DAT076: Report

Sami Jansson Al-Ani, Emil Tervo

March 18, 2025

1 Introduction

A link to our Github repo, please make more note of branch-activity than main-branch history. As of week 12, the application is still work-in-progress. Certain functions such as the private chat or user role are not yet fully implemented. Other aspects like the ER-diagram may also change for the final version. For this project, we have built a software project management system for freelance programmers. The foundational purpose of the application is for software engineers to find professional projects to work with, as well as for project managers to find suitable and qualified members. The application allows users to create a personal profile, assign themselves to either programmer or project manager, and add a description about their work experience, company, searching jobs, etc. Project managers are able to create and upload projects and assign necessary information such as the company name, number of members needed, a description, and a general salary. Project managers may wait until all roles are filled, or close the project to stop more members joining.

A programmer may browse, search, sort, and click on any project uploaded by project managers, and assign themselves to the projects they wish to work with. Members of a project can then contact the project manager in private chats for discussions and negotiations. Once the project manager formally accepts the programmer as a member, the purpose of the application is accomplished.

Users can also rate other members and project leaders/companies through a reputation system with either a like or dislike. The reputation of users will accumulate over time to distinguish veteran expensive programmers, and inexperienced but low-cost programmers. Equally, the reputation system will allow programmers to tell apart serious projects and managers to potentially frivolous companies.

2 User Manual

For the user manual, we assume Node.js and npm are already installed. To install and use the application, clone the Github repository from the link in the introductory section.

1. Open the directory for the project folder (Web-Applications-DAT076) either through the OS file explorer or an IDE.
2. In the server folder, create a '.env' file necessary for the authentication key.
3. Open the '.env' file. In it, add the line:

   ```
   SESSION_SECRET = 'your_key'
   ```

   '*your_key*' is the user's choice of authentication key.
4. Open the **/client** directory and run the command 'npm install' to install all necessary dependencies. Change directory to **/server** and also run 'npm install'.
5. Open a separate terminal (two in total), and for each one, navigate to the **/client** directory and **/server** directory respectively.

6. For each terminal, run the command "npm run dev" to create the client server and backend server necessary for the application.

The application is now ready and should be opened in the client URL (http://localhost:5173 by default).
Once the user opens the application, they are greeted by a login screen, see figure 1. Click on the link underneath ('Register here') to create a new account. Fill in username, first name, last name, password, and e-mail to create the account.

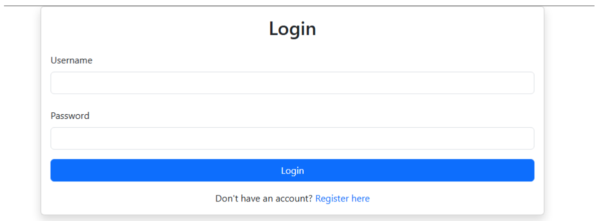
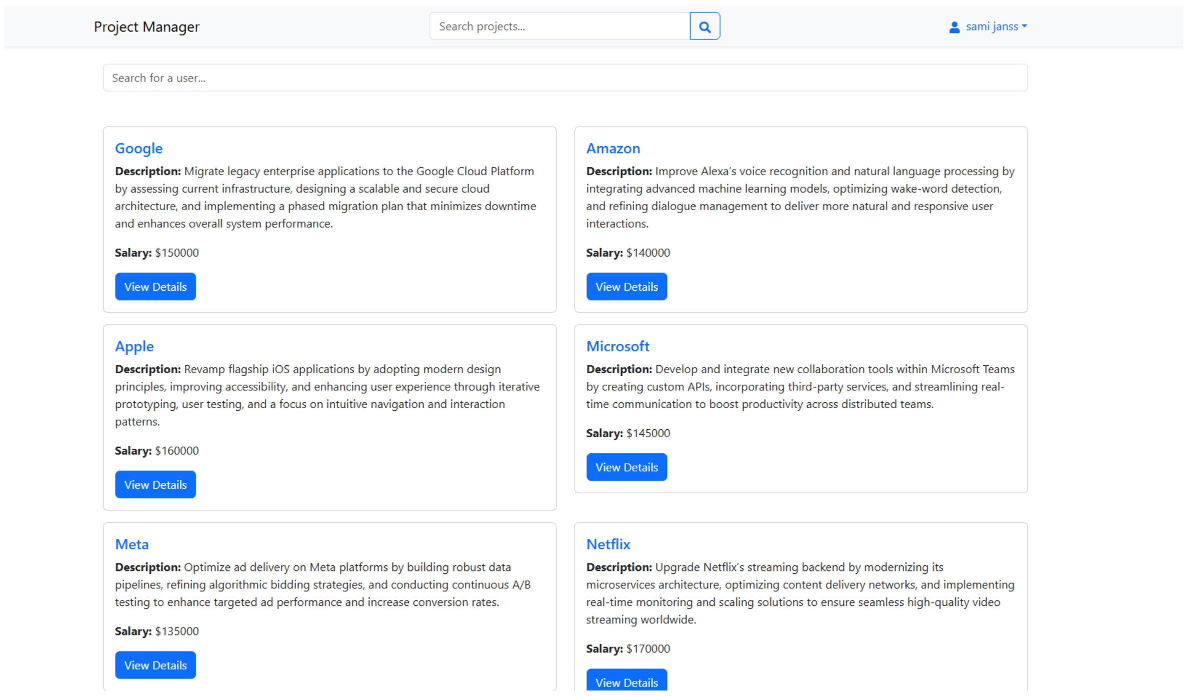


Figure 1: Initial login screen

Once the user finishes and clicks 'Register', they are redirected to the application homepage screen, see figure 2. The user may browse, join, and create new projects. Alternatively, they can also search for keywords in both company name and description to find specific projects, as well as search for other users. The user can also edit their profile, create new projects, or sign out of their account through the drop-down box by clicking on their name at the top right corner of the webpage. Clicking on a project redirects the user to the project page, where additional information such as members and whether the project is open are visible. Here, the user can also start a chat with the project manager.



Google
Description: Migrate legacy enterprise applications to the Google Cloud Platform by assessing current infrastructure, designing a scalable and secure cloud architecture, and implementing a phased migration plan that minimizes downtime and enhances overall system performance.
Salary: $150000
View Details

Amazon
Description: Improve Alexa's voice recognition and natural language processing by integrating advanced machine learning models, optimizing wake-word detection, and refining dialogue management to deliver more natural and responsive user interactions.
Salary: $140000
View Details

Apple
Description: Revamp flagship iOS applications by adopting modern design principles, improving accessibility, and enhancing user experience through iterative prototyping, user testing, and a focus on intuitive navigation and interaction patterns.
Salary: $160000
View Details

Microsoft
Description: Develop and integrate new collaboration tools within Microsoft Teams by creating custom APIs, incorporating third-party services, and streamlining real-time communication to boost productivity across distributed teams.
Salary: $145000
View Details

Meta
Description: Optimize ad delivery on Meta platforms by building robust data pipelines, refining algorithmic bidding strategies, and conducting continuous A/B testing to enhance targeted ad performance and increase conversion rates.
Salary: $135000
View Details

Netflix
Description: Upgrade Netflix's streaming backend by modernizing its microservices architecture, optimizing content delivery networks, and implementing real-time monitoring and scaling solutions to ensure seamless high-quality video streaming worldwide.
Salary: $170000
View Details

Figure 2: Homepage screen

3 Design: Components

- projectView.tsx:

Displays the detailed view of a single project. It shows project information (name, description, salary, status, and member list) and includes a "Back" button and a "Delete Project" button.it uses the URL parameter (id) via React Router's useParams(). project (of type Project — null) holds the project data fetched from the backend. error (string or null) stores any error message encountered during fetching or deletion. In the useEffect(), it makes a GET request to http://localhost:3000/api/projects/$id to retrieve the project details. The handleDelete function sends a DELETE request (with credentials included) to delete the project. After successful deletion, it navigates back to the homepage.

- protectedRoutes.tsx:

Defines the routing structure of the app while enforcing authentication rules. It wraps certain routes in a ProtectedRoute component to ensure only authenticated users can access them, while PublicRoute prevents logged-in users from visiting pages like login or register. Not used externally; instead, it defines internal components (ProtectedRoute and PublicRoute) that accept children as a prop. Uses the useAuth() hook to access authentication state (user and loading) from context. Displays a loading spinner if authentication state is still being determined. It doesn't directly call the backend, instead, it controls access by checking authentication state and redirecting accordingly.

- Header.tsx:

Renders the navigation bar, which includes the brand/logo, a search form, and a user profile dropdown with options like "Edit Profile," "Create Project," and "Sign Out." No external props, it accesses user data via the useAuth() hook. searchQuery Stores the current text entered in the search bar. showDropdown Controls whether the dropdown menu is visible. The search form's handleSearch function updates the URL query (e.g. /?search=...), which can be used by the Homepage to filter projects. Uses the useNavigate hook to route to various pages (profile edit, create project, login on sign out).

- Login.tsx:

Provides a login form for users. It handles user input for username and password, displays any error messages, and triggers authentication. Relies on context for authentication actions. username and password, holds the current input values for credentials. error, stores any error message if login fails. Calls the login method from the useAuth() hook (which in turn makes a POST request to the backend endpoint /api/users/login). On successful login, navigates to the homepage.

- ProjectForm.tsx:

Provides a form for creating a new project. It collects details like the project name, description, salary, open status, and sets the project's roles so that the creator's user ID is included as the first member, uses local state and context. projectData: An object holding the form fields (name, description, salary, open, roles). The roles field is initially set using the authenticated user's id (via useAuth()). On submission, it sends a POST request using axios to /api/projects with the project data. After successful creation, it navigates back to the homepage.

- Register.tsx:

Renders a registration form for new users. It validates user input (username, password, names, email), displays validation errors, and registers the user, manages its own state and uses the auth context.
formData: Holds the registration details (username, password, firstName, lastName, email).
errors: Contains any validation errors for each field.
isSubmitting: Boolean that indicates whether the form submission is in progress.
On submission, it calls the register method from the useAuth() hook, which sends a POST request to /api/users/register. On success, navigates to the homepage.

- ProjectSearch.tsx:

Implements project search functionality by reading the search term from the URL query parameter and fetching matching projects from the backend, extracts query parameters via useLocation().
projects: An array of projects returned from the backend that match the search term.
loading: A boolean indicating whether the data is still being fetched.

error: Stores any error messages encountered during the fetch.
Uses the useEffect() hook to perform a GET request to http://localhost:3000/api/projects?search=$searchTerm. Includes credentials in the request to ensure that session data is sent. Displays the results as project cards (similar in appearance to the Homepage).

- UserSearch.tsx:

Provides a client-side user search/filter component, uses local state.

4 API Documentation

This section provides a detailed specification of the backend API, including supported HTTP methods, endpoints, request bodies, and responses for both successful and error cases. The API is divided into two main routers: `userRouter` for user-related operations and `projectRouter` for project-related operations.

4.1 User API Endpoints

The following endpoints are available under the `/api/users` base path.

4.1.1 Register a New User

- **Verb:** POST
- **Endpoint:** `/register`
- **Request Body:**

```
{
  "username": "string",
  "password": "string"
  // Additional fields as required
}
```

- **Responses:**
  - **Success (201 Created):**

```
{
  "id": "number",
  "username": "string"
  // Other user fields
}
```

  - **Error (400 Bad Request):**

```
{
  "message": "string" // e.g., "Username already exists"
}
```

4.1.2 User Login

- **Verb:** POST
- **Endpoint:** `/login`
- **Request Body:**

```
{
  "username": "string",
  "password": "string"
}
```

- **Responses:**
  - **Success (200 OK):**

    ```
    {
      "id": "number",
      "username": "string"
      // Other user fields
    }
    ```

  - **Error (401 Unauthorized):**

    ```
    {
      "message": "string" // e.g., "Invalid credentials"
    }
    ```

4.1.3 User Logout

- **Verb:** POST
- **Endpoint:** `/logout`
- **Request Body:** None
- **Responses:**
  - **Success (200 OK):**

    ```
    {
      "message": "Logged out successfully"
    }
    ```

4.1.4 Get Current User

- **Verb:** GET
- **Endpoint:** `/me`
- **Request Body:** None
- **Responses:**
  - **Success (200 OK):**

    ```
    {
      "id": "number",
      "username": "string"
      // Other user fields
    }
    ```

  - **Error (401 Unauthorized):**

```
{
  "message": "Not authenticated"
}
```

  - **Error (404 Not Found):**

```
{
  "message": "User not found"
}
```

4.1.5 Get All Users

- **Verb:** GET
- **Endpoint:** `/`
- **Request Body:** None
- **Responses:**
  - **Success (200 OK):**

```
[
  {
    "id": "number",
    "username": "string"
    // Other user fields
  }
]
```

4.1.6 Get User by ID

- **Verb:** GET
- **Endpoint:** `/:id`
- **Request Body:** None
- **Responses:**
  - **Success (200 OK):**

```
{
  "id": "number",
  "username": "string"
  // Other user fields
}
```

4.1.7 Update User

- **Verb:** PATCH
- **Endpoint:** `/:id`
- **Request Body:**

```
{
  "username": "string"
  // Other fields as allowed
}
```

- **Responses:**
  - **Success (200 OK):**

```
{
  "id": "number",
  "username": "string"
  // Updated user fields
}
```

4.1.8 Delete User

- **Verb:** DELETE
- **Endpoint:** `/:id`
- **Request Body:** None
- **Responses:**
  - **Success (204 No Content):** No body

4.2 Project API Endpoints

The following endpoints are available under the `/api/projects` base path.

4.2.1 Get All Projects

- **Verb:** GET
- **Endpoint:** `/`
- **Request Body:** None
- **Responses:**
  - **Success (200 OK):**

```
[
  {
    "idProj": "number",
    "name": "string",
    "description": "string",
    "salary": "number",
    "roles": "string",
    "open": "boolean",
    "employerId": "number",
    "company": "string | null"
  }
]
```

  - **Error (500 Internal Server Error):**

```
{
  "message": "string"
}
```

4.2.2 Get Projects by Logged-in Employer

- **Verb:** GET
- **Endpoint:** `/employer`
- **Request Body:** None
- **Responses:**
  - **Success (200 OK):**

```
[
  {
    "idProj": "number",
    "name": "string",
    "description": "string",
    "salary": "number",
    "roles": "string",
    "open": "boolean",
    "employerId": "number"
  }
]
```

  - **Error (401 Unauthorized):**

```
{
  "message": "Not authenticated"
}
```

  - **Error (403 Forbidden):**

```
{
  "message": "Only employers can view their projects"
}
```

  - **Error (500 Internal Server Error):**

```
{
  "message": "string"
}
```

4.2.3 Get Project by ID

- **Verb:** GET
- **Endpoint:** `/:id`
- **Request Body:** None
- **Responses:**
  - **Success (200 OK):**

```
{
  "idProj": "number",
  "name": "string",
  "description": "string",
  "salary": "number",
  "roles": "string",
  "open": "boolean",
  "employerId": "number",
  "company": "string | null",
  "employerEmail": "string | null"
}
```

– **Error (404 Not Found):**

```
{
  "message": "Project not found"
}
```

– **Error (500 Internal Server Error):**

```
{
  "message": "string"
}
```

4.2.4 Create a New Project

- **Verb:** POST
- **Endpoint: /**
- **Request Body:**

```
{
  "name": "string",
  "description": "string",
  "salary": "number",
  "roles": "string"
}
```

- **Responses:**
  – **Success (201 Created):**

```
{
  "idProj": "number",
  "name": "string",
  "description": "string",
  "salary": "number",
  "roles": "string",
  "open": true,
  "employerId": "number"
}
```

  – **Error (401 Unauthorized):**

```
{
  "message": "Not authenticated"
}
```

  – **Error (403 Forbidden):**

```
{
  "message": "Only employers can create projects"
}
```

- **Error (400 Bad Request):**

```
{
  "message": "string"
}
```

4.2.5 Update a Project

- **Verb:** PUT
- **Endpoint:** `/:id`
- **Request Body:**

```
{
  "name": "string",
  "description": "string",
  "salary": "number",
  "roles": "string",
  "open": "boolean"
}
```

- **Responses:**
  - **Success (200 OK):**

```
{
  "idProj": "number",
  "name": "string",
  "description": "string",
  "salary": "number",
  "roles": "string",
  "open": "boolean",
  "employerId": "number"
}
```

  - **Error (401 Unauthorized):**

```
{
  "message": "Not authenticated"
}
```

  - **Error (403 Forbidden):**

```
{
  "message": "You don't have permission to update this project"
}
```

  - **Error (404 Not Found):**

```
{
  "message": "Project not found"
}
```

- **Error (400 Bad Request):**

```
{
  "message": "string"
}
```

4.2.6 Delete a Project

- **Verb:** DELETE
- **Endpoint:** `/:id`
- **Request Body:** None
- **Responses:**
  - **Success (204 No Content):** No body
  - **Error (401 Unauthorized):**

    ```
    {
      "message": "Not authenticated"
    }
    ```

  - **Error (403 Forbidden):**

    ```
    {
      "message": "You don’t have permission to delete this project"
    }
    ```

  - **Error (404 Not Found):**

    ```
    {
      "message": "Project not found"
    }
    ```

  - **Error (400 Bad Request):**

    ```
    {
      "message": "string"
    }
    ```

4.2.7 Apply to a Project

- **Verb:** POST
- **Endpoint:** `/:id/apply`
- **Request Body:** None
- **Responses:**
  - **Success (201 Created):**

    ```
    {
      "message": "Application submitted successfully"
    }
    ```

  - **Error (401 Unauthorized):**

```
{
  "message": "Not authenticated"
}
```

- **Error (403 Forbidden):**

```
{
  "message": "Only employees can apply to projects"
}
```

- **Error (404 Not Found):**

```
{
  "message": "Project not found"
}
```

- **Error (400 Bad Request):**

```
{
  "message": "string" // e.g., "You have already applied"
}
```

- **Error (500 Internal Server Error):**

```
{
  "message": "string"
}
```

4.2.8 Get Employees for a Project

- **Verb:** GET
- **Endpoint:** `/:id/employees`
- **Request Body:** None
- **Responses:**
  - **Success (200 OK):**

```
[
  {
    "id": "number",
    "projectId": "number",
    "employeeId": "number",
    "joinedAt": "string (ISO date)",
    "status": "string",
    "User": {
      "id": "number",
      "username": "string",
      "firstName": "string",
      "lastName": "string",
      "email": "string"
    }
  }
]
```

  - **Error (401 Unauthorized):**

```
{
  "message": "Not authenticated"
}
```

- **Error (403 Forbidden):**

```
{
  "message": "You don't have permission to view this project's employees"
}
```

- **Error (404 Not Found):**

```
{
  "message": "Project not found"
}
```

- **Error (500 Internal Server Error):**

```
{
  "message": "string"
}
```

4.2.9 Check Application Status for a Project

- **Verb:** GET
- **Endpoint:** `/:id/application-status`
- **Request Body:** None
- **Responses:**
  - **Success (200 OK):**

```
{
  "hasApplied": "boolean"
}
```

  - **Error (401 Unauthorized):**

```
{
  "message": "Not authenticated"
}
```

  - **Error (404 Not Found):**

```
{
  "message": "Project not found"
}
```

  - **Error (500 Internal Server Error):**

```
{
  "message": "string"
}
```

4.2.10 Get All Applications for Current Employee

- **Verb:** GET
- **Endpoint:** `/applications/my`
- **Request Body:** None
- **Responses:**
  - **Success (200 OK):**

```
[
  {
    "projectId": "number",
    "projectName": "string",
    "companyName": "string",
    "role": "string",
    "salary": "number",
    "appliedDate": "string (ISO date)",
    "status": "string",
    "employerEmail": "string | null"
  }
]
```

  - **Error (401 Unauthorized):**

```
{
  "message": "Not authenticated"
}
```

  - **Error (403 Forbidden):**

```
{
  "message": "Only employees can view their applications"
}
```

  - **Error (500 Internal Server Error):**

```
{
  "message": "string"
}
```

4.2.11 Get All Applicants Across Employer's Projects

- **Verb:** GET
- **Endpoint:** `/applicants/all`
- **Request Body:** None
- **Responses:**
  - **Success (200 OK):**

```
[
  {
    "applicationId": "number",
    "projectId": "number",
    "projectName": "string",
    "employeeId": "number",
    "employeeName": "string",
    "employeeEmail": "string",
    "role": "string",
    "appliedDate": "string (ISO date)",
    "status": "string"
  }
]
```

  - **Error (401 Unauthorized):**

    ```
    {
      "message": "Not authenticated"
    }
    ```

  - **Error (403 Forbidden):**

    ```
    {
      "message": "Only employers can view applicants"
    }
    ```

  - **Error (500 Internal Server Error):**

    ```
    {
      "message": "string"
    }
    ```

4.2.12 Update Application Status

- **Verb:** PUT
- **Endpoint:** `/applications/:id/status`
- **Request Body:**

  ```
  {
    "status": "string" // "accepted" or "rejected"
  }
  ```

- **Responses:**
  - **Success (200 OK):**

    ```
    {
      "message": "Application [accepted/rejected] successfully"
    }
    ```

  - **Error (401 Unauthorized):**

    ```
    {
      "message": "Not authenticated"
    }
    ```

  - **Error (403 Forbidden):**

```
{
  "message": "Only employers can update application status"
}
```

- **Error (404 Not Found):**

```
{
  "message": "Application not found"
}
```

- **Error (400 Bad Request):**

```
{
  "message": "Invalid status value"
}
```

- **Error (403 Forbidden):**

```
{
  "message": "You don't have permission to update this application"
}
```

- **Error (500 Internal Server Error):**

```
{
  "message": "string"
}
```

5 ER Diagram

The following ER diagram 3 illustrates the relationships between the `User` and `Project` models, including the many-to-many relationship mediated through the `ProjectEmployee` join table. The `User` entity represents both employers and employees (distinguished by the `userType` attribute), where an employer creates projects and the employees apply to them. The diagram captures the one-to-many relationship between `User` (employer) and `Project`, as well as the many-to-many relationship between `Project` and `User` (employees).

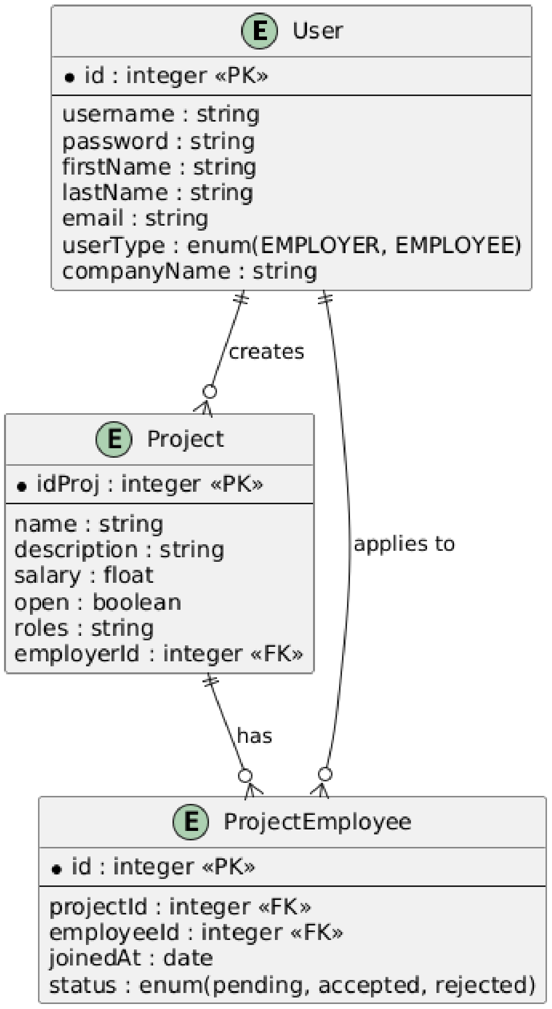


Figure 3: ER Diagram for User and Project Models

6 Responsibilities

Equal contribution between Emil and Sami.