Task Management Application

Sihan Ma, Yuexuan Lu

# Objective and Functionality

## Objective

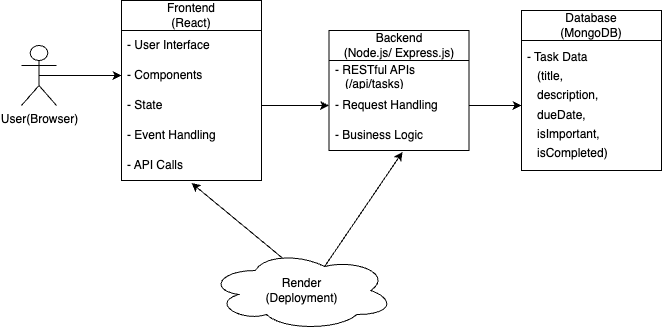
The primary objective of this project was to develop a full-stack web application that enables users to efficiently manage their tasks. Our application aims to provide a user-friendly interface for creating, retrieving, updating, and deleting tasks, along with features for filtering and marking tasks as complete or important. We aimed to create a responsive application accessible on both desktop and mobile devices.

## Functionality

The application provides the following core functionalities:

* **Add New Tasks:** Users can add a task by clicking the add icon on the top right of the screen. A modal will appear directing the users to enter details of a new task (title, description, deadline, check task important, and check task completed).
* **List All Tasks:** When visiting the home page, users can see a list of existing tasks from the database.
* **Update A Task:** Users can click the edit icon on the bottom of a task. A modal will appear directing the users to edit the details of the current task and update.
* **Delete A Task:** Users can delete a task by clicking the delete icon on the bottom of the task. A modal will ask the user to confirm this action.
* **View A Task:** The user can click on a task to see its details. When the task gets too long, There will be some ... and a view details part where clicking on it allows us to expand to see the full details to prevent overflow.
* **Filter Tasks by Category:** Users can filter the tasks that appear on the home page with a list of filters on the sidebar on the left.
* **Toggle Task Completeness:** Users can click on the Complete/Incomplete button on the page to toggle the completeness state of the task. (A complete task will be marked incomplete, and vice versa).
* **Responsive View:** The application is responsive and will adjust the UI for a better user experience on mobile and smaller screens. The number of cards shown and the position of the sidebar moves accordingly.
* **Loading State:** The main page displays a spinner when the page is loading for a better user experience.

# Architecture Diagram



* The application follows a MERN (MongoDB, Express.js, React, Node.js) architecture.
* The Frontend (React) handles the user interface and user interactions. It communicates with the backend via RESTful APIs.
* The API (Node.js/Express.js) serves as the backend, managing data requests and business logic. It interacts with the database.
* The Database (MongoDB) stores the application's data.
* Render is used for deploying the application.
* Data flows from the database to the backend, then to the frontend for display, and vice-versa when users create, update, or delete tasks.

# Tools and Frameworks Used

## Frontend:

* Vite: Build tool for the frontend.
* React: JavaScript library for building the user interface.
* React Bootstrap: UI framework for styling and layout.

## Backend:

* Node.js: JavaScript runtime environment for the server-side.
* Express.js: Web application framework for Node.js.

## Database:

* MongoDB: NoSQL database.
* Mongoose: MongoDB object modeling tool.

## Deployment:

* Render: Cloud hosting platform.

## Other:

* Git: Version control system.

# Roles in Team

## Sihan Ma (@Masihan317):

* Vite setup, main UI (responsive) with React Bootstrap.
* Express Server development, REST API endpoints, MongoDB integration.
* Add task, list all tasks, and delete tasks functionality.
* Documentation.
* Favicon update.
* Loading state of main page.
* Application deployment on Render.

## Yuexuan Lu (@noiorhyun):

* Edit tasks functionality.
* Task filtering on home page.
* Implementing Click button to switch status of tasks between completed and incomplete.
* View Detail Card implementation.
* Handled Overflow of Tasks.
* Worked on most of the report (Objective, Architecture Diagram, etc.)
* Resort the features screenshots

# Challenges, Learnings, and Future improvements

## Challenges

1. One challenge I faced was using React Bootstrap properly. When I finished importing the library, I noticed that while the functionality was there, the Bootstrap styles were not properly applied. I had to read the documentation carefully to find that I needed to add an import css statement in my application. Without it, none of the Bootstrap CSS styles and variables will render properly. This also happened when I first deployed the app on Render. None of the styles were loading correctly and I only had a skeletal app that had the functionality but no style, which worried me but then after a while the styles just loaded and everything was fine.
2. Deployment turned out to be a big challenge for me because my original setup had a separate frontend and backend running, with the Vite config file redirecting the frontend requests to backend. This was not ideal because I then had to deploy the frontend and backend separately, which I didn’t like. Instead, I did some research to change some of the code so that I can now deploy my app from the root directory. I had to build the frontend first using Vite and then run everything from the backend. As I was doing those, I ran into another issue with the versioning of ExpressJS where it just kept on failing and throwing some sort of async error. I had to look stuff up on StackOverflow to find that it was because Express 5 had some built-in bugs. After I downgraded to Express 4 everything was fixed but I would have never guessed that was the reason.
3. One challenge for me is to fix the task details overflow. Since the task card is fixed size. Our goal was to display a reasonable amount of content, indicate when more was available, and maintain a clean layout. First I used CSS truncation, and a combination of useRef, useState, and useEffect was employed to detect precisely when the text was truncated, allowing for the conditional display of a "View details" indicator. Finally, adjusting marginBottom within the fixed-height cards.
4. The task card needed to be clickable to open a "View Details" modal, but also contained interactive elements (toggle, Edit, Delete) that should not trigger this action. The challenge was to differentiate between a card click and a button click within the card. To solve this, I added an onClick handler to open the modal, and event bubbling was identified as the issue, where button clicks were incorrectly triggering the card's click handler. The solution involved using a wrapper function, handleButtonClick, that calls event.stopPropagation() to prevent button click events from propagating up to the card. The original button handlers were then called within this handleButtonClick function to execute their intended actions without also opening the modal.

## Learnings

1. As I mentioned in the first bullet point in the challenges section, I learned the importance of reading documentation. A lot of the time, the answer is already somewhere written in the docs, and we just need to read to get the answer we want.
2. I learned how to build a frontend app and write npm scripts. Vite is such a great tool for bundling things and managing a lot of the overhead in coding. I also learned how to not panic in situations where things don’t work. StackOverflow and Github Issues are great sources where we can find information. It’s very likely that someone ran into similar issues in the past and we can find solutions to things by doing research and experimenting. (Also learned the importance of compatibility which I never really considered a factor that can actually break an app.)
3. I learned to use the filter() method to pick out only the tasks I needed. I figured out how to create rules for this, and how to use more than one rule at a time. I learned how to handle long pieces of text. I used the substring() method to make the text shorter at first. Then, I used React's useState to switch between showing the short text and showing the whole text.
4. I learned the importance of deciding where to manage the state, specifically the currentFilter state. I found that managing this state in the Layout component, rather than the Sidebar, allowed for a more unidirectional data flow, making the application's logic easier to understand and maintain.

## Future Improvements

1. We can add a user login system in the future. It’s better if we have a login system where people don’t share the same task board and can have their own task boards. This will introduce new libraries we can use such as bcrypt for password hashing because we don’t want to store the passwords directly in our database. We can use jwt tokens for cookies and local storage now that we have users with login sessions. We can probably also introduce state management tools like Zustand or React Redux as we now have a global user and tasks we want to share across our app.
2. There are plenty of UI improvements we can make as well. We can show skeleton cards on load. It would also be nice if there’s a dark and a light mode where we can toggle between the 2. We can also add toast messages after our database operations like adding a task to indicate to the user whether the action succeeds or not.
3. One future improvement is to integrate the application with a calendar interface. This would involve displaying task due dates on a calendar, allowing users to visualize their deadlines and schedule their work more effectively. Or this feature would also enable users to create new tasks directly from the calendar.
4. Another potential enhancement is to provide calendar reminders or notifications for upcoming task deadlines. This could involve setting up a system to alert users as deadlines approach, either through in-app notifications, email, or integration with external calendar applications. Such a feature would help users stay on track and avoid missing important deadlines.