This application allows users to create, read, update, and delete (CRUD) tasks or to-do items. The front-end of the application is built using React, which allows for a fast and responsive user interface. The back-end is built using Node.js and Express.js, which provides a flexible and scalable server-side framework. MongoDB is used as the database to store task data. Users can create new tasks, edit or delete tasks, and view all tasks in a list.

These are the components that were created with the application:

**Login component:**

This is a functional component in a React application called "Login" that provides a user interface for logging in to the application.

The first line of the code imports the React and useState libraries from the "react" package. useState is a hook that allows functional components to have state.

The next few lines import some additional libraries from "react-router-dom" and "react-toastify". These are third-party libraries used to create routes and display notifications respectively.

The Login component contains a form with two input fields for email and password, and a "Sign In" button. The values of these fields are stored in the component's state using the useState hook.

When the user clicks the "Sign In" button, the handleLogin function is called. This function first checks if the email and password fields are not empty. If either field is empty, a warning notification is displayed using the toast function from the react-toastify library. If both fields are filled, an asynchronous function called login is called using the fetch API to make a POST request to a local server for login authentication. The response from the server is then checked and if successful, a success notification is displayed using the toast function, and the user is redirected to the home page using the navigate function from "react-router-dom".

Finally, the component returns the UI elements wrapped in a div with the class name "flex items-center justify-center h-screen". The UI displays a two-column layout, one column for the sign-in form and another column with a welcome message and an image. Additionally, the component renders a ToastContainer from the react-toastify library at the bottom of the screen, which will display any notifications. The component is exported as the default export so it can be used in other parts of the application.

**Sign up component:**

This is a React component for a Sign-Up form. It allows a user to enter their name, email, password, and confirm password, and then submit the form. The component uses useState hook to manage state variables for name, email, password, and confirm password fields. It also uses the useNavigate hook from React Router to navigate to the home page after successful sign-up.

The handleSignUp function is triggered when the user clicks the "Sign Up" button. It checks whether all the fields are filled in and whether the password and confirm password fields match. If all the fields are filled in and the passwords match, it makes a POST request to the server with the user's name, email,

and password. If the request is successful, the user is shown a success message, and then navigated to the home page. If there is an error in the request, the user is shown an error message.

The form is styled using Tailwind CSS, and the component also uses the react-toastify library to show success and error messages.

**Task component:**

This is a React component called Tasks that displays a table of tasks fetched from a backend server. The component uses the useState hook to manage the component state, and the useEffect hook to fetch the data from the server when the component mounts.

The getTask function uses fetch to make a GET request to the server endpoint http://localhost:3400/task to retrieve the list of tasks. It then updates the component state with the retrieved data using the setData function.

The handleDeleteTask function makes a DELETE request to the server endpoint http://localhost:3400/task/${$id} to delete a task with the given $id. If the request is successful, it displays a success message using the toast function from the react-toastify package.

The component renders a table with columns for the task title, description,and tools. It maps through the data array using the map function to display a row for each task in the table. The Link component from the react-router-dom package is used to provide a link to edit each task, and an element is used to delete the task.

The ToastContainer component from the react-toastify package is used to display success messages when a task is deleted.

**Home Component:**

This is a React functional component named "Home". It renders a simple dashboard page for an admin, with a navigation bar at the top and two cards that link to other pages. The navigation bar has a title "Admin Dashboard" and three links, "Home", "Settings", and "Logout". The two cards have a title and a brief description of their function and a link to a related page. The first card is titled "Tasks" and links to a page that displays all the tasks, and the second card is titled "Add Task" and links to a page where the admin can add a new task. The code uses tailwindcss classes for styling. The navigation links are anchor tags, but the two cards use React Router's Link component to link to other pages within the React app. The component is exported as the default export.

**Server Component:**

This is a JavaScript code that uses the Express.js framework to build a RESTful API for a web application.

Here are some of the main features of the code:

It imports several dependencies using import statements, including express, cors, body-parser, and mongoose. These dependencies are used to set up the server and interact with a MongoDB database.

It defines a new instance of express() called app.

It uses the cors() middleware to allow cross-origin requests from the frontend.

It connects to a MongoDB database using the mongoose library.

It defines several API endpoints using app.get(), app.post(), app.put(), and app.delete(). These endpoints handle HTTP requests from the frontend, such as creating a new user, logging in a user, adding a new task, getting all tasks, getting a single task by ID, updating a task, and deleting a task.

It listens for requests on a specified port using app.listen().

Here is a breakdown of the code:

The first few lines of the code are import statements. They import various libraries and modules that are required to run the application. These libraries include:

express - a popular Node.js web framework that simplifies the process of building web applications.

cors - a middleware that enables cross-origin resource sharing.

body-parser - a middleware that parses the incoming request bodies in a middleware before the handlers.

mongoose - an Object Data Modeling (ODM) library that provides a simple way to interact with MongoDB databases from Node.js.

userSchema and taskSchema - custom-defined schemas that define the structure of the data models.

const app = express(); creates an instance of the Express application.

app.use(cors()); adds the cors middleware to the application. This enables the application to make cross-origin HTTP requests.

app.use(bodyParser.json()); adds the body-parser middleware to the application. This parses the incoming request body in JSON format.

mongoose.connect(...) connects the application to a MongoDB database. The connection string contains the username and password for the database, as well as the name of the cluster where the database resides.

app.get("/", function (req, res) {...}); defines a handler function for the root route of the application. When a client makes a GET request to the root route, the function sends the string "hello world" as the response.

app.post("/signup", (req, res) => {...}); defines a handler function for the "/signup" route. When a client makes a POST request to this route, the function checks if a user with the specified email address already exists in the database. If the user exists, it sends a response indicating that the user already exists. Otherwise, it creates a new user and saves it to the database.

app.post("/login", (req, res) => {...}); defines a handler function for the "/login" route. When a client makes a POST request to this route, the function checks if a user with the specified email address and password exists in the database. If the user exists and the password is correct, it sends a response indicating that the login was successful. Otherwise, it sends a response indicating that the email or password is incorrect.

app.post("/task", (req, res) => {...}); defines a handler function for the "/task" route. When a client makes a POST request to this route, the function checks if a task with the specified title and description already exists in the database. If the task exists, it sends a response indicating that the task already exists. Otherwise, it creates a new task and saves it to the database.

app.get("/task", function (req, res) {...}); defines a handler function for the "/task" route. When a client makes a GET request to this route, the function retrieves all tasks from the database and sends them as the response.

app.get("/task/:id", (req, res) => {...}); defines a handler function for the "/task/:id" route. When a client makes a GET request to this route with a task ID in the URL, the function retrieves the task with the specified ID from the database and sends it as the response.

app.put("/task/:id", function (req, res) {...}); defines a handler function for the "/task/:id" route. When a client makes a put request to update the task