# 

# **System Design Document**

# 

# **Driven Developers**

# 

# **CSCC01 Project**

# 

# Sprint 2

# 

# 

# Authors: Tamam Makki, Harish Thevakumaran, Hamza Khalid

**Table of Contents**

1. **Class Descriptions: Page 2-15**
   * AddProject (from addproject.js)
   * ProjectList (from projectList.js)
   * App (from server.js)
   * User
   * Projects (from Project.js)
   * EditProfileForm
   * ProfileView
   * Register
   * Dashboard
   * FriendList (from FriendList.js)
   * FriendRequestsNotification.js
   * Notification.js
   * newDiscussion.js
   * Discussions.js
   * Comment.js
   * eventCalendar.js
   * GpaCalculator
   * ConnectionsRouter (from connections.js)
   * SearchUsers (from SearchUsers.js)
   * Index.js
   * reportWebVitals.js
   * setupTests.js
2. **System Interaction with the Environment: Page 12**
   * Dependencies and Assumptions
   * Operating System
   * Programming Languages
   * Frameworks and Libraries
     + - Frontend
       - Backend
     + Development Tools
     + Database
     + Network Configuration
3. **Architecture Diagram: Page 13 - 14**

| Class Name: AddProject (from addproject.js) | |
| --- | --- |
| Parent Class: mongoose.Model  Subclass: None | |
| Responsibilities   * Define and render the project addition form. * Display input fields for project name, description, and link. * Style the form and its elements using the defined styles object. * Handle form submission. * Collect and validate user input. * Send a POST request to the API to create a new project. * Navigate to the project list page upon successful submission. * Handle form cancellation. * Navigate to the project list page when the cancel button is clicked. | Collaborators:   * React (for managing component state and rendering) * useNavigate (for navigation) * localStorage (to get the token for authentication) * fetch (for sending the POST request to the API) * CSS-in-JS styles object (for styling the component) |

| Class Name: ProjectList(from projectList.js) | |
| --- | --- |
| Parent Class:None  Subclass: None | |
| Responsibilities   * fetch and display a list of projects. * Retrieve project data from the server using a GET request. * Render each project with its details, including name, description, link, and profile picture. * Handle user navigation. * Navigate to the 'Add Project' form when the "Add Project" button is clicked. * Navigate back to the dashboard when the "Back to Dashboard" button is clicked. * Handle project deletion. * Send a DELETE request to the server to remove a project. * Update the list of displayed projects upon successful deletion. | Collaborators:   * React (for managing component state and rendering) * useNavigate (for navigation) * localStorage (to get the token for authentication) * fetch (for sending GET and DELETE requests to the API) * CSS-in-JS styles object (for styling the component) |

| Class Name: App(from server.js) |
| --- |
| Parent Class: Express.Application  Subclass: None explicitly defined |
| Responsibilities   * Define the application's routing structure. * Use BrowserRouter to handle routing within the app. * Define Routes and corresponding Route elements for various components: * / - Renders the Login component. * /register - Renders the Register component. * /dashboard - Renders the Dashboard component. * /profile-view - Renders the ProfileView component. * /profile-view/:id - Renders the ProfileView component in read-only mode. * /edit-profile - Renders the EditProfileForm component. * /project-list - Renders the ProjectList component. * /add-project - Renders the AddProject component. * /connect - Renders the SearchUser component. * /notifications - Renders the Notifications component. * /friend-list - Renders the FriendList component. |
| Collaborator: NextResponse from 'next/server'   * express: Web framework for creating the server and handling routing. * mongoose: ODM (Object Data Modeling) library for MongoDB to define schemas and interact with the database. * bcryptjs: Library for hashing passwords. * jsonwebtoken: Library for creating and verifying JWT tokens. * cors: Middleware to enable Cross-Origin Resource Sharing. * multer: Middleware for handling file uploads. * path: Node.js module for working with file and directory paths. * User: Mongoose model representing users in the database. * Project: Mongoose model representing projects in the database. |

| Class Name: User |
| --- |
| Parent Class: mongoose.Model  Subclass: None |
| Responsibilities   * Define the schema for a user. * Specify fields and their types: email, password, fullName, programName, yearOfStudy, gpa, description, profilePicture, interests, courses, friendRequests, and friends. * Set validation rules, such as making email and password required and email unique. * Set default values for certain fields like fullName, programName, yearOfStudy, gpa, description, interests, and courses. * Facilitate CRUD operations in MongoDB. * Enable creation, reading, updating, and deletion of user documents in the MongoDB database. * Ensure that user documents adhere to the defined schema. |
| Collaborator: NextResponse from 'next/server'   * mongoose.Schema (to define the structure of user documents) * mongoose.model (to create the User model) * User model (referenced by the friendRequests and friends fields to establish relationships between users) |

| Class Name: Projects(from Project.js) |  |
| --- | --- |
| Parent Class: mongoose.Model  Subclass: None |  |
| Responsibilities   * Define the schema for a project. * Specify fields and their types: projectName, description, link, and user. * Set validation rules, such as making projectName and user required. * Facilitate CRUD operations in MongoDB. * Enable creation, reading, updating, and deletion of project documents in the MongoDB database. * Ensure that project documents adhere to the defined schema. | Collaborators:   * mongoose.Schema (to define the structure of project documents) * mongoose.model (to create the Project model) * User model (referenced by the user field to establish a relationship between projects and users) |

| Class Name: EditProfileForm | |
| --- | --- |
| Parent Class: React.Component  Subclass: None | |
| Responsibilities   * Render a form to allow users to edit their profile information. * Handle form state using React hooks. * Fetch the current profile data from the server and populate the form fields. * Submit the updated profile data to the server. * Navigate between different views based on user actions. | Collaborators:   * React: Library for building user interfaces and handling component state. * useState and useEffect: React hooks for managing state and side effects. * useNavigate: Hook from react-router-dom for navigating programmatically. * fetch: Function for making HTTP requests to the server. * localStorage: Web API for storing and retrieving authentication tokens. |

| Class Name: ProfileView |
| --- |
| Parent Class: React.Component  Subclass: None |
| Responsibilities:   * Render the user's profile information. * Fetch the profile data from the server. * Display the fetched profile data. * Provide navigation options to edit the profile or return to the dashboard. |
| Collaborators:   * useState * useRouter * setIsRegister |
|  |

| Class Name: AddProject |  |
| --- | --- |
| Parent Class: React.Component  Subclass: None |  |
| Responsibilities   * Render the form for adding a new project. * Manage form state and handle form submission. * Send a POST request to add the new project. * Navigate to the project list on successful submission. * Provide a cancel button to navigate back to the project list without submitting. | Collaborators:   * React: Library for building user interfaces and handling component state. * useState: React hook for managing state. * useNavigate: Hook from react-router-dom for navigating programmatically. * fetch: Function for making HTTP requests to the server. * localStorage: Web API for storing and retrieving authentication tokens. |

| Class Name: ProjectList | |
| --- | --- |
| Parent Class: React.Component  Subclass: None | |
| Responsibilities   * Fetch and display a list of projects from the server. * Provide UI elements to navigate to add new projects and back to the dashboard. * Display each project with its details and a delete button. * Handle deletion of projects by sending a DELETE request to the server and updating the local state. | Collaborators:   * React: Library for building user interfaces and handling component state. * useState: React hook for managing state. * useEffect: React hook for performing side effects in function components. * useNavigate: Hook from react-router-dom for navigating programmatically. * fetch: Function for making HTTP requests to the server. * localStorage: Web API for storing and retrieving authentication tokens.ter |

| Class Name: Register | |
| --- | --- |
| Parent Class: React.Component  Subclass: None | |
| Responsibilities   * Fetch and display a list of projects from the server. (This is not explicitly implemented in the provided code but can be added separately) * Provide UI elements to navigate to add new projects and back to the dashboard. * Display each project with its details and a delete button. (Not implemented in this code) * Handle deletion of projects by sending a DELETE request to the server and updating the local state. (Not implemented in this code) | Collaborators:   * React: Library for building user interfaces and handling component state. * useState: React hook for managing state. * useEffect: React hook for performing side effects in function components. * useNavigate: Hook from react-router-dom for navigating programmatically. * axios: Library for making HTTP requests to the server. * localStorage: Web API for storing and retrieving authentication tokens. |

| Class Name: Dashboard | |
| --- | --- |
| Parent Class: N/A  Subclass: N/A | |
| Responsibilities   * Render and manage the dashboard. * Display user-specific information such as userName, projects, friendRequests, and recommendedConnections. * Provide navigation options for different parts of the application (Profile, Connect, Projects, Notifications, Friends, Event Calendar, Discussions, GPA Calculator). * Integrate FriendRequestsNotification component to show friend request notifications. * Handle navigation and user actions. * Navigate to different routes using useNavigate from react-router-dom. * Fetch and display data related to user profile, projects, friend requests, and recommended connections from the backend. * Handle actions such as sending friend requests and logging out. | Collaborators:   * useNavigate (from react-router-dom) for navigating between routes. * useState (from react) for managing component state. * useEffect (from react) for fetching initial data when the component mounts. * fetch API for interacting with the backend to fetch and update data. * FriendRequestsNotification component to display friend request notifications. |

| Class Name: FriendList (from FriendList.js) | |
| --- | --- |
| Parent Class:None  Subclass: None | |
| Responsibilities   * Render and manage the friend list. * Display the user's friends, including their profile picture and name. * Provide options to view the profile of each friend and to remove friends from the list. * Navigate back to the dashboard. * Handle fetching and updating friends data. * Fetch the list of friends from the backend when the component mounts. * Handle removing a friend from the list by making a DELETE request to the backend. | Collaborators:   * useNavigate (from react-router-dom) for navigating between routes. * useState (from react) for managing component state. * useEffect (from react) for fetching initial data when the component mounts. * fetch API for interacting with the backend to fetch and update data. |

| Class Name: FriendRequestsNotification.js | |
| --- | --- |
| Parent Class:None  Subclass: None | |
| Responsibilities   * Render a notification for friend requests. * Display a message indicating the number of friend requests the user has received. * Only render the notification if there are one or more friend requests. | Collaborators:   * React for creating the component. * friendRequests (prop) passed from the parent component (Dashboard in this case) containing the list of friend requests. |

| Class Name: Notification.js | |
| --- | --- |
| Parent Class:React.Component  Subclass: None | |
| Responsibilities   * Render notifications for friend requests. * Display a list of friend requests with options to accept or reject each request. * Fetch friend requests from the server using an API endpoint. * Handle acceptance and rejection of friend requests via API calls and update the state accordingly. * Display a message if no friend requests are available. | Collaborators:   * React for creating the component. * useState and useEffect hooks from React for managing state and side effects. * fetch API for making HTTP requests to the server. * localStorage for storing and retrieving authentication tokens. |

| Class Name: ConnectionsRouter (from connections.js) | |
| --- | --- |
| Parent Class: None  Subclass: None | |
| Responsibilities:   * Handle routes related to recommended connections. * Implement middleware (verifyToken) to authenticate and decode JWT tokens from request headers. * Define a route (/recommended-connections) to fetch recommended connections for a user based on shared interests. * Use JWT to verify and extract the user ID from the token payload. * Fetch the current user's profile (User.findById(userId)) using * Mongoose or a similar ORM. * Retrieve other users (User.find(...)) who have similar interests as the current user but are not the same user (\_id: { $ne: userId }). * Return recommended connections as JSON response (res.json(recommendedConnections)). * Handle errors such as missing tokens, failed authentication, user not found, or server errors (res.status(...)). | Collaborators:   * express for creating and managing routes. * jwt for token verification and decoding. * User model (presumed to be from Mongoose or a similar ORM) for interacting with user data in the database. * process.env.JWT\_SECRET or default 'your-secret-key' for JWT token verification. |

| Class Name:SearchUsers (from SearchUsers.js) | |
| --- | --- |
| Parent Class:React.Component  Subclass: None | |
| Responsibilities   * Render a search interface for users based on filters. * Display input fields for filtering users by name, academic interests, courses, program, and year. * Allow users to search for other users based on specified filters. * Display search results with user information including profile picture, name, academic interests, courses, program, and year of study. * Enable sending friend requests to users displayed in the search results. * Manage state for input fields (name, academicInterests, courses, program, year), search results (results), and friend requests (friendRequests). * Fetch users based on search criteria using an API endpoint. * Handle sending friend requests to selected users via API calls and update state accordingly. * Display appropriate messages if no users match the search criteria or if there are no search results. | Collaborators:   * React for creating the component and managing state with useState hook. * fetch API for making HTTP requests to the server. * localStorage for storing and retrieving authentication tokens. |

| Class Name: Index.js | |
| --- | --- |
| Parent Class: N/A  Subclass: N/A | |
| Responsibilities   * Imports necessary modules and components (React, ReactDOM, App, reportWebVitals) * Renders the main application component (<App />) within a React.StrictMode wrapper using ReactDOM.createRoot * Includes a call to reportWebVitals() for measuring performance in the app | Collaborators:   * React (React, ReactDOM) * './App' (Assuming this is the main application component) * './reportWebVitals' (Used for performance measurement) |

| Class Name: GpaCalculator.js | |
| --- | --- |
| Parent Class: N/A  Subclass: N/A | |
| Responsibilities   * State Management: Maintain and update state for current courses, potential courses, GPAs, completed credits, error messages, and saved GPAs. * UI Rendering: Render input forms for entering course details, buttons for adding/removing courses, and display calculated GPAs and saved GPAs. * Event Handling: Handle various events such as adding/removing courses, changing course details, calculating GPAs, saving GPAs, and deleting saved GPAs. * API Interaction: Fetch and save GPA data from/to the backend API. * Validation: Validate form inputs before processing GPA calculations. | Collaborators:   * Backend API: Interacts with the backend to fetch and save GPA data. |

| Class Name: Discussion.js | |
| --- | --- |
| Parent Class: N/A  Subclass: N/A | |
| Responsibilities   * Data Representation: Define and structure the data for a discussion post. * Schema Definition: Set up the schema with required fields and data types using Mongoose. * Database Interaction: Facilitate creation, reading, updating, and deletion of discussion documents in the MongoDB database. | Collaborators:   * Mongoose: Utilized for schema definition and model creation. * User Model: References the User model to link discussions to user accounts. |

| Class Name: DiscussionDetail.js | |
| --- | --- |
| Parent Class: N/A  Subclass: N/A | |
| Responsibilities   * Fetch and Display Discussion Details: Retrieve and display detailed information about a specific discussion post. * Fetch and Display Comments: Retrieve and display comments associated with the discussion post. * Handle New Comments: Allow users to submit new comments to the discussion post. * User Interface: Manage the layout and styling of the discussion detail view and comments section.. | Collaborators:   * React: Utilized for building the component and managing state and lifecycle. * React Router (useParams): Used to access the discussionId from the URL. * Axios: Used for making HTTP requests to the backend API. * LocalStorage: Used for retrieving the user's authentication token |

| Class Name: NewDiscussion.js | |
| --- | --- |
| Parent Class: N/A  Subclass: N/A | |
| Responsibilities   * Form Management: Manage the input form for creating a new discussion. * Image Handling: Handle the selection and management of multiple images for the discussion. * Form Submission: Submit the form data, including the images, to the backend API to create a new discussion. * Navigation: Navigate to the discussions list view upon successful creation of a new discussion | Collaborators:   * React: Utilized for building the component and managing state. * React Router (useNavigate): Used for navigating to the discussions list view after successfully creating a new discussion. * Fetch API: Used for making HTTP requests to the backend API. * LocalStorage: Used for retrieving the user's authentication token. |

| Class Name: reportWebVitals.js | |
| --- | --- |
| Parent Class: N/A  Subclass: N/A | |
| Responsibilities   * Defines a function reportWebVitals that takes onPerfEntry as a parameter * Checks if onPerfEntry is a function and then imports the 'web-vitals' module asynchronously * Once the 'web-vitals' module is loaded, it invokes functions (getCLS, getFID, getFCP, getLCP, getTTFB) provided by the module and passes onPerfEntry to each of them | Collaborators:   * 'web-vitals' (module imported dynamically) * onPerfEntry (a function passed as a parameter to reportWebVitals) |

| Class Name: setupTests.js | |
| --- | --- |
| Parent Class: N/A  Subclass: N/A | |
| Responsibilities   * Comments explaining the purpose of the import and its usage. * Imports '@testing-library/jest-dom' which adds custom Jest matchers for asserting on DOM nodes. * Enhances Jest's capabilities to assert on DOM elements using methods like toHaveTextContent provided by '@testing-library/jest-dom'. | Collaborators:   * '@testing-library/jest-dom' (library providing custom Jest matchers for DOM assertions) |

| Class Name: DegreePlanner.js | |
| --- | --- |
| Parent Class: N/A  Subclass: N/A | |
| Responsibilities   * Manage Course State: Handle state for course information (name, code, credits, semester, year) and list of courses. * Fetch Courses: Retrieve the list of courses from the backend when the component mounts. * Add Course: Add a new course to the backend and update the state. * Delete Course: Delete a course from the backend and update the state. * Error Handling: Display error messages when form validation fails or when there are issues with backend communication. * Hover Effects: Manage hover effects for buttons. | Collaborators:   * React: Utilized for building the component and managing state. * Fetch API: Used for making HTTP requests to the backend API. * LocalStorage: Used for retrieving the user's authentication token. |

### 

### 

| Class Name: EventCalendar | |
| --- | --- |
| Parent Class: React.Component  Subclass: None | |
| Responsibilities   * State Management: * Manage state for events, modalOpen, selectedEvent, eventTitle, eventLocation, friends, invitedFriends, startDate, endDate, currentMonthYear, eventInvites, and currentUser. * Data Fetching: * Fetch current user data (fetchCurrentUser) and store it in state. * Fetch friends data (fetchFriends) and store it in state. * Fetch events data (fetchEvents) and store it in state. * Fetch event invites data (fetchEventInvites) and store it in state. * Event Handling: * Handle calendar date selection (handleSelect) to open a modal for creating a new event. * Handle event creation (handleCreateEvent) by sending a POST request to the server and updating the state with the new event. * Handle event click (handleEventClick) to display event details in a modal. * Handle event deletion (handleDeleteEvent) by sending a DELETE request to the server and updating the state. * Handle attendee removal (handleRemoveAttendee) by sending a POST request to the server and updating the state. * Handle invite acceptance (handleAcceptInvite) and rejection (handleRejectInvite) by sending POST requests to the server and updating the state. * Form Handling: * Reset form fields (resetForm) after event creation or cancellation. * Handle form cancellation (handleCancel) by closing the modal and resetting form fields. * Handle inviting friends (handleInviteFriend) by updating the state with selected friends. * Calendar Configuration: * Configure FullCalendar with plugins, initial view, selectable dates, events data, and custom handlers for selection, date setting, and event clicks. * Display the current month and year in the calendar header (handleDatesSet). | Collaborators:   * React: For component creation, state management, and lifecycle methods (useState, useEffect). * FullCalendar: For calendar rendering and interaction (@fullcalendar/react, dayGridPlugin, interactionPlugin). * LocalStorage: For retrieving the JWT token. * Fetch API: For making HTTP requests to the server to fetch and manipulate event and user data. |

### 

| Class Name: AddResource.js | |
| --- | --- |
| Parent Class: N/A  Subclass: N/A | |
| Responsibilities   * Form Management: Handle state for form inputs (title, link, description, tags). * Form Submission: Validate and submit the form data to the backend API. * Redirect on Success: Redirect the user to the resource library upon successful submission. * Error Handling: Display errors when form submission fails. | Collaborators:   * 'React: Utilized for building the component and managing state. * Fetch API: Used for making HTTP requests to the backend API. * LocalStorage: Used for retrieving the user's authentication token. |

| Class Name:ToDo.js | |
| --- | --- |
| Parent Class: N/A  Subclass: N/A | |
| Responsibilities   * Fetch ToDo Items: Retrieve to-do items from the backend API. * Add ToDo Item: Add a new to-do item to the list. * Update ToDo State: Update the state of a to-do item. * Delete ToDo Item: Remove a to-do item from the list. * Filter ToDo Items: Filter to-do items based on their state. * Sort ToDo Items: Sort to-do items by priority. | Collaborators:   * React: Utilized for building the component, managing state, and side effects. * Fetch API: Used for making HTTP requests to the backend API. * LocalStorage: Used for retrieving the user's authentication token. |

### 

### 

### 

### 

### 

### 

### 

### **Description of System Interaction with the Environment**

#### **Dependencies and Assumptions:**

1. **Operating System**:
   * The system is designed to be OS-independent, meaning it can run on Windows, macOS, and Linux. However, development and testing are typically done on a Unix-like environment (Linux or macOS).
2. **Programming Languages**:
   * **JavaScript**: For both client-side (React.js) and server-side (Node.js) development.
   * **HTML/CSS**: For client-side rendering and styling.
3. **Frameworks and Libraries**:
   * **Frontend**:
     + **React.js**: For building the user interface.
     + **React Router**: For handling routing in the single-page application.
     + **Axios**: For making HTTP requests from the client to the server.
   * **Backend**:
     + **Express.js**: For building the server-side RESTful APIs.
     + **Mongoose**: For interacting with MongoDB.
     + **Bcrypt.js**: For password hashing.
     + **Jsonwebtoken**: For user authentication.
     + **Cors**: For enabling Cross-Origin Resource Sharing.
   * **Development Tools**:
     + **Nodemon**: For automatic server restarts during development.
     + **Concurrently**: For running both the client and server concurrently during development.
4. **Database**:
   * **MongoDB**: Used to store user data, including emails, hashed passwords, and profile information.
5. **Network Configuration**:
   * The system requires a stable internet connection for accessing the MongoDB Atlas database.
   * Proper network configurations and permissions are needed, including IP whitelisting for MongoDB Atlas.

#### **Architecture Diagram**

+-------------------------------------------------------------+

| Frontend |

| +---------------------+ +-------------------------+ |

| | Register Component | | Login Component | |

| +---------------------+ +-------------------------+ |

| +---------------------+ +-------------------------+ |

| | Profile Component | | AddProject Component | |

| +---------------------+ +-------------------------+ |

| +---------------------+ +-------------------------+ |

| | ProjectList Component| | Dashboard Component | |

| +---------------------+ +-------------------------+ |

| +---------------------+ +-------------------------+ |

| | FriendList Component| |FriendRequestsNotification|

| +---------------------+ +-------------------------+ |

| +---------------------+ +-------------------------+ |

| | SearchUser Component| | Notifications Component | |

| +---------------------+ +-------------------------+ |

| | | |

| | HTTP Request (Axios) | |

| v v |

| +---------------------+ +-------------------------+ |

| | Profile Component | | Local Storage | |

| | | | (JWT Token) | |

| +---------------------+ +-------------------------+ |

| | | |

| | CSS Styling | |

| v v |

| +---------------------+ +-------------------------+ |

| | App.css | | index.css | |

| +---------------------+ +-------------------------+ |

+-------------------------------------------------------------+

^ ^

| |

| |

v v

+-------------------------------------------------------------+

| Backend |

| +---------------------+ +-------------------------+ |

| | Express.js | | MongoDB | |

| | | | | |

| +---------+-----------+ +------------+------------+ |

| | | |

| | Route Handling | |

| v v |

| +---------------------+ +-------------------------+ |

| | Auth Routes | | User Collection | |

| | (Register, Login) | | | |

| +---------+-----------+ +-------------------------+ |

| | | |

| | Profile Routes | |

| v v |

| +---------------------+ +-------------------------+ |

| | Profile Controller | | Project Collection | |

| +---------------------+ +-------------------------+ |

| | | |

| | Project Routes | |

| v v |

| +---------------------+ +-------------------------+ |

| | Add Project | | | |

| | List Projects | | | |

| +---------------------+ +-------------------------+ |

| | | |

| | Notification Routes | |

| v v |

| +---------------------+ | |

| | Notifications | | |

| +---------------------+ | |

| | | |

| | | |

| v v |

| +---------------------+ +-------------------------+ |

| | Middleware (JWT Auth)| | | |

| +---------------------+ +-------------------------+ |

+-------------------------------------------------------------+