Task Management App with Svelte, Flowbite, Directus, and MySQL - SOP

1. Objective

The purpose of this document is to provide a comprehensive guide for setting up a task management application using Svelte for the frontend, Flowbite for UI components, Directus for backend management, and MySQL as the database and create basic functionalities such as user login and task management.

2. Setting Up the Environment

To start developing the task management application, the development environment needs to be setup with the necessary tools and software.

Installing Node.js (Version 18) and npm

Node.js is a JavaScript runtime that allows you to run JavaScript on the server side, and npm (Node Package Manager) is used to manage dependencies for your project. For this project, we will be using Node.js version 18 as directus accepts Node Version of v18.20.4.

Steps to Install Node.js and npm:

1. **Download Node.js**:
    - Visit the official Node.js website.
    - Download the installer for Node.js version v18.20.4 suitable for the desired operating system (Windows, macOS, or Linux).
2. **Install Node.js**:
    - Follow the installation instructions provided for your operating system:
        - **Windows**: Run the downloaded installer and follow the setup wizard.
        - **macOS**: Run the downloaded .pkg file and follow the setup wizard.
        - **Linux**: Use a package manager to install Node.js. For example, on Ubuntu, you can use:

```
curl -sL https://deb.nodesource.com/setup_18.x | sudo -E bash -

sudo apt install -y nodejs
```

3. **Verify the Installation**:
    - Open a terminal or command prompt and check the installed versions of Node.js and npm:

      node -v

      npm -v

      ```
      PS C:\Users\KSSPL> node -v
      v18.20.4
      PS C:\Users\KSSPL> npm -v
      10.7.0
      ```

    - You should see the version numbers for Node.js (v18.20.4) and npm, indicating that they have been installed successfully. The npm version which is used for this project is 10.7.0.

Installing MySQL

MySQL is a popular relational database management system used to store and manage data for the application.

Steps to Install MySQL:

1. **Download MySQL**:
    - Visit the official MySQL website.
    - Choose the appropriate installer for your operating system (Windows, macOS, or Linux).
2. **Install MySQL**:
    - Follow the installation instructions provided for your operating system:
        - **Windows**: Run the downloaded installer and follow the setup wizard. During the setup process, you can configure MySQL with a root password and choose the default settings.
        - **macOS**: Run the downloaded .dmg file and follow the setup wizard. You can configure MySQL with a root password and choose the default settings.
        - **Linux**: Use a package manager to install MySQL. For example, on Ubuntu, you can use:

          ```
          sudo apt update
          sudo apt install mysql-server
          ```

    - Secure your MySQL installation by running:

      ```
      sudo mysql_secure_installation
      ```

- Follow the prompts to set up a root password and secure your MySQL installation.

3. **Verify the Installation**:
   - Open a terminal or command prompt and check the MySQL service status:

     ```
     sudo service mysql status
     ```

   - You should see a message indicating that the MySQL service is running.

4. **Log in to MySQL**:
   - Log in to the MySQL server using the root account to ensure everything is working correctly:

     ```
     mysql -u root -p
     ```

   - Enter the root password you set during the installation process. You should see the MySQL prompt, indicating that you are logged in to the MySQL server.
   - Create a database named directus:

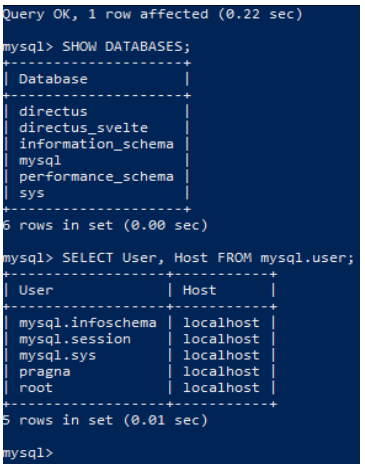
     ```
     CREATE DATABASE directus;
     ```

   - Create a new user and grant them full access to the new database using the following commands:

     ```
     CREATE USER 'your_username'@'localhost'
         IDENTIFIED BY 'your_password';


     GRANT ALL PRIVILEGES ON directus.* TO
         'your_username'@'localhost';


     FLUSH PRIVILEGES;
     ```

   - Check Database with the MySQL prompt:

     `SHOW databases;` to show databass and

     `SELECT User, Host FROM mysql.user;` to check if user is created properly.

With Node.js, npm, and MySQL installed, you are now ready to proceed with setting up the Directus backend and configuring it to work with your MySQL database.

3. Directus Setup with MySQL

Directus is a versatile open-source data platform that provides a user-friendly interface for managing your database. This section outlines the steps to set up Directus with a MySQL database.

Installing Directus

1. **Create a New Directus Project**:
    - Open a terminal or command prompt.
    - Run the following command to create a new Directus project:

```
npx create-directus-project backend
```

    - You will be prompted to choose your database client. Use the arrow keys to select "`MySQL / MariaDB / Aurora`" and press Enter.
2. **Configure Directus with MySQL**:
    - You will be asked to provide the following database configuration details:
        - **Database Host**: Enter `127.0.0.1` (default for local MySQL installations).
        - **Port**: Enter `3306` (default MySQL port).
        - **Database Name**: Enter `directus` (the database you created earlier).
        - **Database User**: Enter username (in my case it's pragna) (the MySQL user).
        - **Database Password**: Enter the password for the MySQL user.
3. **Create an Admin User**:
    - You will be prompted to create an initial admin user for Directus:
        - **Email**: Enter `admin@example.com` (or your preferred admin email).
        - **Password**: Enter a secure password for the admin account.
4. **Project Creation Confirmation**:
    - Once the project is created, you will see a confirmation message with the path to your project directory and configuration file:

```
Your project has been created at F:\Task-Management-App\backend.
The configuration can be found in F:\Task-Management-App\backend/.env
```

5. **Start Directus**:

    Navigate to the project directory and start the Directus server by running:

```
cd backend
npx directus start
```

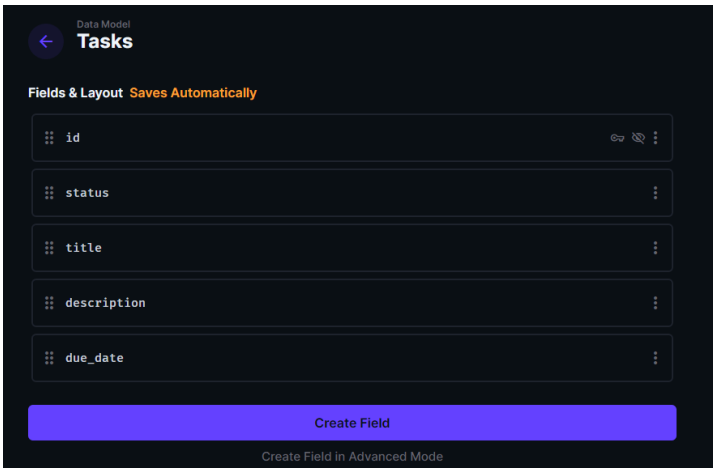
- Directus will start and you can access the admin interface through your web browser at `http://localhost:8055` (default port).

4. Configuring Directus for Task Management

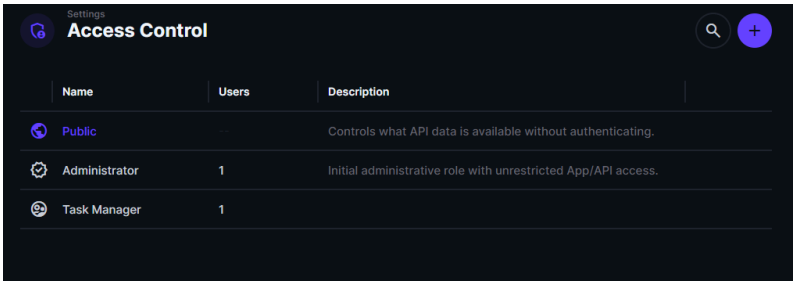
After setting up Directus, the next steps involve configuring it for managing tasks and setting up user authentication.

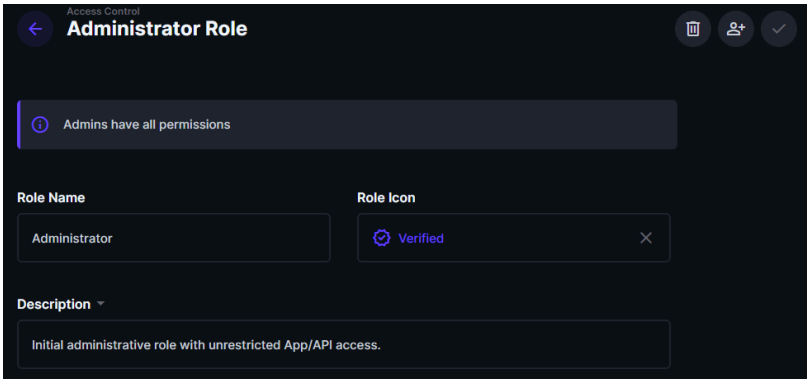
Configuring Directus for Task Management

1. **Log in to Directus Admin Interface**:
   - Open a web browser and navigate to http://localhost:8055.
   - Log in using the admin credentials you created (admin@example.com and the password).
2. **Create a Collection for Tasks**:
   - In the Directus admin interface, navigate to the Settings -> Data Model section.
   - Click on the "+ Create Collection" button.
   - Enter a name for the collection, e.g., tasks (in my case tasks)
   - Add fields to the tasks collection:
     - **title**: Add a field of type input for the task title.
     - **description**: Add a field of type Textarea for the task description.
     - **status**: Add a field of type Dropdown for task status (e.g., to-do, in-progress, done - this can be changed by clicking on status -> select Interface and modify data in “Choices”).
     - **due_date**: Add a field of type Datetime for the task due date.
   - Save the collection.



3. **Set Up Permissions**:
    - Go to the Settings -> Access Control section.
    - Configure permissions for different roles (e.g., Admin, Task Manager) to ensure they can create, read, update, and delete tasks as needed.
    - For the Admin role, ensure full permissions for the tasks collection.

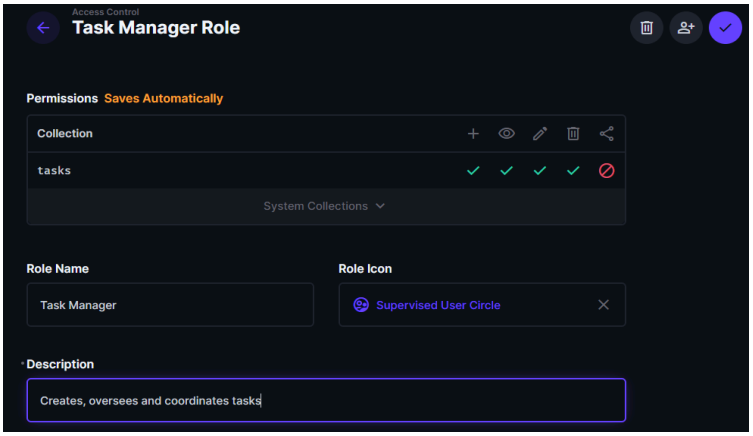




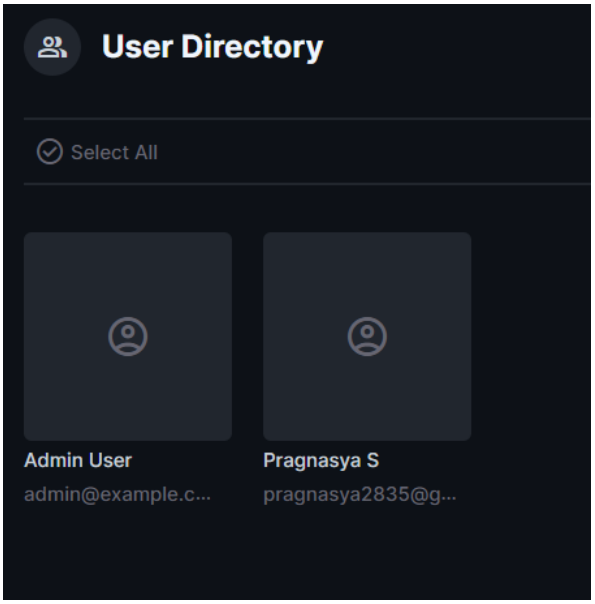
Implementing User Authentication

Directus comes with built-in user authentication. Here's how you can set it up:

1. **Create Additional User Roles**:
    - In the "Access Control" section create roles if necessary, such as Manager or Employee, depending on the needs of your application.
    - Configure these roles with the appropriate permissions for managing tasks.



2. **Create Users**:
    - Go to the “User Directory” section.
    - Click on the "+ Create Item" button.
    - Enter user details such as Firstname, Lastname, email, password and assign role based on their access needs (scroll below to see the Admin Options).
    - Save the new users.



3. **Test Authentication**:

    To ensure that the authentication system in Directus works correctly, you can use Postman to test the user login and role-based access. Follow these steps:

    - Open Postman and create a new request to test user authentication.
    - Use the POST method to send a request to the Directus login endpoint:

```
POST http://localhost:8055/auth/login
```

    - In the request body, include the credentials for the user you want to test:

```
{
  "email": "user@example.com",
  "password": "userpassword"
}
```

    - Replace `user@example.com` and `userpassword` with the credentials for the user you are testing.
    - Send the request and verify that you receive a successful response with a token.

With Directus configured for task management and user authentication, you can now move on to integrating the frontend using Svelte and Flowbite.

5. Frontend Integration with Svelte and Flowbite

In this section, a basic Svelte project will be setup and is integrated with Flowbite to create a user-friendly interface. You will also use the Directus SDK to interact with the Directus API.

Setting Up a Basic Svelte Project

1. **Create a New Svelte Project**:
    - Open a terminal or command prompt.
    - Use Vite to create a new Svelte project by running:
        - `npm create vite@latest frontend --template svelte`
    - This command sets up a new Svelte project in a directory named frontend.
2. **Navigate to the Project Directory**:
    - Change to the newly created project directory:

```
cd frontend
```

3. **Install Project Dependencies**:
    - Install the required npm packages:

```
npm install
```

4. **Start the Development Server**:
    - Start the Svelte development server:

```
npm run dev
```

    - Open your browser and go to `http://localhost:5173` to see your Svelte application running.

Integrating Flowbite

1. **Install Flowbite**:
    - Flowbite is a component library for Tailwind CSS. To use it with Svelte, first, ensure that Tailwind CSS is installed in your Svelte project.
    - Install Flowbite and Tailwind CSS using npm:

```
npm install tailwindcss flowbite
```

2. **Configure Tailwind CSS**:

- Initialize Tailwind CSS in your Svelte project:

```
npx tailwindcss init
```

- Update the tailwind.config.js file to include Flowbite and configure paths to your CSS files:

3. **Add Tailwind CSS to Your Project**:

```
const flowbitePlugin = require('flowbite/plugin');
/** @type {import('tailwindcss').Config} */
module.exports = {
  content: ['./src/**/*.{html,js,svelte,ts}',
'./node_modules/flowbite-svelte/**/*.{html,js,svelte,ts}'],
  darkMode: 'class', // Changed 'selector' to 'class' for dark mode
  theme: {
    extend: {
      colors: {
        // flowbite-svelte
        primary: {
          50: '#FFF5F2',
          100: '#FFF1EE',
          200: '#FFE4DE',
          300: '#FFD5CC',
          400: '#FFBCAD',
          500: '#FE795D',
          600: '#EF562F',
          700: '#EB4F27',
          800: '#CC4522',
          900: '#A5371B'
        }
      }
    }
  },
  plugins: [flowbitePlugin]
};
```

- Include the Tailwind CSS directives in your src/app.css file:

```
@import 'flowbite/dist/flowbite.css';

@tailwind base;
@tailwind components;
@tailwind utilities;
```

Implementing Directus APIs

1. **Configure the Directus APIs**:

- Create a file named config/auth.js in your src directory and configure the Directus APIs:

  Login - http://localhost:8055/auth/login

  Logout - http://localhost:8055/auth/logout

2. **Add Forms for Login, Adding and Deleting Tasks**:

   - Implement Login form and functionality to add and delete tasks using the Directus APIs. Example code for fetching all tasks, adding, deleting, updating(optional) a task:

```
async function fetchTasks() {
    try {
      const response = await fetch("http://localhost:8055/items/tasks", {
        headers: {
          Authorization: `Bearer ${sessionStorage.getItem("token")}`,
        },
      });

      if (response.ok) {
        const data = await response.json();
        tasks = data.data || [];
      } else {
        console.error(
          "Failed to fetch tasks",
          response.status,
          response.statusText
        );
        tasks = [];
      }
    } catch (error) {
      console.error("Error fetching tasks", error);
      tasks = [];
    } finally {
      loading = false;
    }
  }

async function addTask() {
    try {
      const response = await fetch("http://localhost:8055/items/tasks", {
        method: "POST",
        headers: {
          "Content-Type": "application/json",
          Authorization: `Bearer ${sessionStorage.getItem("token")}`,
        },
        body: JSON.stringify(newTask),
      });
```

```
    if (response.ok) {
      const data = await response.json();
      tasks = [...tasks, data.data];
      resetEditing();
    } else {
      console.error(
        "Failed to add task",
        response.status,
        response.statusText
      );
    }
  } catch (error) {
    console.error("Error adding task", error);
  }
}

async function updateTask(editForm) {
  console.log("updateTask is called",editForm);
  try {
    const response = await fetch(
      `http://localhost:8055/items/tasks/${editForm.id}`,
      {
        method: "PATCH",
        headers: {
          "Content-Type": "application/json",
          Authorization: `Bearer ${sessionStorage.getItem("token")}`,
        },
        body: JSON.stringify(editForm),
      }
    );

    if (response.ok) {
      const data = await response.json();
      tasks = tasks.map((task) =>
        task.id === newTask.id ? data.data : task
      );
      resetEditing();
    } else {
      console.error(
        "Failed to update task",
        response.status,
        response.statusText
      );
    }
  } catch (error) {
    console.error("Error updating task", error);
  }
}

async function handleDeleteTask(taskId) {
  try {
    const response = await fetch(
```

```
        `http://localhost:8055/items/tasks/${taskId}`,
        {
          method: "DELETE",
          headers: {
            Authorization: `Bearer ${sessionStorage.getItem("token")}`,
          },
        }
      );

      if (response.ok) {
        tasks = tasks.filter((task) => task.id !== taskId);
      } else {
        console.error(
          "Failed to delete task",
          response.status,
          response.statusText
        );
      }
    } catch (error) {
      console.error("Error deleting task", error);
    }
  }
```

5. **Conclusion**

In this project, the task management application using Directus as the backend, MySQL as the database, and Svelte with Flowbite for the frontend is successfully setup. Directus is configured to handle task management with a MySQL database, enabling functionalities for creating, updating, and deleting tasks while ensuring secure user authentication. The frontend was built using Svelte, with Flowbite providing a modern, responsive UI.

This setup demonstrates a complete workflow from backend configuration to frontend implementation, delivering a fully functional task management system as per the requirement.