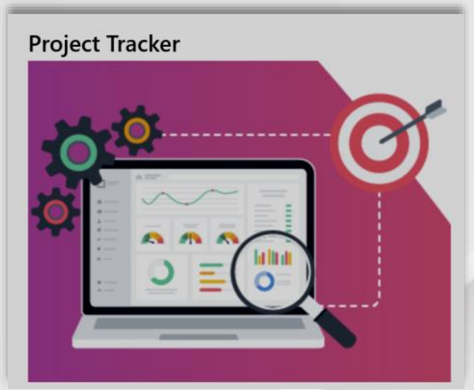
Project Tracker

by Code Crafters
Tarak
Prasanna
Aneesh
Pavan
Akshay

Roles

- Everyone acts as a designer, developer, and tester for each component.
- Everyone acts as a scrum master for each sprint meeting.
- Everyone also acts as product owner/stakeholder to the other components to provide feedback, validation and enhancement ideas

Overview



What is it?

- Project Tracker is a user-friendly web application for managing projects, contracts, and employees in organizations.

What does it do?

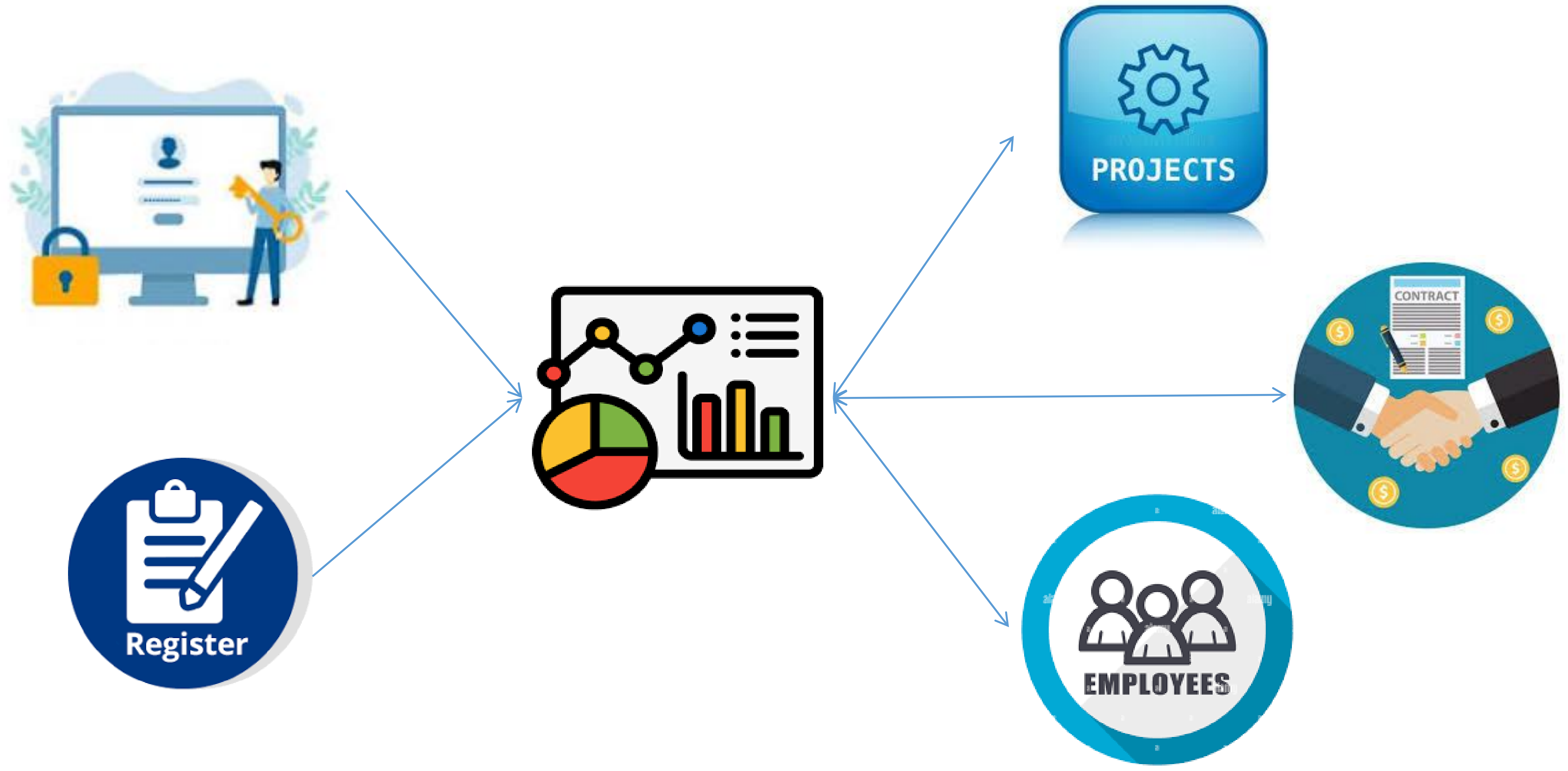
- It helps admins and admin privilege users to add, edit, and delete projects, contracts, and employees, and approve user roles.

Why use it?

- It replaces traditional methods like Excel, offering easier access, stats, and secure authentication.

Project Tracker

Outline



Technical Requirements

Frontend:

- Programming Languages: JavaScript
- Front-end Framework: React
- Version Control: GitHub
- Authentication: Implement secure user authentication like JWT(JSON web token) Authentication

Backend:

- Back-end Framework: Node.js(Express.js), Mocha(unit test framework)
- Authentication: Implement secure user authentication like JWT(JSON web token) Authentication

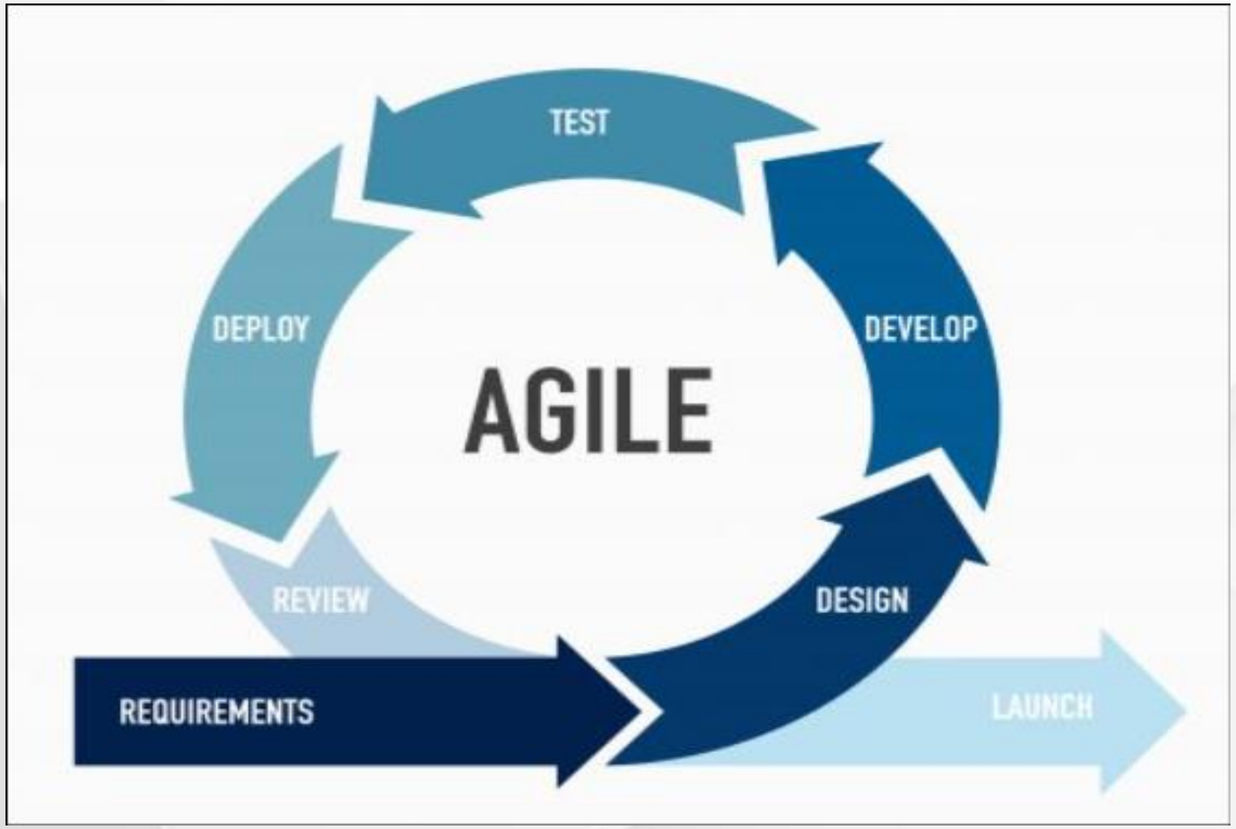
Database:

- Database: MongoDB
- ODM: Mongoose

Tools:

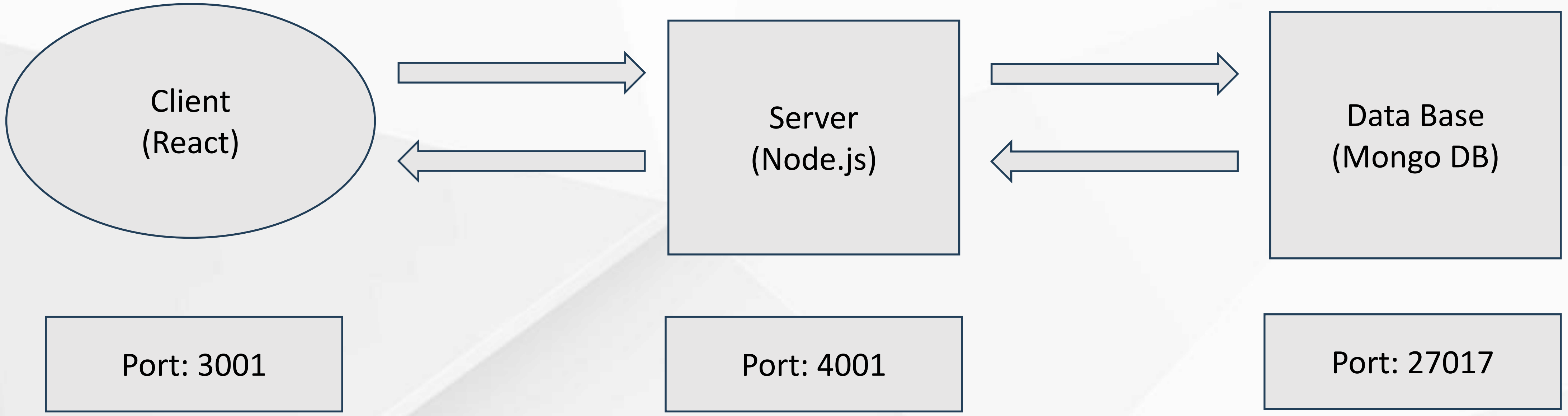
- Postman
- Visual Studio
- Studio 3T

Methodology



- Agile Methodology:
- Weekly sprints
- Weekly meetings – Teams or in-person.
- Functionality discussions and enhancements ideas.
- Everyone acted as scrum master, developer.
- Everyone acted as stakeholder and product owner to validate and enhance the other teammate component development.
- Backlog refinement

Architecture



Components

- User Registration
- Admin User Creation
- User Login
- Users
- Dashboard
- Projects
- Contracts
- Employees
- Settings

User Registration

Introduction

- To access the application, an employee should register with their details.
- Registered user is in active mode.
- Only admin users/admin privilege users can activate the registered users.

Requirements

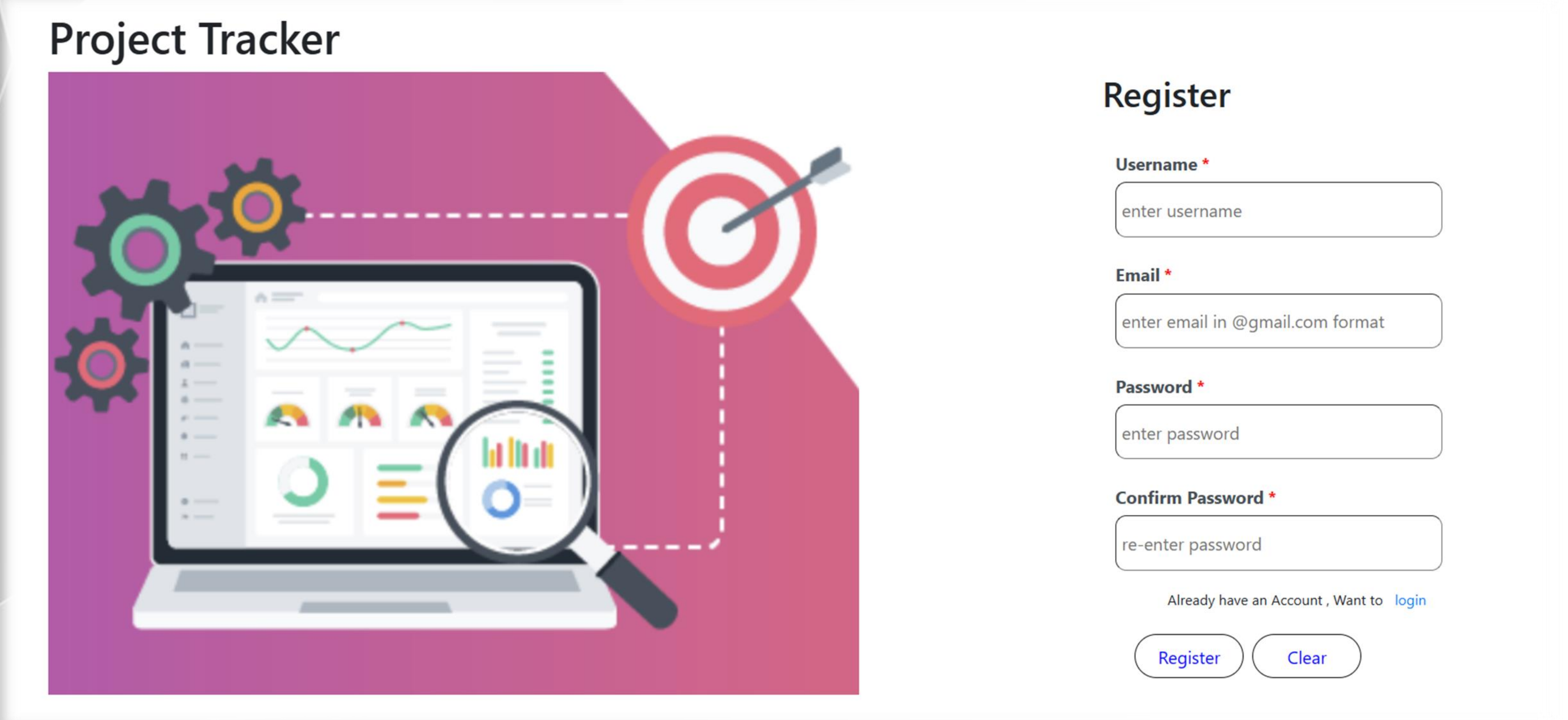
A register form will provide with following fields.

- Username: Unique, no suggestions, error for duplicates.
- Email: Valid format, case-insensitive, unique, error for duplicates.
- Password: Min 8 characters, enforce complexity (uppercase, lowercase, digit, special symbol), masked input, error for invalid passwords, no suggestions.
- Confirm Password: Must match password, error for mismatch.
- Login Link: Navigate to the login page and Clear the login form upon click.
- Register Button: Validate payload and show proper registration status.

User story

As an employee, I want to sign up for a new user account on the platform so that I can access its features and functionalities. I will need to provide a distinct username, email address, and password

Register Form UI



Project Tracker

Register

Username *

enter username

Email *

enter email in @gmail.com format

Password *

enter password

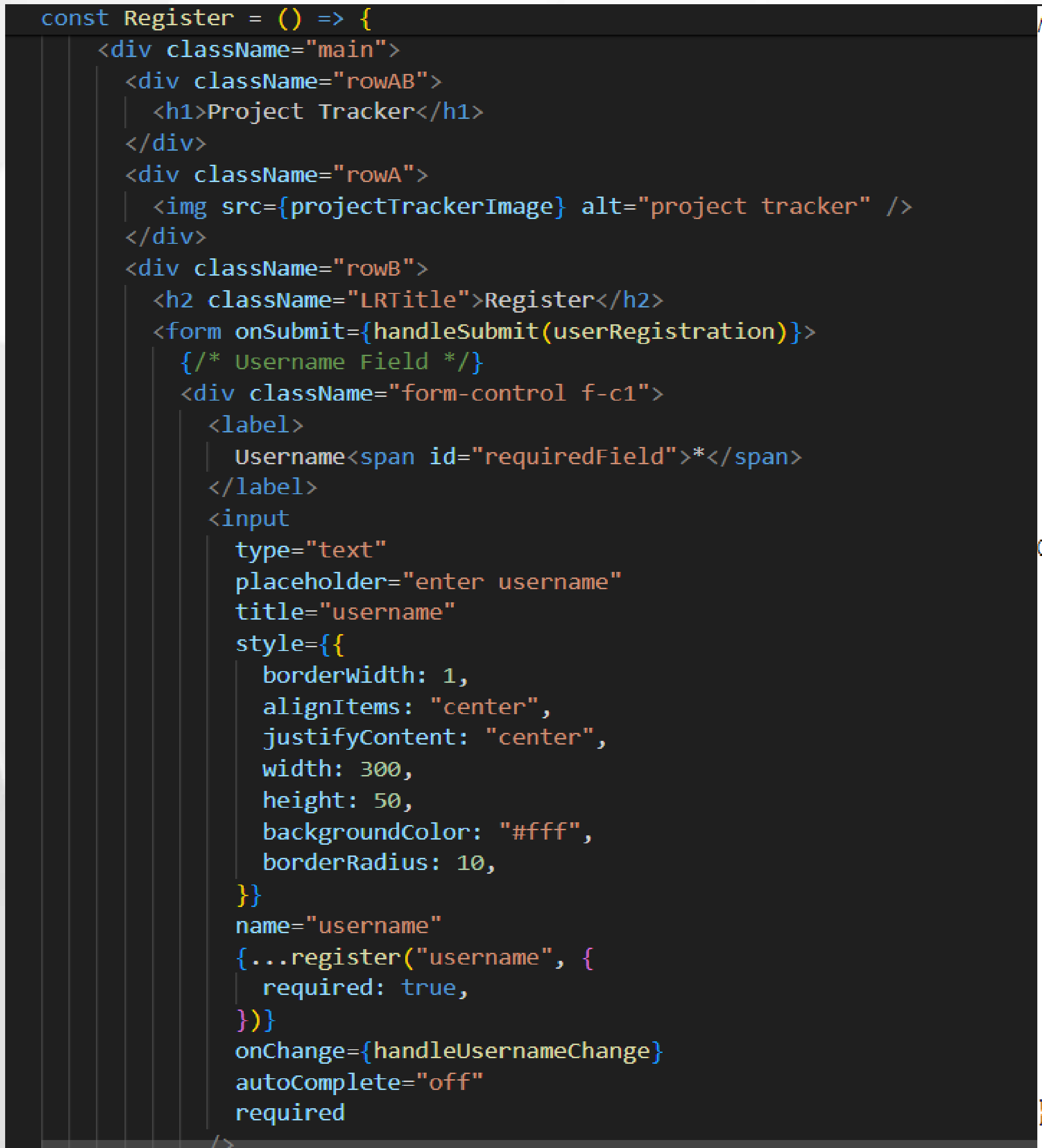
Confirm Password *

re-enter password

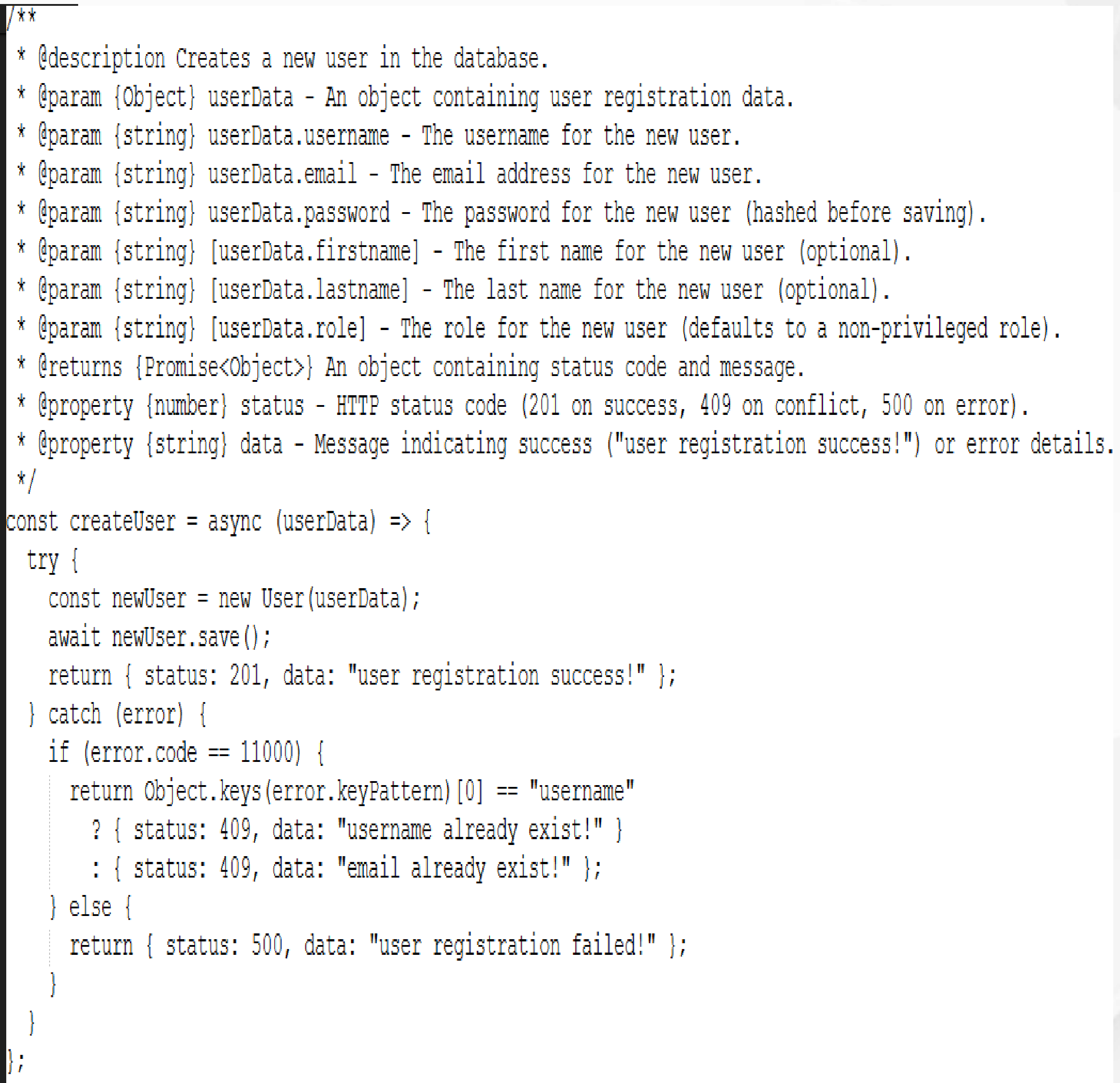
Already have an Account , Want to login

Project Tracker

Code Snippet



```
const Register = () => {
    <div className="main">
      <div className="rowAB">
        <h1>Project Tracker</h1>
      </div>
      <div className="rowA">
        <img src={projectTrackerImage} alt="project tracker" />
      </div>
      <div className="rowB">
        <h2 className="LRTitle">Register</h2>
        <form onSubmit={handleSubmit(userRegistration)}>
          {/* Username Field */}
          <div className="form-control f-c1">
            <label>
              Username<span id="requiredField">*</span>
            </label>
            <input
              type="text"
              placeholder="enter username"
              title="username"
              style={{
                borderWidth: 1,
                alignItems: "center",
                justifyContent: "center",
                width: 300,
                height: 50,
                backgroundColor: "#fff",
                borderRadius: 10,
              }}
              name="username"
              {...register("username", {
                required: true,
              })}
              onChange={handleUsernameChange}
              autoComplete="off"
              required
```



```
/**
 * @description Creates a new user in the database.
 * @param {Object} userData - An object containing user registration data.
 * @param {string} userData.username - The username for the new user.
 * @param {string} userData.email - The email address for the new user.
 * @param {string} userData.password - The password for the new user (hashed before saving).
 * @param {string} [userData.firstname] - The first name for the new user (optional).
 * @param {string} [userData.lastname] - The last name for the new user (optional).
 * @param {string} [userData.role] - The role for the new user (defaults to a non-privileged role).
 * @returns {Promise<Object>} An object containing status code and message.
 * @property {number} status - HTTP status code (201 on success, 409 on conflict, 500 on error).
 * @property {string} data - Message indicating success ("user registration success!") or error details.
 */
const createUser = async (userData) => {
  try {
    const newUser = new User(userData);
    await newUser.save();
    return { status: 201, data: "user registration success!" };
  } catch (error) {
    if (error.code == 11000) {
      return Object.keys(error.keyPattern)[0] == "username"
        ? { status: 409, data: "username already exist!" }
        : { status: 409, data: "email already exist!" };
    } else {
      return { status: 500, data: "user registration failed!" };
    }
  }
};
```

API’s

1.Get registered users

Endpoint: /v/getUsers

Method: GET

2.User Register

Endpoint: /v/register

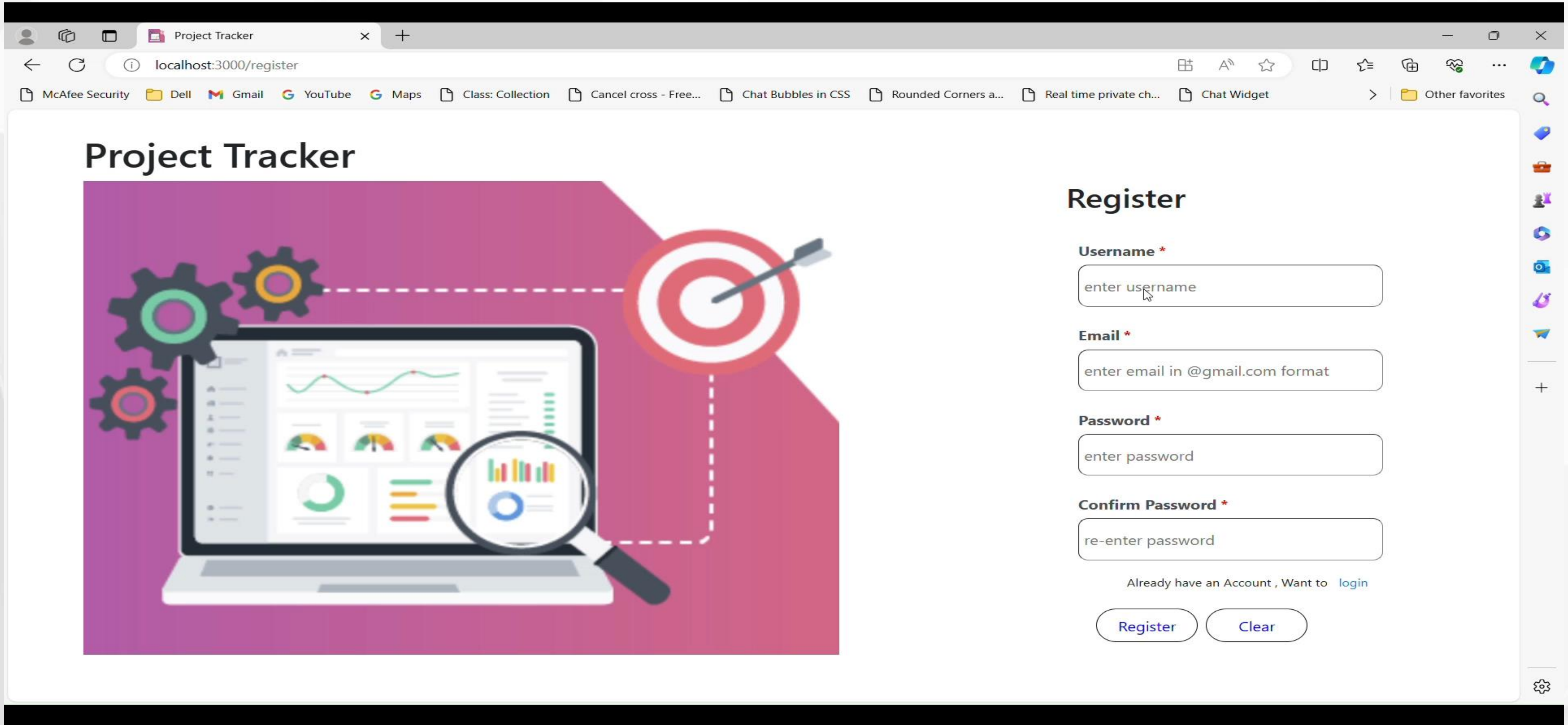
Method: POST

Request payload :

{"username": "Test", "email": "test@gmail.com", “password”: “Test@1234”, "confirm_password": "Test@1234"}

Evaluation Criteria

Secure user registration form with validation unique username, email, and strong password with confirmation, clear error messages, and successful registration/failure toast notification.



Project Tracker

Register

Username *

enter username

Email *

enter email in @gmail.com format

Password *

enter password

Confirm Password *

re-enter password

Already have an Account , Want to login

Register Clear

Admin User Creation

Introduction

- In the project tracker web application, the admin user holds the highest level of authority and responsibility.
- Privileges
  - Access to all features and functionalities of the project tracker web application.
  - Authority to create and modify user accounts.
  - Oversight of project/employee data, including creation, editing, and deleting.

Requirements

REST API to create an admin user for the application

User Story

As a system administrator, I want to create a new admin user through a secure API endpoint, providing their username, email, password, and optional name information, so that I can manage system access and assign administrative privileges

Admin User API format

API Type : REST
Method : POST
End point : /v/adminUser
Authorization : Basic Auth
Request Body :

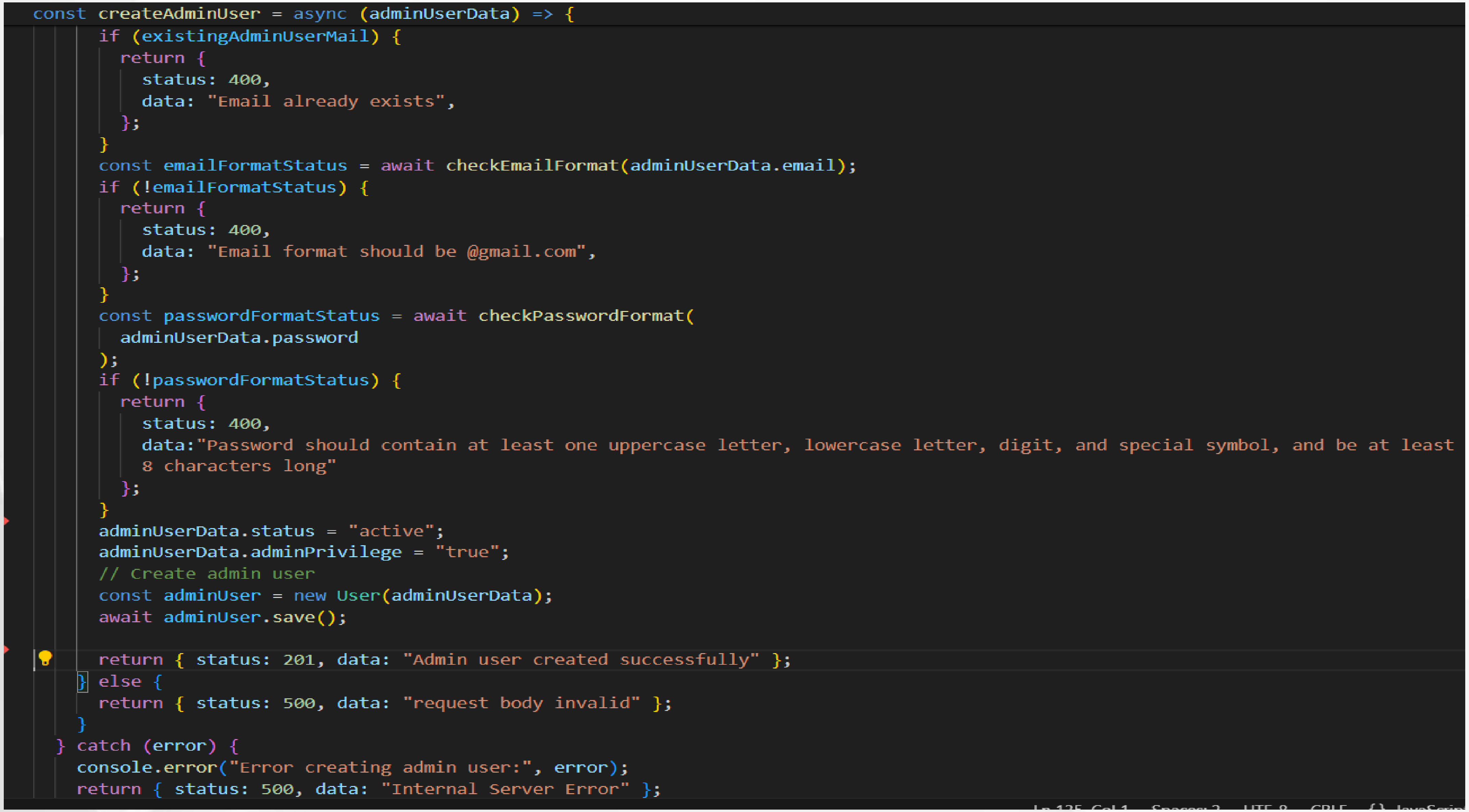
```
{
      "username": "adminUser",
      "email": "test@gmail.com",
      "password":"Pwd@1234",
      "firstname": "fname",
      "lastname": "lname",
      "role":"admin"
  }
```

Response :

```
{
    "status": 201,
    "data": "Admin user created successfully"
}
```

Project Tracker

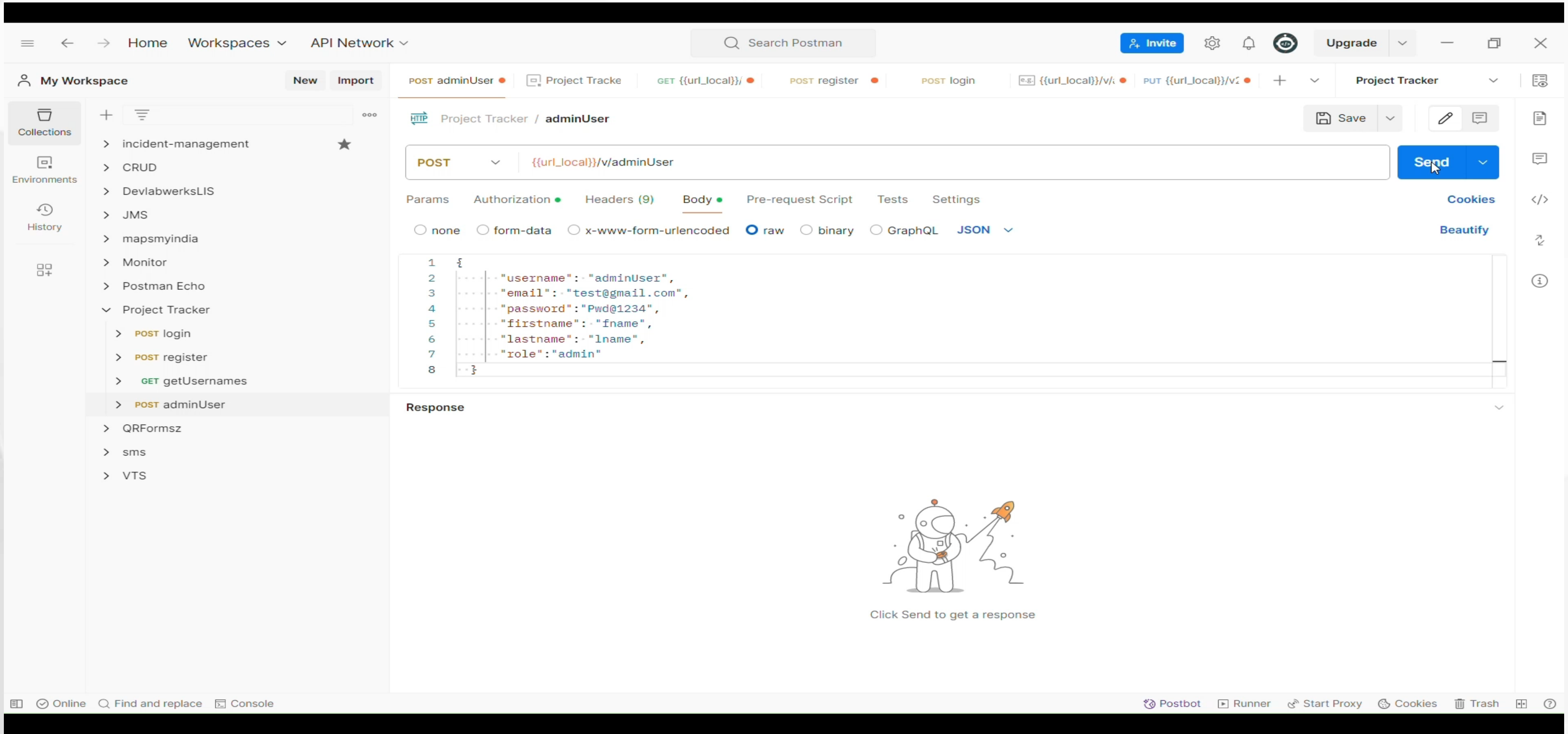
Code Snippet



```
const createAdminUser = async (adminUserData) => {
      if (existingAdminUserMail) {
        return {
          status: 400,
          data: "Email already exists",
        };
      }
      const emailFormatStatus = await checkEmailFormat(adminUserData.email);
      if (!emailFormatStatus) {
        return {
          status: 400,
          data: "Email format should be @gmail.com",
        };
      }
      const passwordFormatStatus = await checkPasswordFormat(
        adminUserData.password
      );
      if (!passwordFormatStatus) {
        return {
          status: 400,
          data:"Password should contain at least one uppercase letter, lowercase letter, digit, and special symbol, and be at least
          8 characters long"
        };
      }
      adminUserData.status = "active";
      adminUserData.adminPrivilege = "true";
      // Create admin user
      const adminUser = new User(adminUserData);
      await adminUser.save();

      return { status: 201, data: "Admin user created successfully" };
    } else {
      return { status: 500, data: "request body invalid" };
    }
  } catch (error) {
    console.error("Error creating admin user:", error);
    return { status: 500, data: "Internal Server Error" };
```

Evaluation Criteria

Functionally creates admin via POST request with validated unique username, email, and strong password.



POST {{url_local}}/v/adminUser

```
{
    "username": "adminUser",
    "email": "test@gmail.com",
    "password":"Pwd@1234",
    "firstname": "fname",
    "lastname": "lname",
    "role":"admin"
}
```

User Login

Introduction

- Active registered users can log in to the application to access the features based on the privileges.
- When the user tries to log in, the data gets verified with the backend database and gets authenticated

Requirements

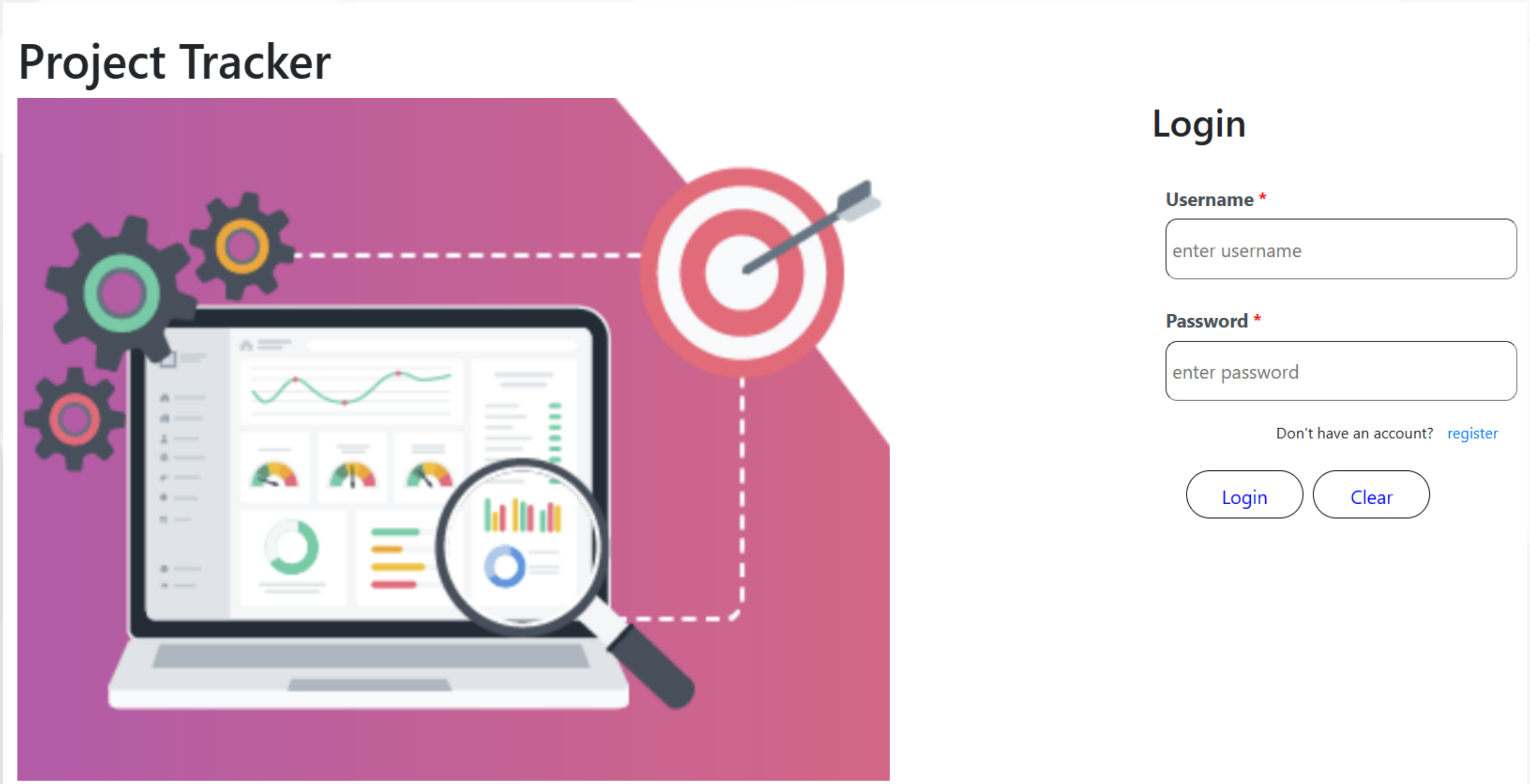
A login form with the following fields

- Username: Text Field (mandatory), no suggestions.
- Password: Text Field (mandatory), Minimum 8 characters, Enforce complexity (uppercase, lowercase, digit, special symbol).
- Register Link: Navigate to the register form and clear the login form upon clicking.
- Login Button: Submits form data

User Story

As a registered user, I want to log in to the application using my username and password so that I can access its features and functionalities

User Login Form UI


Project Tracker

Login

Username *

enter username

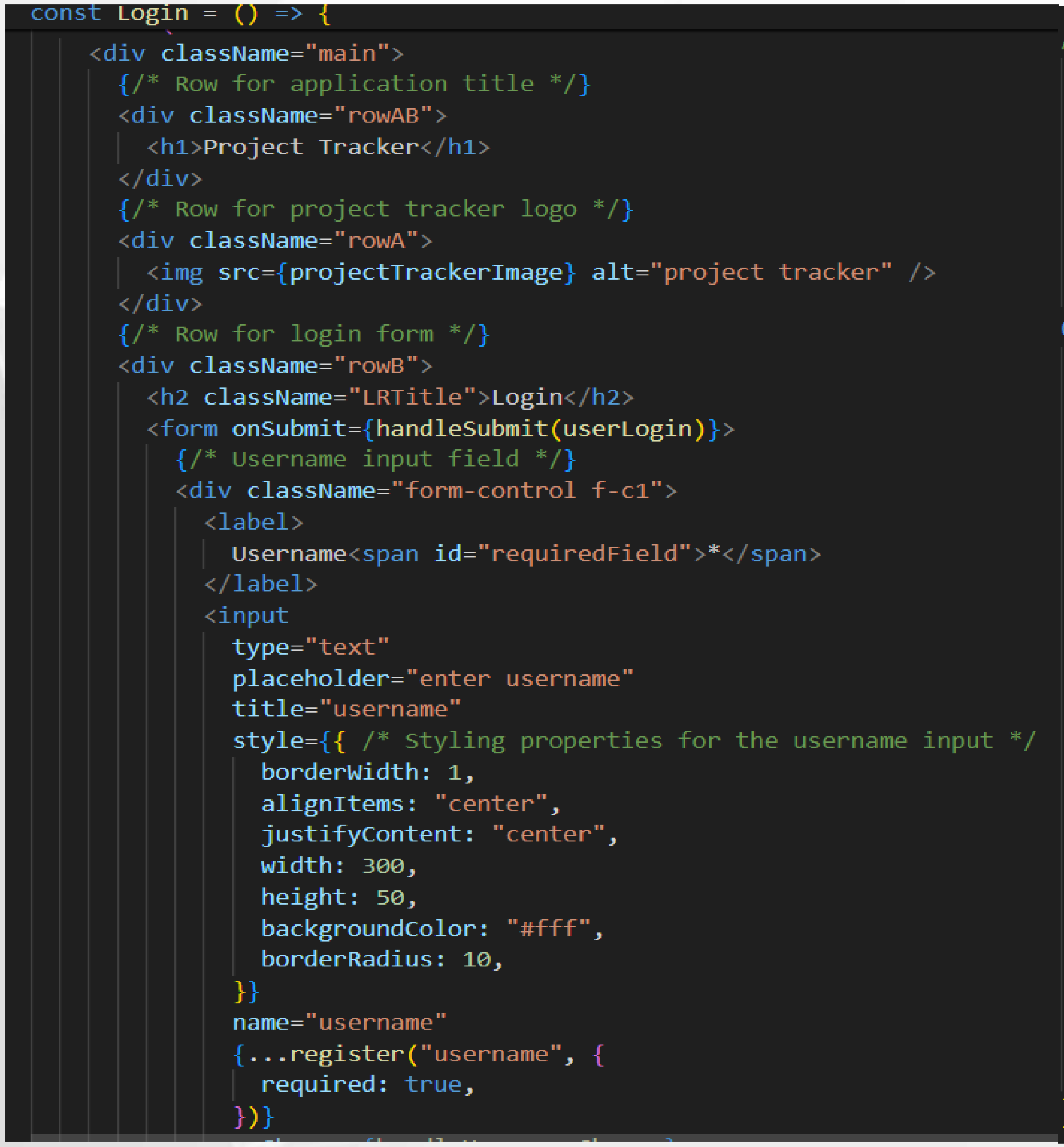
Password *

enter password

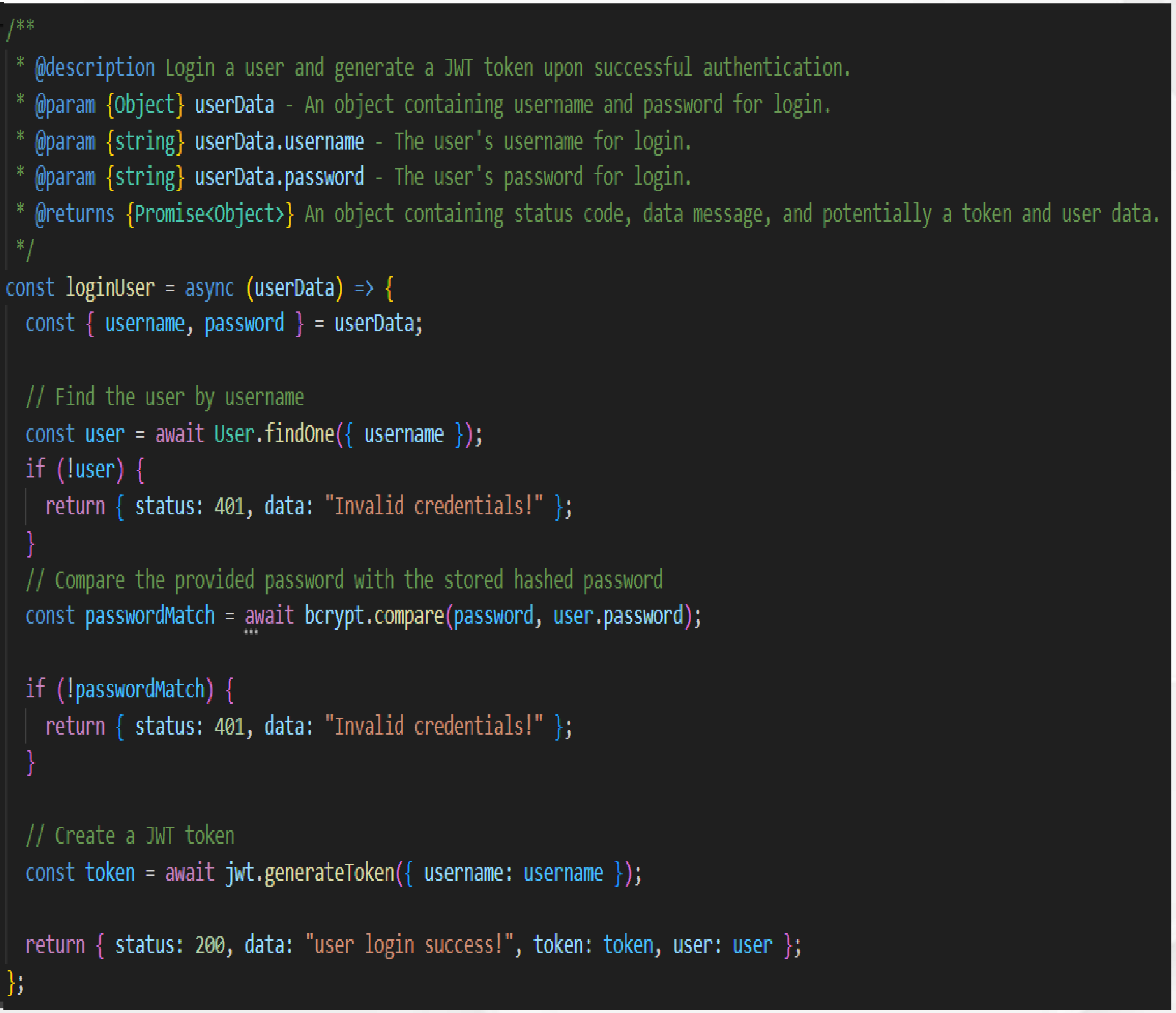
Don't have an account? register

Project Tracker

Code Snippet



```
const Login = () => {
    <div className="main">
      {/* Row for application title */}
      <div className="rowAB">
        <h1>Project Tracker</h1>
      </div>
      {/* Row for project tracker logo */}
      <div className="rowA">
        <img src={projectTrackerImage} alt="project tracker" />
      </div>
      {/* Row for login form */}
      <div className="rowB">
        <h2 className="LRTitle">Login</h2>
        <form onSubmit={handleSubmit(userLogin)}>
          {/* Username input field */}
          <div className="form-control f-c1">
            <label>
              Username<span id="requiredField">*</span>
            </label>
            <input
              type="text"
              placeholder="enter username"
              title="username"
              style={{ /* Styling properties for the username input */
                borderWidth: 1,
                alignItems: "center",
                justifyContent: "center",
                width: 300,
                height: 50,
                backgroundColor: "#fff",
                borderRadius: 10,
              }}
              name="username"
              {...register("username", {
                required: true,
              })}
```



```
/**
 * @description Login a user and generate a JWT token upon successful authentication.
 * @param {Object} userData - An object containing username and password for login.
 * @param {string} userData.username - The user's username for login.
 * @param {string} userData.password - The user's password for login.
 * @returns {Promise<Object>} An object containing status code, data message, and potentially a token and user data.
 */
const loginUser = async (userData) => {
  const { username, password } = userData;

  // Find the user by username
  const user = await User.findOne({ username });
  if (!user) {
    return { status: 401, data: "Invalid credentials!" };
  }
  // Compare the provided password with the stored hashed password
  const passwordMatch = await bcrypt.compare(password, user.password);

  if (!passwordMatch) {
    return { status: 401, data: "Invalid credentials!" };
  }

  // Create a JWT token
  const token = await jwt.generateToken({ username: username });

  return { status: 200, data: "user login success!", token: token, user: user };
};
```

API's

1.Get registered users

End point : /v/getUsers

Method : GET

2.User Login

End point : /v1/login

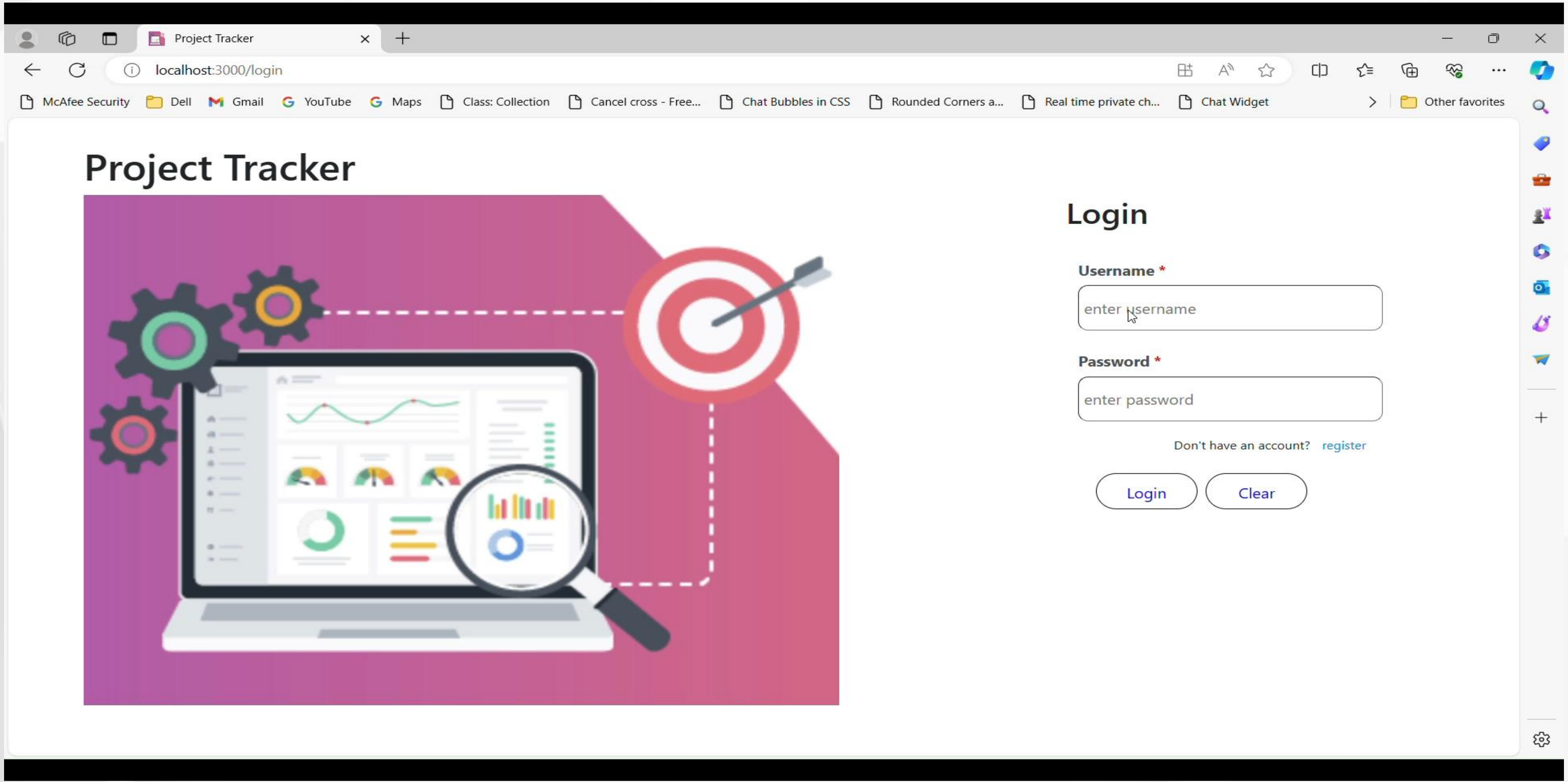
Method : POST

Request payload :

```
{"username":"adminUser","password":"Pwd@1234"}
```

Evaluation Criteria

Secure user login form with validate username and strong password, masked input, and clear error messages. Form clears after submission.



Users

Introduction

The user page will allow to display user’s privileges and also allow admin privilege users to update privileges except admin role users. All the changes act like a query to the database and the tables get modified accordingly

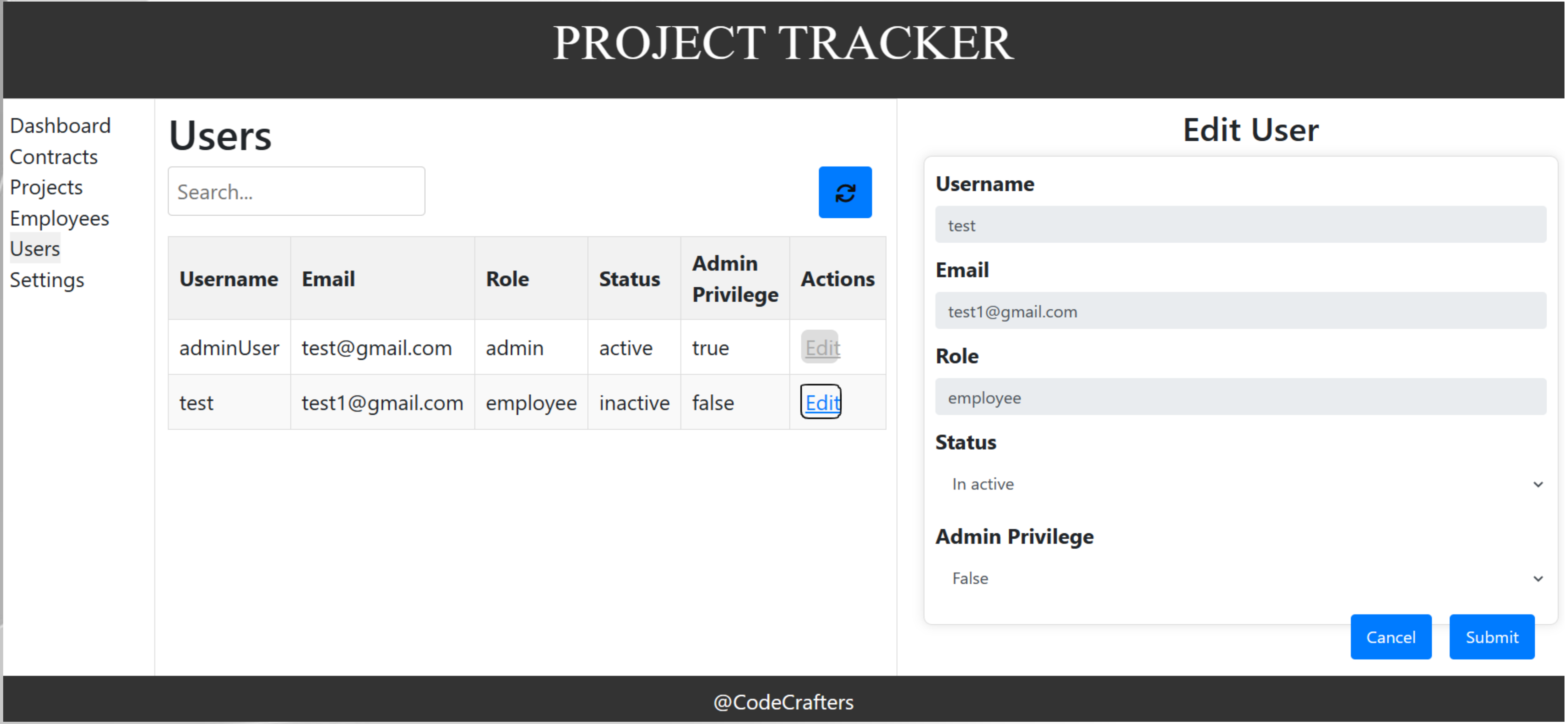
Requirements

- A table displays all registered users with:
  Username, Email, Status, Role, Admin Privilege
- Edit Option: Only admins can edit users. Clicking "Edit" opens the edit form for the selected user.
- Edit User Form:
  - Pre-populated with existing user data: Username (read-only), Email, Status (editable dropdown) Role (editable dropdown), Admin Privilege (editable checkbox)
- Allows admins to modify: User Status, User Role, Admin Privilege

User Story

As an admin privilege user, I would like to examine user information, including username, email, status, role, and admin privilege, and also update status and privilege.

Users Tab UI



PROJECT TRACKER

Users

Username	Email	Role	Status	Admin Privilege	Actions
adminUser	test@gmail.com	admin	active	true	Edit
test	test1@gmail.com	employee	inactive	false	Edit

Edit User

Username: test

Email: test1@gmail.com

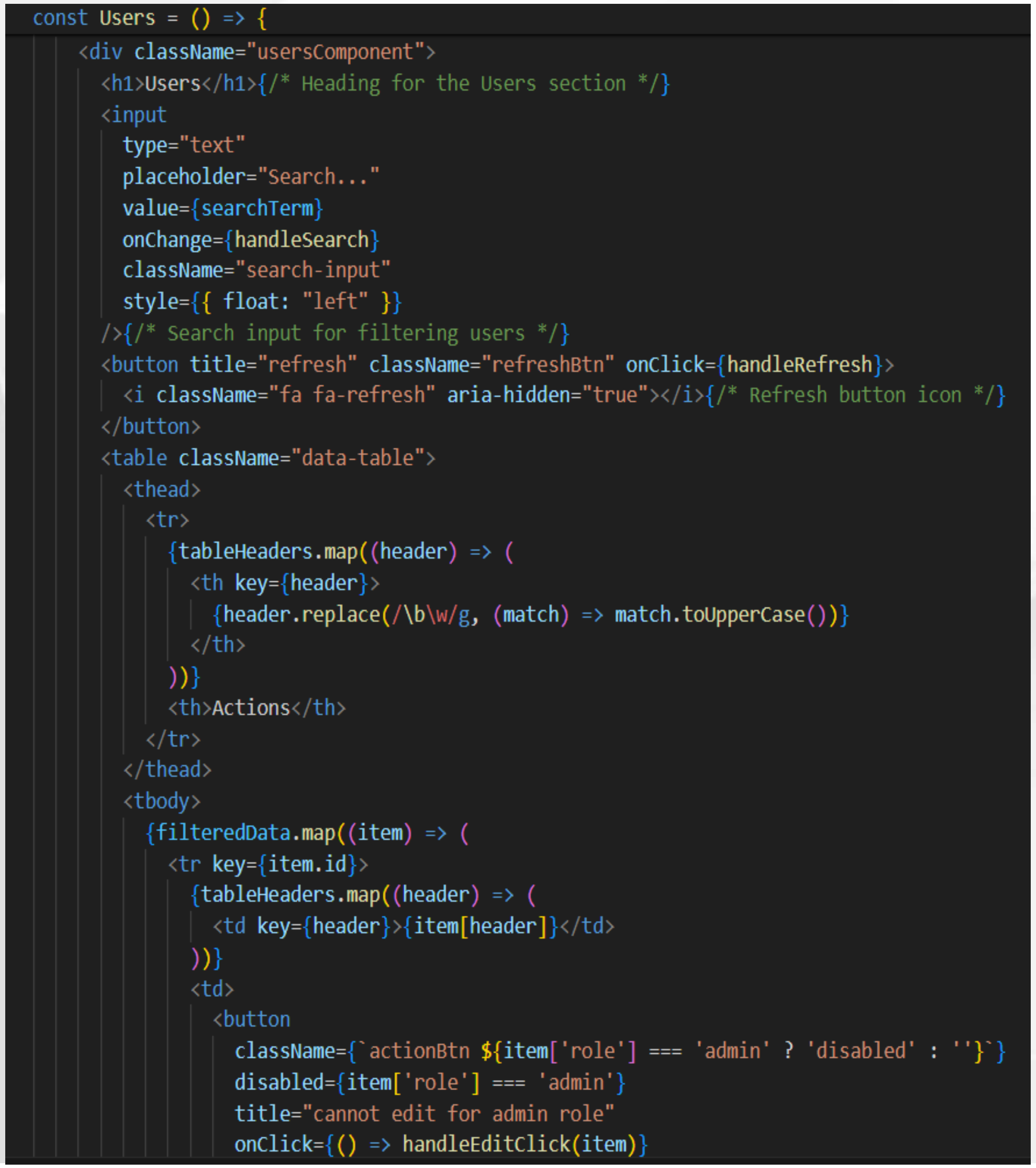
Role: employee

Status: In active

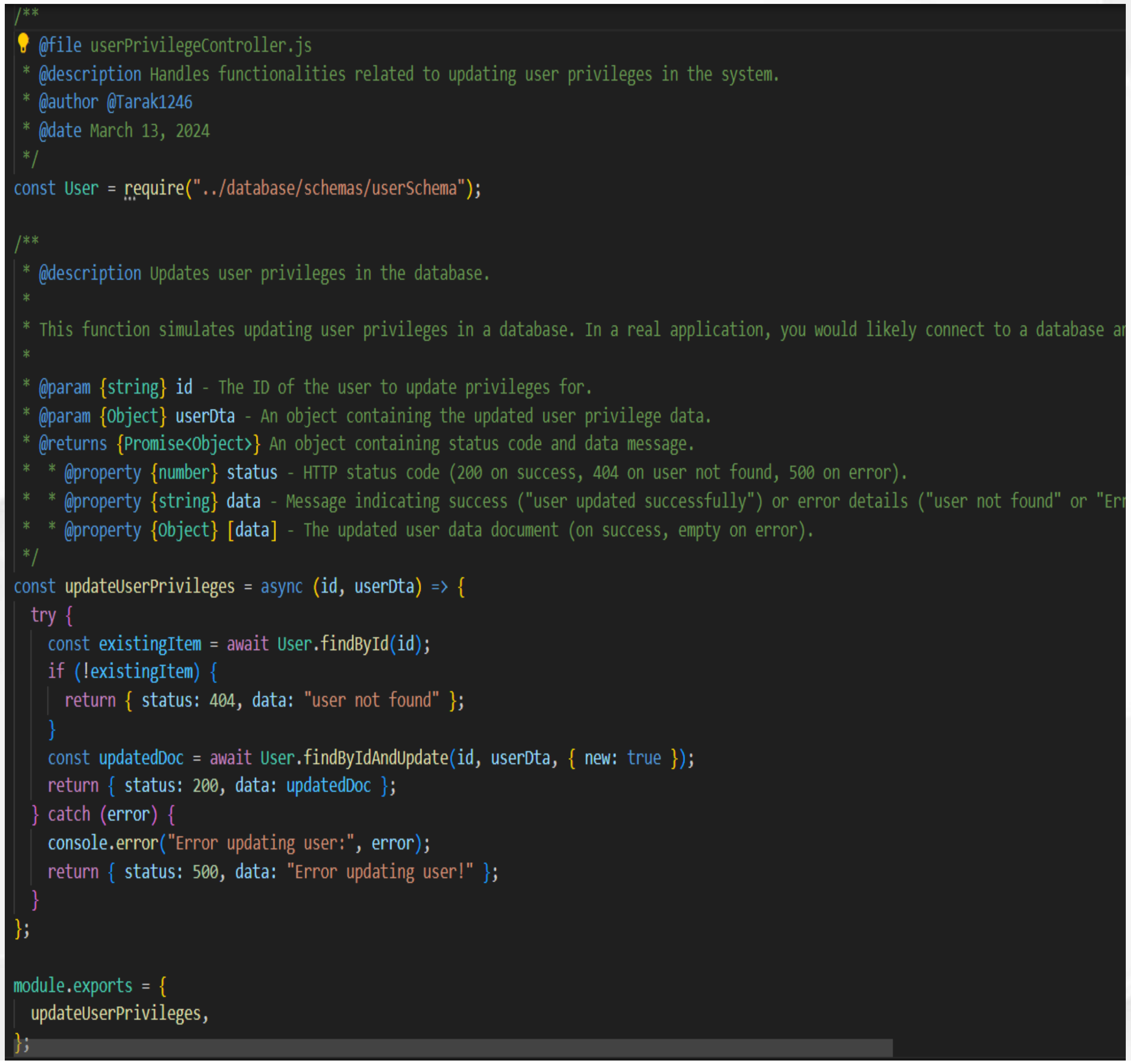
Admin Privilege: False

Project Tracker

Code Snippet



```
const Users = () => {
  <div className="usersComponent">
    <h1>Users</h1>{/* Heading for the Users section */}
    <input
      type="text"
      placeholder="Search..."
      value={searchTerm}
      onChange={handleSearch}
      className="search-input"
      style={{ float: "left" }}
    />{/* Search input for filtering users */}
    <button title="refresh" className="refreshBtn" onClick={handleRefresh}>
      <i className="fa fa-refresh" aria-hidden="true"></i>{/* Refresh button icon */}
    </button>
    <table className="data-table">
      <thead>
        <tr>
          {tableHeaders.map((header) => (
            <th key={header}>
              {header.replace(/\b\w/g, (match) => match.toUpperCase())}
            </th>
          ))}
          <th>Actions</th>
        </tr>
      </thead>
      <tbody>
        {filteredData.map((item) => (
          <tr key={item.id}>
            {tableHeaders.map((header) => (
              <td key={header}>{item[header]}</td>
            ))}
            <td>
              <button
                className={`actionBtn ${item['role'] === 'admin' ? 'disabled' : ''}`}
                disabled={item['role'] === 'admin'}
                title="cannot edit for admin role"
                onClick={() => handleEditClick(item)}
```



```
/**
 * @file userPrivilegeController.js
 * @description Handles functionalities related to updating user privileges in the system.
 * @author @Tarak1246
 * @date March 13, 2024
 */
const User = require("../database/schemas/userSchema");

/**
 * @description Updates user privileges in the database.
 *
 * This function simulates updating user privileges in a database. In a real application, you would likely connect to a database ar
 *
 * @param {string} id - The ID of the user to update privileges for.
 * @param {Object} userDta - An object containing the updated user privilege data.
 * @returns {Promise<Object>} An object containing status code and data message.
 *   * @property {number} status - HTTP status code (200 on success, 404 on user not found, 500 on error).
 *   * @property {string} data - Message indicating success ("user updated successfully") or error details ("user not found" or "Err
 *   * @property {Object} [data] - The updated user data document (on success, empty on error).
 */
const updateUserPrivileges = async (id, userDta) => {
  try {
    const existingItem = await User.findById(id);
    if (!existingItem) {
      return { status: 404, data: "user not found" };
    }
    const updatedDoc = await User.findByIdAndUpdate(id, userDta, { new: true });
    return { status: 200, data: updatedDoc };
  } catch (error) {
    console.error("Error updating user:", error);
    return { status: 500, data: "Error updating user!" };
  }
};

module.exports = {
  updateUserPrivileges,
};
```

API’s

1. **Get registered users**

   Endpoint: /v/getUsers

   Method: GET

   Authorization: JSON Web Token

2. **Update user information**

   End point : /v2/updateUserPrivileges

   Method : PUT

   Authorization : JSON Web Token

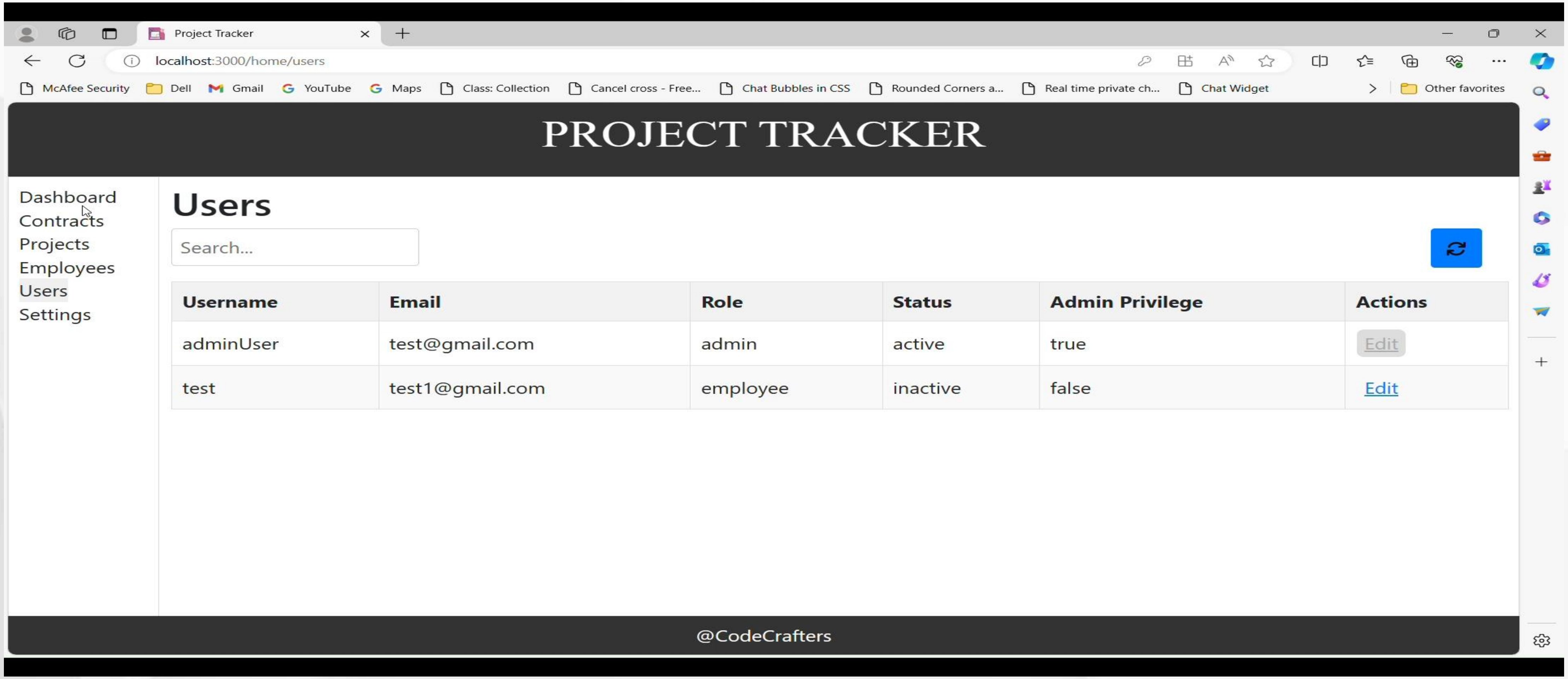
   Request payload :

   {"username":"tarak123","email":"taraksai@gmail.com","role":"employee","status":"active"," admin privilege":"true","adminPrivilege":"true"}

Project Tracker

Evaluation Criteria

Admin-only user management page listing all users in a table with username, email, status, role, and admin privilege. The edit option allows admins to edit user data (status, role, and potential privilege) within a dedicated form



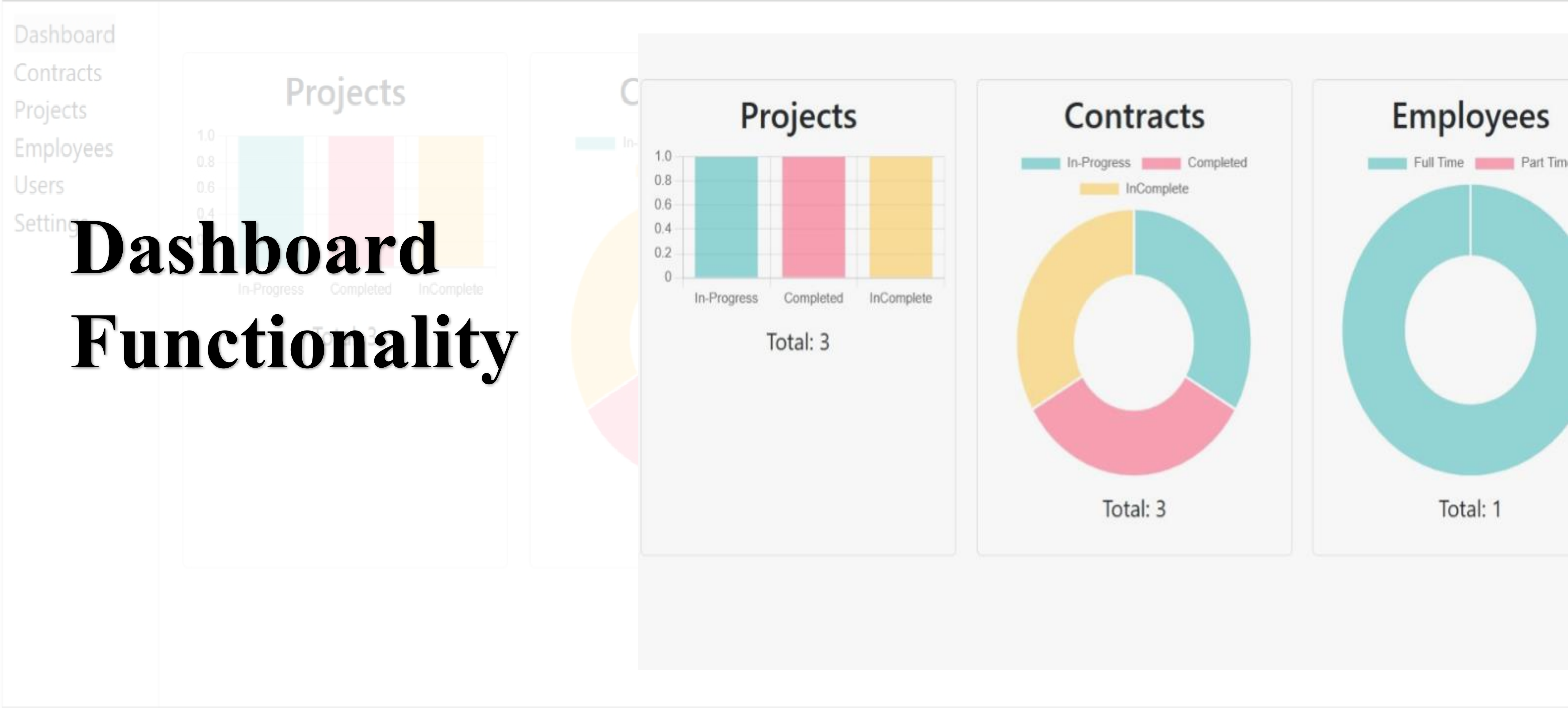
Users

Username	Email	Role	Status	Admin Privilege	Actions
adminUser	test@gmail.com	admin	active	true	Edit
test	test1@gmail.com	employee	inactive	false	Edit

Project Tracker

PROJECT TRACKER

Dashboard Functionality

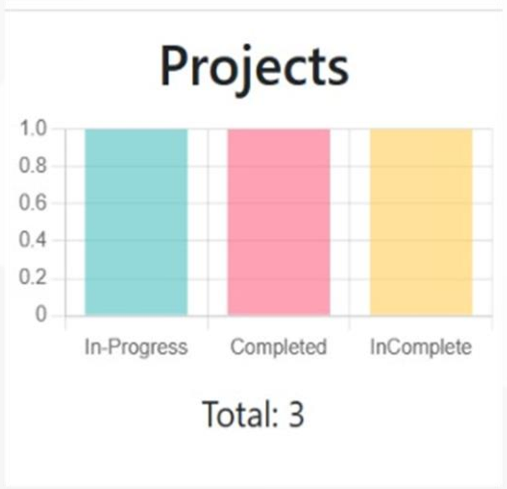


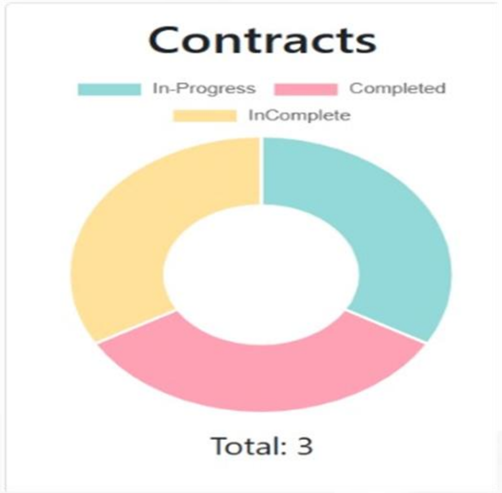
Introduction

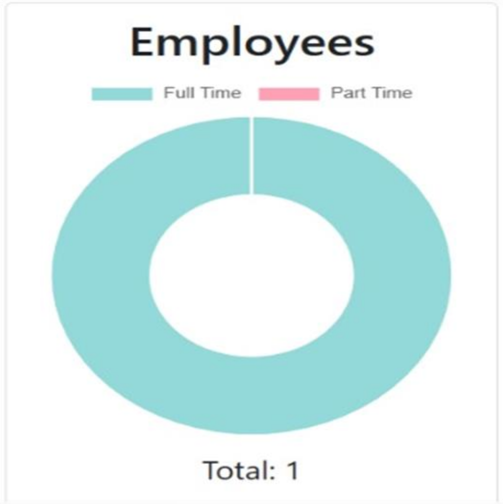
- The Dashboard module aims to provide users with a graphical representation
- Privileges:
  1. Access to Graphical Representation
  2. Access to Metrics
  3. Dynamic Updates
  4. Responsive Design
  5. Compatibility
- User Story:

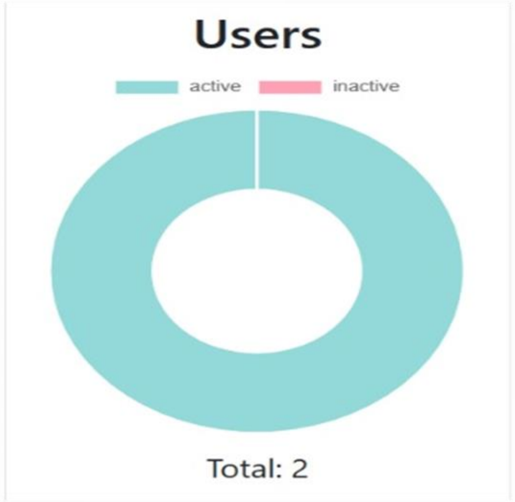
Dashboard developers provide dynamic visual modules for the system, showing project progress and employee information via graphs/charts that automatically reload as the underlying data changes.

Requirements









Projects

Any changes made to the Projects tab can seen in the Projects chart here

Contracts

The Contracts data can be easily tracked here by checking the pie chart

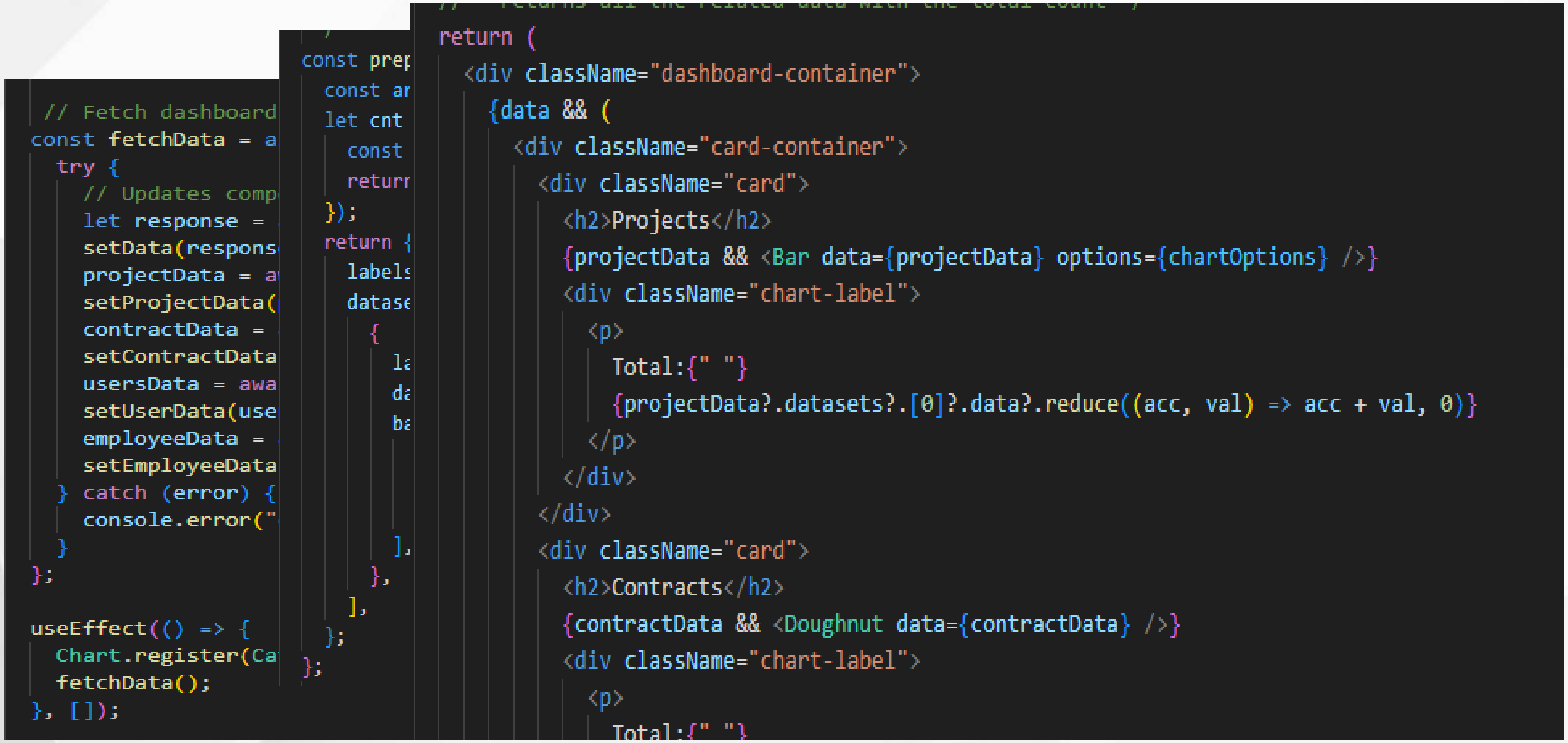
Employees

All the status of the employees is tracked here for the fast representation

Users

The Users data is shown here

Code

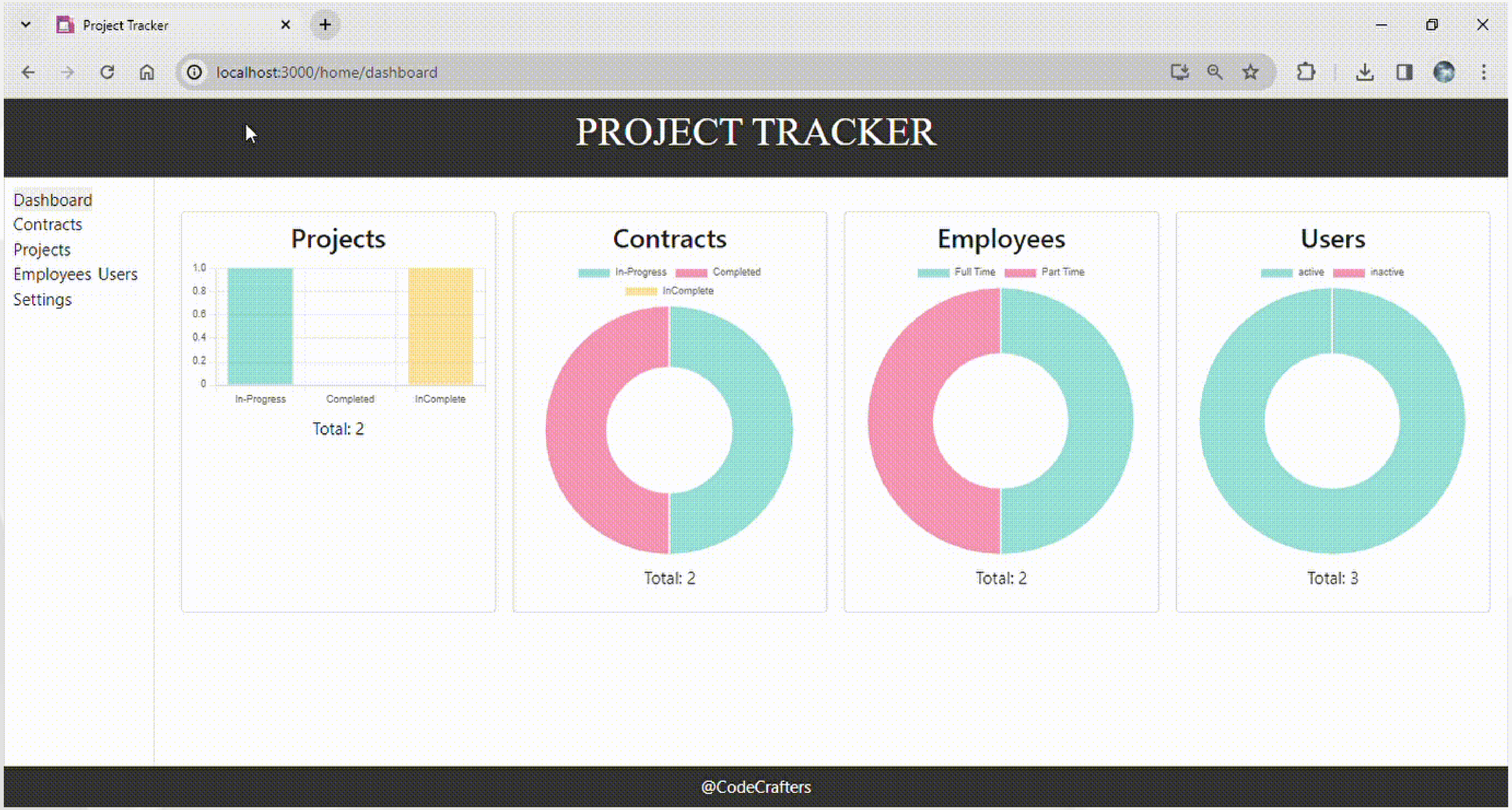


```
// Fetch dashboard
const fetchData = a
  try {
    // Updates comp
    let response =
    setData(respons
    projectData = a
    setProjectData(
    contractData =
    setContractData
    usersData = awa
    setUserData(use
    employeeData =
    setEmployeeData
  } catch (error) {
    console.error("
  }
};

useEffect(() => {
  Chart.register(Ca
  fetchData();
}, []);
```

```
*/
const prep
  const ar
  let cnt
    const
    return
  });
  return {
    labels
    datase
      {
        la
        da
        ba




        ],
      },
    ],
  };
};
```

```
// returns all the related data with the total count */
return (
  <div className="dashboard-container">
    {data && (
      <div className="card-container">
        <div className="card">
          <h2>Projects</h2>
          {projectData && <Bar data={projectData} options={chartOptions} />}
          <div className="chart-label">
            <p>
              Total:{" "}
              {projectData?.datasets?.[0]?.data?.reduce((acc, val) => acc + val, 0)}
            </p>
          </div>
        </div>
        <div className="card">
          <h2>Contracts</h2>
          {contractData && <Doughnut data={contractData} />}
          <div className="chart-label">
            <p>
              Total:{" "}
```

Evaluation criteria



Projects Component

Introduction:

- Projects tab serves as a central hub for managing project details within our organizational project management platform.
- It consolidates all project related information, offering a single point of access to stay updated on projects status and details.
- Comprehensive functionality and performing tasks seamlessly: Provide the ability to add, edit, delete and view the projects across the organization.
- Real-time data updates.
- User friendly interface and display data in tabular format.

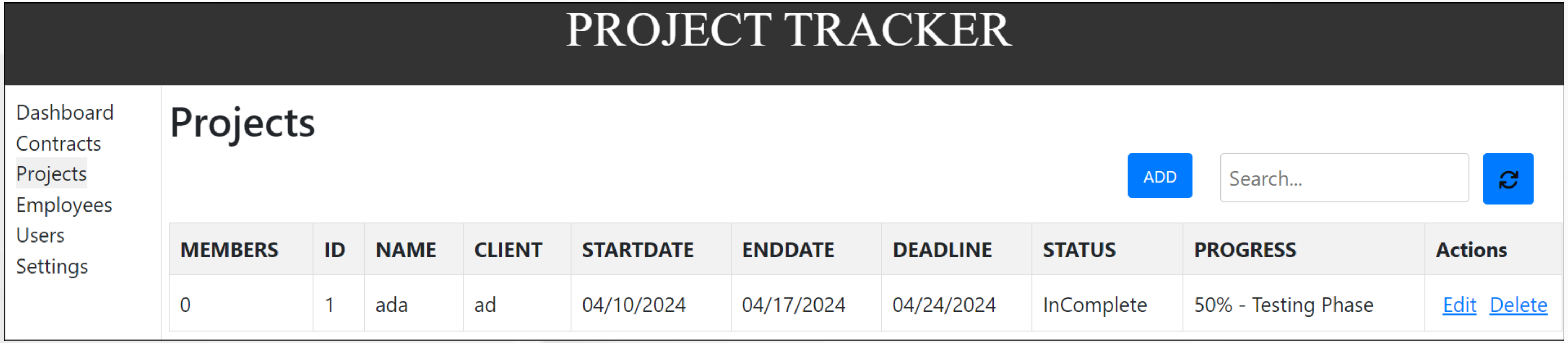
User Story:

- As a user, employee, or admin in the organization, I want to efficiently access, manage(add, edit, delete) and track project details.
- As a user, I require critical project information to be presented in a user-friendly tabular format, including project name, dates, members, progress, and status.
- As a user, it is crucial that certain project details are unique, preventing duplicate entries and ensuring data integrity.
- As a user, I expect a responsive and friendly user interface, with complete with features like search, confirmation dialogs, and feedback messages to facilitate seamless project management.

Projects Component – Requirements

- ❑ Projects Listing:
- Display all project records in the tabular view.
- Include key data items ID, Name, members, client, Start/End date, deadline, progress & status, actions.
- ❑ Action Buttons:
- Each project record provides an action buttons for user to edit and delete them.
- Search box to find specific information.
- Add button action to create a new project record and update in database.
- ❑ The projects component presents projects details dynamically, reflecting the changes in the database in the real time.
- ❑ The ID and name of the project should be unique.
- ❑ Members field should represent the number of people involved in the project.

Projects Tab UI

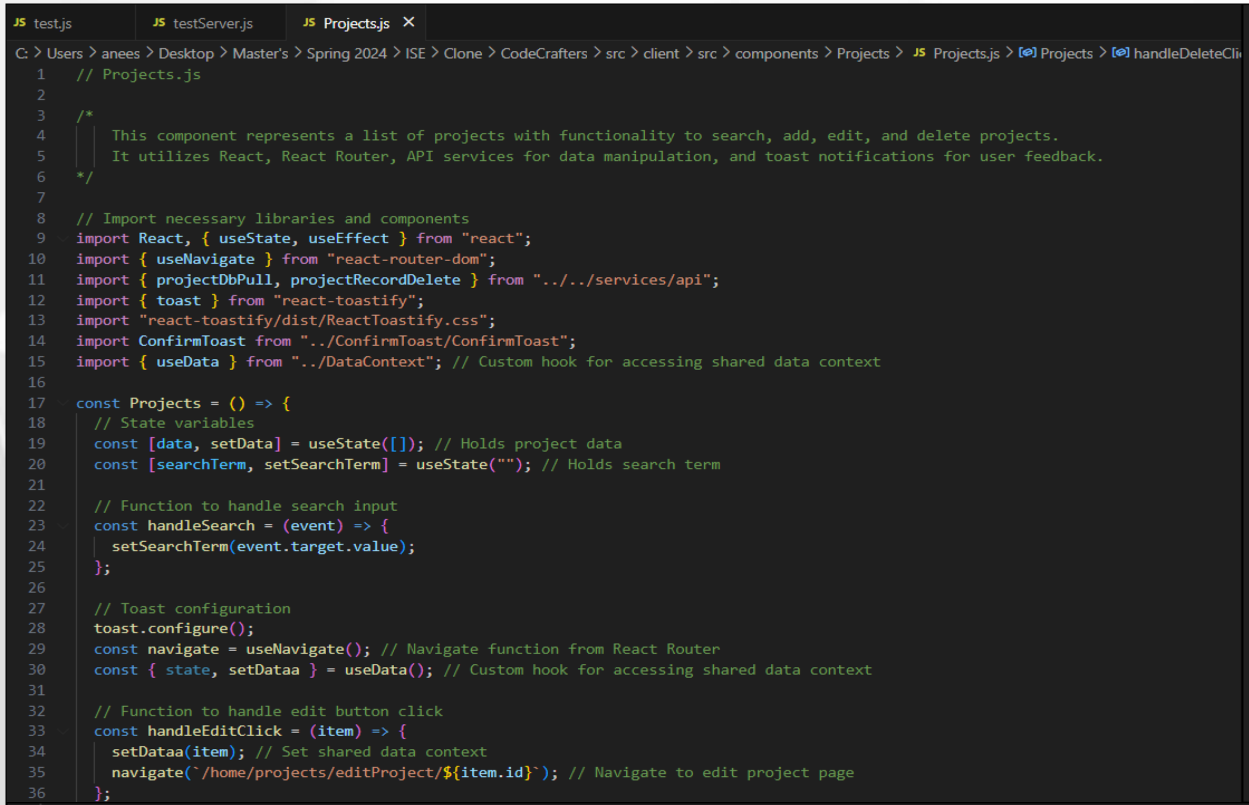


Projects

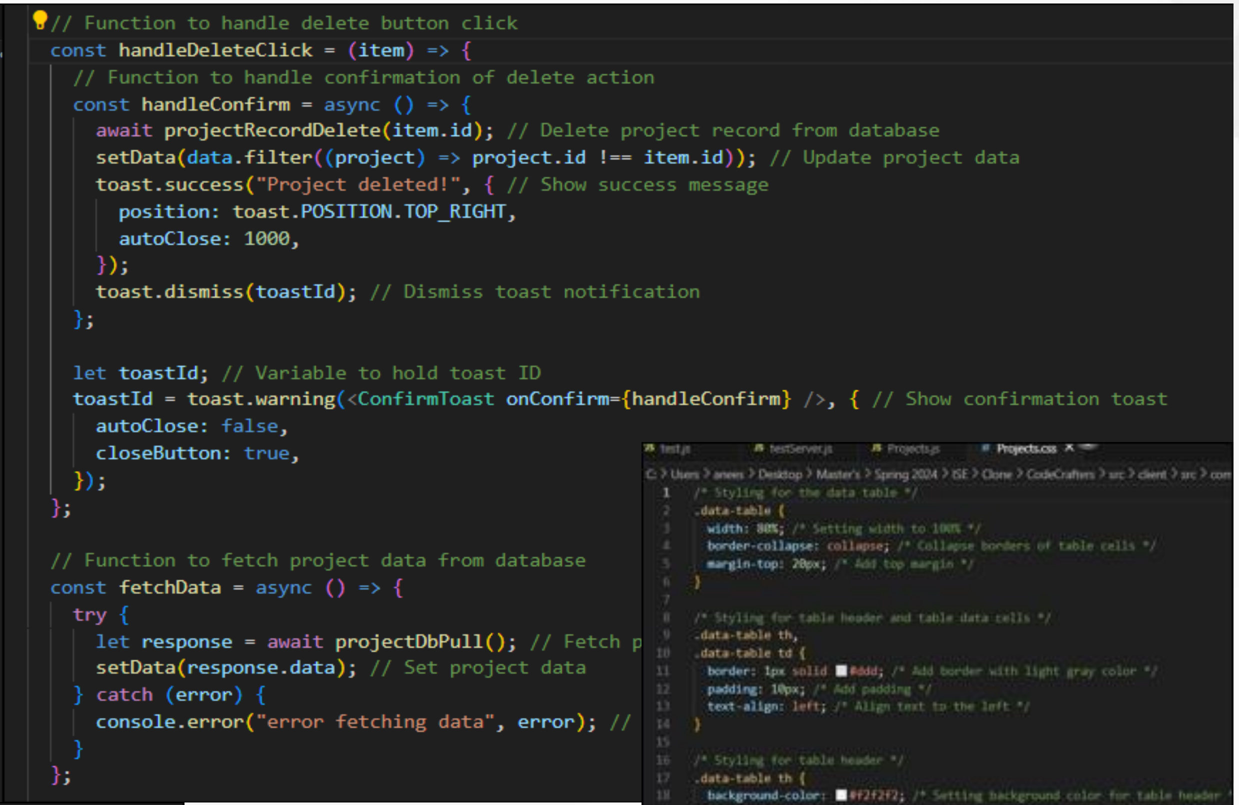
MEMBERS	ID	NAME	CLIENT	STARTDATE	ENDDATE	DEADLINE	STATUS	PROGRESS	Actions
0	1	ada	ad	04/10/2024	04/17/2024	04/24/2024	InComplete	50% - Testing Phase	Edit Delete

Project Tracker

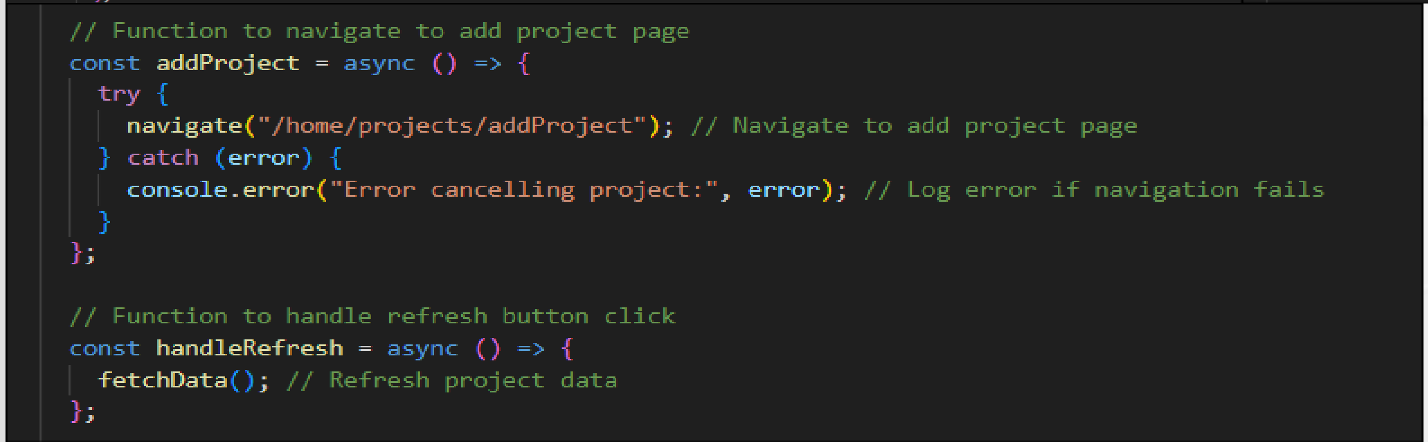
Code Snippet


test.js
testServer.js
Projects.js
C: > Users > anees > Desktop > Master's > Spring 2024 > ISE > Clone > CodeCrafters > src > client > src > components > Projects > Projects.js > Projects > handleDeleteCli

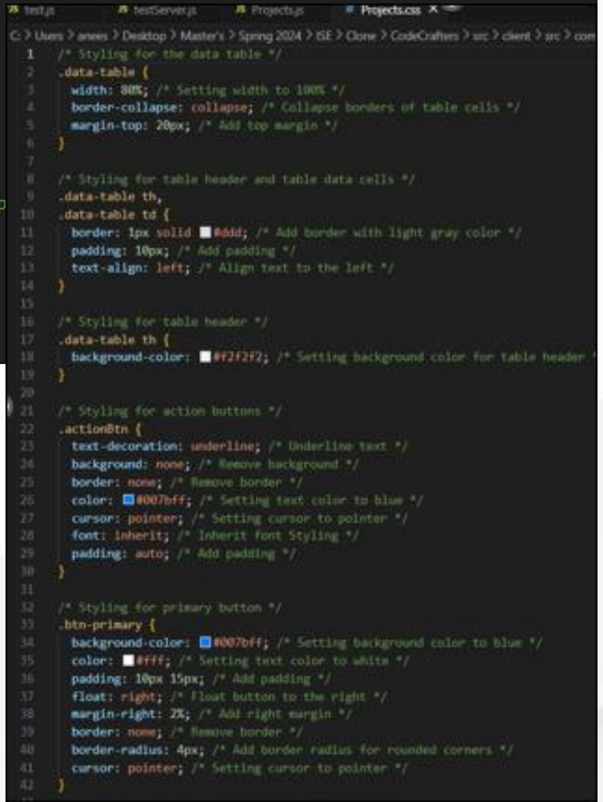
```javascript
// Projects.js

/*
  This component represents a list of projects with functionality to search, add, edit, and delete projects.
  It utilizes React, React Router, API services for data manipulation, and toast notifications for user feedback.
*/

// Import necessary libraries and components
import React, { useState, useEffect } from "react";
import { useNavigate } from "react-router-dom";
import { projectDbPull, projectRecordDelete } from "../../services/api";
import { toast } from "react-toastify";
import "react-toastify/dist/ReactToastify.css";
import ConfirmToast from "../ConfirmToast/ConfirmToast";
import { useData } from "../DataContext"; // Custom hook for accessing shared data context

const Projects = () => {
  // State variables
  const [data, setData] = useState([]); // Holds project data
  const [searchTerm, setSearchTerm] = useState(""); // Holds search term

  // Function to handle search input
  const handleSearch = (event) => {
    setSearchTerm(event.target.value);
  };

  // Toast configuration
  toast.configure();
  const navigate = useNavigate(); // Navigate function from React Router
  const { state, setDataa } = useData(); // Custom hook for accessing shared data context

  // Function to handle edit button click
  const handleEditClick = (item) => {
    setDataa(item); // Set shared data context
    navigate(`/home/projects/editProject/${item.id}`); // Navigate to edit project page
  };
```



```javascript
// Function to handle delete button click
const handleDeleteClick = (item) => {
  // Function to handle confirmation of delete action
  const handleConfirm = async () => {
    await projectRecordDelete(item.id); // Delete project record from database
    setData(data.filter((project) => project.id !== item.id)); // Update project data
    toast.success("Project deleted!", { // Show success message
      position: toast.POSITION.TOP_RIGHT,
      autoClose: 1000,
    });
    toast.dismiss(toastId); // Dismiss toast notification
  };

  let toastId; // Variable to hold toast ID
  toastId = toast.warning(<ConfirmToast onConfirm={handleConfirm} />, { // Show confirmation toast
    autoClose: false,
    closeButton: true,
  });
};

// Function to fetch project data from database
const fetchData = async () => {
  try {
    let response = await projectDbPull(); // Fetch p
    setData(response.data); // Set project data
  } catch (error) {
    console.error("error fetching data", error); //
  }
};
```



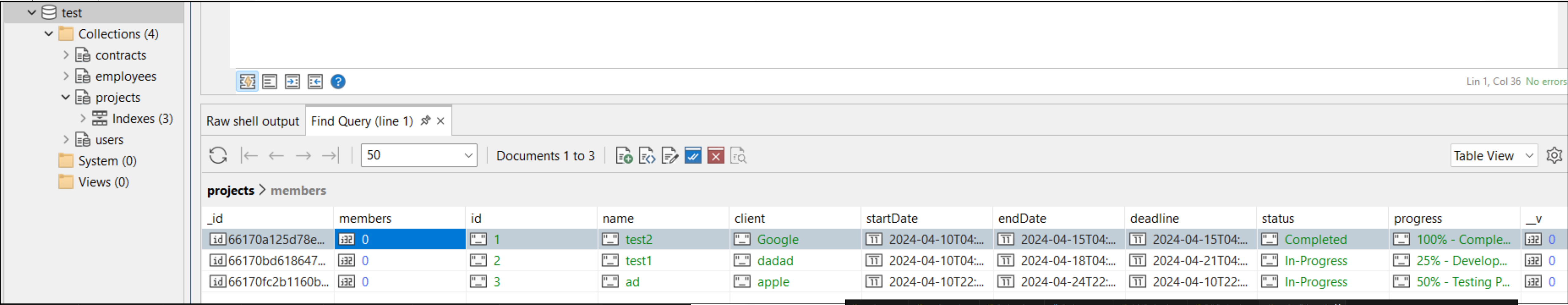
```javascript
// Function to navigate to add project page
const addProject = async () => {
  try {
    navigate("/home/projects/addProject"); // Navigate to add project page
  } catch (error) {
    console.error("Error cancelling project:", error); // Log error if navigation fails
  }
};

// Function to handle refresh button click
const handleRefresh = async () => {
  fetchData(); // Refresh project data
};
```


test.js
testServer.js
Projects.js
Projects.css
C: > Users > anees > Desktop > Master's > Spring 2024 > ISE > Clone > CodeCrafters > src > client > src > com

```css
/* Styling for the data table */
.data-table {
  width: 80%; /* Setting width to 100% */
  border-collapse: collapse; /* Collapse borders of table cells */
  margin-top: 20px; /* Add top margin */
}

/* Styling for table header and table data cells */
.data-table th,
.data-table td {
  border: 1px solid #ddd; /* Add border with light gray color */
  padding: 10px; /* Add padding */
  text-align: left; /* Align text to the left */
}

/* Styling for table header */
.data-table th {
  background-color: #f2f2f2; /* Setting background color for table header *
}

/* Styling for action buttons */
.actionBtn {
  text-decoration: underline; /* Underline text */
  background: none; /* Remove background */
  border: none; /* Remove border */
  color: #007bff; /* Setting text color to blue */
  cursor: pointer; /* Setting cursor to pointer */
  font: inherit; /* Inherit font Styling */
  padding: auto; /* Add padding */
}

/* Styling for primary button */
.btn-primary {
  background-color: #007bff; /* Setting background color to blue */
  color: #fff; /* Setting text color to white */
  padding: 10px 15px; /* Add padding */
  float: right; /* Float button to the right */
  margin-right: 2%; /* Add right margin */
  border: none; /* Remove border */
  border-radius: 4px; /* Add border radius for rounded corners */
  cursor: pointer; /* Setting cursor to pointer */
}
```

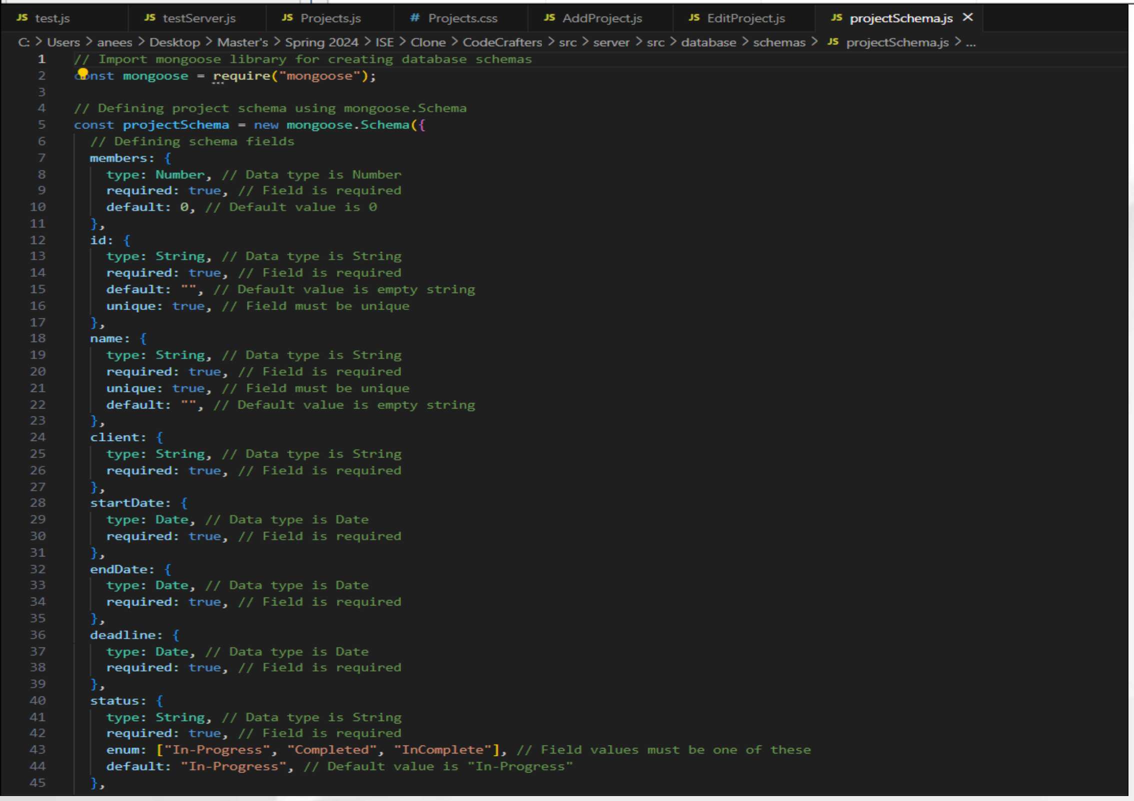
Project Tracker

Database Schema

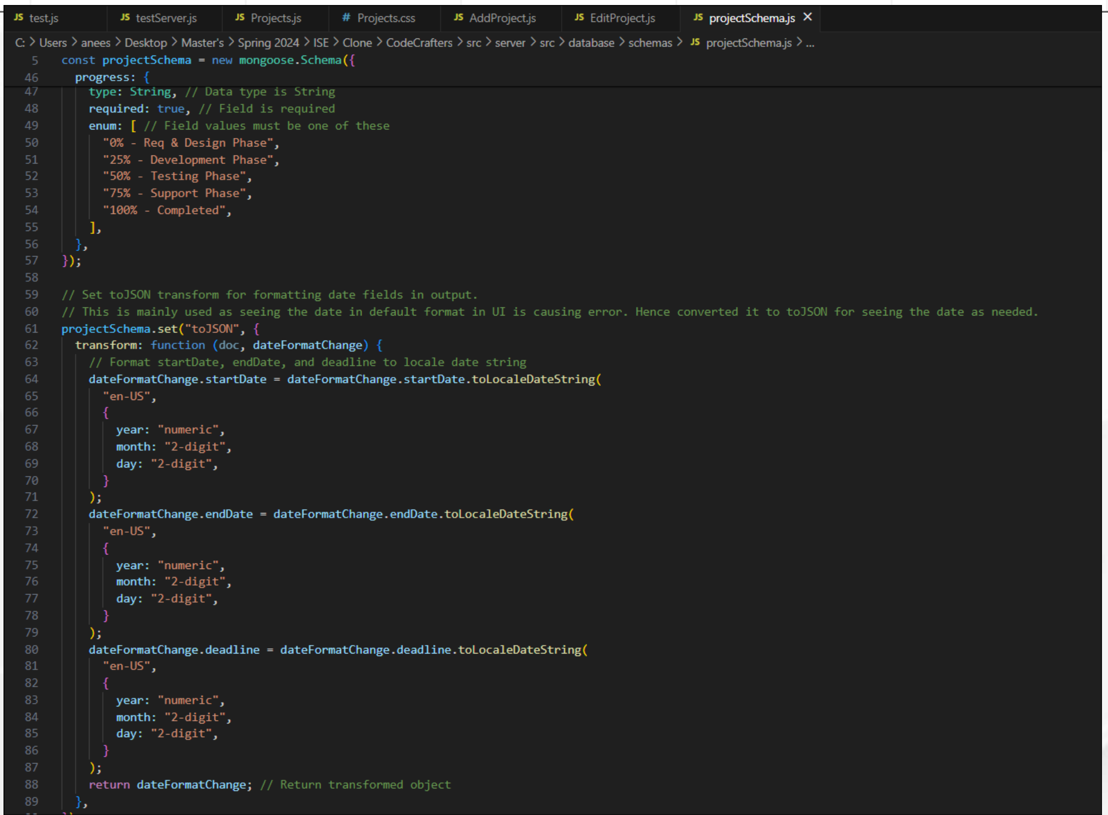


projects > members

_id	members	id	name	client	startDate	endDate	deadline	status	progress	__v
66170a125d78e...	0	1	test2	Google	2024-04-10T04:...	2024-04-15T04:...	2024-04-15T04:...	Completed	100% - Comple...	0
66170bd618647...	0	2	test1	dadad	2024-04-10T04:...	2024-04-18T04:...	2024-04-21T04:...	In-Progress	25% - Develop...	0
66170fc2b1160b...	0	3	ad	apple	2024-04-10T22:...	2024-04-24T22:...	2024-04-10T22:...	In-Progress	50% - Testing P...	0


test.js
testServer.js
Projects.js
Projects.css
AddProject.js
EditProject.js
projectSchema.js
C: > Users > anees > Desktop > Master's > Spring 2024 > ISE > Clone > CodeCrafters > src > server > src > database > schemas > projectSchema.js > ...

```javascript
// Import mongoose library for creating database schemas
const mongoose = require("mongoose");

// Defining project schema using mongoose.Schema
const projectSchema = new mongoose.Schema({
  // Defining schema fields
  members: {
    type: Number, // Data type is Number
    required: true, // Field is required
    default: 0, // Default value is 0
  },
  id: {
    type: String, // Data type is String
    required: true, // Field is required
    default: "", // Default value is empty string
    unique: true, // Field must be unique
  },
  name: {
    type: String, // Data type is String
    required: true, // Field is required
    unique: true, // Field must be unique
    default: "", // Default value is empty string
  },
  client: {
    type: String, // Data type is String
    required: true, // Field is required
  },
  startDate: {
    type: Date, // Data type is Date
    required: true, // Field is required
  },
  endDate: {
    type: Date, // Data type is Date
    required: true, // Field is required
  },
  deadline: {
    type: Date, // Data type is Date
    required: true, // Field is required
  },
  status: {
    type: String, // Data type is String
    required: true, // Field is required
    enum: ["In-Progress", "Completed", "InComplete"], // Field values must be one of these
    default: "In-Progress", // Default value is "In-Progress"
  },
```

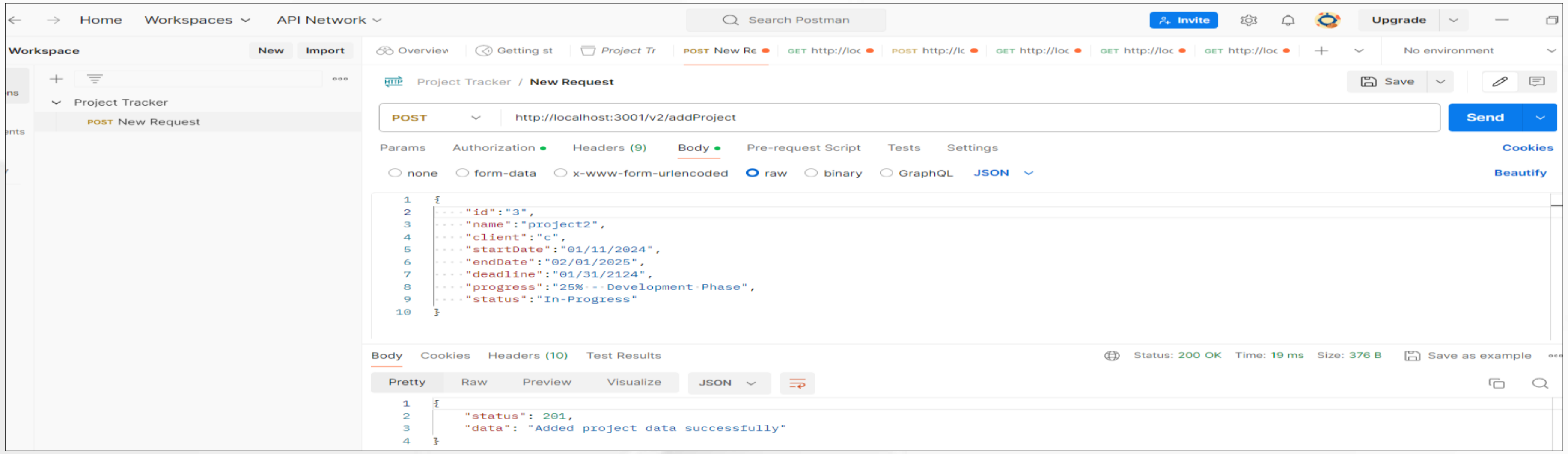

test.js
testServer.js
Projects.js
Projects.css
AddProject.js
EditProject.js
projectSchema.js
C: > Users > anees > Desktop > Master's > Spring 2024 > ISE > Clone > CodeCrafters > src > server > src > database > schemas > projectSchema.js > ...

```javascript
const projectSchema = new mongoose.Schema({
  progress: {
    type: String, // Data type is String
    required: true, // Field is required
    enum: [ // Field values must be one of these
      "0% - Req & Design Phase",
      "25% - Development Phase",
      "50% - Testing Phase",
      "75% - Support Phase",
      "100% - Completed",
    ],
  },
});

// Set toJSON transform for formatting date fields in output.
// This is mainly used as seeing the date in default format in UI is causing error. Hence converted it to toJSON for seeing the date as needed.
projectSchema.set("toJSON", {
  transform: function (doc, dateFormatChange) {
    // Format startDate, endDate, and deadline to locale date string
    dateFormatChange.startDate = dateFormatChange.startDate.toLocaleDateString(
      "en-US",
      {
        year: "numeric",
        month: "2-digit",
        day: "2-digit",
      }
    );
    dateFormatChange.endDate = dateFormatChange.endDate.toLocaleDateString(
      "en-US",
      {
        year: "numeric",
        month: "2-digit",
        day: "2-digit",
      }
    );
    dateFormatChange.deadline = dateFormatChange.deadline.toLocaleDateString(
      "en-US",
      {
        year: "numeric",
        month: "2-digit",
        day: "2-digit",
      }
    );
    return dateFormatChange; // Return transformed object
  },
```

Project Tracker

Projects API

To add a project
API Type : REST
Method : POST
End point : /v2/addProject
Authorization : Bearer Auth



Project Tracker / New Request

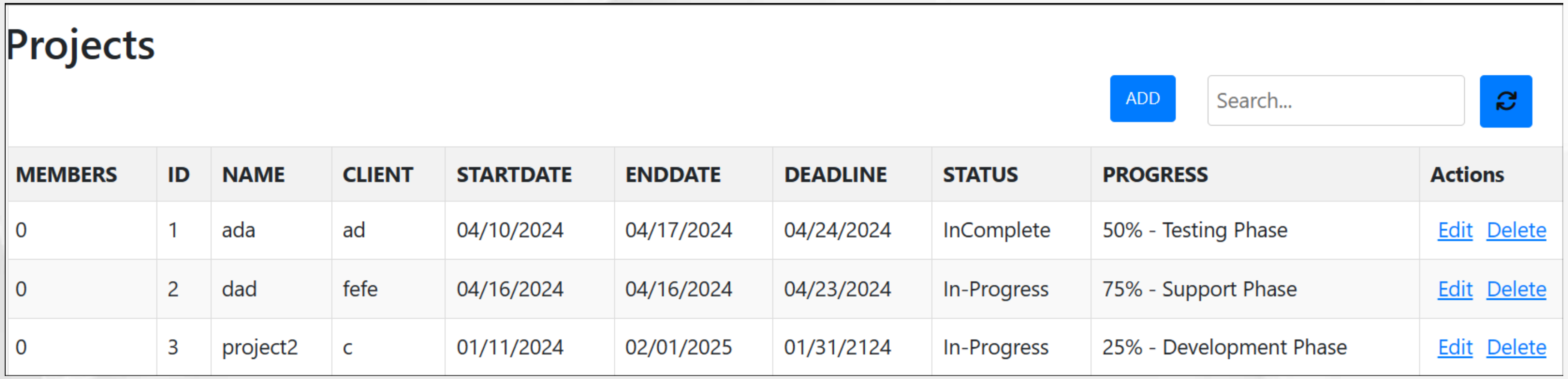
POST http://localhost:3001/v2/addProject

Body: none / form-data / x-www-form-urlencoded / [✓ raw] / binary / GraphQL — JSON

```
{
    "id":"3",
    "name":"project2",
    "client":"c",
    "startDate":"01/11/2024",
    "endDate":"02/01/2025",
    "deadline":"01/31/2124",
    "progress":"25% - Development Phase",
    "status":"In-Progress"
}
```

Body | Cookies | Headers (10) | Test Results — Status: 200 OK Time: 19 ms Size: 376 B

```
{
    "status": 201,
    "data": "Added project data successfully"
}
```

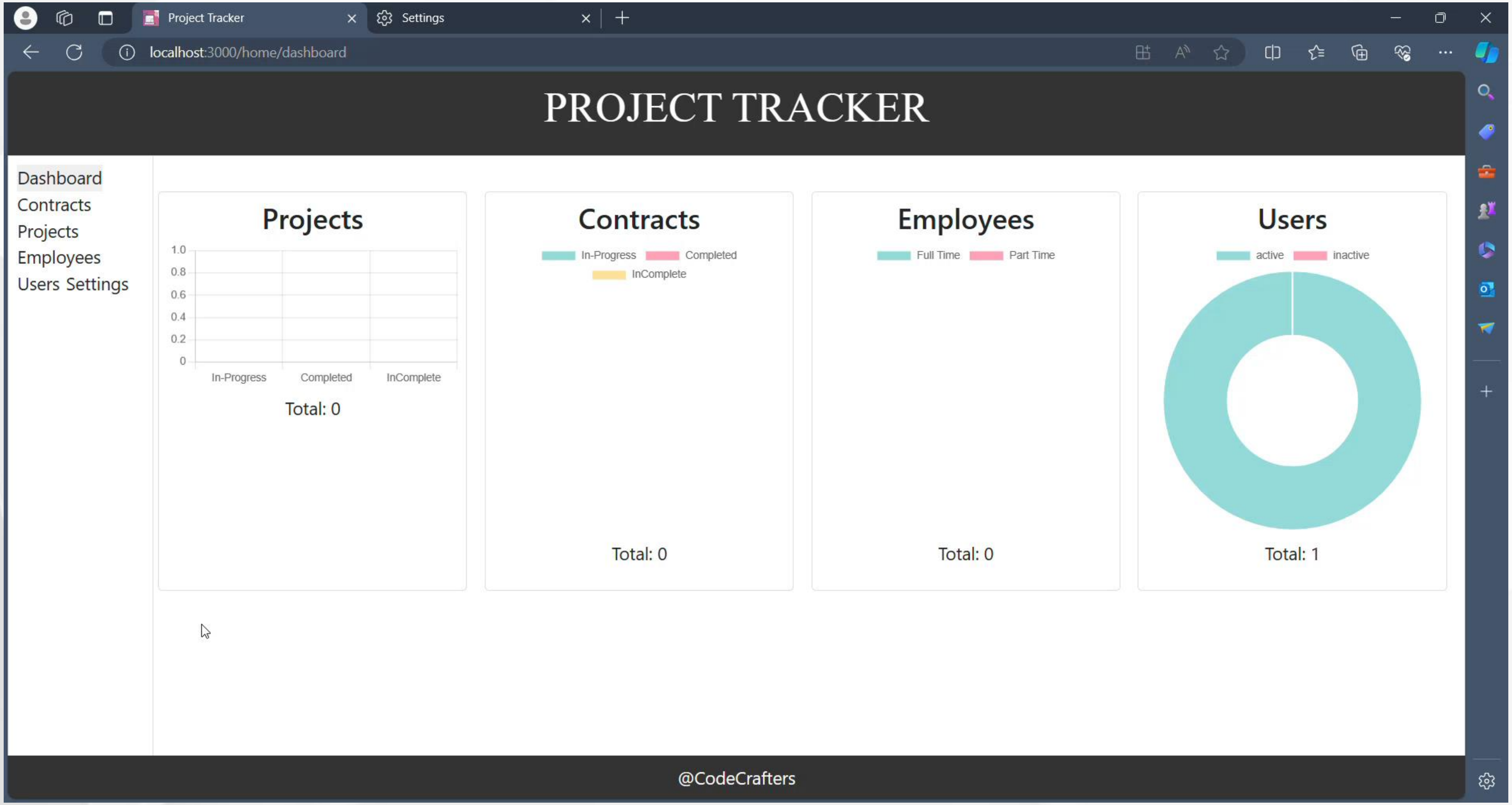


Projects

MEMBERS	ID	NAME	CLIENT	STARTDATE	ENDDATE	DEADLINE	STATUS	PROGRESS	Actions
0	1	ada	ad	04/10/2024	04/17/2024	04/24/2024	InComplete	50% - Testing Phase	Edit Delete
0	2	dad	fefe	04/16/2024	04/16/2024	04/23/2024	In-Progress	75% - Support Phase	Edit Delete
0	3	project2	c	01/11/2024	02/01/2025	01/31/2124	In-Progress	25% - Development Phase	Edit Delete

Project Tracker

Projects Tab


Project Tracker
Settings
localhost:3000/home/dashboard

PROJECT TRACKER

Dashboard
Contracts
Projects
Employees
Users Settings

Projects

Total: 0

Contracts

Total: 0

Employees

Total: 0

Users

Total: 1

@CodeCrafters

Contracts Component

Introduction

The Contracts component serves as a dynamic contract management solution. It offers a table-top representation of contracts with search capabilities for convenient browsing. Contracts may be added, edited, and deleted by users, and the table reflects any real-time adjustments. This guarantees a current, well-organized, and easily accessible contract management platform.

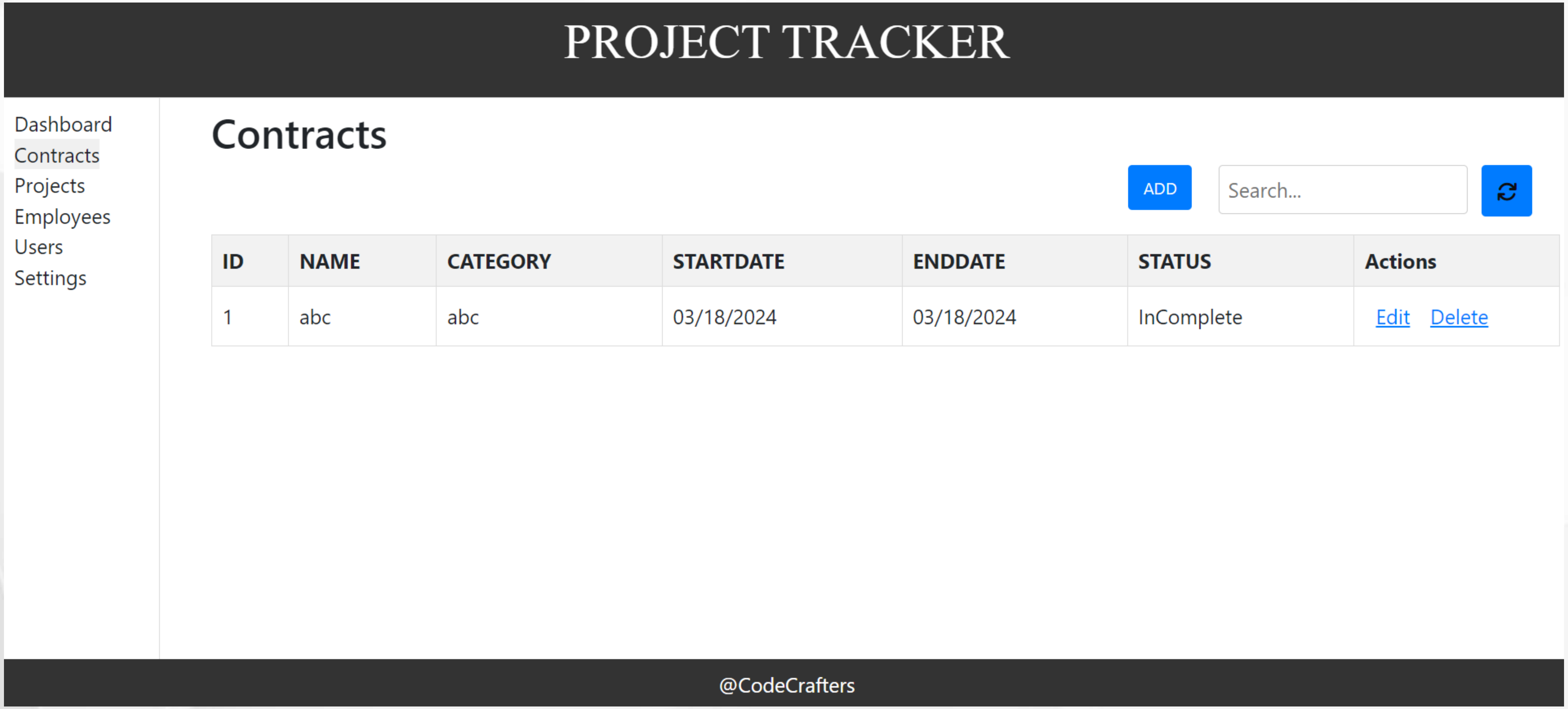
Contracts Component User Stories

- List Contracts: As a user, I want to see all contracts in a table format, so that I can easily navigate through them.
- Search Contracts: As a user, I want to quickly find contracts by typing into a search box and see dynamic results, so that I can get instant feedback.
- Add New Contract: As a user, I want to add new contracts using a form and expect the system to validate my data so that I can ensure its correctness and completeness.
- Edit Contracts: As a user, I want to edit contracts using a form that pops up when I click an "Edit" button so that I can easily modify contract details.
- Delete Contracts: As a user, I want to delete contracts with a confirmation step, so that I can prevent unintentional deletions.
- Real-time Updates: As a user, I want the table to be immediately updated after I edit or delete a contract so that I can see the current state of the contracts.

Contracts Component Requirements

- List Contracts: Display contracts in a tabular format for easy navigation.
- Search Functionality: Provide a search box for users to find contracts by name with dynamic results.
- Add New Contract: Include an "Add New Contract" button or link and a form for data entry. Validate the data for correctness and completeness.
- Edit Contracts: Allow users to edit contracts using an "Edit" button or editable field in each contract row.
- Delete Contracts: Include a "Delete" button or checkbox in each contract row. Ask users to confirm before deletion.
- Real-time Updates: Update the table immediately after a contract is edited or deleted.

Contracts Component UI



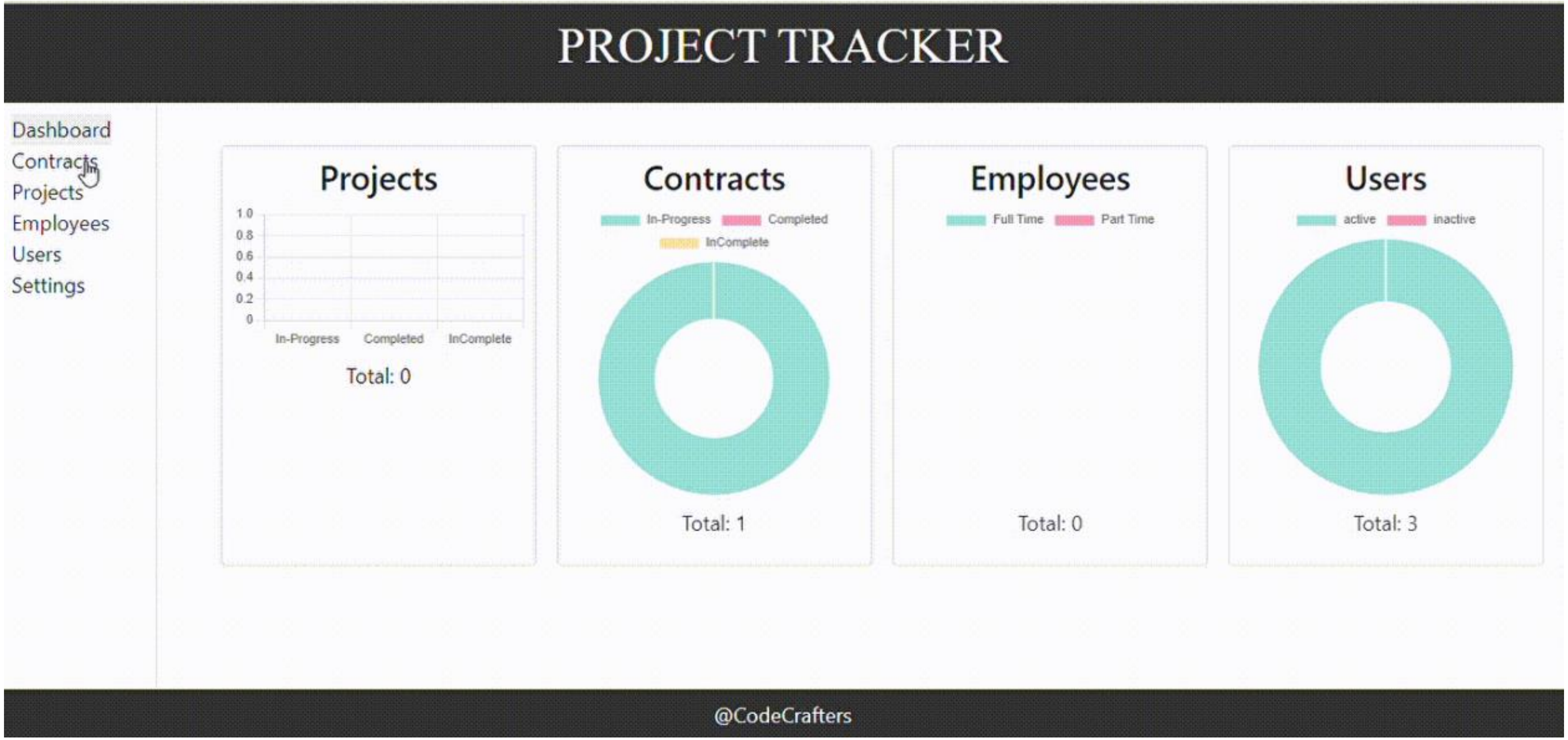
PROJECT TRACKER

Contracts

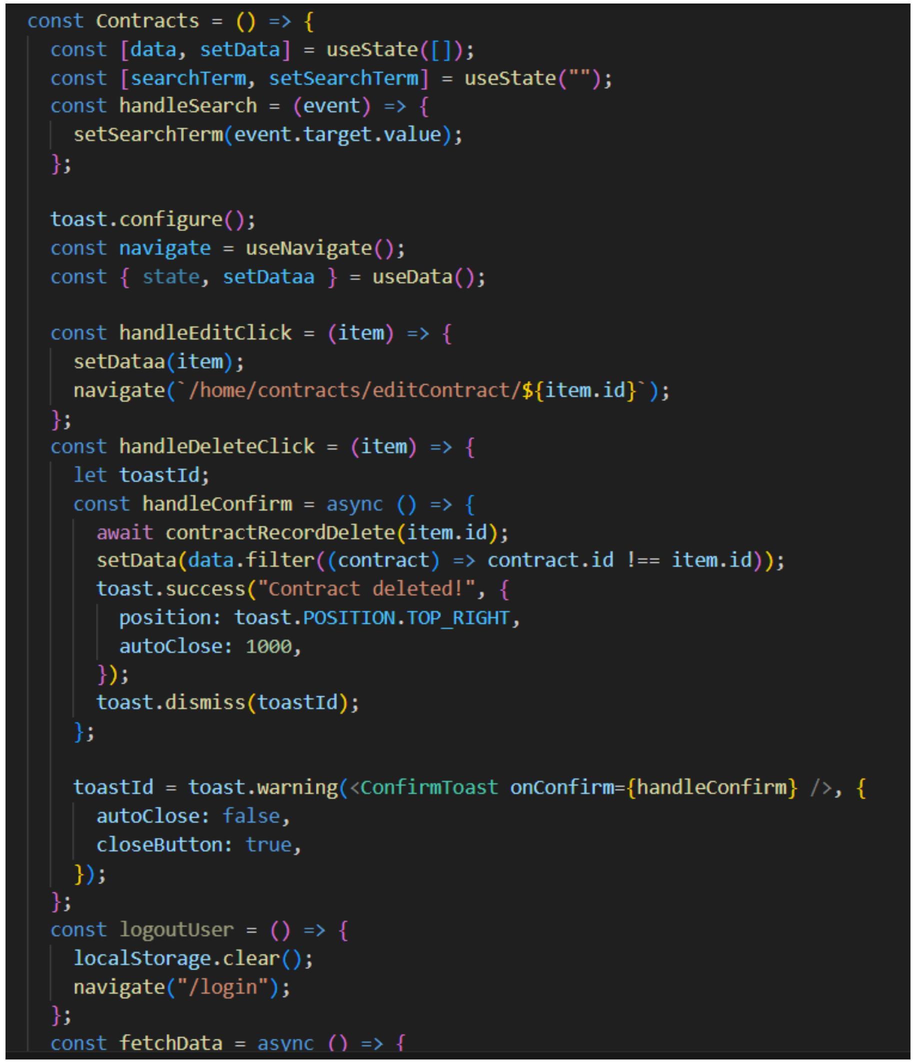
ID	NAME	CATEGORY	STARTDATE	ENDDATE	STATUS	Actions
1	abc	abc	03/18/2024	03/18/2024	InComplete	Edit Delete

@CodeCrafters

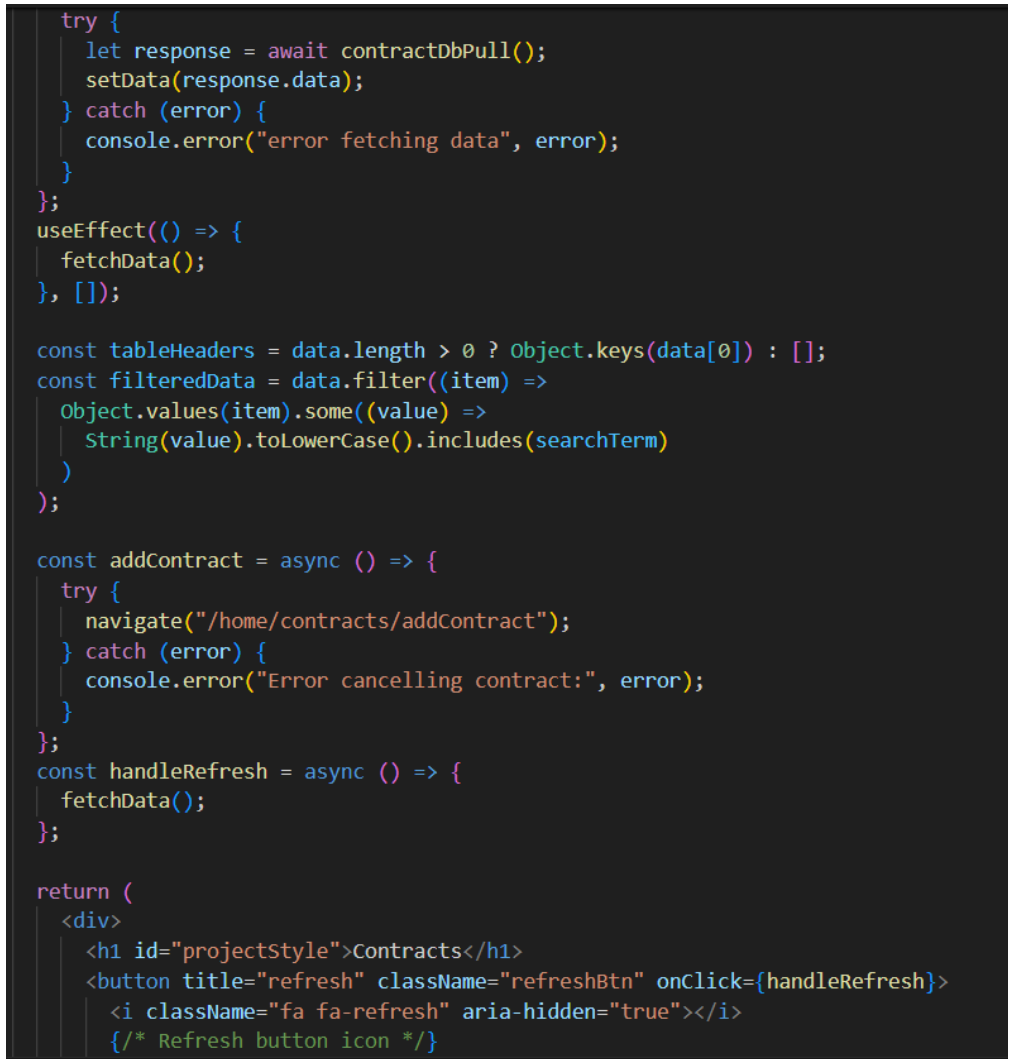
Contracts Component UI



Contracts Component Code



```
const Contracts = () => {
  const [data, setData] = useState([]);
  const [searchTerm, setSearchTerm] = useState("");
  const handleSearch = (event) => {
    setSearchTerm(event.target.value);
  };

  toast.configure();
  const navigate = useNavigate();
  const { state, setDataa } = useData();

  const handleEditClick = (item) => {
    setDataa(item);
    navigate(`/home/contracts/editContract/${item.id}`);
  };
  const handleDeleteClick = (item) => {
    let toastId;
    const handleConfirm = async () => {
      await contractRecordDelete(item.id);
      setData(data.filter((contract) => contract.id !== item.id));
      toast.success("Contract deleted!", {
        position: toast.POSITION.TOP_RIGHT,
        autoClose: 1000,
      });
      toast.dismiss(toastId);
    };

    toastId = toast.warning(<ConfirmToast onConfirm={handleConfirm} />, {
      autoClose: false,
      closeButton: true,
    });
  };
  const logoutUser = () => {
    localStorage.clear();
    navigate("/login");
  };
  const fetchData = async () => {
```



```
    try {
      let response = await contractDbPull();
      setData(response.data);
    } catch (error) {
      console.error("error fetching data", error);
    }
  };
  useEffect(() => {
    fetchData();
  }, []);

  const tableHeaders = data.length > 0 ? Object.keys(data[0]) : [];
  const filteredData = data.filter((item) =>
    Object.values(item).some((value) =>
      String(value).toLowerCase().includes(searchTerm)
    )
  );

  const addContract = async () => {
    try {
      navigate("/home/contracts/addContract");
    } catch (error) {
      console.error("Error cancelling contract:", error);
    }
  };
  const handleRefresh = async () => {
    fetchData();
  };

  return (
    <div>
      <h1 id="projectStyle">Contracts</h1>
      <button title="refresh" className="refreshBtn" onClick={handleRefresh}>
        <i className="fa fa-refresh" aria-hidden="true"></i>
        {/* Refresh button icon */}
```

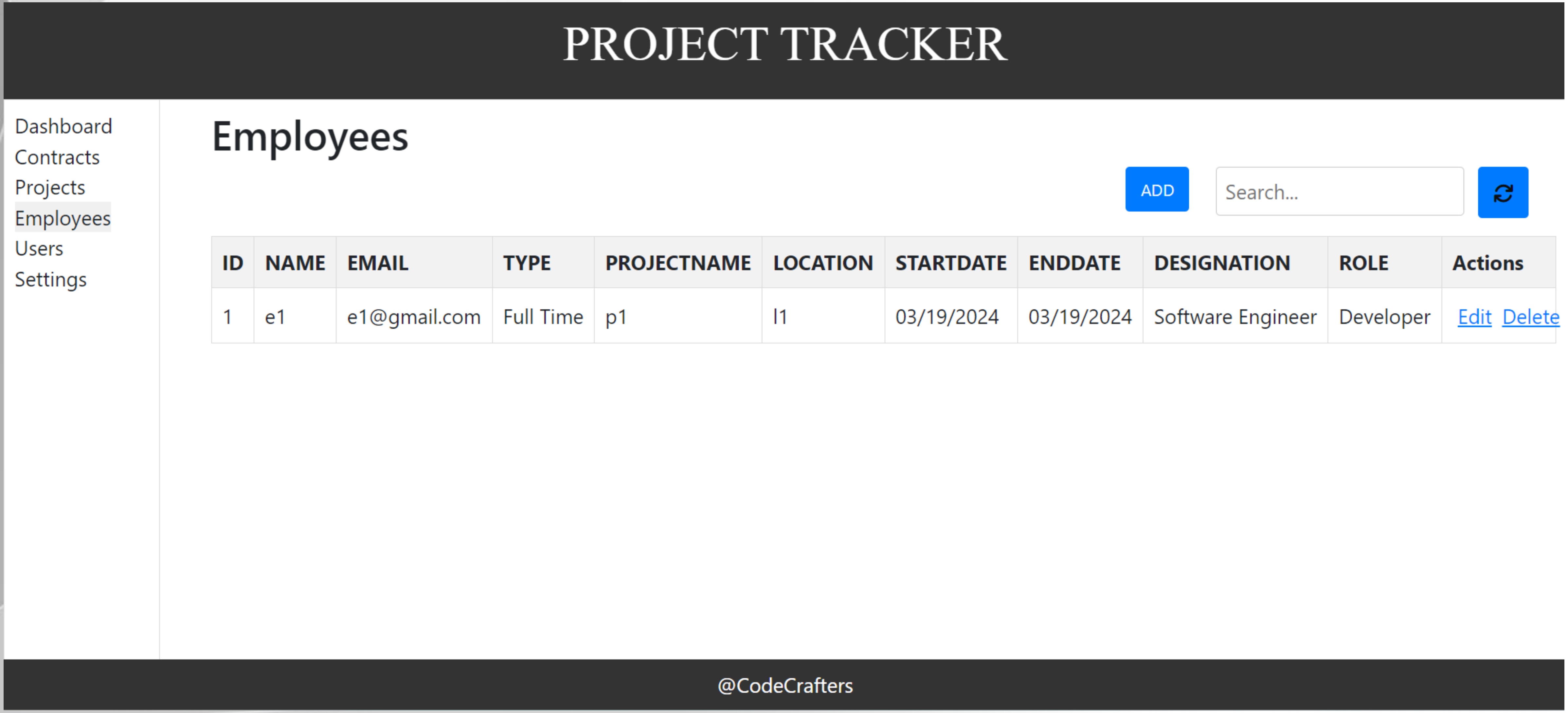
Employees Component

- Purpose of the Employees component.
- Ability to modify, delete, and create employee details.
- Immediate updates to the database.
- Data synchronization for real-time updates.
- Managers:
  - Admin role with the ability to modify employee details.
  - Can update employee projects, location, allocation dates, and roles.
- Other Users:
  - Can view employee details and updates in real-time.

Employee Component User Story

- As a user, I want to see a list of all employees in a table format.
- As a user, I want to search for employees by entering their names in a search box.
- As a user, I want to add a new employee to the system.
- As a user, I want to edit the details of an existing employee.
- As a user, I want to delete an existing employee.
- As a user, I want the employee table to update in real-time after edit or delete operations.

Employee UI



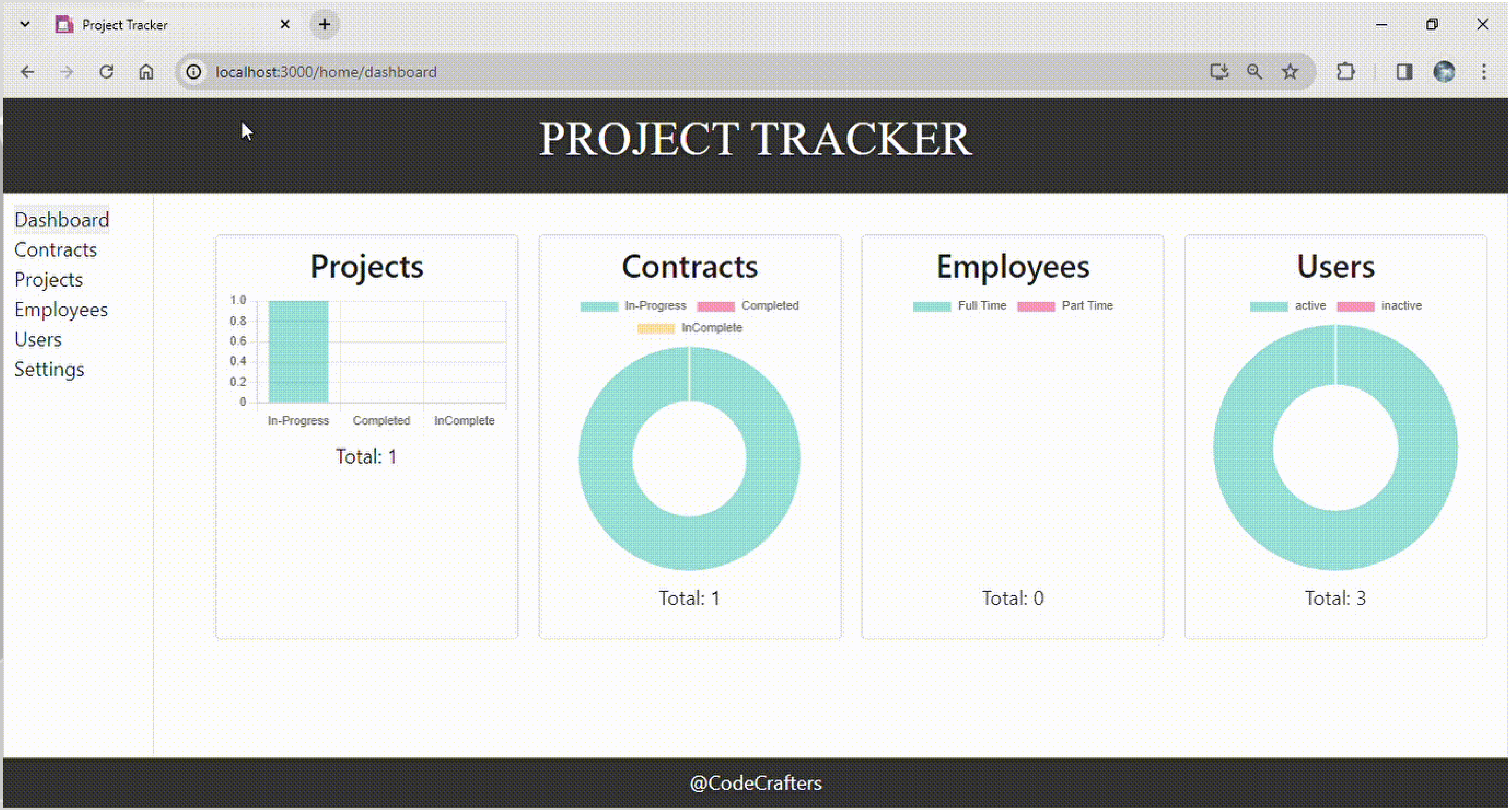
PROJECT TRACKER

Employees

ID	NAME	EMAIL	TYPE	PROJECTNAME	LOCATION	STARTDATE	ENDDATE	DESIGNATION	ROLE	Actions
1	e1	e1@gmail.com	Full Time	p1	l1	03/19/2024	03/19/2024	Software Engineer	Developer	Edit Delete

@CodeCrafters

Employee UI


Project Tracker

localhost:3000/home/dashboard

PROJECT TRACKER

- Dashboard
- Contracts
- Projects
- Employees
- Users
- Settings

Projects

Total: 1

Contracts

Total: 1

Employees

Total: 0

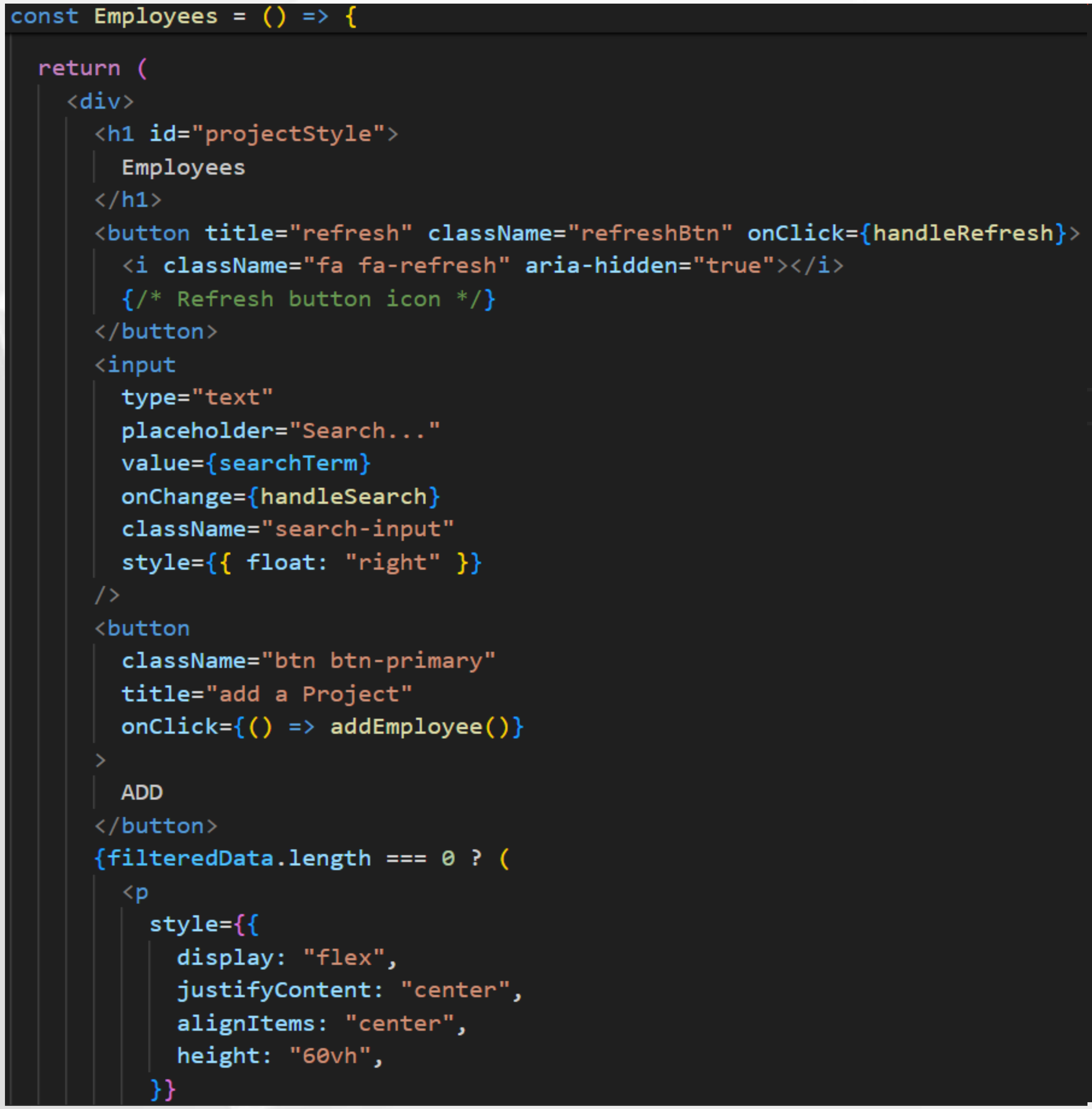
Users

Total: 3

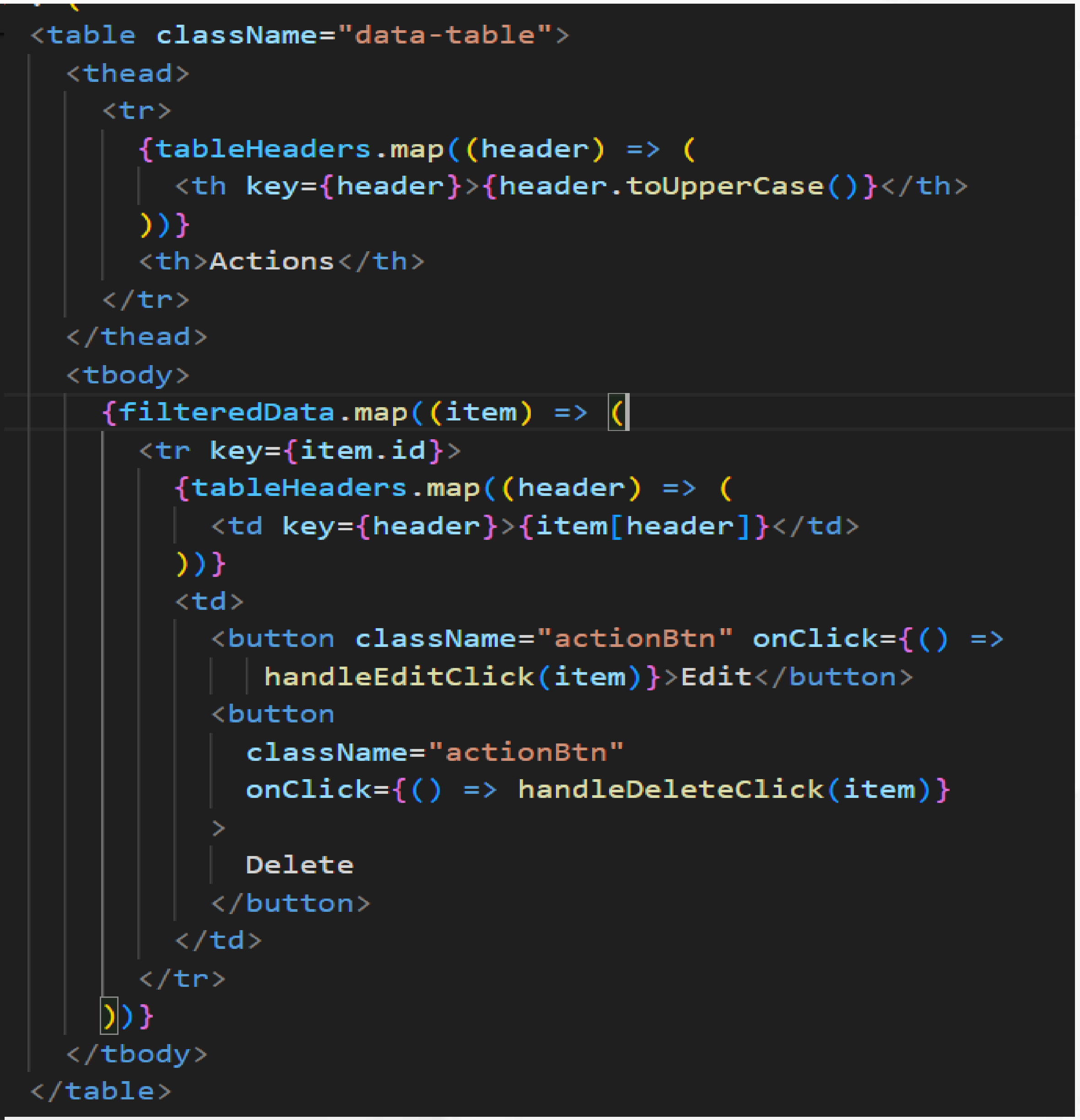
@CodeCrafters

Project Tracker

Code Snippet



```
const Employees = () => {

  return (
    <div>
      <h1 id="projectStyle">
        Employees
      </h1>
      <button title="refresh" className="refreshBtn" onClick={handleRefresh}>
        <i className="fa fa-refresh" aria-hidden="true"></i>
        {/* Refresh button icon */}
      </button>
      <input
        type="text"
        placeholder="Search..."
        value={searchTerm}
        onChange={handleSearch}
        className="search-input"
        style={{ float: "right" }}
      />
      <button
        className="btn btn-primary"
        title="add a Project"
        onClick={() => addEmployee()}
      >
        ADD
      </button>
      {filteredData.length === 0 ? (
        <p
          style={{
            display: "flex",
            justifyContent: "center",
            alignItems: "center",
            height: "60vh",
          }}
```

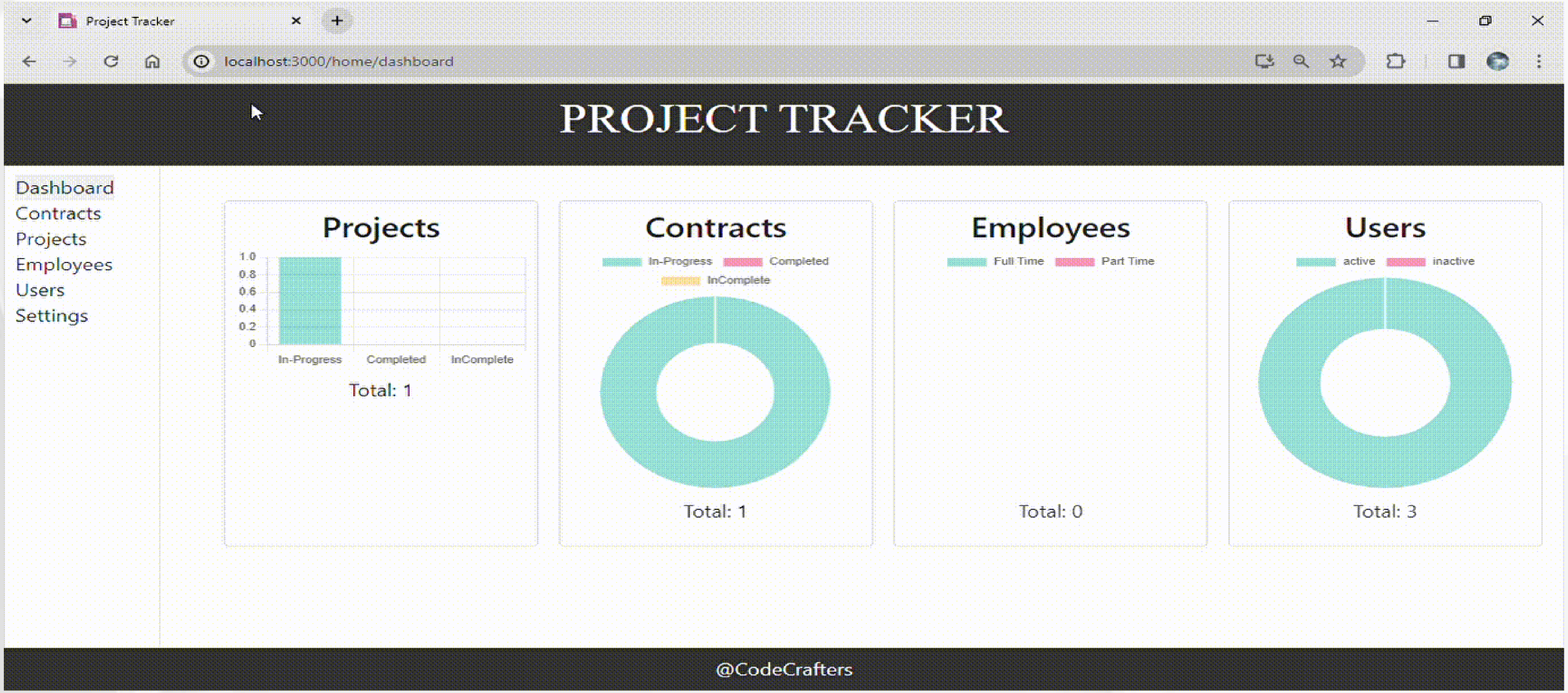


```
<table className="data-table">
  <thead>
    <tr>
      {tableHeaders.map((header) => (
        <th key={header}>{header.toUpperCase()}</th>
      ))}
      <th>Actions</th>
    </tr>
  </thead>
  <tbody>
    {filteredData.map((item) => (
      <tr key={item.id}>
        {tableHeaders.map((header) => (
          <td key={header}>{item[header]}</td>
        ))}
        <td>
          <button className="actionBtn" onClick={() =>
            handleEditClick(item)}>Edit</button>
          <button
            className="actionBtn"
            onClick={() => handleDeleteClick(item)}
          >
            Delete
          </button>
        </td>
      </tr>
    ))}
  </tbody>
</table>
```

Project Tracker

Evaluation Criteria

The table should display columns for ID, Name, Type, Project Name, Location, Allocation Start Date, Allocation End Date, Email, Designation, Role, and Actions (Edit, Delete).



Settings

Introduction

The user settings page will allow users to see/update personal information. All the changes act as a query to the database and the tables get modified accordingly

Requirements

- Profile Display: Shows the user's current information:
    - Username (read-only)
    - Email Address
    - First Name
    - Last Name
- Edit Profile:
  - Edit Button: Triggers a form to edit user data.
  - Edit Form:
    - Pre-populated with existing data for:
    - Email Address (editable)
    - First Name (editable)
    - Last Name (editable)
    - Save Button: Submits changes and updates displayed information.
    - Logout: Provides a safe way for users to log out of the platform

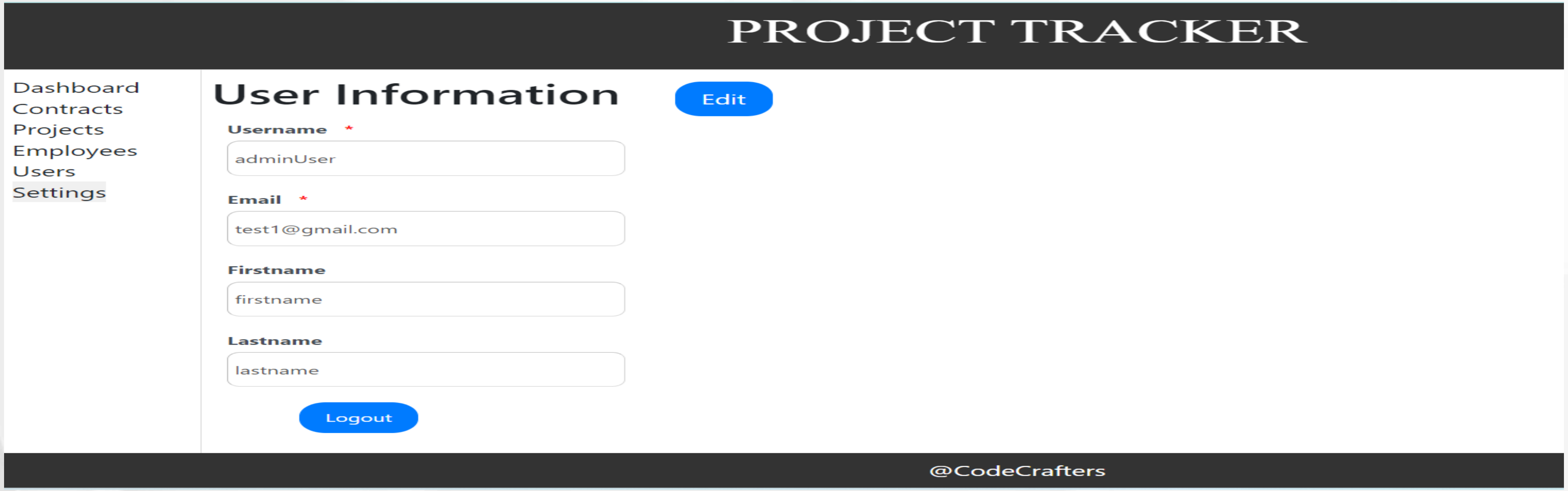
Project Tracker

Settings User Story

As a registered user, I want to view my current profile information, including username, email address, first name, and last name, so that I can easily see and manage my account details.

As a registered user, I want to edit my profile information, including email address, first name, and last name, within certain bounds, so that I can keep my information accurate and up-to-date

UI



User Information

Username *
adminUser

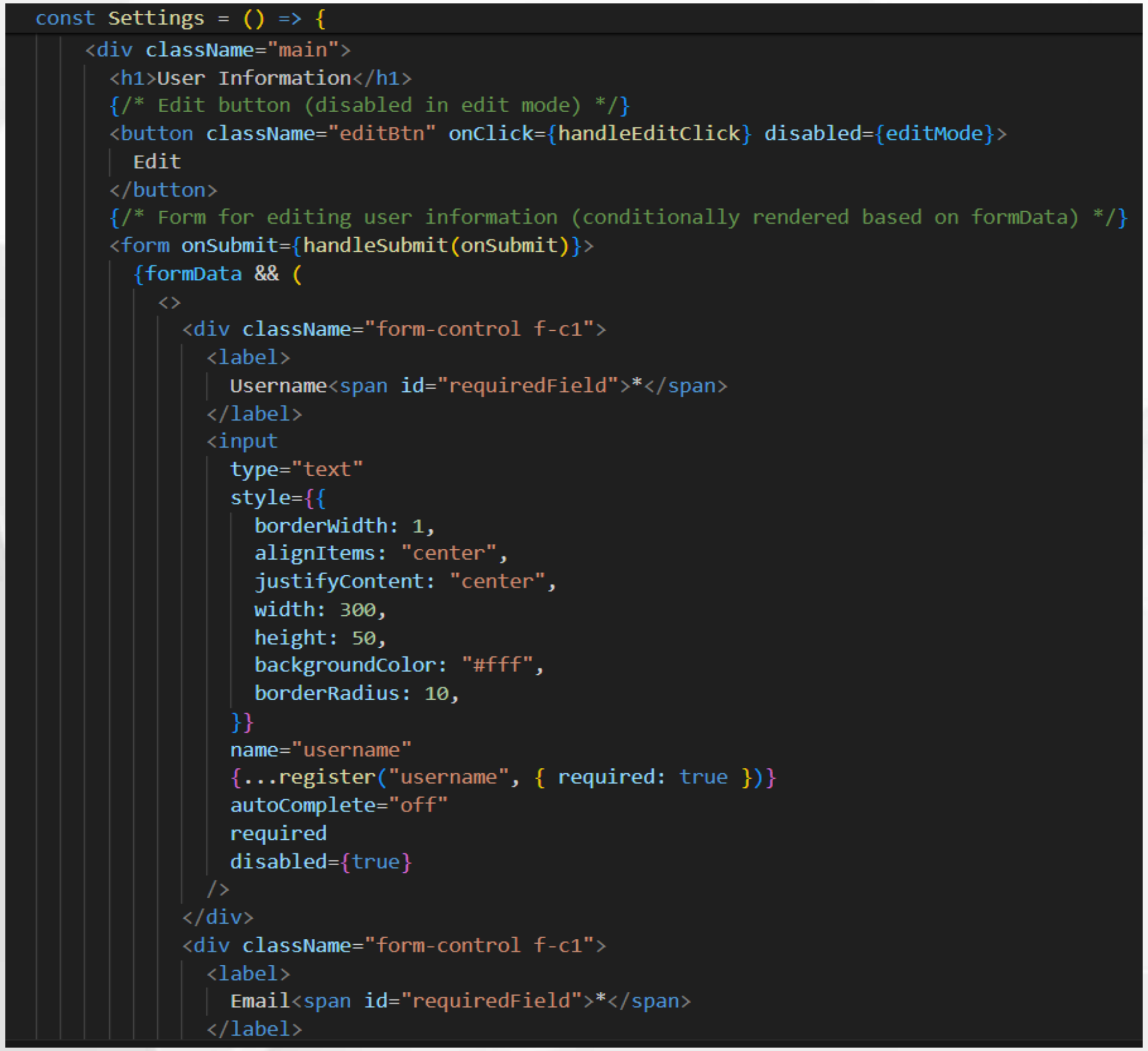
Email *
test1@gmail.com

Firstname
firstname

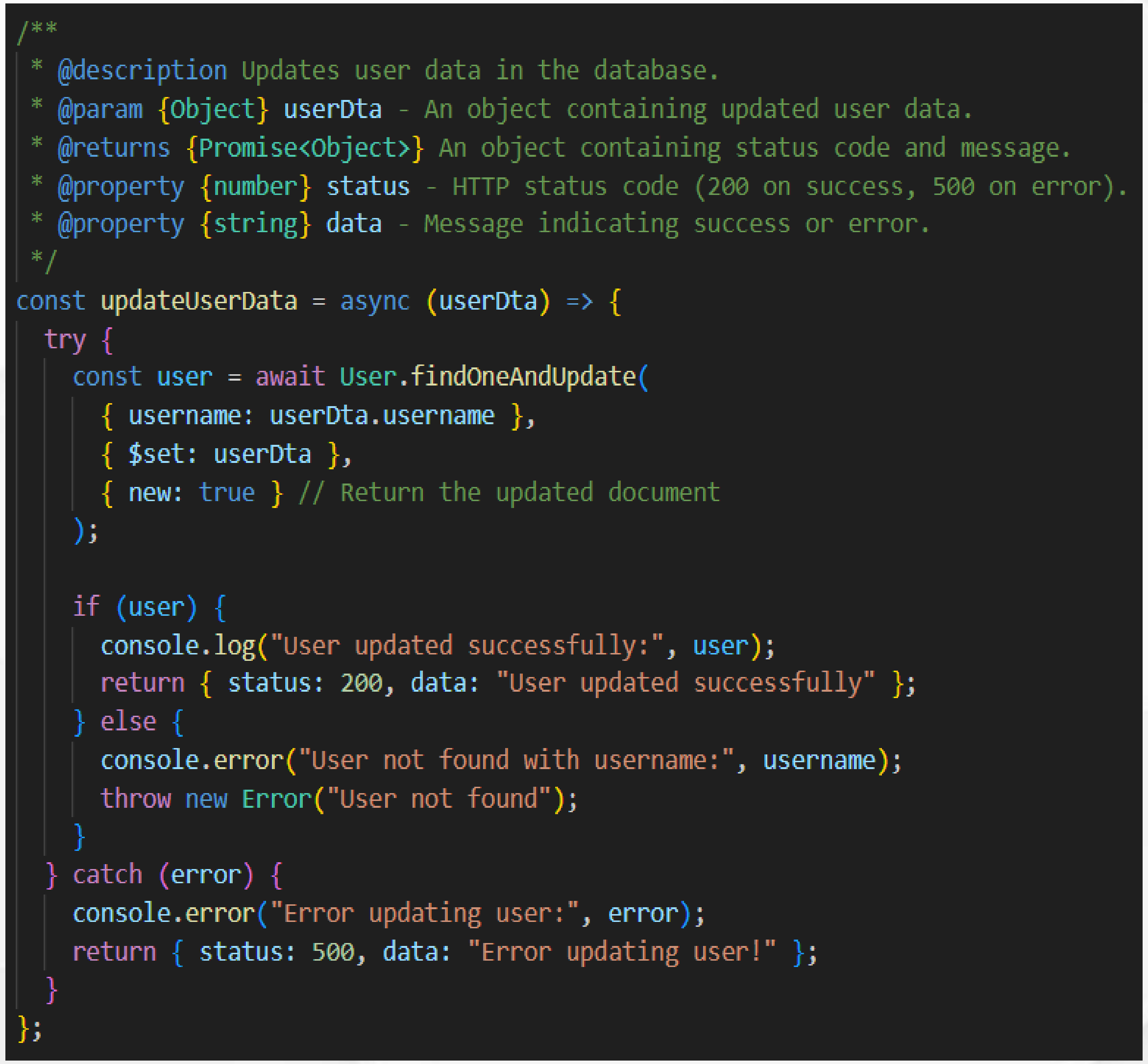
Lastname
lastname

Project Tracker

USERS TAB Code Snippet



```jsx
const Settings = () => {
    <div className="main">
      <h1>User Information</h1>
      {/* Edit button (disabled in edit mode) */}
      <button className="editBtn" onClick={handleEditClick} disabled={editMode}>
        Edit
      </button>
      {/* Form for editing user information (conditionally rendered based on formData) */}
      <form onSubmit={handleSubmit(onSubmit)}>
        {formData && (
          <>
            <div className="form-control f-c1">
              <label>
                Username<span id="requiredField">*</span>
              </label>
              <input
                type="text"
                style={{
                  borderWidth: 1,
                  alignItems: "center",
                  justifyContent: "center",
                  width: 300,
                  height: 50,
                  backgroundColor: "#fff",
                  borderRadius: 10,
                }}
                name="username"
                {...register("username", { required: true })}
                autoComplete="off"
                required
                disabled={true}
              />
            </div>
            <div className="form-control f-c1">
              <label>
                Email<span id="requiredField">*</span>
              </label>
```



```javascript
/**
 * @description Updates user data in the database.
 * @param {Object} userDta - An object containing updated user data.
 * @returns {Promise<Object>} An object containing status code and message.
 * @property {number} status - HTTP status code (200 on success, 500 on error).
 * @property {string} data - Message indicating success or error.
 */
const updateUserData = async (userDta) => {
  try {
    const user = await User.findOneAndUpdate(
      { username: userDta.username },
      { $set: userDta },
      { new: true } // Return the updated document
    );

    if (user) {
      console.log("User updated successfully:", user);
      return { status: 200, data: "User updated successfully" };
    } else {
      console.error("User not found with username:", username);
      throw new Error("User not found");
    }
  } catch (error) {
    console.error("Error updating user:", error);
    return { status: 500, data: "Error updating user!" };
  }
};
```

USERS API’s

1. Get logged-in user info

   Endpoint : /v2/getLoggedinUserData/{username}

   Method: GET

   Authorization: JSON Web Token

2. Update user data

   End point : /v2/updateUserData

   Method: POST

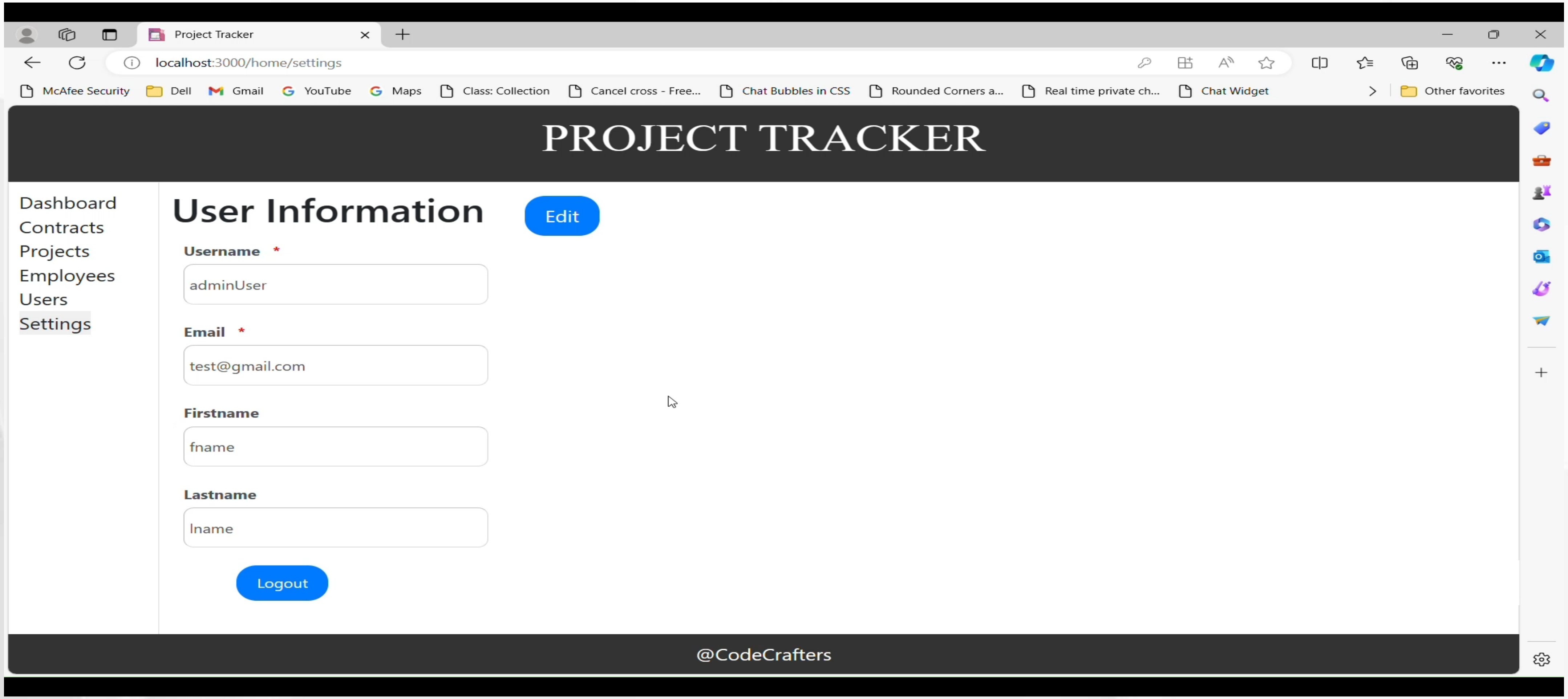
   Authorization: JSON Web Token

   Request payload:
   {"username":"adminUser","email":"test1@gmail.com","firstname":"firstname","lastname":"lastnam"}

Project Tracker

Evaluation Criteria

User settings page showing current username, email, first name, and last name. Offers an "Edit" button to update editable information (first name, last name, and email format) with validation and error messages. Allows saving changes, undoing edits, and secure logout.



PROJECT TRACKER

Dashboard
Contracts
Projects
Employees
Users
Settings

User Information

Edit

Username *

adminUser

Email *

test@gmail.com

Firstname

fname

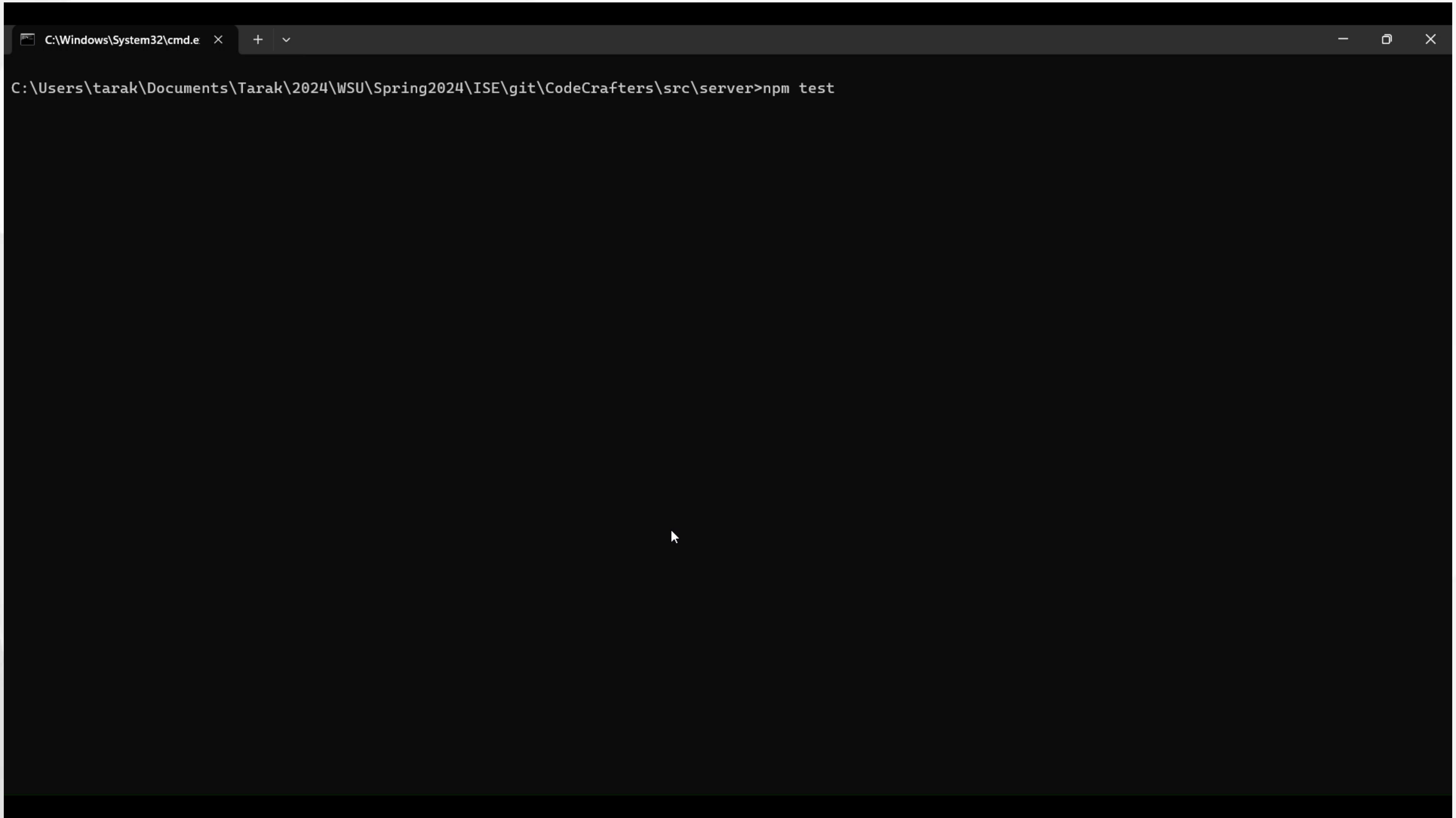
Lastname

lname

Logout

@CodeCrafters

Unit Test Cases



```
C:\Users\tarak\Documents\Tarak\2024\WSU\Spring2024\ISE\git\CodeCrafters\src\server>npm test
```

What have we learned?

- Software Development Process.
- Exposed to GitHub.
- Team collaboration.
- Agile Methodology.

Questions?

THANK YOU