INFWAD Project description

Office Calendar Application

Introduction

Calendify is a young tech startup focused on creating solutions that encourage employees to return to the office more frequently. For this practical case, you will *design and develop* a calendar web application designed to be used in an office environment. The web application will allow employees and managers to:

- Check when their colleagues are planning to work in the office
- Sign up for company events like boardgame nights, hack-a-thons or team events
- Book meeting rooms directly through the application

This project gives you the opportunity to work on a real-world application that balances technical development with usability and collaboration.

Collaboration & Workflow

- You may work in teams of four students
- A shared GitHub repository is *highly recommended* to manage your code and track your team's progress
- Good teamwork and communication will be as important as your final product. Make use of your experience with Agile and SCRUM.

Project Goal & Value

The purpose of this project is to *design & develop* a social agenda application that helps employees and managers to easily see office and event attendance. Currently, it's difficult for teams to know:

- Who will be in the office.
- Who is attending specific events.
- How to effectively manage and encourage participation.

Your application will solve these problems by:

- Providing a clear, shared overview of office presence and event attendance.
- Making it easy for employees to sign up for events and book meeting rooms.
- Allowing managers to announce events, view attendance history, and send push notifications.

What makes your product stand out is its simplicity, accessibility, and inclusivity:

- It will be **intuitive** to use for all employees, regardless of technical skill.
- It will be **usable** across devices (desktop, tablet, mobile).
- It will be **accessible** for people with disabilities (for example, conform to WCAG)
- It will be more than just a calendar—it will function as a social hub that motivates employees to collaborate and connect in person.

Possible Entity Relationship Diagram for the Database:

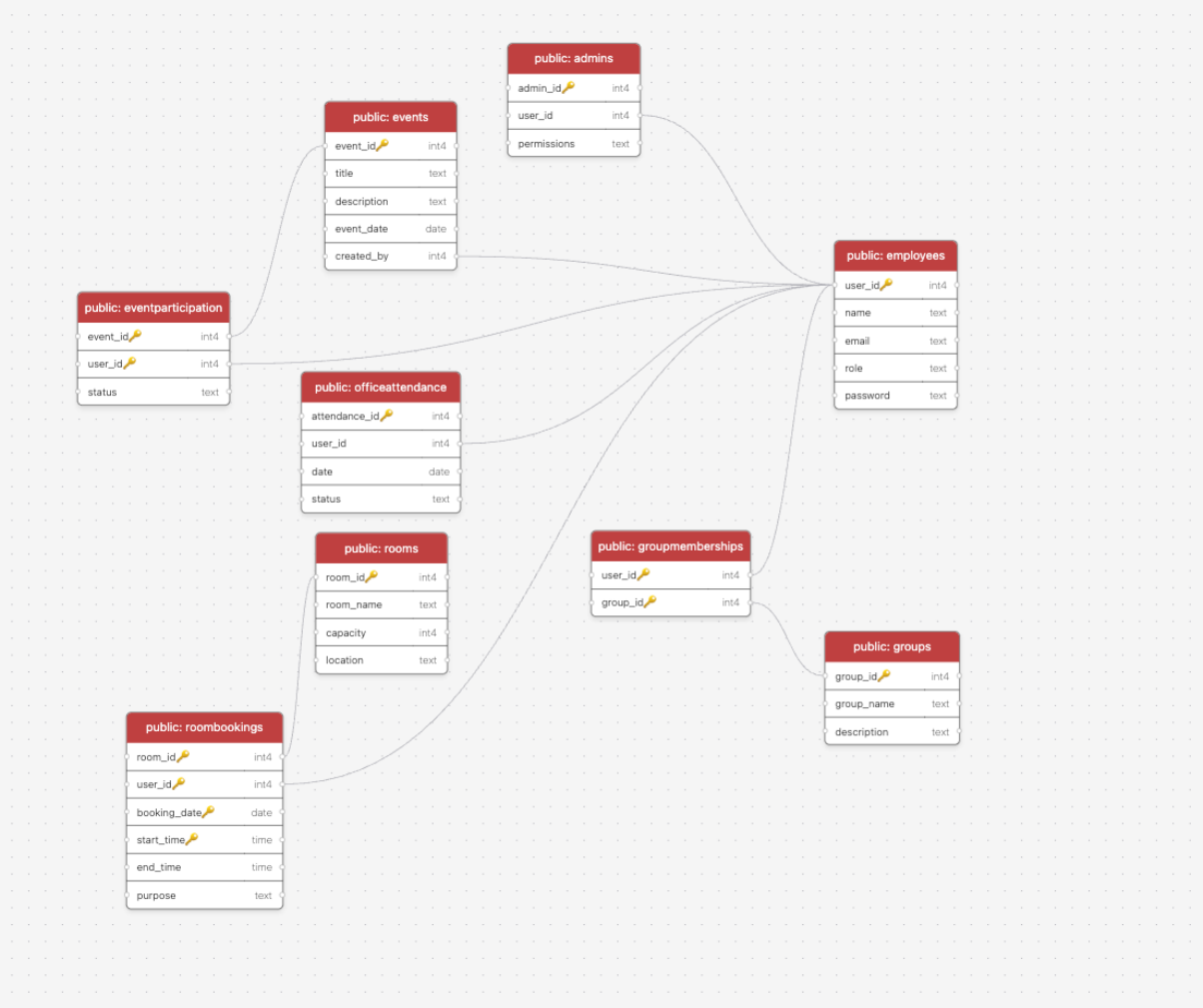
Given for this project is the database design shown in the diagram below. There is a standalone Admin table that will be used to be able to login to the administrator dashboard. The Employee is also able to login and register.

An Employee can attend many Events, and an Event can be attended by many Employees, this is a so called many-to-many relationship solved in the relational database by the table EventParticipation. The OfficeAttendance specifies if an Employee is present in the office.

Employees can book rooms for meetings. Employees might belong to different groups (departments), a group of employees might consist of no or some employees.

Possible modifications:

- Add permission changes (read/update) of events/room bookings based on group's membership.
- Add location to the Events using the Rooms table.



Features & Requirements

Below you will find the project requirements that the student needs to implement during the web development course. Before you can start you will need to have a development environment ready with C#/.NET and Node.js.

Frontend: React & Typescript

The upcoming tasks will focus on building the front-end of the application. Each front-end feature is. Connected to a back-end feature that must also be implemented in the final product in the final product for the application to work properly. In the initial development phase, since the backend is not yet ready at start, you will use mock data to test and build the user interface before connecting it to the back-end you will develop. No specific front-end design is provided. Creating your designs or mock-ups, however, is optional but strongly recommended, as it could help you better visualize how your application should look and behave.

1.1. Develop a React component that represents the login screen

- The login screen (react component) consumes the login API that is going to be built in the back-end part of the application.
- The component displays messages it receives from the API *(i.e. Password is incorrect).*
- When the admin user is logged in, the component should redirect to the dashboard page.
- When an unauthorized user tries to reach the administrator dashboard page, the application should redirect to the login screen.

1.1. Develop an administrative dashboard from where the admin can do the following:

- View a table with an overview of all (calendar) events that are in the system.
- Admin can use a form to submit a new (calendar) event to a POST endpoint, where the endpoint will be later implemented on the back-end.
- Admin can edit events via an UPDATE endpoint, where the endpoint will be later implemented on the back-end.
- Admin can delete events (a pop-up confirms the deletion before actually deleting the event from the back-end).
- Admin should be able to view a list of attendees that are signed up for an event.

1.3. Homepage of the application should show a list of available events

- The list should only show the events that are in the future.
- When you click an event, application takes you to a page containing more details about that specific event.
- Homepage is only visible for logged in users, if the user is not logged in, the application should redirect to the login screen.

1.4. Attend a specific event via a form

- Form matches the structure of the POST endpoint (that is going to be implemented in the back-end phase) to track attendance for the event.
- Users can delete events from their calendar.

1.5. Logged in users can book a room for a meeting

- The user can decide to book an available room if available for a particular day/time range.
- Overview of room availabilities will change accordingly as they are booked.

1.6. Multi-Page Navigation with React Router

- Navigation between pages updates the URL to reflect the current view.
- When visiting a specific URL (copy-pasting to URL bar of the browser), the correct component/page loads.
- URL parameters (e.g., filters, IDs) are supported and passed correctly through React Router to the API endpoints.

1.7. User can specify its attendance in the office

- Design and implement a user interface that allows users to specify their attendance.
- The UI design is up to you – choose a solution that you wish.
- Attendance data should be stored and retrievable (data persistence part is going to be implemented in the back-end side).

Backend: ASP.Net

This section outlines back-end tasks. Read all requirements first, as many depend on earlier steps. Structure your solution to maximize code reuse (shared services, validation, DTOs, and utilities) so later features build on earlier work. All data must be stored in a relational database that implements the provided ERD. Use the supplied domain model and instructions as your source of truth or *inspiration* when designing entities, relationships, and persistence logic. Avoid schema changes unless explicitly requested. In short: build the backend in ASP.NET, keep it modular and reusable, and persist data exactly as defined by the ERD and model.

2.1. The student needs to implement a login system for an admin user. This will lay the basic fundament for authorization to our application.

- Login system will consist of a POST call that receives a username and password and checks if the password is correct with what is in the database. This endpoint will also register a session on the server.
- POST endpoint returns a success message if the password is correct, else it returns a reasonable error message of what went wrong.
- Create a GET endpoint that returns a Boolean value based on if the session is registered or not. Additionally, return the name of the admin user that is logged in.
- The login logic and endpoints need to be separated into a Service and a Controller.

2.2. The student needs to create a CRUD (Create, Read, Update, Delete) API controller for the Event entity. This is so that the admin can manage events.

- The Read operation needs to be a GET endpoint and must be public. This endpoint will also be used by the front-end to display an overview of events.
- Include the reviews and attendees to the event entity in the response of the GET operation.
- The other endpoints (Create, Update and Delete) needs to be protected and requires an admin to be logged in to continue with the execution of the method
- The following methods are used to implement the CRUD operations inside the Controller: Create => POST, Read => GET, Update => PUT, Delete => DELETE.
- The JSON body of the Create endpoint needs to contain the following attributes: Title, description, date, start time, end time and location.
- Make it possible to attach a review to a specific event.

2.3. Extend the login service in such a way that it can handle multiple roles.

- A normal user/employee must be able to login to the application.
- A user must register in order to use the application, registration can be handled by the login controller.
- There must be a clear distinction between admin and user roles. The user can only view events and eventually attend them. Only an admin can create, update and delete events.

2.4. Make a POST endpoint inside a new controller that allows a logged in user to attend a new event.

- The JSON body to submit an attendance should contain a user id and an event id.
- Endpoint returns the event that the user has attended.
- Endpoint is responsible to check the availability of the event based on the date and start time.
    - If the above condition fails, the endpoint needs to provide an error message of what went wrong.

- Inside the same controller, there should be a GET endpoint that allows a logged-in user to view the list of attendees.
- The controller can delete events that the user attended, in case the user cannot attend the event anymore.

2.5. Make a new controller that allows a logged-in user to modify its attendance in the office.

- Endpoints are protected by a login.
- Endpoints are accessible by the user that is logged-in, the user may only edit its own attendance.
- You can re-use the code from the login service.
- API endpoint returns a reasonable error message If the user tries to book a date that is already occupied

2.6. Make a new controller that allows a logged-in user to book a room.

- Endpoints are protected by a login.
- Endpoints are accessible by the user that is logged-in, the user may only book an available room.
- You can re-use the code from the login service.
- API endpoint returns a reasonable error message If the user tries to book a date that is already occupied

Custom feature

3.1. There must also be at least one custom feature implemented in the project to distinguish the application from others

- Decide with your team what you would like to do (if you'd like to learn more about front-end animation, think of a front-end heavy feature, for example), and create a clear description and requirements for this.
- The feature must first be proposed to your teacher and approved. Make sure you do this before you start implementing it.