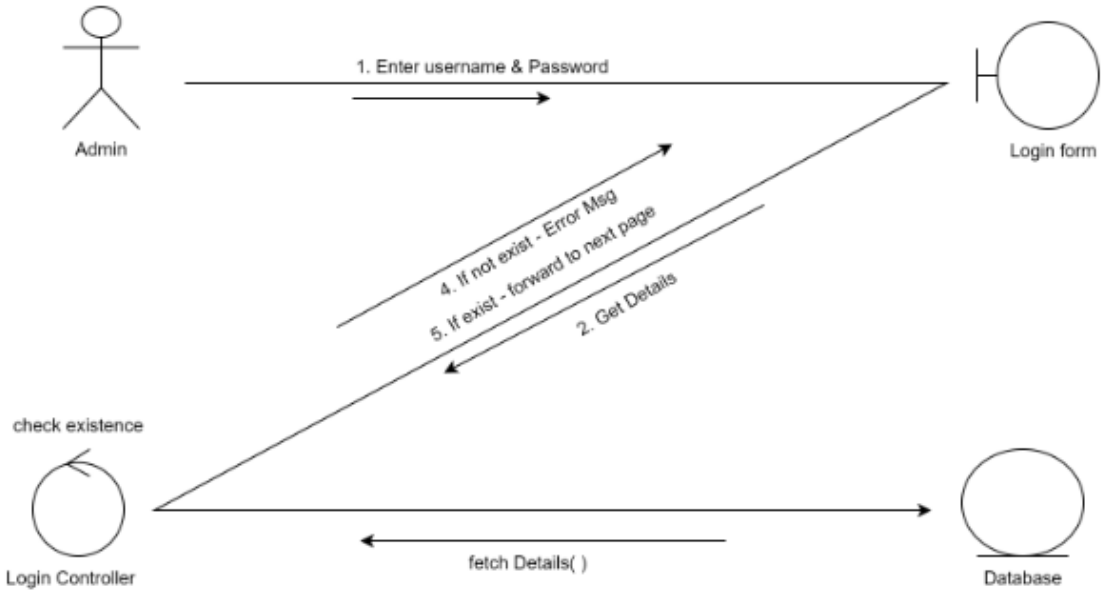
The diagram showcases the integration between various components such as the GitHub repository, GitHub Actions workflows, Docker Hub, AWS EC2 instance, MongoDB Atlas (or any other MongoDB hosting service), and Route 53 for DNS routing. It visualizes the connectivity and interactions between these components to provide a clear understanding of the deployment workflow.

3.4 Sequence Diagram

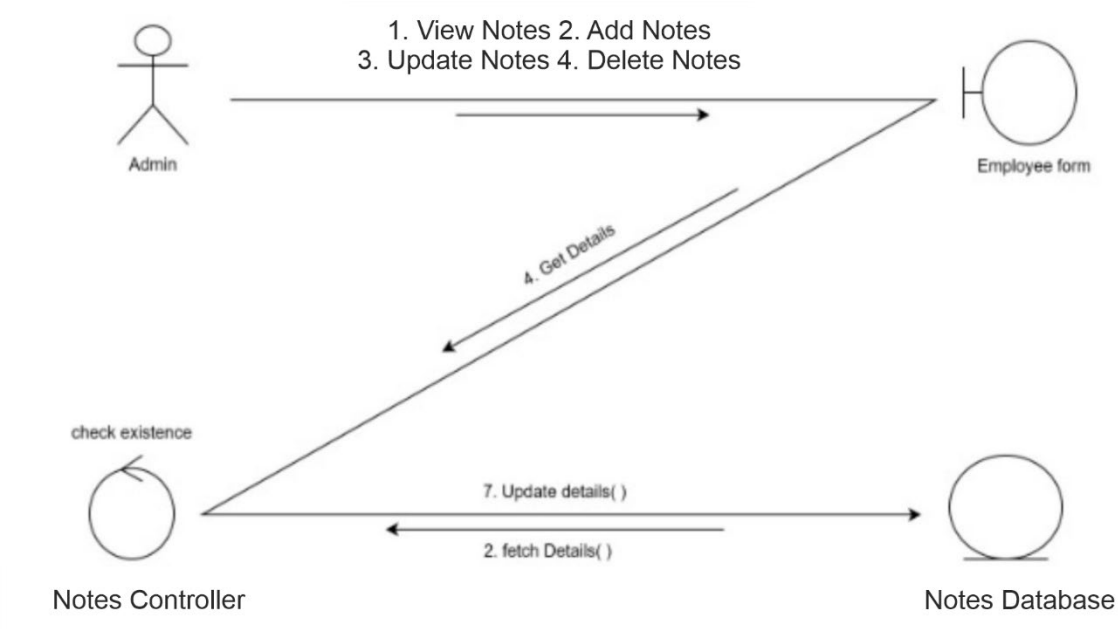


3.5 Collaboration Diagram

For Admin Login

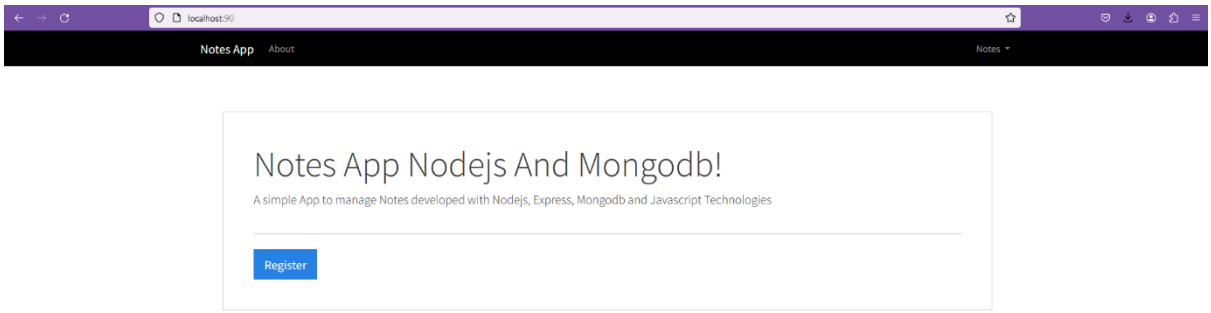


For Admin

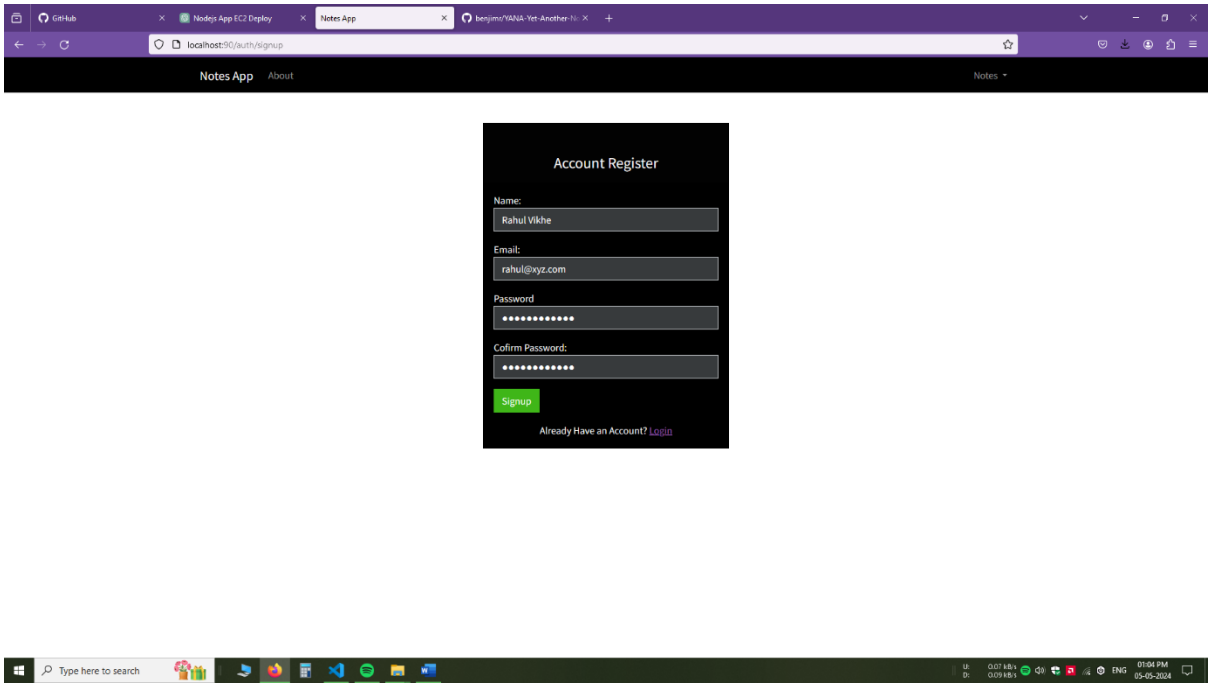


3.6 User interface design (screens etc.)

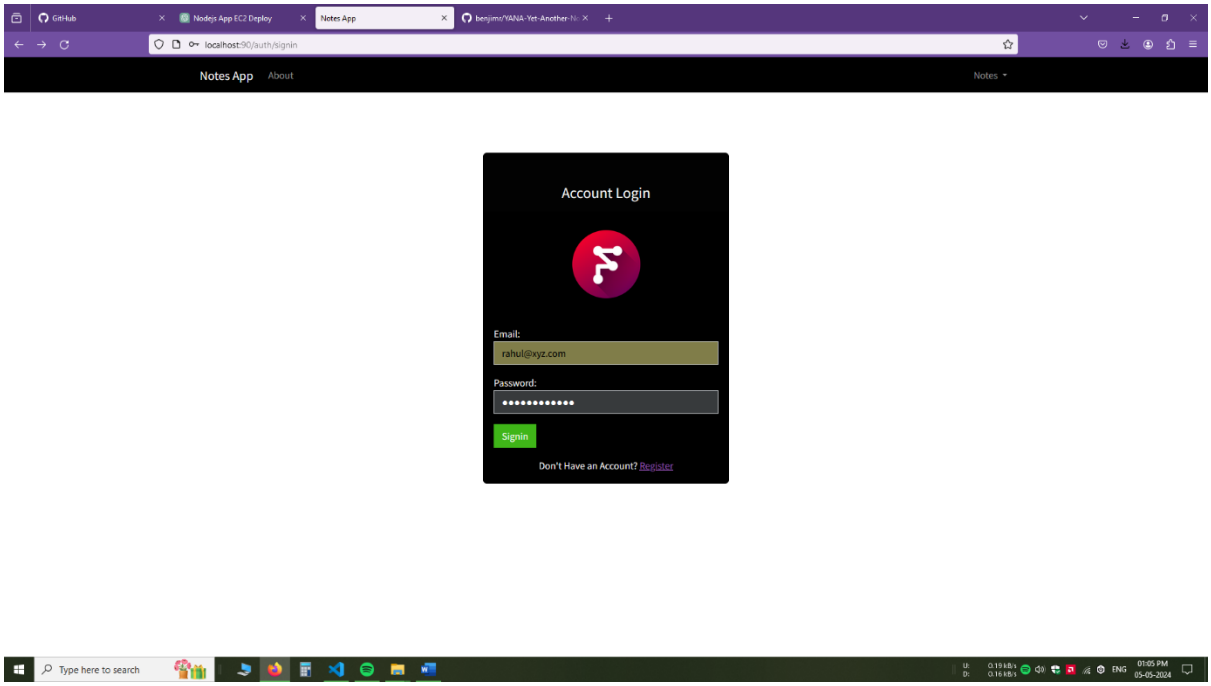
Homepage



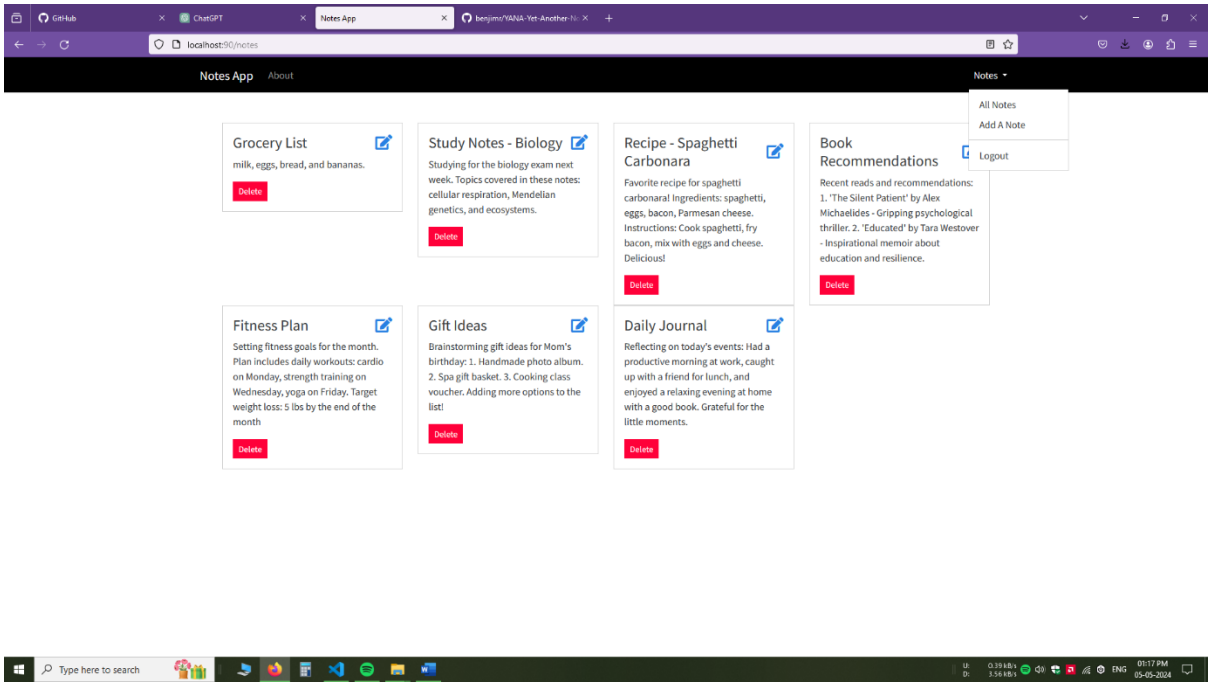
Registration form



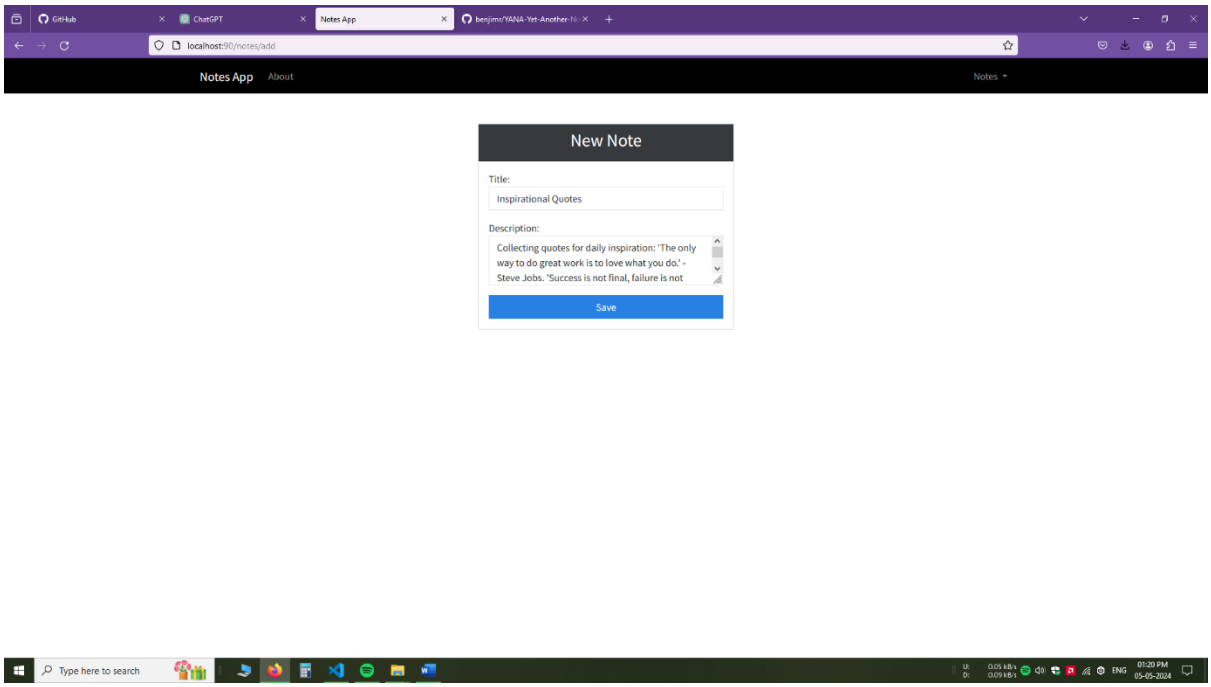
Login page



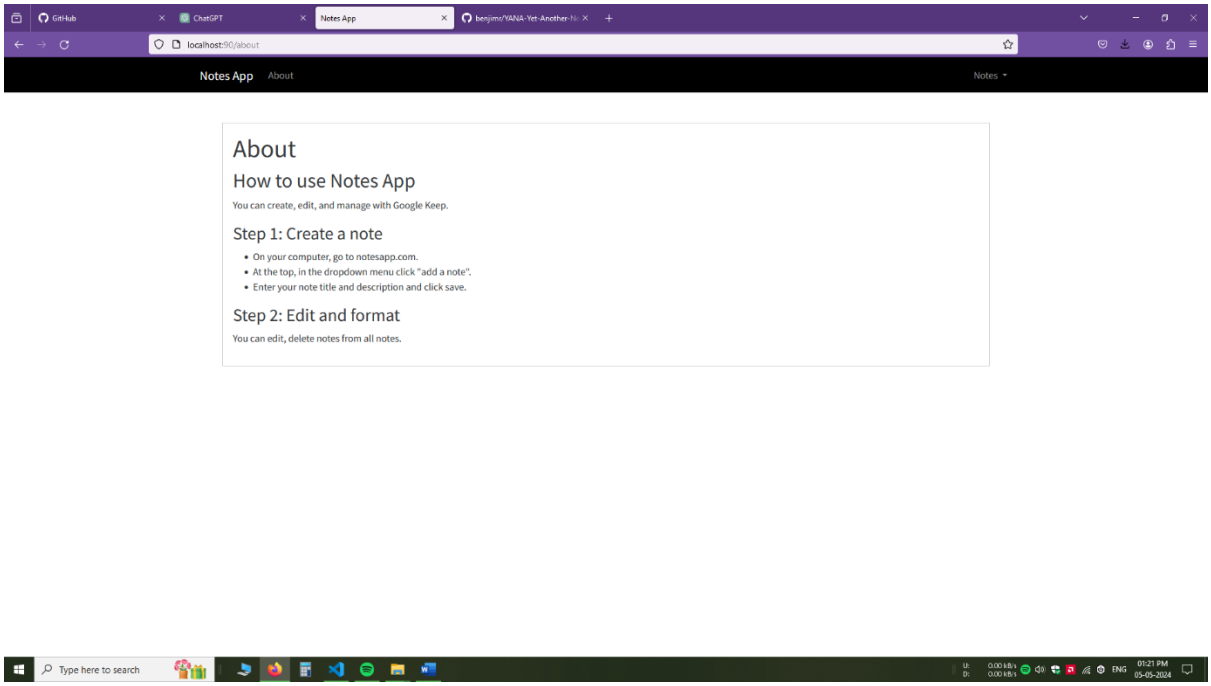
User Interface (All Notes)



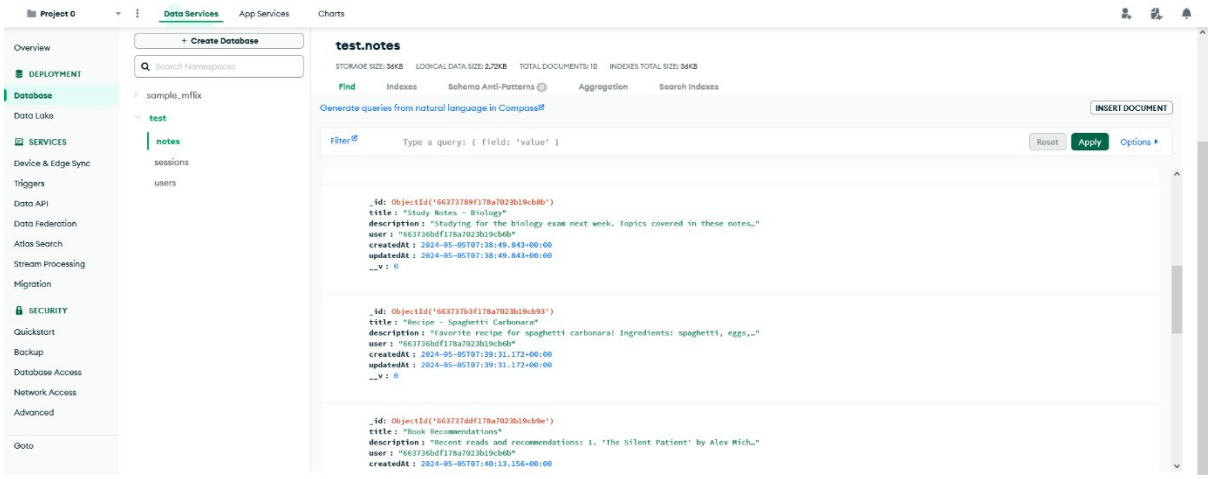
Add Note form



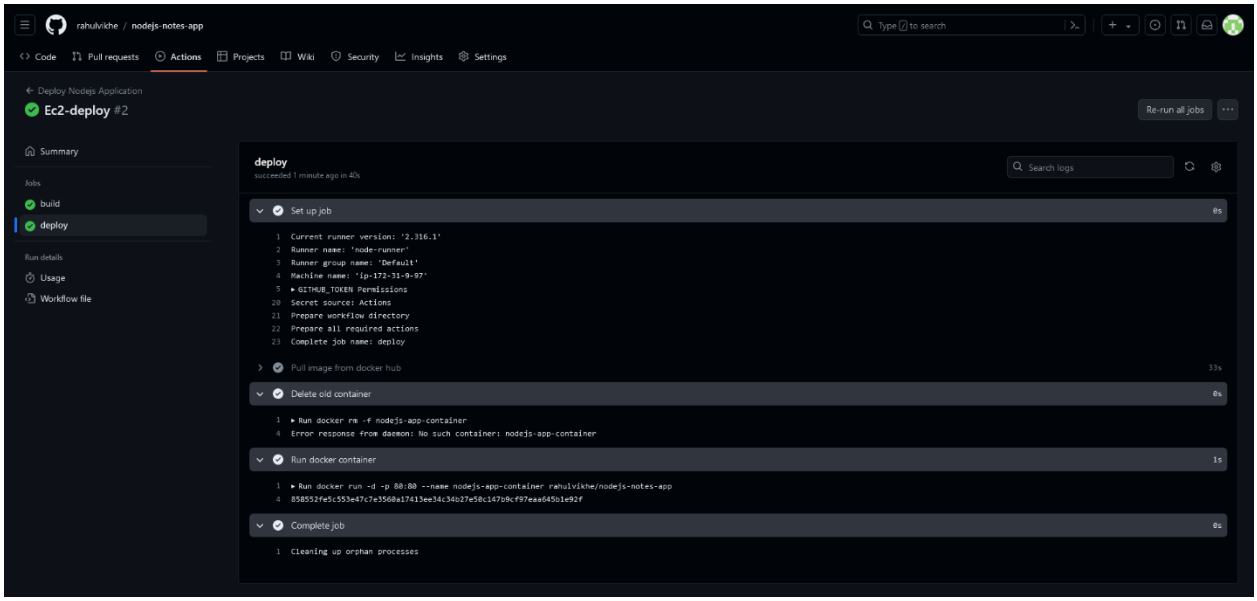
About Page



Atlas MongoDB

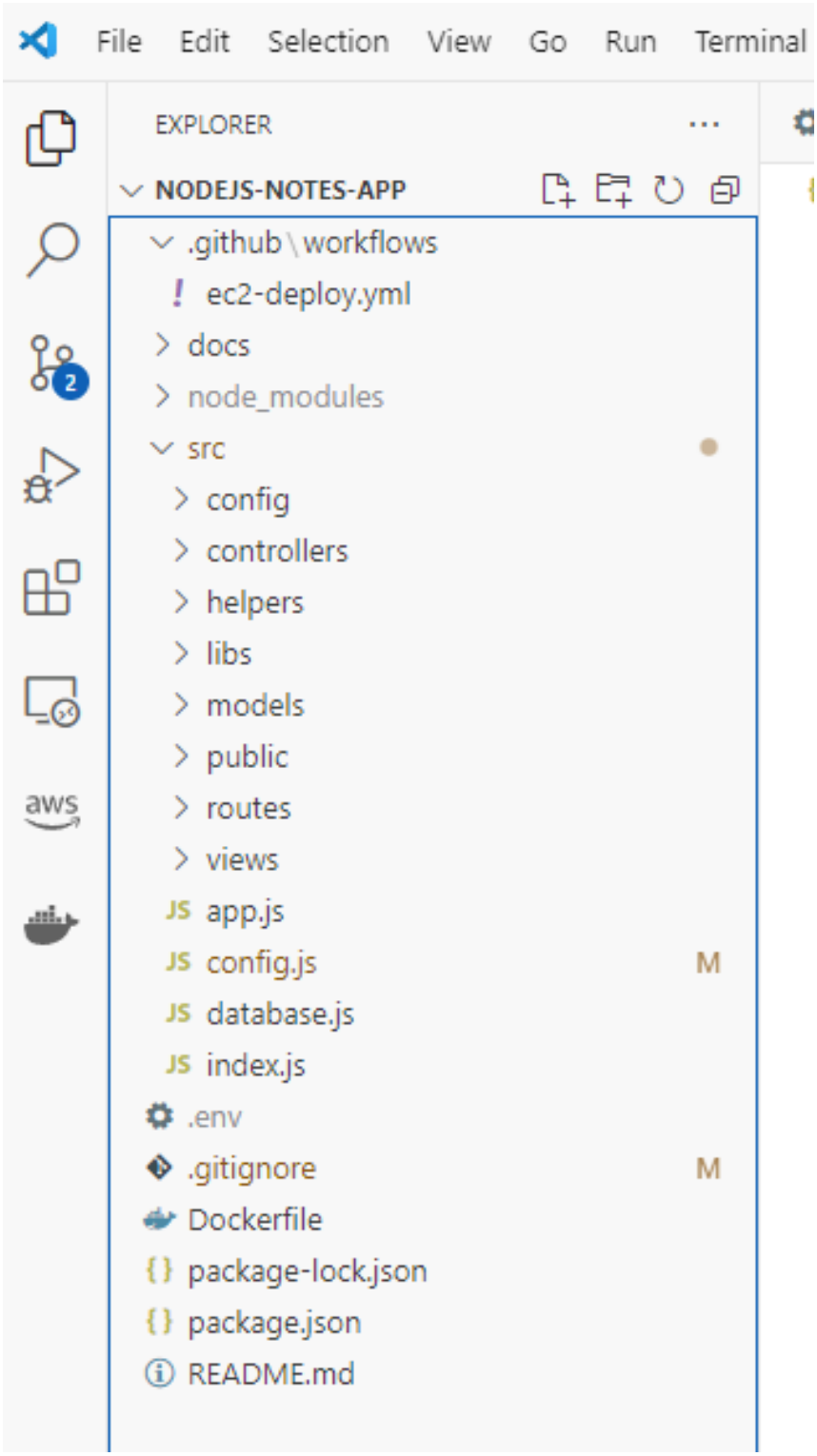


Github Actions



3.7 Sample Code

Code Structure



Package.json

```
{
  "name": "nodejs-notes-app",
  "version": "2.0.1",
  "description": "Notes App",
  "main": "index.js",
  "type": "module",
  "scripts": {
    "dev": "nodemon src/index.js",
    "start": "node src/index.js"
  },
  "keywords": [],
  "author": "",
  "license": "ISC",
  "dependencies": {
    "bcryptjs": "^2.4.3",
    "connect-flash": "^0.1.1",
    "connect-mongo": "^4.6.0",
    "express": "^4.18.1",
    "express-handlebars": "^6.0.6",
    "express-session": "^1.17.3",
    "method-override": "^3.0.0",
    "mongoose": "^6.5.2",
    "morgan": "^1.10.0",
    "passport": "^0.6.0",
    "passport-local": "^1.0.0"
  },
  "devDependencies": {
    "dotenv": "^16.0.1",
    "handlebars": "^4.7.7",
    "nodemon": "^3.1.0"
  }
}
```

main.hbs

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8">
    <title>Notes App</title>
    <!-- BOOTSTRAP - BOOTSWATCH THEME - COSMO -->
    <link rel="stylesheet" href="https://bootswatch.com/5/cosmo/bootstrap.min.css">
    <!-- FONT AWESOME -->
    <link rel="stylesheet" href="https://use.fontawesome.com/releases/v5.5.0/css/all.css"
integrity="sha384-
B4dIYHKNBt8Bc12p+WXckhzcICo0wtJAoU8YZTY5qE0Id1GSseTk6S+L3BlXeVIU"
crossorigin="anonymous">
    <!-- CUSTOM CSS -->
    <link rel="stylesheet" href="/css/main.css">
  </head>
  <body>

    {{> navigation }}

    <main class="container p-5">
      <div class="col-md-4 offset-md-4">
        {{> messages }}
        {{> errors }}
      </div>
      {{{ body }}}
    </main>

    <!-- Scritps -->
    <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.2.0-
beta1/dist/js/bootstrap.bundle.min.js"></script>
  </body>
</html>
```

signin.hbs

```
<div class="row">
  <div class="col-md-4 mx-auto">
    <div class="card mt-5 bg-black text-white rounded">
      <div class="card-header text-center pt-5">
        <h1 class="h4 text-white">
          Account Login
        </h1>
      </div>
      <img
        class="rounded-circle mx-auto d-block logo m-4"
        src="/img/logo.png"
        alt="Logo"
      />
      <div class="card-body">
        <form action="/auth/signin" method="POST">
          <div class="mb-3">
            <label for="email">Email:</label>
            <input
              type="email"
              class="form-control bg-dark text-white"
              name="email"
              placeholder="Email"
              autofocus
            />
          </div>
          <div class="mb-3">
            <label for="passswore">Password:</label>
            <input
              type="password"
              class="form-control bg-dark text-white"
              name="password"
              placeholder="Password"
            />
          </div>
          <button class="btn btn-success btn-block">
            Signin
          </button>
        </form>
      </div>

      <p class="text-center">Don't Have an Account? <a href="/auth/signup" class="text-
info">Register</a></p>
```

```
    </div>
  </div>
</div>
```

signup.hbs

```
<div class="row">
  <div class="col-md-4 mx-auto">
    <div class="card bg-black text-white">
      <div class="card-header pt-5">
        <h4 class="text-center">
          Account Register
        </h4>
      </div>
      <div class="card-body">
        <form action="/auth/signup" method="POST">
          <div class="mb-3">
            <label for="name">Name:</label>
            <input
              type="text"
              class="form-control bg-dark text-white"
              name="name"
              placeholder="Name"
              value="{{name}}"
            />
          </div>
          <div class="mb-3">
            <label for="email">Email:</label>
            <input
              type="email"
              class="form-control bg-dark text-white"
              name="email"
              placeholder="Email"
              value="{{email}}"
            />
          </div>
          <div class="mb-3">
            <label for="password">Password</label>
            <input
              type="password"
              class="form-control bg-dark text-white"
              name="password"
              placeholder="Password"
```

```
            value="{{password}}"
          />
        </div>
        <div class="mb-3">
          <label for="confirm_password">Cofirm Password:</label>
          <input
            type="password"
            class="form-control bg-dark text-white"
            name="confirm_password"
            placeholder="Confirm Password"
            value="{{confirm_password}}"
          />
        </div>
        <button class="btn btn-success btn-block">
          Signup
        </button>
      </form>
    </div>
    <p class="text-center">Already Have an Account?
      <a href="/auth/signin" class="text-info">Login</a></p>
  </div>
 </div>
</div>
```

404.hbs

```
<div class="row">
 <div class="col-md-4 offset-md-4">
  <div class="card card-body">
   <h1>Not Found</h1>
   <p>This page does not exists</p>
   <a href="/">Return to home</a>
  </div>
 </div>
</div>
```

about.hbs

```
<section class="card">
 <div class="card-body">
  <h1>About</h1>
  <h2>How to use Notes App</h2>
  <p>
   You can create, edit, and manage with Google Keep.
```

```
    </p>
    <h3>Step 1: Create a note</h3>
    <ul>
      <li>On your computer, go to notesapp.com.</li>
      <li>At the top, in the dropdown menu click "add a note".</li>
      <li>Enter your note title and description and click save.</li>
    </ul>
    <h3>Step 2: Edit and format</h3>
    <p>
      You can edit, delete notes from all notes.
    </p>
  </div>
</section>
```

error.hbs

```
<div class="row">
  <div class="col-md-4 offset-md-4">
    <div class="card card-body">
      <h1>Internal Server Error</h1>
      <p>{{error}}</p>
      <a href="/">Return to home</a>
    </div>
  </div>
</div>
```

index.hbs

```
<div class="card card-body p-5 mt-4">
  <h1 class="display-4">Notes App Nodejs And Mongodb!</h1>
  <p class="lead">A simple App to manage Notes developed with Nodejs, Express,
    Mongodb and Javascript Technologies</p>
  <hr class="my-4" />

  <div>
    <a
      class="btn btn-primary btn-lg"
      href="/auth/signup"
      role="button"
    >Register</a>
  </div>
</div>
```

app.js

```
import express from "express";
import exphbs from "express-handlebars";
import session from "express-session";
import methodOverride from "method-override";
import flash from "connect-flash";
import passport from "passport";
import morgan from "morgan";
import MongoStore from "connect-mongo";
import { dirname, join } from "path";
import { fileURLToPath } from "url";

import { MONGODB_URI, PORT } from "./config.js";

import indexRoutes from "./routes/index.routes.js";
import notesRoutes from "./routes/notes.routes.js";
import userRoutes from "./routes/auth.routes.js";
import "./config/passport.js";

// Initializations
const app = express();
const __dirname = dirname(fileURLToPath(import.meta.url));

// settings
app.set("port", PORT);
app.set("views", join(__dirname, "views"));

// config view engine
const hbs = exphbs.create({
  defaultLayout: "main",
  layoutsDir: join(app.get("views"), "layouts"),
  partialsDir: join(app.get("views"), "partials"),
  extname: ".hbs",
});
app.engine(".hbs", hbs.engine);
app.set("view engine", ".hbs");

// middlewares
app.use(morgan("dev"));
app.use(express.urlencoded({ extended: false }));
app.use(methodOverride("_method"));
app.use(
  session({
```

```
    secret: "secret",
    resave: true,
    saveUninitialized: true,
    store: MongoStore.create({ mongoUrl: MONGODB_URI }),
  })
);
app.use(passport.initialize());
app.use(passport.session());
app.use(flash());

// Global Variables
app.use((req, res, next) => {
  res.locals.success_msg = req.flash("success_msg");
  res.locals.error_msg = req.flash("error_msg");
  res.locals.error = req.flash("error");
  res.locals.user = req.user || null;
  next();
});

// routes
app.use(indexRoutes);
app.use(userRoutes);
app.use(notesRoutes);

// static files
app.use(express.static(join(__dirname, "public")));

app.use((req, res, next) => {
  return res.status(404).render("404");
});

app.use((error, req, res, next) => {
  res.status(error.status || 500);
  res.render("error", {
    error,
  });
});

export default app;
```

config.js

```
import { config } from "dotenv";
config();

export const PORT = process.env.PORT || 90;
export const MONGODB_URI =
`mongodb+srv://rahulvikhe:${process.env.MONGO_PASSWORD}@dev-
cluster.idtyynl.mongodb.net/?retryWrites=true&w=majority&appName=dev-cluster`;
```

database.js

```
import mongoose from "mongoose";
import { MONGODB_URI } from "./config.js";

mongoose.set('strictQuery', false);

try {
  const db = await mongoose.connect(MONGODB_URI);
  console.log("Connected to ", db.connection.name);
} catch (error) {
  console.error(error);
}

mongoose.connection.on("connected", () => {
  console.log("Mongoose is connected");
});

mongoose.connection.on("disconnected", () => {
  console.log("Mongoose is disconnected");
});
```

Index.js

```
import app from "./app.js";
import { createAdminUser } from "./libs/createUser.js";
import "./database.js";

async function main() {
  await createAdminUser();
```

```
  app.listen(app.get("port"));

  console.log("Server on port", app.get("port"));
  console.log("Environment:", process.env.NODE_ENV);
}

main();
```

src\routes\auth.routes.js

```
import { Router } from "express";
import {
  renderSignUpForm,
  signup,
  renderSigninForm,
  signin,
  logout,
} from "../controllers/auth.controllers.js";

const router = Router();

// Routes
router.get("/auth/signup", renderSignUpForm);

router.post("/auth/signup", signup);

router.get("/auth/signin", renderSigninForm);

router.post("/auth/signin", signin);

router.get("/auth/logout", logout);

export default router;
```

src\routes\notes.routes.js

```
import { Router } from "express";
import {
  renderNoteForm,
  createNewNote,
  renderNotes,
  renderEditForm,
```

```
  updateNote,
  deleteNote,
} from "../controllers/notes.controller.js";
import { isAuthenticated } from "../helpers/auth.js";

const router = Router();

// New Note
router.get("/notes/add", isAuthenticated, renderNoteForm);

router.post("/notes/new-note", isAuthenticated, createNewNote);

// Get All Notes
router.get("/notes", isAuthenticated, renderNotes);

// Edit Notes
router.get("/notes/edit/:id", isAuthenticated, renderEditForm);

router.put("/notes/edit-note/:id", isAuthenticated, updateNote);

// Delete Notes
router.delete("/notes/delete/:id", isAuthenticated, deleteNote);

export default router;
```

src\libs\createUser.js

```
import User from "../models/User.js";

export const createAdminUser = async () => {
  const userFound = await User.findOne({ email: "admin@localhost" });

  if (userFound) return;

  const newUser = new User({
    username: "admin",
    email: "admin@localhost",
  });

  newUser.password = await newUser.encryptPassword("adminpassword");

  const admin = await newUser.save();
```

```
  console.log("Admin user created", admin);
};
```

src\controllers\auth.controllers.js

```
import User from "../models/User.js";
import passport from "passport";

export const renderSignUpForm = (req, res) => res.render("auth/signup");

export const signup = async (req, res) => {
  let errors = [];
  const { name, email, password, confirm_password } = req.body;
  if (password !== confirm_password) {
    errors.push({ text: "Passwords do not match." });
  }

  if (password.length < 4) {
    errors.push({ text: "Passwords must be at least 4 characters." });
  }

  if (errors.length > 0) {
    return res.render("auth/signup", {
      errors,
      name,
      email,
      password,
      confirm_password,
    });
  }

  // Look for email coincidence
  const userFound = await User.findOne({ email: email });
  if (userFound) {
    req.flash("error_msg", "The Email is already in use.");
    return res.redirect("/auth/signup");
  }

  // Saving a New User
  const newUser = new User({ name, email, password });
  newUser.password = await newUser.encryptPassword(password);
```

```
  await newUser.save();
  req.flash("success_msg", "You are registered.");
  res.redirect("/auth/signin");
};

export const renderSigninForm = (req, res) => res.render("auth/signin");

export const signin = passport.authenticate("local", {
  successRedirect: "/notes",
  failureRedirect: "/auth/signin",
  failureFlash: true,
});

export const logout = async (req, res, next) => {
  await req.logout((err) => {
    if (err) return next(err);
    req.flash("success_msg", "You are logged out now.");
    res.redirect("/auth/signin");
  });
};
```

src\controllers\notes.controller.js

```
import Note from "../models/Note.js";

export const renderNoteForm = (req, res) => res.render("notes/new-note");

export const createNewNote = async (req, res) => {
  const { title, description } = req.body;
  const errors = [];
  if (!title) {
    errors.push({ text: "Please Write a Title." });
  }
  if (!description) {
    errors.push({ text: "Please Write a Description" });
  }
  if (errors.length > 0)
    return res.render("notes/new-note", {
      errors,
      title,
      description,
    });
```

```
  const newNote = new Note({ title, description });
  newNote.user = req.user.id;
  await newNote.save();
  req.flash("success_msg", "Note Added Successfully");
  res.redirect("/notes");
};

export const renderNotes = async (req, res) => {
  const notes = await Note.find({ user: req.user.id })
    .sort({ date: "desc" })
    .lean();
  res.render("notes/all-notes", { notes });
};

export const renderEditForm = async (req, res) => {
  const note = await Note.findById(req.params.id).lean();
  if (note.user != req.user.id) {
    req.flash("error_msg", "Not Authorized");
    return res.redirect("/notes");
  }
  res.render("notes/edit-note", { note });
};

export const updateNote = async (req, res) => {
  const { title, description } = req.body;
  await Note.findByIdAndUpdate(req.params.id, { title, description });
  req.flash("success_msg", "Note Updated Successfully");
  res.redirect("/notes");
};

export const deleteNote = async (req, res) => {
  await Note.findByIdAndDelete(req.params.id);
  req.flash("success_msg", "Note Deleted Successfully");
  res.redirect("/notes");
};
```

src\config\passport.js

```
import passport from "passport";
import { Strategy as LocalStrategy } from "passport-local";

import User from "../models/User.js";
```

```
passport.use(
  new LocalStrategy(
    {
      usernameField: "email",
    },
    async (email, password, done) => {
      // Match Email's User
      const user = await User.findOne({ email: email });

      if (!user) {
        return done(null, false, { message: "Not User found." });
      }

      // Match Password's User
      const isMatch = await user.matchPassword(password);
      if (!isMatch)
        return done(null, false, { message: "Incorrect Password." });

      return done(null, user);
    }
  )
);

passport.serializeUser((user, done) => {
  done(null, user.id);
});

passport.deserializeUser((id, done) => {
  User.findById(id, (err, user) => {
    done(err, user);
  });
});
```

Dockerfile

```
FROM node:latest

# Set the working directory
WORKDIR /app

# Copy package.json and package-lock.json
COPY package*.json ./

# Install dependencies
RUN npm install

# Copy the rest of the application
COPY . .

# Copy .env file
COPY .env ./

# Expose the port
EXPOSE 80

# Start the application
CMD ["npm", "start"]
```

.github\workflows\ec2-deploy.yml

```
name: Deploy Nodejs Application

on:
  push:
    branches:
      - main

jobs:
  build:
    runs-on: ubuntu-latest
    steps:
      - name: Checkout source
        uses: actions/checkout@v4
      - name: Create .env file
        run: echo "MONGO_PASSWORD=${{ secrets.MONGO_PASSWORD }}" >> .env
```

```
      - name: Login to docker hub
        run: docker login -u ${{ secrets.DOCKER_USERNAME }} -p ${{
secrets.DOCKER_PASSWORD }}
      - name: Build docker image
        run: docker build -t rahulvikhe/nodejs-notes-app .
      - name: Publish image to docker hub
        run: docker push rahulvikhe/nodejs-notes-app:latest

  deploy:
    needs: build
    runs-on: self-hosted
    steps:
      - name: Pull image from docker hub
        run: docker pull rahulvikhe/nodejs-notes-app:latest
      - name: Delete old container
        run: docker rm -f nodejs-app-container
      - name: Run docker container
        run: docker run -d -p 80:80 --name nodejs-app-container rahulvikhe/nodejs-notes-app
```

Note: Visit https://github.com/rahulvikhe/nodejs-notes-app for full source code

3.8 Table specification:

Field	Data type	Description
id	Long	Unique identifier for the note
Title	String	Title or heading of the note.
Content	String	Text content of the note.
Created_Date:	String	Date and time when the note was created.
Last_Updated_Date	String	Date and time when the note was last updated.
description	Text	Brief description of the note
username	String	user's name.

Field: Represents the attribute or column name in the Book entity.
Data Type: Specifies the data type of each attribute (e.g., String, Long).
Description: Provides a brief description of the purpose or meaning of each attribute.

3.9 Drawbacks & Limitations

Lack of Collaboration Features: Users may miss the ability to collaborate with others on notes, such as sharing and real-time editing, which can be useful for group projects or shared tasks.

Limited Organization Options: App doesn't include features like labels, tags, or color coding, users might find it challenging to organize and categorize their notes effectively, especially as the number of notes grows.

No Reminders or Alerts: Without reminders or alerts functionality, users might struggle to stay on top of important tasks or deadlines associated with their notes.

No Integration with Other Apps or Services: Users may miss the ability to integrate their notes with other apps or services they use regularly, such as Google Drive, Evernote, or task management apps, which can streamline workflows and enhance productivity.

No Archiving or Trash Functionality: Without the ability to archive or move notes to a trash folder, users may find it challenging to manage and declutter their note collections effectively.

Lack of Customization Options: App doesn't offer customization options for fonts, themes, or layouts, users may feel limited in personalizing their note-taking experience to suit their preferences.

3.10 Proposed Enhancements:

Advanced Organization Options: We will introduce a variety of organizational tools such as labels, tags, categories, and folders, giving users more flexibility in organizing and structuring their notes. Users can create custom labels and categories, making it easier to find and manage notes based on their specific needs.

Reminder and Notification System: Our app will feature a robust reminder and notification system, allowing users to set reminders for individual notes with options for time-based alerts, location-based notifications, and recurring reminders. Users will receive timely reminders via push notifications or email, helping them stay organized and on top of their tasks.

Powerful Search Functionality: We will enhance the search functionality with advanced search filters, enabling users to quickly find notes using keywords, tags, labels, date ranges, and more. Users can refine their search results and navigate through their notes effortlessly, saving time and increasing productivity.

Enhanced Security Features: We will prioritize the security and privacy of our users' data by implementing advanced security features such as end-to-end encryption, biometric authentication, passcode lock, and remote wipe functionality. These measures will safeguard users' sensitive information and provide peace of mind.

Archiving and Trash Functionality: Our app will include options for archiving notes to declutter the main interface while still retaining access to archived content. Deleted notes will be moved to a trash or recycle bin, where they can be restored or permanently deleted as needed.

Customization Options: We will offer users a wide range of customization options to personalize their note-taking experience, including themes, fonts, colors, and layout preferences. Users can customize the appearance of individual notes or entire notebooks to suit their preferences and style.

References

Express.js Official Documentation: https://expressjs.com/

Mongoose Official Documentation: https://mongoosejs.com/docs/guide.html

Passport.js Official Documentation: http://www.passportjs.org/docs/

Dockerfile Reference: https://docs.docker.com/reference/dockerfile/

AWS EC2 Documentation: https://docs.aws.amazon.com/ec2/

Github Runners: https://docs.github.com/en/actions/hosting-your-own-runners