

**COMS 319**  
**Final Project Documentation**  
**TaskPro**

**Date:** 10th May 2023

**Student 1 Name:** Aryan Rao

**Student 2 Name:** Nishi Kant

**Professor name:**

Abraham Netzahualcoy Aldaco Gastelum

# Index

1. Project Description.....	page 3
2. The diagram to describe overall functioning of the Software.....	page 4
3. Graphical of the architecture.....	page 5
4. Main code explanation.....	page 6
5. View Explanation.....	page 9
6. Manual of Installation.....	page 13
7. Copy of the code, JSON files.....	page 16

# Project Description:

TaskPro is a MERN (MongoDB, Express.js, React, Node.js) application that allows users to manage tasks collaboratively within groups. It provides features for user authentication, task management, and group messaging.

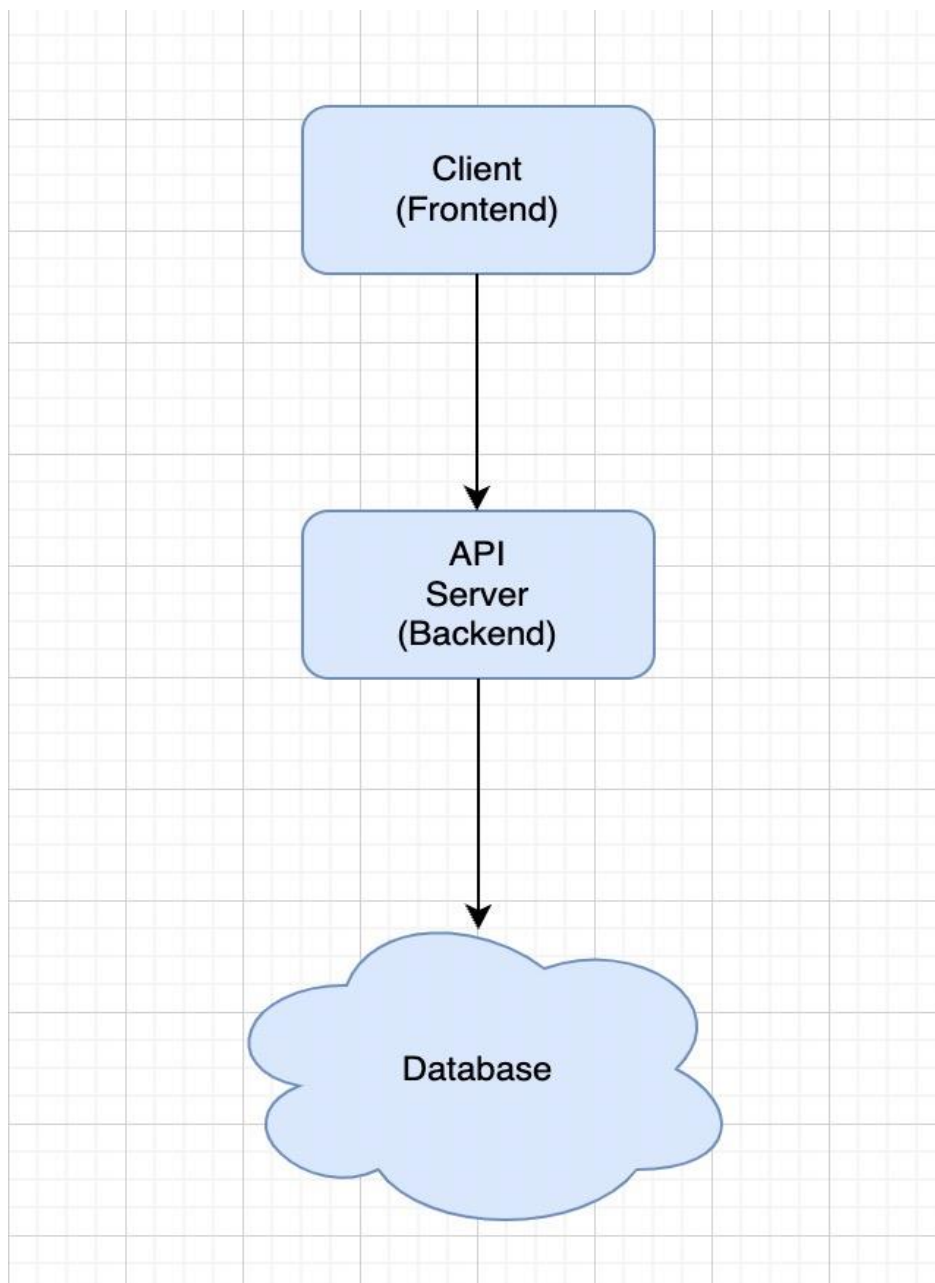
The application enables users to register an account or login if they already have one. This authentication system ensures that each user has a unique identity within the application.

Once logged in, users can create tasks within their assigned groups. Each task consists of a description and a status indicating whether it is completed or not. Users can create new tasks, mark tasks as completed, and delete tasks as needed. This allows for efficient task management and tracking of progress within the group.

In addition to task management, the application also facilitates group communication through messaging. Users can send messages to the group, allowing for collaboration, discussion, and coordination among group members. The messages are stored and displayed in real-time, allowing users to stay updated on group discussions.

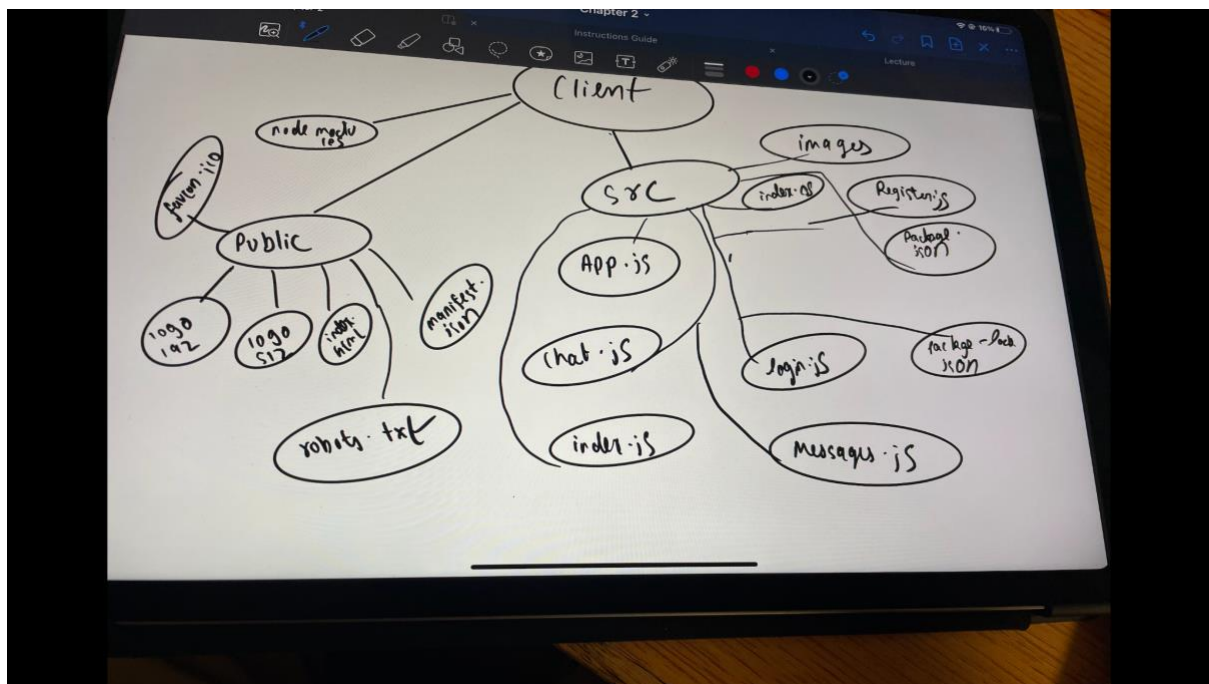
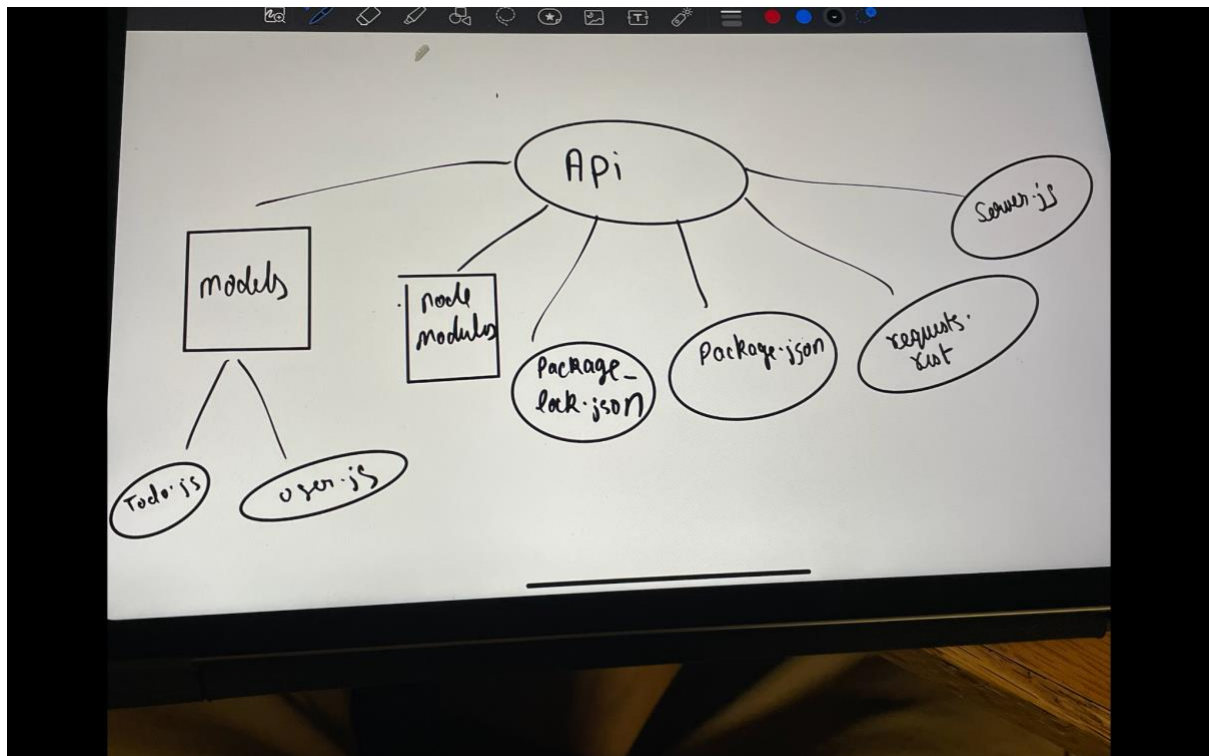
The collaborative tasks keeper leverages the MERN stack to provide a seamless user experience. The frontend is built using React, which provides a responsive and interactive user interface. The backend is implemented with Node.js and Express.js, handling requests from the client and managing the business logic. Data is stored in a MongoDB database, ensuring persistence and scalability. Overall, the collaborative tasks keeper simplifies and streamlines task management within groups by providing a user-friendly interface, real-time updates, and collaborative messaging. It enhances productivity and coordination among group members, making it easier to organize and complete tasks in a collaborative setting.

## Diagram:



1. The client, which represents the frontend application (e.g., a web browser), interacts with the API server to send requests and receive responses.
2. The API server handles incoming requests from the client and performs the necessary operations to process those requests.
3. The API server communicates with the database to store and retrieve data. In your case, you're using MongoDB as the database.
4. The database stores the data required for your application. It can be queried and modified by the API server based on the incoming requests.
5. The API server sends back responses to the client, which can include data fetched from the database or status messages indicating the success or failure of the requested operations.

# Graphical Architecture:



# Main Code Explanation:

## App.js:

This is a javascript file named APP.js. This basically renders the user interface of our application called TaskPro. It imports components from other files mainly Login, register and Todos. It also imports dependencies such as router, route, routes.

In the App function, the useState hook is used to declare a state variable named aboutViewActive and its corresponding update function setAboutViewActive. This state variable is used to keep track of whether the "About" view is currently active or not.

A function named toggleAboutView is defined which toggles the aboutViewActive state variable. The function is passed as an event handler to a button in the UI, which toggles the view when clicked.

Another function named renderAboutView is defined which returns a JSX element containing information about the application and its creators. This function is used to render the "About" view in the UI.

Inside the return statement, the Router component is used to set up the routes for the application. The routes component is used to define the different paths and their corresponding components, such as Register, Login, and Todos.

There is also a div element with a button that toggles the "About" view, and a conditional rendering of the renderAboutView function using the aboutViewActive state variable.

Finally, the App component is exported as the default export of this module, which can be used by other components or files.

## Login.js

This code defines a React component called Login which displays a login form for the user to enter their username and password. The component imports useState and axios from external modules, as well as Link and useNavigate from react-router-dom.

When the user submits the login form, the handleLogin function is called. This function first prevents the default form submission behavior using e.preventDefault(). It then sends a request to the server using axios.post with the user's entered username and password .

If the login attempt is successful, the server sends back a response containing a message field, which is logged to the console using console.log(response.data.message).

Finally, the navigate object is used to redirect the user to the Todos page using navigate('/todos').

The Login component also renders a form with two input fields for the user to enter their username and password, respectively.

## Messages.js

This is a React component called "Messages" which displays a list of messages fetched from an API endpoint and allows users to add new messages.

The component initializes two state variables: `messages` and `newMessage`. `messages` is an empty array that will store the messages fetched from the API. `newMessage` is an empty string that will hold the value of the new message that the user types into an input field.

The `useEffect` hook is used to fetch the initial list of messages from the API when the component is mounted. The `fetchMessages` function sends a GET request to the `/messages` endpoint and updates the `messages` state variable with the response data.

The `sendMessage` function sends a POST request to the same `/messages` endpoint to add a new message to the list. It uses the `newMessage` state variable as the message text in the request body. If the request is successful, the `newMessage` state variable is cleared and the list of messages is refetched using `fetchMessages`.

The return statement renders the list of messages using the `map` function. It also renders an input field and a "Send" button that triggers the `sendMessage` function when clicked.

## Register.js

This is a code for a React component that implements a registration form. The component uses the `useState` hook to manage the state of the input fields for the username and password. The `useNavigate` hook is used from the `react-router-dom` package to handle client-side navigation.

When the `Register` component is rendered, it displays a form that has two input fields for the username and password. When the user enters a value in either field and submits the form, the `handleRegister` function is called.

The `handleRegister` function is an asynchronous function that uses the `axios` library to make a POST request to a registration endpoint at `http://localhost:3001/register`. The request payload includes the username and password entered by the user. If the registration is successful, the server responds with a success message that is logged to the console, and the user is redirected to the login page using the `navigate` function.

If there is an error in the registration process, the error is logged to the console.

## Todos.js

This code defines a React component called `Todos`. It imports `useEffect` and `useState` from the `react` module and the `Messages` component from a local file.

The component defines several states using the `useState` hook:

- `todos`: an array to store the to-do list items

- `popupActive`: a boolean to determine whether or not the add to-do popup is visible
- `newTodo`: a string to store the text of the new to-do item
- `filter`: a string to store the current filter for displaying to-dos (all, complete, or incomplete)
- `sort`: a string to store the current sort order for displaying to-dos (oldest or newest)

The component then defines an effect using the `useEffect` hook to fetch the to-dos from a local API when the component is mounted. The API URL is defined as a constant variable `api_base`.

The component defines several functions:

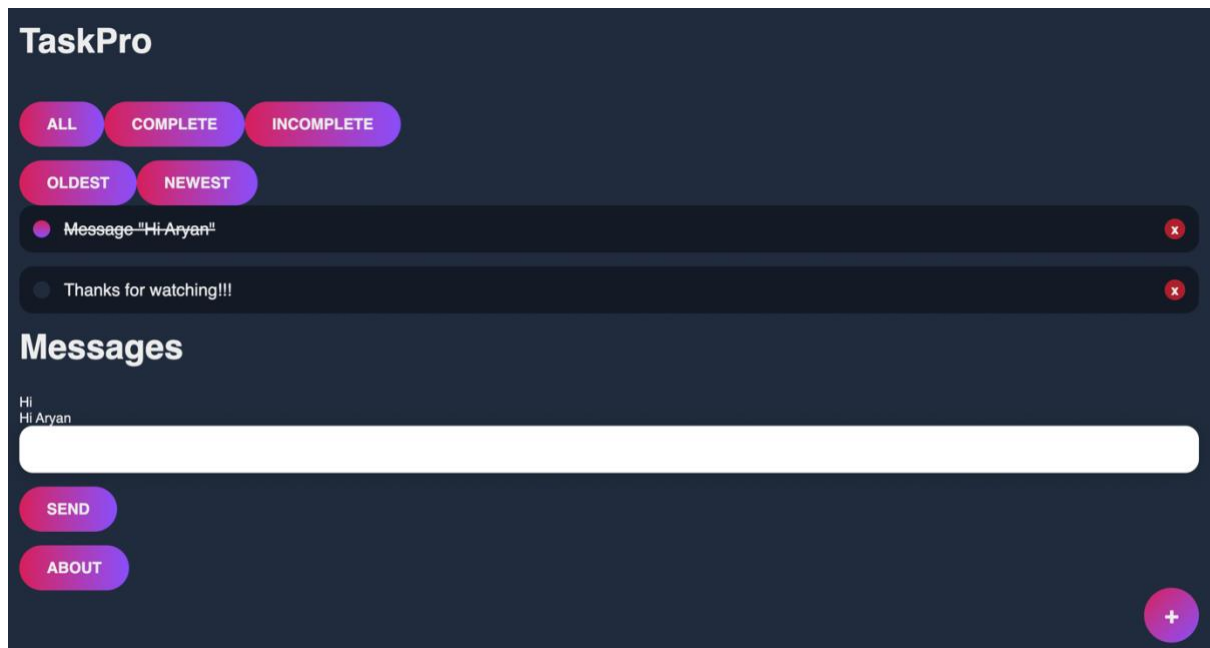
- `GetTodos`: a function to fetch the to-dos from the API and set them to the `todos` state
- `completeTodo`: a function to mark a to-do item as complete in the database and update the `todos` state to reflect the change
- `addTodo`: a function to add a new to-do item to the database and update the `todos` state to include the new item
- `deleteTodo`: a function to delete a to-do item from the database and update the `todos` state to remove the deleted item
- `renderFilterButtons`: a function to render buttons for filtering to-dos by all, complete, or incomplete
- `toggleFilter`: a function to update the `filter` state based on the selected filter button
- `toggleSort`: a function to update the `sort` state based on the selected sort button
- `renderSortButtons`: a function to render buttons for sorting to-dos by oldest or newest
- `renderTodos`: a function to render the to-dos based on the current filter and sort states
- `renderAddPopup`: a function to render the add to-do popup with an input field and create task button

Finally, the component returns JSX code to render the filter and sort buttons, the to-dos, the add to-do button, and the add to-do popup when active. The component also renders the `Messages` component. The JSX code calls the various functions defined earlier to render the dynamic content.



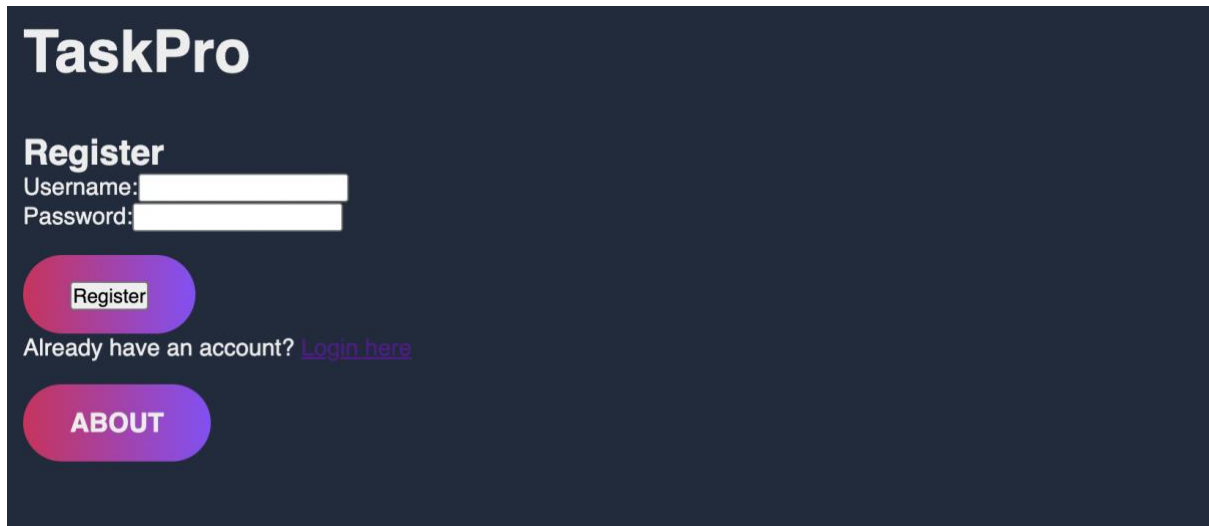
# View Explanation:

## Main Page:



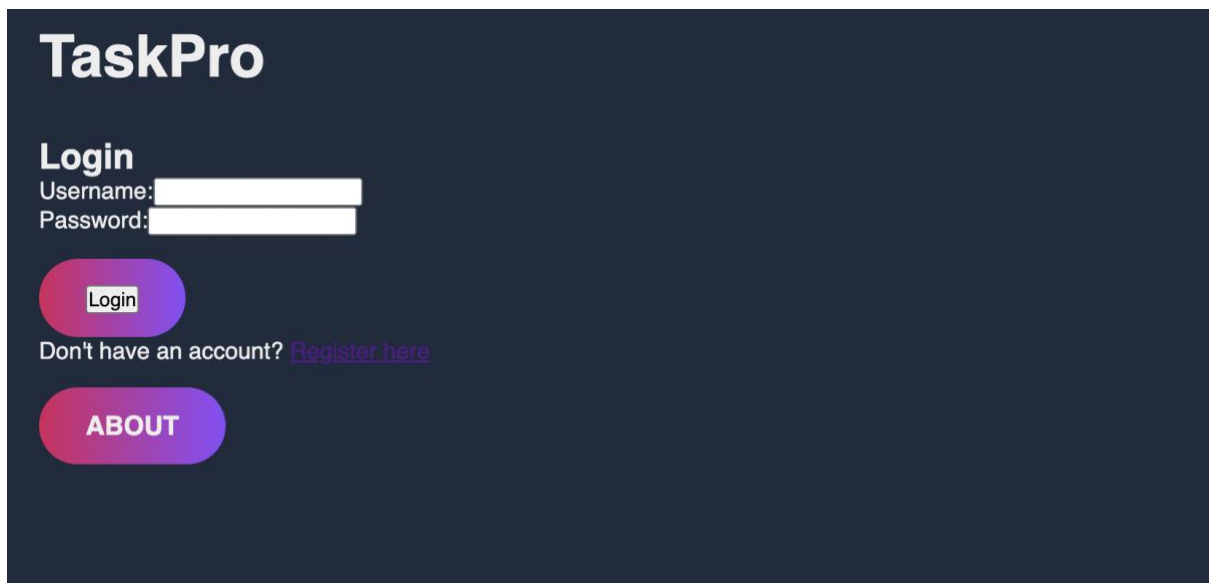
This is how our Main page looks. User can see all his tasks, make more, delete or mark them as complete. “Completed” shows the tasks he has completed and similarly not completed. There’s also “Oldest” and “Newest” which lets the users see the latest and oldest tasks. The cross button in red deletes a task while simply clicking on a task makes it complete. The plus button in bottom right lets a user add a tasks. There is messages section which displays all the texts in a group chat and lets a user send messages. The about pahe shows the details of the creators.

## Register:

The image shows the 'TaskPro' registration page. It has a dark blue background. At the top left is the 'TaskPro' logo in white. Below it is the heading 'Register' in white. There are two white input fields for 'Username:' and 'Password:'. Below these is a rounded purple button with the text 'Register' in white. Underneath the button is the text 'Already have an account?' followed by a blue link 'Login here'. At the bottom is another rounded purple button with the text 'ABOUT' in white.

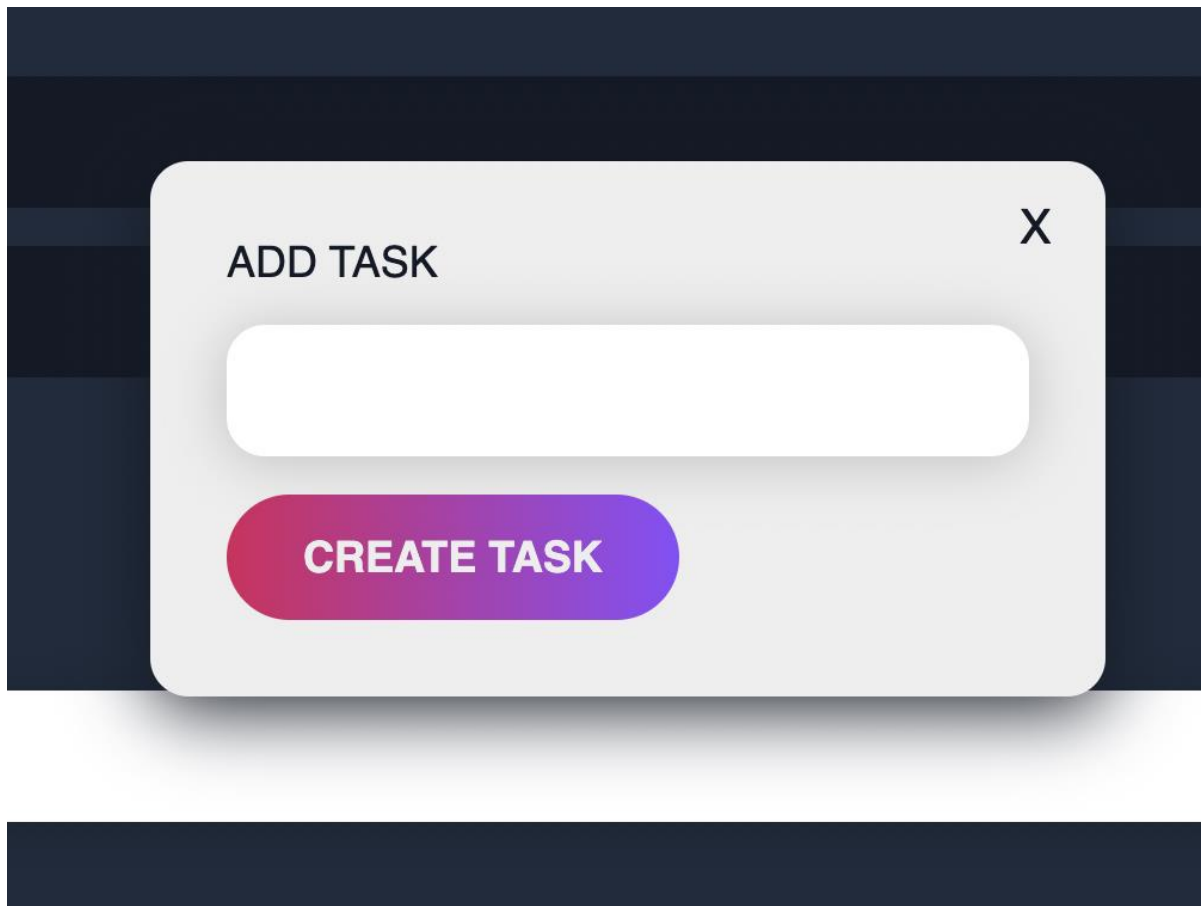
This is how our registration page looks. Users can enter a “username” and “password” and are then directed to the login page. If they already have an account, there’s an option to simply login.

## Login:

The image shows the 'TaskPro' login page. It has a dark blue background. At the top left is the 'TaskPro' logo in white. Below it is the heading 'Login' in white. There are two white input fields for 'Username:' and 'Password:'. Below these is a rounded purple button with the text 'Login' in white. Underneath the button is the text 'Don't have an account?' followed by a blue link 'Register here'. At the bottom is another rounded purple button with the text 'ABOUT' in white.

This is our login page which upon successful authentication redirects the users to the tasks and basically our application. If a user doesn’t have an account, he can simply register.

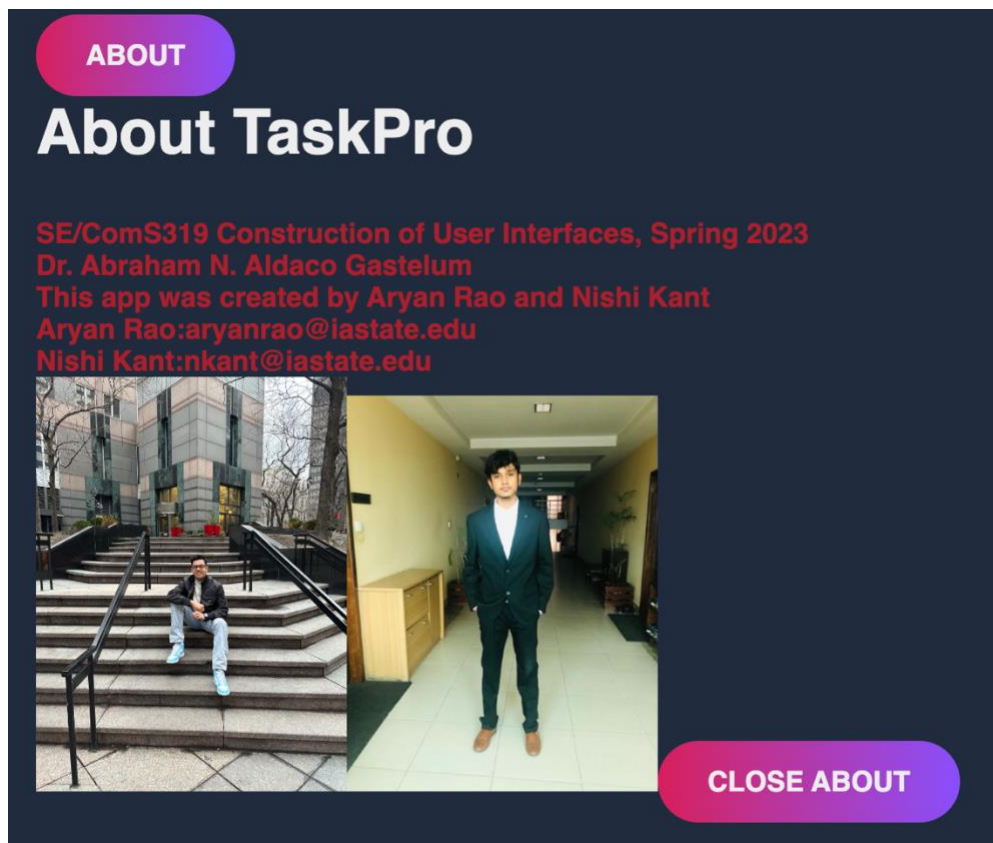
## Add Task:



The image shows a mobile application interface with a dark blue background. A white dialog box with rounded corners is centered on the screen. The dialog box has the text "ADD TASK" in the top left and a close button "X" in the top right. Below the title is a white text input field. At the bottom of the dialog box is a button with a purple-to-pink gradient and the text "CREATE TASK".

This is our ADD Task view which lets a user create a task. A user enters the text and clicking “Create Task” will make a task.

## About:



This is our “About” view which shows the details of the creators, professor and the images.

# Manual of Installation:

React:

Installing React on Mac

## 1. Install Node.js

- First, you need to install Node.js on your Mac. You can download the installer from the official Node.js website <https://nodejs.org/en/download/>.

## 2. Install Create React App

- Open your terminal and run the following command to install Create React App: `npx create-react-app my-app`
- This will create a new React application in a folder called my-app.

## 3. Run the React app

- Once the installation is complete, navigate to the my-app directory by running the command `cd my-app`
- Run the command `npm start` to start the development server.
- You can now view the React app in your browser at `http://localhost:3000`

Installing React on Windows

## 1. Install Node.js

- First, you need to install Node.js on your Windows machine. You can download the installer from the official Node.js website <https://nodejs.org/en/download/>.

## 2. Install Create React App

- Open your Command Prompt or PowerShell and run the following command to install Create React App: `npx create-react-app my-app`
- This will create a new React application in a folder called my-app.

## 3. Run the React app

- Once the installation is complete, navigate to the my-app directory by running the command `cd my-app`
- Run the command `npm start` to start the development server.

## Mongoose:

For Mac:

1. Open Terminal.
2. Navigate to your project folder using the `cd` command.
3. Run `npm init -y` to initialize a new Node.js project with default settings.
4. Run `npm install mongoose` to install the latest version of Mongoose.
5. Start using Mongoose in your Node.js project by requiring it in your code with `const mongoose = require('mongoose')`.

For Windows:

1. Open Command Prompt or PowerShell.
2. Navigate to your project folder using the `cd` command.
3. Run `npm init -y` to initialize a new Node.js project with default settings.
4. Run `npm install mongoose` to install the latest version of Mongoose.
5. Start using Mongoose in your Node.js project by requiring it in your code with `const mongoose = require('mongoose')`.

## Express:

For Mac:

1. Open the terminal.
2. Navigate to your project directory.
3. Run the following command to initialize npm package:

```
npm init -y
```

Run the following command to install Express as a dependency:

```
npm install express
```

For Windows:

1. Open the command prompt.
2. Navigate to your project directory.
3. Run the following command to initialize npm package:

```
npm init -y
```

Run the following command to install Express as a dependency

```
npm install express
```

## Nodejs:

For Mac:

1. Go to the official Node.js website <https://nodejs.org/en/>
2. Click on the "Download" button for the LTS version of Node.js, which is recommended for most users.
3. Once the download is complete, double-click the installer package to launch the Node.js installer.
4. Follow the instructions in the installer to complete the installation.
5. To check if Node.js is installed correctly, open the Terminal app and type `node -v`. The version number should be displayed in the Terminal.

For Windows:

1. Go to the official Node.js website <https://nodejs.org/en/>
2. Click on the "Download" button for the LTS version of Node.js, which is recommended for most users.
3. Once the download is complete, double-click the installer package to launch the Node.js installer.
4. Follow the instructions in the installer to complete the installation.
5. To check if Node.js is installed correctly, open the Command Prompt app and type `node -v`. The version number should be displayed in the Command Prompt.

## Nodemon:

1. Open your terminal or command prompt.
2. Install Node.js (if not already installed) by running the command:
  - For Mac: `brew install node`
  - For Windows: download the installer from the Node.js website and follow the installation instructions.
3. Once Node.js is installed, run the following command to install nodemon globally:

```
npm install -g nodemon
```

1. Wait for the installation to complete.
2. To verify that nodemon is installed correctly, run the command:

```
nodemon --version
```

# Code Images:

## Api-package.json

```
{
  "name": "todo-api",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "start": "nodemon server.js"
  },
  "author": "Aryan Rao",
  "license": "ISC",
  "dependencies": {
    "bcryptjs": "^2.4.3",
    "cors": "^2.8.5",
    "express": "^4.18.2",
    "mongoose": "^7.1.0",
    "socket.io": "^4.6.1",
    "socket.io-client": "^4.6.1"
  },
  "devDependencies": {
    "nodemon": "^2.0.22"
  }
}
```

## Server.js:

```
const express = require('express');
const mongoose = require('mongoose');
const cors = require('cors');
const path = require('path');
const bcrypt = require('bcryptjs');
const messages = [];

const app = express();

app.use(express.static(path.join(__dirname, 'client')));
app.use(express.json());
app.use(cors());

mongoose.connect('mongodb://127.0.0.1:27017/react-todo', {
```



```

    useNewUrlParser: true,
    useUnifiedTopology: true
  }).then(() => console.log("Connected to MongoDB")).catch(console.error);

// Models
const Todo = require('./models/Todo');
const User = require('./models/User');
const Message = mongoose.model('Message', new mongoose.Schema({
  message: String,
}));

app.get('/messages', async (req, res) => {
  try {
    const messages = await Message.find();
    res.json(messages);
  } catch (error) {
    console.error('Error fetching messages:', error);
    res.status(500).json({ message: 'Internal server error' });
  }
});

app.post('/messages', async (req, res) => {
  const { message } = req.body;
  try {
    const newMessage = new Message({ message });
    await newMessage.save();
    res.json({ success: true });
  } catch (error) {
    console.error('Error saving message:', error);
    res.status(500).json({ message: 'Internal server error' });
  }
});

// User Registration
app.post('/register', async (req, res) => {
  const { username, password } = req.body;
  try {
    // Check if the user already exists
    const existingUser = await User.findOne({ username });
    if (existingUser) {
      return res.status(400).json({ message: 'Username already exists' });
    }

    // Hash the password

```

```

    const hashedPassword = await bcrypt.hash(password, 10);

    // Create a new user
    const newUser = new User({ username, password: hashedPassword });
    await newUser.save();

    res.status(200).json({ message: 'Registration successful' });
  } catch (error) {
    console.error(error);
    res.status(500).json({ message: 'Internal server error' });
  }
});

// User Login
app.post('/login', async (req, res) => {
  const { username, password } = req.body;
  try {
    // Find the user
    const user = await User.findOne({ username });
    if (!user) {
      return res.status(400).json({ message: 'Invalid username or password'
});
    }

    // Compare passwords
    const isPasswordValid = await bcrypt.compare(password, user.password);
    if (!isPasswordValid) {
      return res.status(400).json({ message: 'Invalid username or password'
});
    }

    res.status(200).json({ message: 'Login successful' });
  } catch (error) {
    console.error(error);
    res.status(500).json({ message: 'Internal server error' });
  }
});

app.get('/todos', async (req, res) => {
  const todos = await Todo.find();

  res.json(todos);
});

```

```

app.post('/todo/new', (req, res) => {
  const todo = new Todo({
    text: req.body.text
  })

  todo.save();

  res.json(todo);
});

app.delete('/todo/delete/:id', async (req, res) => {
  const result = await Todo.findByIdAndDelete(req.params.id);

  res.json({ result });
});

app.get('/todo/complete/:id', async (req, res) => {
  const todo = await Todo.findById(req.params.id);

  todo.complete = !todo.complete;

  todo.save();

  res.json(todo);
});

app.put('/todo/update/:id', async (req, res) => {
  const todo = await Todo.findById(req.params.id);

  todo.text = req.body.text;

  todo.save();

  res.json(todo);
});

app.listen(3001);

```

## Client- manifest.json

```
{
  "short_name": "React App",
  "name": "Create React App Sample",
  "icons": [
    {
      "src": "favicon.ico",
      "sizes": "64x64 32x32 24x24 16x16",
      "type": "image/x-icon"
    },
    {
      "src": "logo192.png",
      "type": "image/png",
      "sizes": "192x192"
    },
    {
      "src": "logo512.png",
      "type": "image/png",
      "sizes": "512x512"
    }
  ],
  "start_url": ".",
  "display": "standalone",
  "theme_color": "#000000",
  "background_color": "#ffffff"
}
```

## App.js:

```
{
  "short_name": "React App",
  "name": "Create React App Sample",
  "icons": [
    {
      "src": "favicon.ico",
      "sizes": "64x64 32x32 24x24 16x16",
      "type": "image/x-icon"
    },
    {
      "src": "logo192.png",
      "type": "image/png",
      "sizes": "192x192"
    },
    {

```

```

        "src": "logo512.png",
        "type": "image/png",
        "sizes": "512x512"
    }
],
"start_url": ".",
"display": "standalone",
"theme_color": "#000000",
"background_color": "#ffffff"
}

```

## Index.css

```

:root {
  --primary: #D81E5B;
  --secondary: #8A4EFC;
  --light: #EEE;
  --light-alt: #61759b;
  --dark: #131A26;
  --dark-alt: #202B3E;
}

* {
  margin: 0;
  padding: 0;
  box-sizing: border-box;

  font-family: "Fira Sans", sans-serif;
}

body {
  background-color: var(--dark-alt);
  color: var(--light);
}

.App {
  padding: 32px;
}

h1 {
  font-size: 40px;
  font-weight: 700;
  margin-bottom: 32px;
}

```

```

h3{
  font-size: 18px;
  color: #AF1E2D;
}

h4 {
  font-size: 18px;
  color: var(--light-alt);
  text-transform: uppercase;
  font-weight: 400;
  margin-bottom: 16px;
}

.todo {
  position: relative;
  background-color: var(--dark);
  padding: 16px;
  border-radius: 16px;
  display: flex;
  align-items: center;
  transition: 0.5s;
  cursor: pointer;
  margin-bottom: 16px;
}

.todo:hover {
  opacity: 0.8;
}

.todo .checkbox {
  width: 20px;
  height: 20px;
  margin-right: 16px;
  border-radius: 50%;
  background-color: var(--dark-alt);
  background-image: linear-gradient(to bottom, transparent, transparent);
  transition: 0.4s;
}

.todo.is-complete .checkbox {
  background-color: var(--primary);
  background-image: linear-gradient(to bottom, var(--primary), var(--secondary));
}

```

```

.todo .text {
  font-size: 20px;
}

.todo.is-complete .text {
  text-decoration: line-through;
}

.todo .delete-todo {
  position: absolute;
  top: 50%;
  transform: translateY(-50%);
  right: 16px;
  color: var(--light);
  width: 24px;
  height: 24px;
  border-radius: 50%;
  background-color: #AF1E2D;
  display: flex;
  align-items: center;
  justify-content: center;
  font-weight: 700;
}

.addPopup {
  position: fixed;
  bottom: 32px;
  right: 32px;
  display: flex;
  align-items: center;
  justify-content: center;

  width: 64px;
  height: 64px;
  border-radius: 999px;
  font-size: 28px;
  font-weight: 900;
  color: var(--light);
  background-color: var(--primary);
  background-image: linear-gradient(to bottom right, var(--primary), var(--secondary));
  cursor: pointer;
}

.popup {

```

```

    position: fixed;
    top: 50%;
    left: 50%;
    transform: translate(-50%, -50%);

    width: 100%;
    max-width: 400px;
    background-color: var(--light);
    padding: 32px;
    border-radius: 16px;
    box-shadow: 0px 4px 32px var(--dark);
}

.about-image {
    width: 200px;
    height: auto;
}

.closePopup {
    position: absolute;
    top: 16px;
    right: 16px;
    width: 20px;
    height: 20px;
    font-size: 20px;
    color: var(--dark);
    cursor: pointer;
}

.popup h3 {
    color: var(--dark);
    margin-bottom: 16px;
    font-weight: 400;
    text-transform: uppercase;
}

.add-todo-input {
    appearance: none;
    outline: none;
    border: none;
    background-color: #FFF;
    padding: 16px;
    border-radius: 16px;
    width: 100%;
    box-shadow: 0px 2px 24px rgba(0, 0, 0, 0.2);
    font-size: 20px;
}

```



```
.button {
  padding: 16px 32px;
  border-radius: 999px;
  background-image: linear-gradient(to right, var(--primary), var(--secondary));
  display: inline-block;
  font-weight: 700;
  text-transform: uppercase;
  font-size: 18px;
  margin-top: 16px;
  text-align: center;
  cursor: pointer;
}
```

## Index.js

```
import React from 'react';
import ReactDOM from 'react-dom/client';
import './index.css';
import App from './App';

const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(
  <React.StrictMode>
    <App />
  </React.StrictMode>
);
```

## Login

```
import React, { useState } from 'react';
import axios from 'axios';
import { Link, useNavigate } from 'react-router-dom';

function Login() {
  const [username, setUsername] = useState('');
  const [password, setPassword] = useState('');
  const navigate = useNavigate();

  const handleLogin = async (e) => {
    e.preventDefault();
```

```

    try {
      const response = await axios.post('http://localhost:3001/login', { username,
password });
      console.log(response.data.message);

      // Set isLoggedIn to true upon successful login

      // Redirect to the Todos page
      navigate('/todos');
    } catch (error) {
      console.error(error);
    }
  };

  return (
    <div>
      <h2>Login</h2>
      <form onSubmit={handleLogin}>
        <div>
          <label>Username:</label>
          <input type="text" value={username} onChange={ (e) =>
setUsername(e.target.value)} />
        </div>
        <div>
          <label>Password:</label>
          <input type="password" value={password} onChange={ (e) =>
setPassword(e.target.value)} />
        </div>
        <div className='button'>
          <button type="submit">Login</button></div>
        </form>

        <p>Don't have an account? <Link to="/register">Register here</Link></p>
      </div>
    );
  }

export default Login;

```

# Messages.js

```
import React, { useState, useEffect } from 'react';

const api_base = 'http://localhost:3001';

function Messages() {
  const [messages, setMessages] = useState([]);
  const [newMessage, setNewMessage] = useState('');

  useEffect(() => {
    fetchMessages();
  }, []);

  const fetchMessages = async () => {
    try {
      const response = await fetch(api_base + '/messages');
      const data = await response.json();
      setMessages(data);
    } catch (error) {
      console.error('Error fetching messages:', error);
    }
  };

  const sendMessage = async () => {
    try {
      const response = await fetch(api_base + '/messages', {
        method: 'POST',
        headers: {
          'Content-Type': 'application/json',
        },
        body: JSON.stringify({ message: newMessage }),
      });
      const data = await response.json();
      if (data.success) {
        setNewMessage('');
        fetchMessages();
      }
    } catch (error) {
      console.error('Error sending message:', error);
    }
  };

  return (
    <div>
```

```

    <h1>Messages</h1>
    <div>
      {messages.map((messageObj, index) => (
        <p key={index}>{messageObj.message}</p>
      ))}

    </div>
    <input
      className='add-todo-input'
      type="text"
      value={newMessage}
      onChange={ (e) => setNewMessage(e.target.value)}
    />
    <div className='button' onClick={sendMessage}>Send</div>
  </div>
);
}

export default Messages;

```

## Register.js

```

import React, { useState } from 'react';
import axios from 'axios';
import { Link, useNavigate } from 'react-router-dom';

function Register() {
  const [username, setUsername] = useState('');
  const [password, setPassword] = useState('');
  const navigate = useNavigate();

  const handleRegister = async (e) => {
    e.preventDefault();

    try {
      const response = await axios.post('http://localhost:3001/register', {
        username, password });
      console.log(response.data.message);

      // Redirect to the Login page
      navigate('/login');
    }
  }
}

```

```

    } catch (error) {
      console.error(error);
    }
  };

  return (
    <div>
      <h2>Register</h2>
      <form onSubmit={handleRegister}>
        <div>
          <label>Username:</label>
          <input type="text" value={username} onChange={(e) =>
setUsername(e.target.value)} />
        </div>
        <div>
          <label>Password:</label>
          <input type="password" value={password} onChange={(e) =>
setPassword(e.target.value)} />
        </div>
        <div className='button'>
          <button type="submit">Register</button></div>
        </form>

        <p>Already have an account? <Link to="/login">Login here</Link></p>
      </div>
    );
  }

export default Register;

```

## Todos.js

```

import React, { useEffect, useState } from 'react';
import Messages from './Messages';

const api_base = 'http://localhost:3001';

function Todos() {
  const [todos, setTodos] = useState([]);
  const [popupActive, setPopupActive] = useState(false);
  const [newTodo, setNewTodo] = useState("");

```

```

const [filter, setFilter] = useState("all");
const [sort, setSort] = useState("oldest");

useEffect(() => {
  GetTodos();
}, []);

const GetTodos = () => {
  fetch(api_base + '/todos')
    .then(res => res.json())
    .then(data => setTodos(data))
    .catch((err) => console.error("Error: ", err));
}

const completeTodo = async id => {
  const data = await fetch(api_base + '/todo/complete/' + id).then(res =>
res.json());
  setTodos(todos => todos.map(todo => {
    if (todo._id === data._id) {
      todo.complete = data.complete;
    }

    return todo;
  }));
}

const addTodo = async () => {
  const data = await fetch(api_base + "/todo/new", {
    method: "POST",
    headers: {
      "Content-Type": "application/json"
    },
    body: JSON.stringify({
      text: newTodo
    })
  }).then(res => res.json());
  setTodos([...todos, data]);

  setPopupActive(false);
  setNewTodo("");
}

const deleteTodo = async id => {
  const data = await fetch(api_base + '/todo/delete/' + id, { method: "DELETE"
}).then(res => res.json());

```

```

    setTodos(todos => todos.filter(todo => todo._id !== data.result._id));
  }

  const renderFilterButtons = () => (
    <div className="filter-buttons">
      <div className="button" onClick={() => toggleFilter("all")}>All</div>
      <div className="button" onClick={() =>
toggleFilter("complete")}>Complete</div>
      <div className="button" onClick={() => toggleFilter("incomplete")}
>Incomplete</div>
    </div>
  );

  const toggleFilter = (filterName) => {
    setFilter(filterName);
  };

  const toggleSort = (sortName) => {
    setSort(sortName);
  };

  const renderSortButtons = () => (
    <div className="sort-buttons">
      <div className="button" onClick={() => toggleSort("oldest")}
>Oldest</div>
      <div className="button" onClick={() => toggleSort("newest")}
>Newest</div>
    </div>
  );

  const renderTodos = () => {
    let filteredTodos = todos.filter(todo => {
      if (filter === "all") {
        return true;
      } else if (filter === "complete") {
        return todo.complete;
      } else {
        return !todo.complete;
      }
    });

    let sortedTodos = filteredTodos.sort((a, b) => {
      if (sort === "newest") {
        return b.date - a.date;
      } else {

```

```

        return a.date - b.date;
    }
});

return (
    <div className="todos">
        {sortedTodos.length > 0 ? sortedTodos.map(todo => (
            <div className={
                "todo" + (todo.complete ? " is-complete" : "")
            } key={todo._id} onClick={() => completeTodo(todo._id)}>
                <div className="checkbox"></div>

                <div className="text">{todo.text}</div>

                <div className="delete-todo" onClick={() =>
deleteTodo(todo._id)}>x</div>
            </div>
        )) : (
            <h3>You currently have no tasks</h3>
        )}
    </div>
);
};

const renderAddPopup = () => (
    <div className="popup">
        <div className="closePopup" onClick={() => setPopupActive(false)}>
            X
        </div>
        <div className="content">
            <h3>Add Task</h3>
            <input
                type="text"
                className="add-todo-input"
                onChange={(e) => setNewTodo(e.target.value)}
                value={newTodo}
            />
            <div className="button" onClick={addTodo}>
                Create Task
            </div>
        </div>
    </div>
);

```



```

    return (
      <>
        {renderFilterButtons()}
        {renderSortButtons()}
        {renderTodos()}

        <div className="addPopup" onClick={() => setPopupActive(true)}>
          +
        </div>

        {popupActive && renderAddPopup()}

        <Messages />
      </>
    );
  }
}

export default Todos;

```

## Client-package.json

```

{
  "name": "client",
  "version": "0.1.0",
  "private": true,
  "dependencies": {
    "@testing-library/jest-dom": "^5.16.5",
    "@testing-library/react": "^13.4.0",
    "@testing-library/user-event": "^13.5.0",
    "axios": "^1.4.0",
    "react": "^18.2.0",
    "react-dom": "^18.2.0",
    "react-router-dom": "^6.11.1",
    "react-scripts": "5.0.1",
    "socket.io-client": "^4.6.1",
    "web-vitals": "^2.1.4"
  },
  "scripts": {
    "start": "react-scripts start",
    "build": "react-scripts build",
    "test": "react-scripts test",
    "eject": "react-scripts eject"
  },

```

```
"eslintConfig": {
  "extends": [
    "react-app",
    "react-app/jest"
  ]
},
"browserslist": {
  "production": [
    ">0.2%",
    "not dead",
    "not op_mini all"
  ],
  "development": [
    "last 1 chrome version",
    "last 1 firefox version",
    "last 1 safari version"
  ]
}
```