Practical case
Office Calendar Application

Calendify is a new tech startup that wants to build a new solution to make employees go back to the office more often.

For this practical case the student will build a calendar application for an office. From the web application the employees and managers are able to see when co-workers are going to the office. From the application employees can also sign up for events.

Working on the project

The students are allowed to work in groups of a maximum of 4 students. It is recommended to work together in a collaborative GitHub repository.

What is the purpose of this project?

The goal of this project is to create a 'social agenda' application that will make it clear to *Calendify* when employees will come to work and show up to certain events like cooking nights or getaways. This app should also encourage employees to show up to work physically as much as possible.

What problem does this project solve?

The website makes it easier for employees to see who will come to the office/events. It will also be easier for managers to announce and manage attendance to events

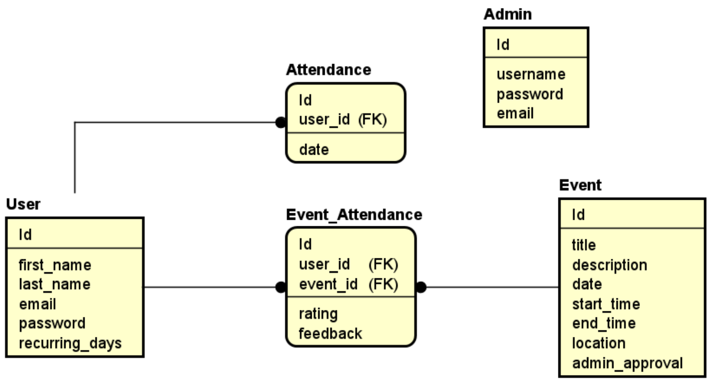
How your product is better than existing solutions

It is very intuitive and very easy to use and is created for all types of users. The app makes clear who is attending what and managers are capable of viewing attendance history. Push notifications can be sent when a new event is planned; the app will be easily accessible and functions on any device.

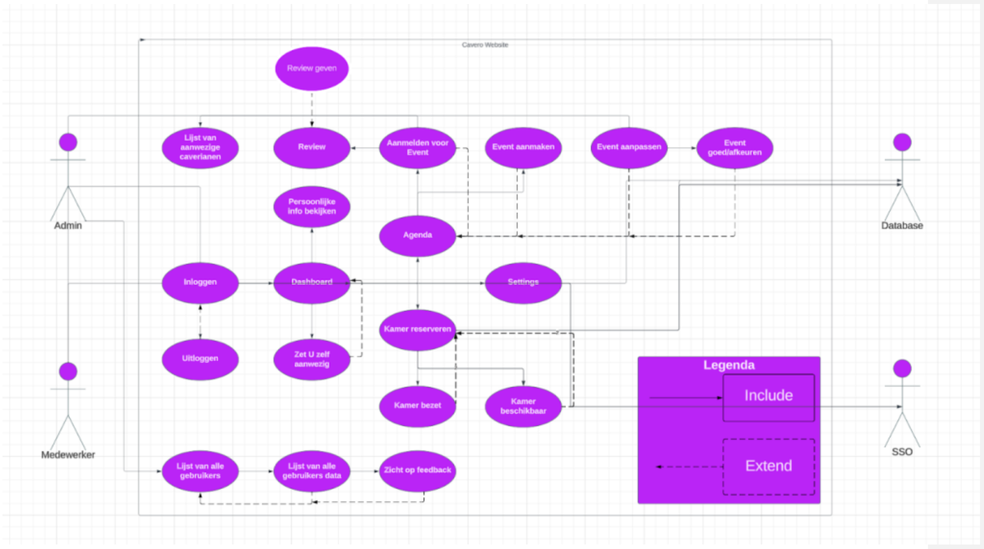
ERD

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 User is also able to login and register.

A User can Attend many Events and a User can place many Reviews to a single Event. The Attendance specifies if a User is present in the office.



Usecase diagram



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 a given database model.

Backend

Below you will find the tasks you need to implement for the backend. Read all the requirements carefully. Re-use of code is required since the requirements are dependent on each other.

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

- The 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.
- The POST endpoint should return a success message if the password is correct, else it should return reasonable feedback 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 in a Service and a Controller.

1.2. The student needs to create a CRUD(Create, Read, Update, Delete) API controller for the Event entity. This is so that the admin is able to 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 in order to continue with the execution of the method
- The following methods need to be 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.
- Reuse the login Service created in the previous task.

1.3. 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.

1.4. 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.
- The endpoint should return the event the user has attended.
- The endpoint should be 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 a feedback message of what went wrong.
- Inside the same controller there should be a protected GET endpoint that allows a logged-in user to view the list of attendees.
- The controller also needs to be able to delete events that the user attended, in case the user is not able to attend the event anymore.

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

- The endpoints should be protected by a login.
- The endpoints should only be accessible by the user that is logged-in, the user may only edit its own attendance.
- Re-use the code from the login service.
- If the user tries to book a date that is already occupied, then the API should return a reasonable error message.

Frontend: React

The following tasks will focus on building the front-end of the application. Every front-end requirement depends on a back-end requirement that needs to be present in the final product in order to complete the assignment. There is no design given for the front-end, creating a design or mock-ups are optional, but may help to get an insight on how the application should work.

2.1. The student needs to implement a React component that represents the login screen

- The component needs to consume the login API build earlier.
- The component needs to display 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

2.2. The student needs to build an administrative dashboard from where the admin can do the following:

- View a table with an overview of all events that are in the system
- There needs to be a form that the admin can use to submit a new event to the earlier created POST endpoint
- The admin needs to be able to edit events
- The admin needs to be able to delete events, when deleting a show there needs to be a popup that asks for confirmation before the actual delete request takes place
- The admin should be able to view a list of attendees that are signed up for the event.

2.3. The 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 on an event you should go to a page containing more details about that specific event.
- This is a private application, so the homepage should only be visible for logged in users, if the user is not logged in it should redirect to the login screen.

2.4. Make it possible through a form to attend a specific event

Commented [K(1]: Employees also need to be able to remove events from their calendar

- The form needs to match the structure of the earlier made POST endpoint to create an attendance
- The user also needs to be able to delete events from their calendar.

2.5. Make it possible for logged in users to place a review with feedback under an event.

- Only logged in users should be able to place feedback for an event.
- The student may decide what rating system will be used (stars, range selection etc.).
- The average rating score should be visible next to an event in the front-end.

2.6. Separate the React Application in different pages using the React Router

- You should see the URL change when you navigate through the site
- A specific component should be loaded when you visit the site directly from a specific URL
- URL parameters for filtering should also be passed through the application and to the API endpoints through the React router

2.7. The user needs to be able to specify its attendance in the office.

- Create a user interface solution yourself for this requirement. Custom feature

Custom feature

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

- The feature has been proposed to the teacher and approved.

Grading & Final delivery

For grading of the ***project*** part of this course there will be two assessments a mid-term (backend) and a final assessment (front-end and connection to web API), a repair assessment will take place at the end of the course. The grade will be individual. This will be based on the assessment of the final product (up to that moment), corrected by the assessment of the individual contribution.
The repair assessment will follow the same criteria but applied on the full project (backend, front-end, integration).

We expect the student to have implemented a fully functioning web application with both a backend and a frontend. All the backend requirements have a frontend dependency that needs to be implemented with it in order to be suitable for grading.

Requirement	Dependent on	Indicator*
1.1	2.1.	
1.2	2.2, 2.3, 2.5.	
1.3	2.1.	
1.4	2.4.	
1.5.	2.7	
2.1.	1.1.	
2.2.	1.2.	
2.3.	1.2.	
2.4.	1.4.	
2.5.	1.2.	
2.6.	-	
2.7.	1.5.	

*Indicator can be: verry poor, insufficient, satisfactory or excellent. This will be based on how many bullet points are met that are mentioned below each requirement.

Indicator	Very poor (1 point)	Insufficient (5 points)	Satisfactory (7.5 points)	Excellent (10 points)
The student proofs to have a clear understanding of the Web App technologies and is able to show this by implementing the features based on the given requirements	The feature is not present at all in the final web application, so the student didn't show its understanding of the web technology for the given topic.	The feature is present in the final web application, but not does not match every requirement that is given	The feature is present in the final web application and meets all the given requirements that are given.	The feature is present in the final web application, meets all the requirements that are given and the code standards from the theory lessons are well applied.

Commented [I(2]: For grading of the ***project*** part of this course there will be two assessments a mid-term (backend) and a final assessment (front-end and connection to web API), a repair assessment will take place at the end of the course. The grade will be individual. This will be based on the assessment of the final product (up to that moment), corrected by the assessment of the individual contribution.
The repair assessment will follow the same criteria but applied on the full project (backend, front-end, integration).
We expect the student to have implemented a fully functioning web application with both a backend and a frontend. All the backend requirements have a frontend dependency that needs to be implemented with it in order to be suitable for grading.