

**Wydział Informatyki**

**Katedra Baz Danych**

Specjalizacja Baz Danych

**Tu Alexander, Zembrzuska Magdalena, Zviezdin Illia**

numery albumów: s20290, s20983, s21120

**Stworzenie narzędzia do edukacji studentów kierunków medycznych za pomocą symulacji badań EKG**

Praca inżynierska  
dr Ida Jokisz

Warszawa, styczeń 2022

# **Streszczenie**

Głównym celem pracy było stworzenie funkcjonalnej i intuicyjnej aplikacji internetowej do pomocy przy kształceniu studentów kierunków medycznych na polskich uczelniach. Aplikacja miała na celu stworzenie narzędzia zarówno dla studentów jak i ich profesorów oraz możliwość łatwego sprawdzenia wiedzy w zakresie analizy wykresów uzyskanych podczas wykonywania badania diagnostycznego - elektrokardiogramu i na tej podstawie oceny zdrowia pacjenta, a także zdiagnozowania przebytych chorób czy obecnych zmian morfologicznych.

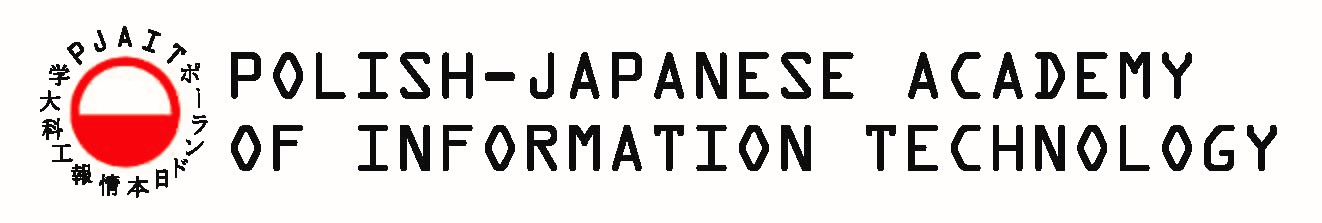
Praca zawiera szczegółowy opis funkcjonalności dostarczanych przez aplikację internetową, architekturę tej aplikacji, opis sposobów w jaki różne części projektu są ze sobą związane, modele zapewniające wizualizacje pewnych konceptów, oprogramowanie użyte do budowy aplikacji wraz z uzasadnieniem wyboru danego oprogramowania oraz instrukcje użytkowania aplikacji internetowej wraz z testami.

Pierwszy rozdział…

Drugi rozdział…

Trzeci rozdział…

**Słowa kluczowe:** elektrokardiogram, edukacja, aplikacja internetowa, aplikacja fullstack, aplikacja medyczna, aplikacja szkoleniowa



**Faculty of Information Technology**

**Chair of Databases**

Databases Specialization

**Tu Alexander, Zembrzuska Magdalena, Zviezdin Illia**

student no.: s20290, s20983, s21120

**Creation of a tool for education of students of medical related studies with simulating ECG**

Bachelor of Engineering Thesis  
dr Ida Jokisz

Warsaw, January 2022

# **Abstract**

The main goal of the project was to create a functional and intuitive web application to help educate students of medical faculties at Polish universities. The purpose of the application was to create a tool for both students and their professors, as well as an opportunity to easily test their knowledge in the field of analysis of graphs obtained during a diagnostic test - electrocardiogram and, on this basis, assess the patient's health, as well as diagnose past diseases or current morphological changes.

The paper contains a detailed description of the functionalities provided by the web application, the architecture of this application, a description of the ways in which various parts of the project are related to each other, models that provide visualization of certain concepts, software used to build the application along with the justification for the choice of the given software and instructions for using the web application along with tests.

The first chapter…

The second chapter…

Third chapter…

**Keywords:** electrocardiogram, education, web application, full-stack application, medical application, training application

[**Streszczenie**](#_rxwmnwmts3kh) **2**

[**Abstract**](#_jiahd0ppfy1a) **4**

[**List of Figures**](#_rjgl8669y2bj) **8**

[**1. Introduction**](#_2y35zjayo40n) **10**

[1.1. Aim](#_e1ziuinsvlox) 11

[1.2. Scope](#_pdjj4h298f) 12

[1.3. Overview of current solutions](#_vxjp7g3fqp69) 12

[1.3.1. ecg-quiz.com](#_dbfx7j5i7c9) 13

[1.3.2. medschool.co](#_v5g3bl9lk77z) 14

[**2. Requirements**](#_xwboq5cybl3) **15**

[2.1. Functional requirements](#_n4wj38wk4vj0) 15

[2.1.1. Description of the system functionalities](#_5szumyym13pq) 16

[2.1.1.1. Actors](#_mbrhit4tap6g) 16

[2.1.1.2. Functionalities](#_3qc04jxjz8je) 16

[2.1.2. Use case diagram](#_macoik2ob8qe) 18

[2.1.3. Entity diagram](#_901bvt7w3fc6) 19

[2.1.4. Use case scenarios](#_ws1y58megn62) 20

[2.2. Non-functional requirements](#_wm7cdmh9pi30) 33

[2.2.1. Usability](#_xlzj8l4llup0) 33

[2.2.2. Security](#_1744dzagtkar) 34

[**3. Application architecture**](#_miw6uzgv4ok3) **36**

[3.1. Description of the prototypes](#_vsay8x2k3j87) 36

[3.2. General description of the architecture](#_vi9p9wr111vo) 36

[3.3. C4 Model](#_97b5feo4z3d3) 36

[3.3.1. Level 1](#_7770jq4n1ibf) 37

[3.3.2. Level 2](#_qjkl9vstdzt4) 39

[**4. Software**](#_pbmkz0lglkua) **41**

[4.1. Front end](#_99ananrxw8cp) 41

[4.1.1. Programming language - Typescript](#_531i2rv8ehjd) 41

[4.1.2. Libraries/Frameworks](#_6kn9mostgc32) 42

[4.1.2.1. React](#_yhxagtmypvjn) 42

[4.1.2.2. i18next](#_9ps5fionotzo) 42

[4.1.2.3 React query (tanstack query)](#_84oqy8rjz162) 43

[4.1.2.4. React-chartjs](#_rm9tgpnrpctg) 44

[4.1.2.5. Material UI](#_k1dbt7ull7mx) 44

[4.2. Back end](#_v9t10nmp8nsw) 44

[4.2.1. Programming language - C#](#_87qmlsriekzf) 44

[4.2.2. Libraries/Frameworks](#_8lfz86rm1lsx) 45

[4.2.2.1. ASPNET](#_ml71o0rfqtfi) 45

[4.2.2.2. EntityFrameworkCore (EFC)](#_395dp6wol6oi) 45

[4.2.2.3. Pomelo EntityFrameworkCore MySql](#_te4n4d5hra5k) 46

[4.2.2.4. Gmail API](#_t40243i4b2pl) 46

[4.3. Persistence layer - Database](#_94rd46n603fw) 46

[4.3.1. MySql](#_ezjpanvvdant) 47

[4.4. Codebase showcase](#_o49akx33j14a) 47

[4.4.1. Frontend](#_jc555oxosg3i) 47

[4.4.1.1. API request](#_ioizyn266gt6) 47

[4.4.1.2. ReactQuery Hooks](#_6awojxy86xlv) 48

[4.4.1.3 ReactQuery features](#_6gv7wpbytnga) 51

[4.4.2 Backend](#_4tiv4naq9dyz) 52

[4.4.2.1 Setting up Pomelo EFC](#_m9ril65kbsm8) 52

[4.4.2.2 Authentication](#_a00wd67ilcgk) 52

[4.4.2.3 Gmail API](#_ihv7zrdshkvo) 53

[4.4.2.4 Login endpoint](#_1pnql4apz6hc) 56

[**5. Testing and deployment**](#_f9ry5oolc21c) **58**

[5.1 Unit Tests](#_mgv94l1esw17) 58

[5.2 Deployment](#_79bb43v899eq) 58

[**6 Documentation for the users**](#_hrjnndhyxm0z) **59**

[6.1 Main pages before login](#_ehvtz3u07tic) 59

[6.1.1 Login page](#_3nzbmsh531k) 59

[6.1.2 Register page](#_1ji8nuvk9qi7) 59

[6.1.3 Reset password page](#_xpbinykd3z97) 59

[6.2 Pages after login](#_vvfkpn3ndehf) 59

[6.2.1 Page layout](#_iftdeseqxn6) 59

[6.2.2 Landing page - Dashboard](#_9qtvfkew0t0s) 60

[6.2.3 Task section](#_ed2qub75piwl) 60

[6.2.3.1. Tasks to do](#_jx8w24ek4eap) 60

[6.2.3.2. Task History](#_f2cw208iy5y5) 60

[6.2.4 Group section](#_8gholpndj8as) 60

[6.2.4.1. Groups](#_t7bouj740bvx) 61

[6.2.4.2. Group management](#_gd5qgi9mth2n) 61

[**Bibliography**](#_gg7idqg1uaeh) **62**

# List of Figures

2.1 Use case diagram

2.2 Entity diagram

3 Level 1 C4 model

4 Level 2 C4 model

5 Dashboard

6 Example of test for GetTasksByUserAsync()

7 Actors

8 Functionalities

**Abbreviations and Acronyms**

ECG Electrocardiogram

JS Javascript

TS Typescript

SPA Single Page Application

API Application programming interface

D3 Data-Driven Documents

SQL Structured Query Language

OAuth Open Authorization

C4 context, containers, components, and code

C# Csharp

URL Uniform Resource Locator

DOM Document Object Model

ASP.NET Active Server Pages Network Enabled Technologies

CRUD CREATE, READ, UPDATE and DELETE

ORM Object-relational mapping

UML Unified Modeling Language

REST representational state transfer

JSON JavaScript Object Notation

JWT JSON Web Token

IOC Inversion of Control

HTTP Hypertext Transfer Protocol

SMTP Simple Mail Transfer Protocol

# **1. Introduction**

Public health should be an important matter to every person in society and public health is highly correlated to the education of young doctors in our medical universities. It is such a shame that we do not use the tools we have available these days to take this education to the next level. The creation of more applications to help young people learn is very important. According to studies*“(...)mobile learning is highly motivating for students”*[17]. Such applications should be created with decent regard to intuitiveness and simplicity so they would be easy to use.

The electrocardiogram is an important diagnostic test, it is a recording of the heart's electrical activity from which a medical professional can assess the current condition of the patient, notice some past diseases, but also, with high probability, start diagnosing diseases or look for their signs. Creating an application that simulates an ECG test and thus gives future doctors the opportunity to be better prepared through more exercises was filling the gap. With the best knowledge on the day of writing this work, such an application available in polish did not exist.

The purpose of this document is to provide a thorough overview of the methods and tools used to create the application, as well as the architectural philosophies that guided its development. It will include descriptions of the various functionalities of the system and diagrams illustrating the various processes within the application. The goal is to give a comprehensive understanding of how the application operates and the reasoning behind its design.

One of the main objectives of this document is to detail the different technologies and frameworks that were utilized in the creation of the application. This will include programming languages, libraries, and any other tools that were integral to the development process. Additionally, the document will delve into the various architectural patterns and principles that were followed, such as modularity, separation of concerns, and scalability.

Overall, this document aims to provide a comprehensive understanding of the inner workings of the application and the thought process behind its design. By presenting a clear and detailed explanation of the methods and tools used, as well as the architectural philosophies that guided the development, it is hoped that this document will serve as a valuable resource for anyone looking to gain a deeper understanding of the application.

## **1.1. Aim**

This application's purpose is to offer a platform for ECG simulations. This program was created with the intention of helping teachers assess their students' grasp of electrocardiograms and giving students a practice and review tool for these crucial medical data.

By utilizing this program, professors will be able to evaluate their students' understanding of ECG interpretation, which is essential for identifying and treating a variety of cardiac diseases. On the other end, students will be able to practice detecting ECG patterns and recognizing the illnesses that might cause them. They will get a deeper understanding of this complicated subject matter as a result, and they will also be more equipped to handle the difficulties that await them in the world of healthcare.

This application's overall goal is to aid in the teaching and training of students in the field of electrocardiography so they can master the recognition and interpretation of electrocardiograms. This application intends to assist students in developing the information and skills necessary to thrive in their employment and have a positive influence on their patients' lives by offering a virtual environment for practicing and learning.

## **1.2. Scope**

As an educational platform, it is crucial that this application is designed in a way that allows users to easily access and utilize all of its functionalities. To achieve this, we have focused on creating a simple and intuitive interface for our users.

One of the key features of this application is the ability to visualize electrocardiograms, which are graphical representations of the electrical activity of the heart. These are an essential part of the application, as they provide the basis for learning and practice in interpreting these important medical records. To ensure that our visualizations are as accurate and realistic as possible, we have chosen to use actual electrocardiogram images as the basis for our visualizations. This means that the visualizations are based on real case scenarios, providing a highly accurate representation of what users can expect to encounter in their professional careers.

Overall, the scope of this application is to provide an educational platform for professors and students to learn about and practice interpreting electrocardiograms. By using real electrocardiogram images as the basis for our visualizations, we aim to provide a valuable and realistic tool for those seeking to improve their knowledge and skills in this field.

## **1.3. Overview of current solutions**

Unfortunately all polish similar applications are unavailable to the authors as they are not medical professionals. To the best of our knowledge, there does not exist an application like the one described in this paper available in the Polish language. There are some English language applications that meet the conditions of current solutions and they will be analysed in this subchapter.

### **1.3.1. ecg-quiz.com**

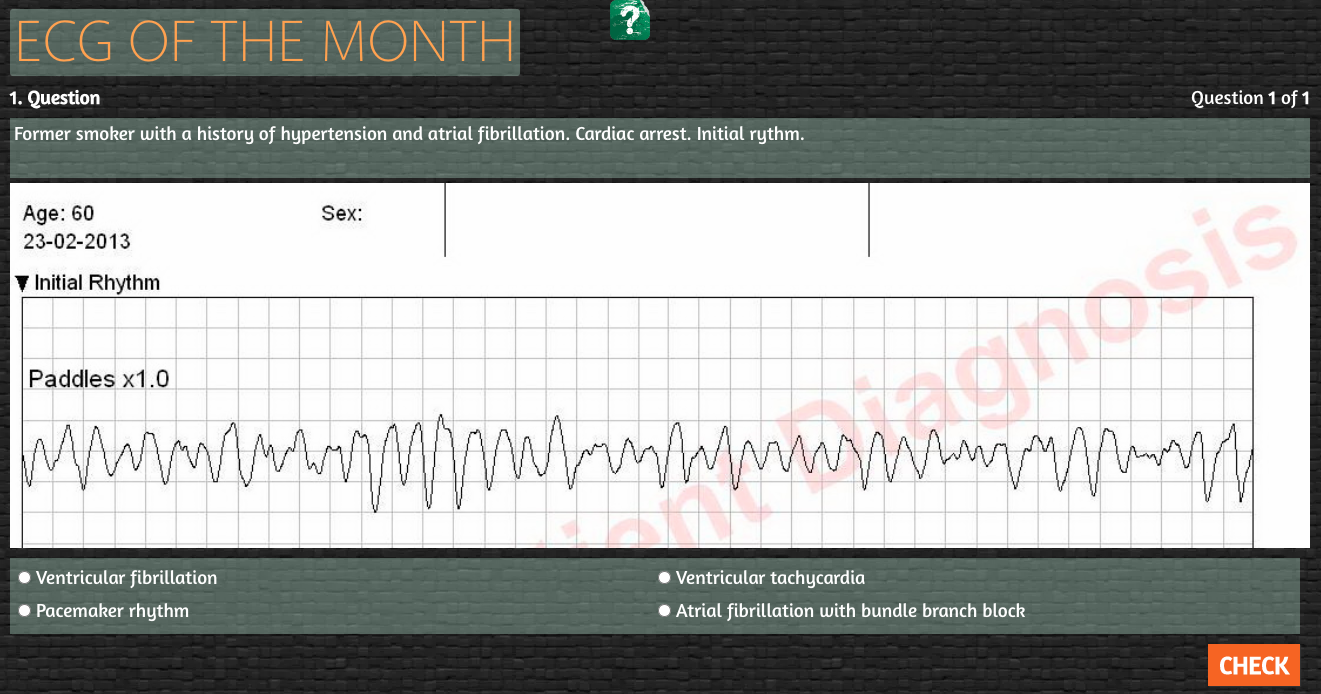


Figure 1.1

### 1.3.2. medschool.co

### 

Figure 1.2 Figure 1.3

# 

# 2. Requirements

## 2.1. Functional requirements

The functional requirements of the application are the specific tasks and functions that it is intended to perform. These requirements serve as the foundation for the design and development of the application and must be carefully considered in order to ensure that the application meets the needs of its users.

In order to clearly understand the functional requirements of the application, it is necessary to consider the various factors that may affect its performance. These factors may include the intended user base, the operating environment, and any external systems or services that the application may need to interact with. Additionally, it is important to consider the various scenarios that the application may encounter, as this will help to identify any potential challenges or issues that may arise during its operation.

Once the functional requirements of the application have been identified, the next step is to design and develop the application in a way that meets these requirements. This may involve choosing specific programming tools and frameworks, designing the backend and front end of the application, and creating an architecture for the database. It is also important to carefully test the application and ensure that it functions correctly in all scenarios and to carefully plan the deployment process in order to ensure a smooth transition to live operation.

Overall, the functional requirements of the application are a crucial factor in its design and development and must be carefully considered in order to ensure that the application meets the needs of its users and operates smoothly in all scenarios.

### 2.1.1. Description of the system functionalities

The application is designed to offer a range of functionalities to its users. These functionalities include basic features such as user authentication and authorization, as well as more specialized features that are specific to the intended use of the application.

#### 2.1.1.1. Actors

Professor - Professors have administrative access to students and can assign tasks

Student - Students have basic access to functionalities

Basic administrator - Pseudo admin has minimum administrative access to all users

#### 2.1.1.2. Functionalities

| ID | Functionality | Description |
| --- | --- | --- |
| 1 | Login | The system will check the given credentials during login and allow users to access the app given the correct info. |
| 2 | Register | The system sends a confirmation email when a new account is registered. |
| 3 | Logout | Users can logout |
| 4 | Create group | A new group will be created based on the doctor’s request when creating the group and a unique group code for that group will be generated upon creation. |
| 5 | Assign task | The assign task functionality will allow doctors to assign tasks to chosen groups. |
| 6 | Activate account | Users will be able to activate account which they have registered |
| 7 | Start a task | Users will be able to access the tasks assigned to them and start solving them. |
| 8 | Reset password | Users will be able to reset password if they lost their credentials |
| 9 | Remove group | Professors will be able to remove groups which they have created |
| 10 | Remove user from group | Professors will be able to remove users that are in the group they created |
| 11 | Remove task from group | Professors will be able to remove tasks they have assigned to their groups |
| 12 | Regenerate group code | Professors will be able to regenerate group code of their groups |
| 13 | Join group | Users will be able to join a group given corresponding group code |
| 14 | Update role | Pseudo admin has permission to update user’s role |

### 2.1.2. Use case diagram

# 

Figure 2.1

# 

### 2.1.3. Entity diagram

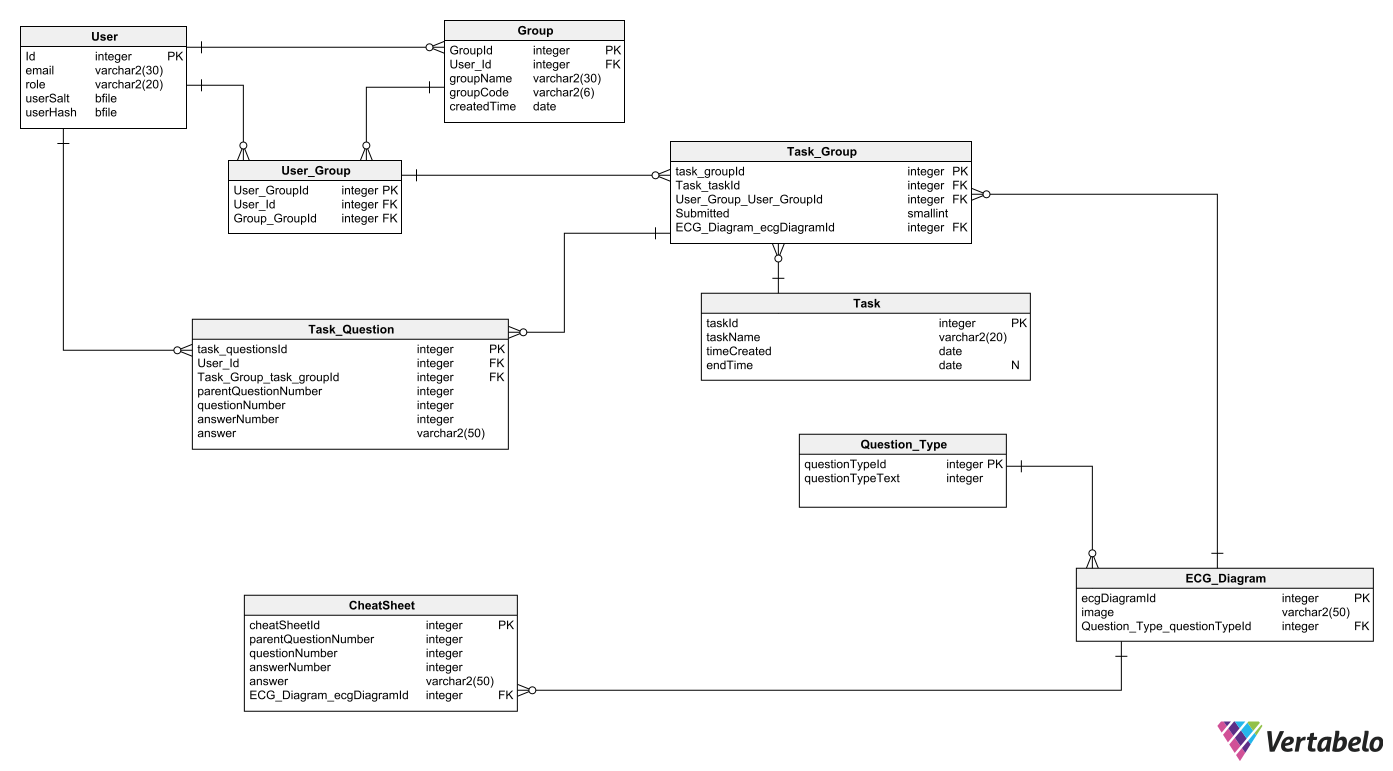


Figure 2.2

### 

### 2.1.4. Use case scenarios

**Use case 1 - Log in**

**Description:** A student or professor can be authorized into service and authenticated based on provided credentials

**Primary Actors:** Student, Professor

**Preconditions:** The user is a valid and verified person who has access to the system **Postconditions:** The user is directed to the landing page after login

**Main Success Scenario**

1. The user clicks the Login button

2. A credential form is displayed

3. The user enters credential

4. The system checks the validity of credentials

5. The landing page after login is displayed

**Extensions**

1. No valid credentials have been entered

(a) System informs the user of invalid credential

(b) Flow returns to step 2 of basic flow

**Frequency of use:** very frequent

**Priority:** high

#### 

**Use case 2 - Register**

**Description:** A user is allowed to register an account using an email belonging to the school’s domain.

**Primary Actors:** Abstract User

**Preconditions:** The user is a valid and verified person who has access to the system.

**Postconditions:** The user is granted login permission by the system.

**Main Success Scenario**

1. The user clicks the Register button

2. The app redirects the user to the registration page with form

3. The user enters their email and password for registration

4. The system sends a validation email to the user’s email address

5. The system grants login permission to the corresponding account once validation is completed

**Extensions**

1. The user enters an email not belonging to the school’s domain

(a) The system informs the user of the wrong domain

(b) Flow returns to Step 3 of main scenario

2. The user does not verify the email

**Frequency of use:** often

**Priority:** high

**Use case 3 - Log out**

**Description:** A student or professor can log out of the web application when they need to

**Primary Actors:** Professor, Student

**Preconditions:** The user is a valid and verified person who has access to the system and is already logged in.

**Postconditions:** The user is logged out of the web application.

**Main Success Scenario**

1. The user clicks the Log out button

2. The system logs the user’s account out

3. The user is taken to the main login page

**Extensions**

1. The user is already logged out

(a) The user is shown a successful logout interface

**Frequency of use:** often

**Priority:** high

**Use case 4 - Create a group**

**Description:** Allow professors to create groups for other users to join

**Primary Actors:** Professor

**Preconditions:** The user is a valid and verified person who has access to the system and is already logged in

**Postconditions:** A new group is created

**Main Success Scenario**

1. The user selects Create Group on the Group Management page

2. The system verifies the user’s permission

3. The user is shown a form to enter the name of group

4. The user confirms the creation of group

5. The system generates a unique 6-character long code for the group

**Extensions**

1. The user does not have permission to create a group

(a) System does not create a group

**Frequency of use:** often

**Priority:** medium

**Use case 5 - Assign Task**

**Description:** The assign task functionality will allow doctors to assign tasks to chosen groups.

**Primary Actors:** Professor

**Preconditions:** The user is a valid and verified person who has access to the system and is already logged in

**Postconditions:** New task is assigned to the corresponding group

**Main Success Scenario**

1. The user selects the desired topic and enters the task name to assign to the chosen group

2. The system verifies the user’s permission

3. The system creates a new task for group

4. The page informs the user of the successful completion

**Extension**

1. User does not have permission to assign a task

(a) System does not create new task

**Frequency of use:** often

**Priority:** high

**Use case 6 - Activate account**

**Description:** Users will be able to activate the account which they have registered

**Primary Actors:** Professor, Student

**Preconditions:** The user is a valid and verified person who has access to the system

**Postconditions:** The account is activated

**Main Success Scenario**

1. The user accesses the URL that holds the activation token provided in email

2. The system accepts the requests and checks if the activation token is valid

3. The system activates the account

4. The page informs the user of successful activation

**Extensions**

1. The activation token is invalid

(a) The system does not proceed with the request

2. The account is already activated

(a) The system does not proceed with the request

**Frequency of use:** often

**Priority:** high

**Use case 7 - Start a task**

**Description:** Users will be able to access the tasks assigned to them and start solving them

**Primary Actors:** Professor, Student

**Preconditions:** The user is a valid and verified person who has access to the system

**Postconditions:** The chosen task is displayed

**Main Success Scenario**

1. The user selects a task to start

2. The system takes the user to the task page

**Extensions**

1. The user is not verified

(a) The system does not proceed with the request

**Frequency of use:** often

**Priority:** high

**Use case 8 - Reset password**

**Description:** Users will be able to reset their password if they lost their credentials

**Primary Actors:** Professor, Student

**Preconditions:** The user is a valid and verified person who has access to the system

**Postconditions:** The password is reset

**Main Success Scenario**

1. The user enters the page to reset the password

2. The user enters the corresponding email address

3. The system checks if email exists

4. The system sends URL with password token to the given email address

5. The user accesses the given URL provided in email

6. The page shows a form for the user to enter desired new password

7. The user submits the reset request

8. The system checks if the request is valid

9. The system resets the password

10. The page informs the user of successful reset

**Extensions**

1. The email does not exist

(a) The system does not proceed with the request

2. The password token is not valid

(a) The system does not proceed with the request

**Frequency of use:** often

**Priority:** high

**Use case 9 - Remove a group**

**Description:** Professors are authorized to remove groups created by them

**Primary Actors:** Professor

**Preconditions:** The user is a valid and verified person who has access to the system and is already logged in

**Postconditions:** Group is removed

**Main Success Scenario**

1. The user selects the Remove button of the desired group

2. The system verifies the user’s permission

3. The system removes group

4. The page shows a success message to notify the user

**Extensions**

1. The user does not have permission to remove the group

(a) System does not remove group

**Frequency of use:** rarely

**Priority:** low

**Use case 10 - Remove user from group**

**Description:** Professors will be able to remove users that are in the group they created

**Primary Actors:** Professor

**Preconditions:** The user is a valid and verified person who has access to the system

**Postconditions:** The user is removed from the group

**Main Success Scenario**

1. The user clicks on the Remove button for the chosen user

2. The system checks if the user has permission

3. The system removes the selected user from group

4. The page updates the list

**Extensions**

1. The user does not have permission

(a) The system does not proceed with request

**Frequency of use:** rarely

**Priority:** low

**Use case 11 - Remove task from group**

**Description:** Professors will be able to remove tasks they have assigned to their groups

**Primary Actors:** Professor

**Preconditions:** The user is a valid and verified person who has access to the system **Postconditions:** The chosen task is removed from the group

**Main Success Scenario**

1. The user clicks on the Remove button on the desired task
2. The system checks if the user has permission
3. The system removes the task from group
4. The page updates the list

**Extensions**

1. The user does not have permission to remove task
2. The system does not carry out the request

**Frequency of use:** rarely

**Priority:** medium

**Use case 12 - Regenerate group code**

**Title:** Regenerate group code

**Description:** Professors will be able to regenerate group code of their groups

**Primary Actors:** Professor

**Preconditions:** The user is a valid and verified person who has access to the system **Postconditions:** The group code for the group is regenerated

**Main Success Scenario**

1. The user clicks on the Regenerate code button
2. The system checks if the user has permission
3. The system regenerates group code
4. The page updates the group code

**Extensions**

1. The user does not have permission
2. The system does not proceed with the request

**Frequency of use:** rarely

**Priority:** medium

**Use case 13 - Join group**

**Title:** Join a group

**Description:** Users will be able to join a group given corresponding group code **Primary Actors:** Professor, Student

**Preconditions:** The user is a valid and verified person who has access to the system **Postconditions:** The user joins the group

**Main Success Scenario**

1. The user enters the group code and submits request
2. The system checks if the request is valid
3. The system adds the user to the group with the corresponding group code
4. The page updates the list

**Extensions**

1. The request is not valid
2. The system does not proceed with request
3. The user is already in the group
4. The system does not proceed with request

**Frequency of use:** rarely

**Priority:** medium

**Use case 14 - Update role**

**Title:** Start task

**Description:** Pseudo admin will be able to update users' role

**Primary Actors:** Pseudo admin

**Preconditions:** The user is a valid and verified person who has access to the system

**Postconditions:** The user chosen by the pseudo admin will have role updated

**Main Success Scenario**

1. The user enters the desired email address and role
2. The user submits the request
3. The system checks if the user has permission
4. The system updates role
5. The system notifies the user of a successful update

**Extensions**

1. The request is not valid
2. The system does not carry out the request
3. The email address does not exist
4. The system does not carry out the request

**Frequency of use:** often

**Priority:** high

## 2.2. Non-functional requirements

### 2.2.1. Usability

It is crucial that the medical application is designed with usability in mind, as it will be used by a variety of non-technical users. This means that all features and functionality should be easily accessible and intuitive to use.

To achieve this, the app should have a coherent and logical layout, with clear and concise instructions for each feature. This will help medical students quickly and easily navigate to the resources and information they need, without being bogged down by the complexities of the technology.

Additionally, the app should be user-friendly and straightforward to use, so that medical students can focus on their studies rather than struggling with the app itself. By prioritizing usability, the app can be a valuable tool for medical students, helping them to efficiently access the resources and information they need to succeed in their studies.

### 2.2.2. Security

Ensuring the security of an application is a complex task that requires constant vigilance and attention. Despite our best efforts, it is possible that there may be overlooked issues or vulnerabilities that could compromise the security of the system. In the case of an internal application intended for use by a university, it is important to take extra precautions to ensure the safety of sensitive information and data.

One key factor to consider is that the application will be standalone, meaning that it will not be connected to any external systems or networks. This can significantly reduce the potential for a secondary breach, as there will be no link to other servers or databases. However, it is still important to take steps to secure the application itself, such as implementing strong passwords, enabling two-factor authentication, and regularly updating the system to fix any identified vulnerabilities.

In addition to technical measures, it is also important to consider the role of user education in maintaining the security of the application. By providing training and resources to users on how to protect their passwords and use the system responsibly, it is possible to significantly reduce the risk of a security breach.

The usage of OAuth for authentication and permission is another step made to improve the security of the application. Users can securely authorize access to their data and resources, such as user profiles and server resources, using the open standard known as OAuth, without disclosing their login information. Since it enables users to log in using their university credentials and lowers the likelihood of password-related security breaches, this can be very helpful for an internal application. Additionally, because users don't need to remember numerous sets of login credentials, OAuth can offer a more smooth and more user-friendly experience.

# 

# 3. Application architecture

## 3.1. Description of the prototypes

The prototypes that are being developed are expected to possess a variety of features and capabilities. These include the ability to perform various functions, such as those that have been previously mentioned. It is important that the prototypes are able to fulfil all of the requirements and specifications that have been outlined in order to be deemed successful and effective. Therefore, it is essential that they are thoroughly tested and evaluated to ensure that they possess the necessary functionality to meet the intended goals and objectives.

## 3.2. General description of the architecture

The application will be composed of React Typescript for the front end, C# for the backend and MySql for the database.

## 3.3. C4 Model

*“When you ask someone in the construction business to describe a building’s design visually, they will likely show you site plans, floor plans, elevation views, cross-section views, and detailed drawings. Comparatively, if you ask a software developer to represent the software architecture of a software system using diagrams, you’ll probably end up with a muddled mess of boxes and lines... inconsistent notation (colour coding, shapes, line styles, etc.), ambiguous naming, unlabeled relationships, generic terminology, missing technology choices, mixed abstractions, etc.”* [15]

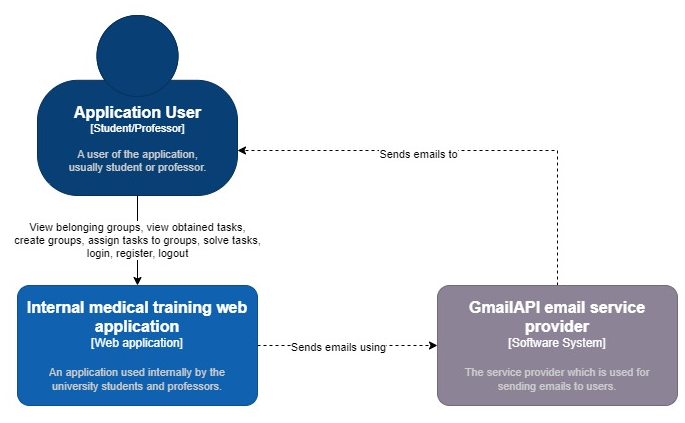
The statement above has never been more true. From the perspective of a software developer, it is often very hard to communicate visually about how a certain software works, sometimes the visual representation given is too in-depth or too vague, making things more complicated than it needs to be.

While Unified Modeling Language (UML), ArchiMate and SysML do exist as a form of visual communication between software developers and recipients, it is still lacklustre in delivering the idea as it is simply too complex. An alternative to the previously mentioned ideas is the C4 model. [15]

The C4 model provides an ”abstraction-first” approach to diagramming software architecture, based on how software architects and developers think about and create software. The C4 model emphasises simplicity for it to be easy to pick up. Furthermore, the C4 model is divided into 4 levels, not all of them must be used, only those that are important to deliver the idea. [15]

In the following section, there are two levels of the C4 model presented, level 1 and level 2, both of which will provide a visual concept of the architecture of the application, as well as an explanation of the diagrams in question.

### 3.3.1. Level 1



In the level 1 C4 model described, the system consists of three main components: the application users, the web application, and the email service provider. The application users are students and professors who interact with the web application to perform various tasks. These tasks include viewing the groups they belong to, viewing the tasks they are assigned, joining groups, registering, changing their password, logging in and out, and more.

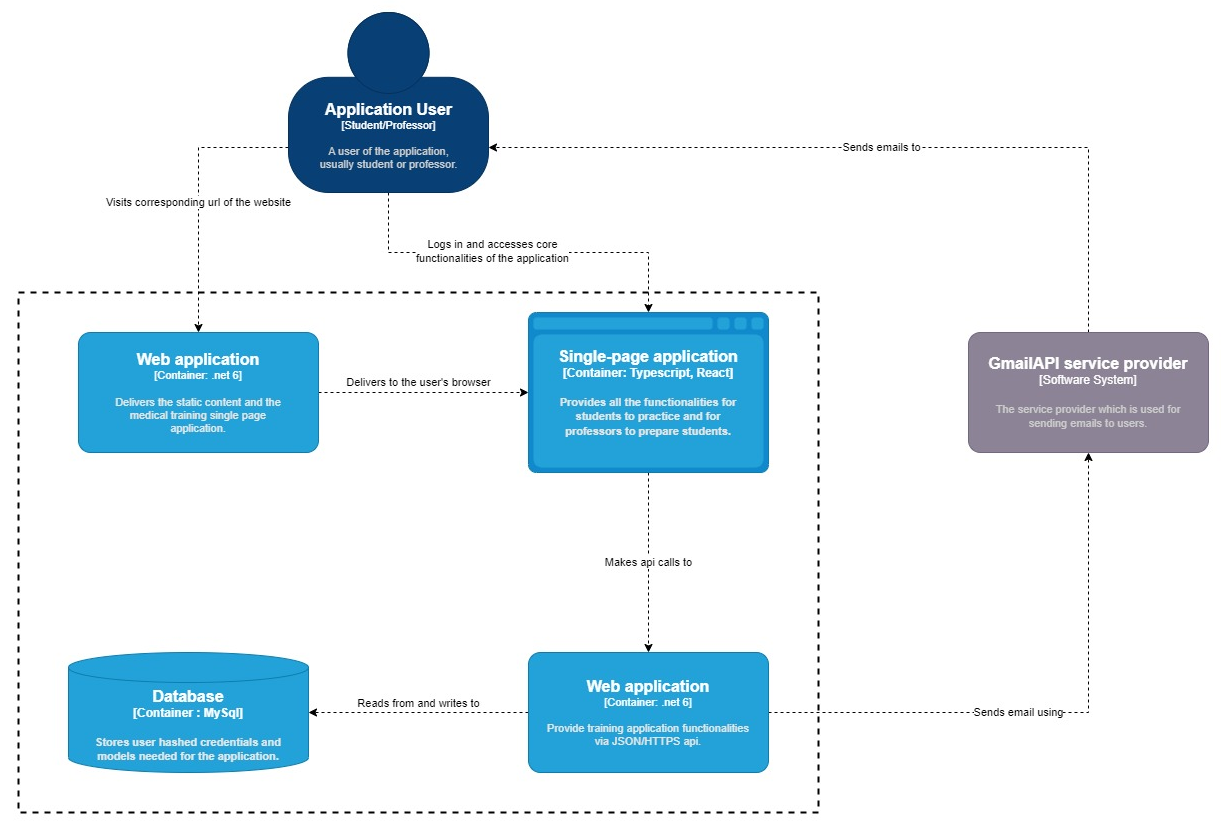
The web application is responsible for managing the interactions between the application users and the email service provider. It provides the interface through which the application users can perform these tasks and communicate with the email service provider. It also stores and retrieves data about the application users, such as their login credentials and group membership information.

The email service provider is a separate component that is used for sending emails to the application users. In this system, it is used specifically for registering an account. When a new application user registers, the web application sends a request to the email service provider to send an email to the user with a link to confirm their registration.

The level 1 C4 model for this system describes the high-level relationships and responsibilities of the different components. The application users interact with the web application to perform various tasks, and the web application communicates with the email service provider to send emails when necessary. Each component plays a specific role in the system and works together to provide the desired functionality.

There are a few potential improvements that could be made to this system based on the level 1 C4 model description. One potential improvement would be to add more functionality to the email service provider, such as the ability to send emails for tasks or group updates. This would allow the professors to easily communicate with their students and make it easier for students to stay informed about their tasks and group activities.

### 3.3.2. Level 2



The second level of the C4 model provides a more detailed view of the system architecture, focusing on the relationships and responsibilities of the various components within the system. The system consists of four main components: the application users, the web application, the backend, and the Gmail API service provider.

The application users are students and professors who interact with the system through the web application. When they visit the URL of the web application, it delivers the necessary content and a single-page application (SPA). The SPA is a client-side application that runs in the user's web browser and provides the interface through which the application users can access the core functionality of the system.

The web application is responsible for serving the SPA to the application users and handling their interactions with it. When the application users perform an action within the SPA, such as logging in or accessing a group, the SPA makes an API call to the backend to request the necessary data or to perform the desired action.

The backend is a separate component that handles most of the processes within the system. It is responsible for reading and writing data to and from the database, as well as processing the API requests made by the SPA. It is also responsible for communicating with the Gmail API service provider, which is used to send emails to the application users.

The Gmail API service provider is a third-party component that is integrated into the system to provide email functionality. It is used specifically to send emails to the application users, such as when they register for an account.

Overall, the second level of the C4 model for this system describes the relationships and responsibilities of the various components within the system. The application users interact with the web application and SPA to access the core functionality of the system, and the web application communicates with the backend to retrieve data and perform actions. The backend, in turn, communicates with the Gmail API service provider to send emails when necessary.

# 

# 4. Software

## 4.1. Front end

The front end is undeniably a very important part of the application. Often underestimated but it must be taken into account that user experience is responsible for much of the success or lack of it. Choosing the right tools for the right project is not so much difficult as full of uncertainty. Such potentially unimportant things as colours or arrangement of buttons have a great impact on how willingly the user will return to such an application.

### 4.1.1. Programming language - Typescript

One of the reasons why Typescript was chosen over Javascript is that Typescript is strongly typed, whereas Javascript is not, this difference makes it easier to know which part of the code is causing the problem. To put it simply, Typescript is much safer, it helps you catch and fix errors before you run your code, making it easier to spot and fix bugs. Also, it makes it easier to refactor your code, as the type system helps you understand how different parts of your code are related and how they are used. As Typescript is, in layman’s terms, a typed Javascript, it has a large and active community of users and developers, providing a wealth of resources and support for those working with the language. It has excellent documentation and a large number of resources available, making it easy to learn and use. In addition, it has strong support for object-oriented programming concepts, such as classes, interfaces, and inheritance, making it a good choice for projects that need to use these features.

### 4.1.2. Libraries/Frameworks

Obviously the aforementioned Typescript would not be sufficient in such a project. For this reason, various libraries have been used in the application to provide a high-quality front-end experience.

#### 4.1.2.1. React

1. Rather than going for a mobile or desktop app, the need for the application to be web-based is rather important for this project as the intended users will most likely find it more convenient to access the app via a web browser. Therefore, in this project, we decided to use the React library.
2. React was our choice for this project because using vanilla Javascript would be downright nonsensical. In comparison to Angular and Vue, React is rather easier to pick up, which is quite important as this allows the team to learn the library with relative ease and start creating complex web applications.
3. React has a large and active community of users and developers, providing a wealth of resources and support for those working with the library.
4. React is highly efficient and fast, with a virtual DOM that allows for fast rendering and updates to the user interface, making the application very reactive.
5. React has strong support for reusable components, making it easy to build and maintain large and complex applications.

#### 4.1.2.2. i18next

1. The university, for which this app is designed, has both Polish-speaking students and English-speaking students, so there is a need to localize the app. The i18next library provides tools which make localization simpler than it needs to be.
2. i18next has a large and active community of users and developers, providing lots of resources and support for those working with the library.
3. i18next is highly configurable, allowing you to customize how it handles translations and language detection, meaning users using English as their default language on browsers will automatically see English translations.
4. i18next is lightweight and efficient, with a small footprint that won't add unnecessary bloat to the project.
5. i18next supports a wide range of platforms and frameworks, including web, mobile, and server-side applications. For our application, it was simpler to store the translations on the front end and i18next does provide such a feature.

#### 4.1.2.3 React query (tanstack query)

1. React query takes away much of the boilerplate code that is required when writing fetch requests for API. There are many functionalities, which when used with QueryClient, make updating and removing data on the page as simple as calling a single function.
2. React Query is a lightweight and efficient library for fetching, caching, and updating data in your React applications.
3. React Query has a simple and intuitive API that makes it easy to use and integrate into our project, letting us focus on sending requests to the backend as opposed to building our own API.
4. React Query has built-in support for handling common data fetching tasks, such as pagination, filtering, and sorting, making it a good choice for projects that need to work with large or complex data sets.
5. React Query has a flexible and configurable caching system that allows you to control how data is fetched and stored, making it a good choice for projects that need to optimize performance or manage data efficiently.
6. React Query has a large and active community of users and developers, providing a wealth of resources and support for those working with the library.
7. React Query is lightweight and efficient, with a small footprint that won't add unnecessary bloat to the project.
8. React Query has a built-in dev tool which helps us ensure that the requests are in fact being called and the data that is received is indeed the correct data that is being called for.

#### 4.1.2.4. React-chartjs

1. React Chart.js is a relatively customizable library that allows developers to create charts that fit the specific needs and design of their applications. This means it helps us to focus on passing the data into the React component rather than building everything from scratch.
2. It is easy to use and integrate into a React project, with a simple API and numerous pre-built components. In reference to the previous point, Chart.js is not as customizable as say D3js but that is why it is chosen because in D3js you have to define charts yourself from the ground up.
3. It has a wide range of chart types available, including bar charts, line charts, pie charts, and more. Given the purpose of our project, it was important to have multiple options to see which fits best for the given component we want to use Chart.js in.

#### 4.1.2.5. Material UI

1. //something

## 4.2. Back end

### 4.2.1. Programming language - C#

1. The greatest reason for the choice of C# over other languages is the fact that every member has had experience creating full-stack applications with C# before.
2. C# has a big community, meaning there are lots of resources to refer to.
3. On top of that, C# is a statically-typed language, which means that it catches errors at compile-time rather than runtime. This can lead to more stable and reliable code. [4]
4. C# also has strong support for test-driven development, which can help ensure that our code is not full of bugs and easy to maintain.

### 4.2.2. Libraries/Frameworks

#### 4.2.2.1. ASPNET

1. ASP.NET is the framework chosen to be used for this application. There are many advantages to ASP.NET, it is cross-platform, and high-performance but it is first and foremost an open-source framework.
2. ASP.NET is a competitor with many other frameworks such as spring, Django, PHP laravel and nodejs express to name a few. While other frameworks do the job, the majority of the team member has experience with C#.
3. Strong support and a large community, ASP.NET is a mature, widely-used web development framework with a strong developer community and good documentation and resources which is extremely useful for creating an app.
4. ASP.NET has many built-in features and tools that help increase productivity and speed up the development process.
5. ASP.NET has a number of built-in security features, such as validation and encryption, to help protect web applications and keep data safe.

#### 4.2.2.2. EntityFrameworkCore (EFC)

1. Entity Framework Core is a lightweight, extendable, open-source, cross-platform version of the well-maintained Entity Framework data access technology. [5]
2. In comparison to Dapper, a micro ORM alternative to EFC, EFC offers an abstraction from SQL, allowing developers to focus on writing code in C#.
3. Furthermore, developing with Dapper requires more time because the reason for choosing Dapper is typically to write your own SQL as opposed to EFC where SQL is optimized under the hood.

#### 4.2.2.3. Pomelo EntityFrameworkCore MySql

1. The reason for the use of Pomelo EFC is simply due to the fact that there seems to be no alternative as of the time of choosing it. While there are other similar alternatives, for example, Npgsql, it is used with a different database and that is a topic to be discussed later.
2. Pomelo EFC does not replace EFC completely, it only extends it as a provider for databases that are compatible with MySql, having this Object-relational mapping (ORM) helps us to connect object code to a MySql database.

#### 4.2.2.4. Gmail API

1. Gmail API has built-in support for OAuth 2.0 authentication, which makes it easier to securely authenticate users and access their data. [6]
2. Gmail API is free to use for the first 15 GB of data transferred per month, which should be sufficient for most small to medium-sized applications.
3. Gmail API was chosen over SendGrid mainly because there was a reason to choose it. Given the scale of the application and how the only usage of email is for registration and resetting passwords, the final decision was Gmail API.

## 4.3. Persistence layer - Database

Without a doubt, such an application would not have the right to work without a well-designed and well-implemented database. In this case, all the information needed to create a quiz is stored in a specially designed database and without it, creating a good and efficient application would be practically impossible.

### 4.3.1. MySql

MySql is a good database choice as the purpose of our application is to be used internally for a university so it was in the university’s and our best interest to keep things simple and as cheap as possible. Therefore, an open-source database was the one chosen as it is a cost-effective choice. It has a long history and has been widely adopted by many organizations, making it a tried and tested choice for the project, among other benefits. It also supports a wide range of data types and data storage options, including support for large volumes of data and support for different storage engines. In this case, images containing ECG diagrams have to be stored, which MySQL does support.

## 4.4. Codebase showcase

### 4.4.1. Frontend

#### 4.4.1.1. API request

The most important part of the front end is arguably the API requests. Without Application programming interface (API) requests there is no way to get data from the backend, rendering the front end essentially pointless. Therefore, it is important that these functions that handle API requests work properly.

The project is a CREATE, READ, UP-DATE and DELETE (CRUD) application that is built with the representational state transfer (RESTful) API architectural style in mind. Oftentimes, the endpoints will need to have a payload that is relatively complex, therefore it is usually delivered in JSON format. On top of that, because OAuth2 is used in this project as a security measure, every request that requires validation must send a token [10], in this case, JSON web token (JWT). [13]

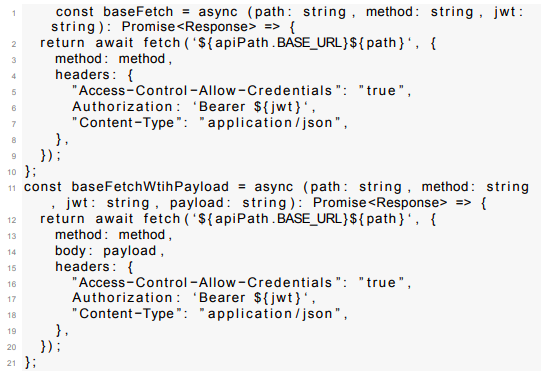


Figure 4.1

The two functions baseFetch and baseFetchWithPayload above are core components of the frontend for API requests. Most API requests that require data from the database or make changes to the database will use one of these two functions by giving the required parameters, along with the JWT inside the Aurtorization header.

#### 4.4.1.2. ReactQuery Hooks

One of the more important parts of the frontend is arguably the custom hooks that take care of API requests. While they do not look like the most complex code out of the entire codebase, the functionalities they provide are important and the code under the hood of ReactQuery is extremely sophisticated. Let’s have a look at the landing page - Dashboard.

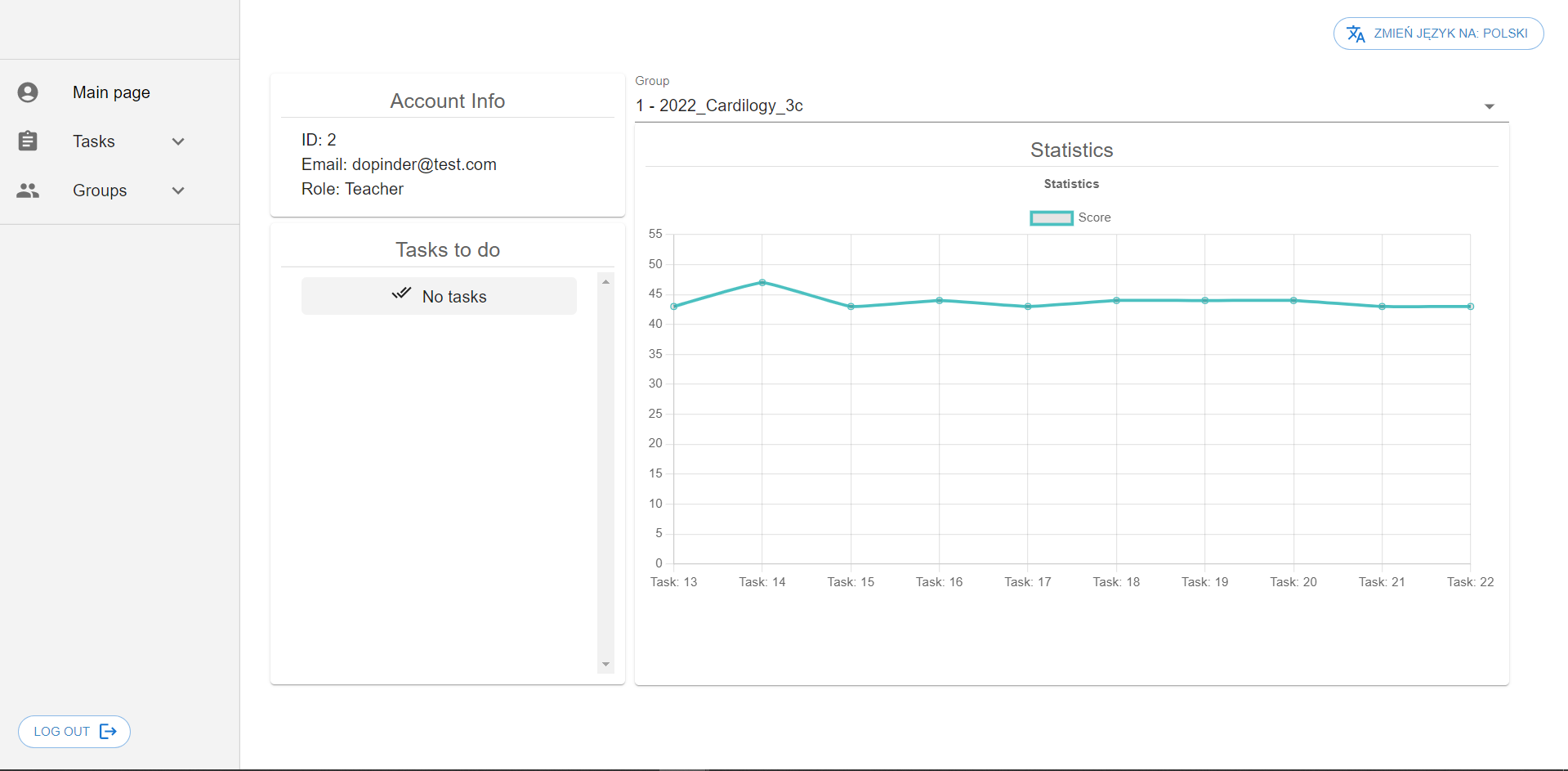


Figure 4.2

The dashboard is divided into essentially three main parts, UserDetailCard, TodoTasksCard and UserStatsCard. UserDetailCard shows the information about the user’s account (ID, Email, Role). TodoTasksCard shows tasks that haven’t been done yet. And most importantly the UserStatsCard provides a line chart that shows the score of every task for a given group.

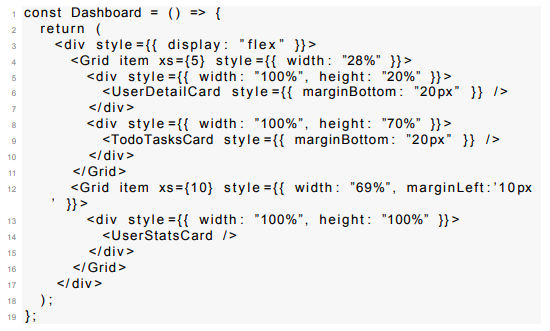


Figure 4.3

Each component inside Dashboard makes an API request to retrieve the necessary data and they do it with the help of custom hooks that

use React Query.

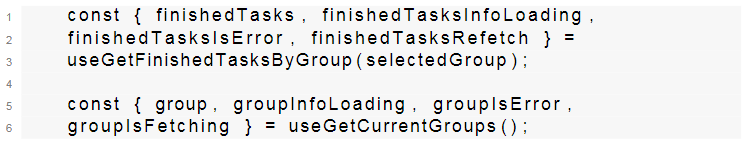
UserDetailCard uses the following hook:



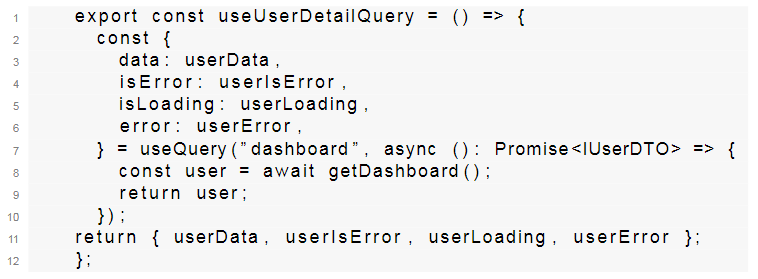
TodoTasksCard uses the following hook:



UserStatsCard uses the following hooks:

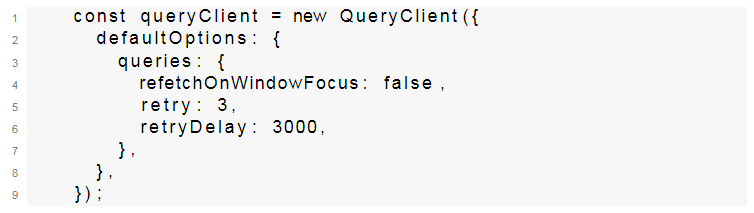


Having a look at an example, say useUserDetailQuery, if we go deeper and look at what this hook does, you will notice that it is calling another function with useQuery. The function within useQuery utilizes the previously mentioned baseFetch to fetch the details about the account that called the corresponding endpoint.



#### 4.4.1.3 ReactQuery features

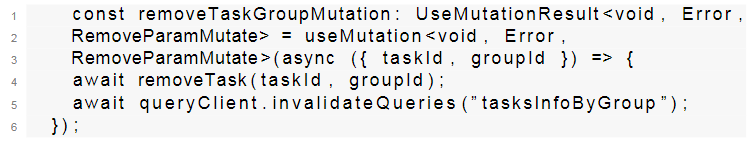
One of the most important ReactQuery components is the QueryClient, it provides many methods that can be used to fetch and cache a query, all of which provide a different purpose and method of querying.



One of the methods from QueryClient is refetchQueries, it accepts a string argument queryKey which is used to select the correct query for making the API request. To be more specific, useUserDetailQuery has queryKey ”dashboard”, meaning if there is a need to refresh the data, you need only to call the refetchQueries function and pass ”dashboard” as its argument.

You may have noticed that in the useUserDetailQuery snippet, useQuery was called within the hook. The useQuery hook is the most basic feature of ReactQuery and it is primarily used for reading data. From the useQuery hook you may receive the data itself, boolean value of whether it is loading, whether there was an error and more.

However, let’s say that we would like to remove data, say from a list, then we cannot use useQuery. The reason for that is due to the fact that useQuery is first and foremost used for read-only, not writing. In order to modify data we must use another helpful hook called useMutation.

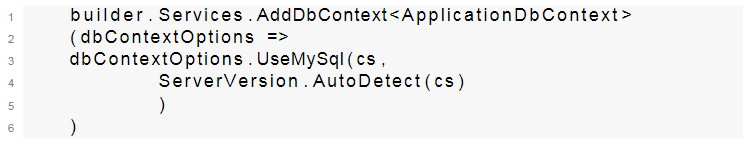


In this code snippet, the useMutation hook makes an API request removeTask to remove a task that was assigned to the corresponding group, then it invalidates the query with a queryKey ”tasksInfoByGroup”; this queryKey comes from another hook that was used to first fetch a list of tasks.

### 4.4.2 Backend

#### 4.4.2.1 Setting up Pomelo EFC

One of the most crucial parts of the project is to set up the database driver in order to use the ORM. Thankfully, this can be done very easily by using AddDbContext after installing the Pomelo EFC package.



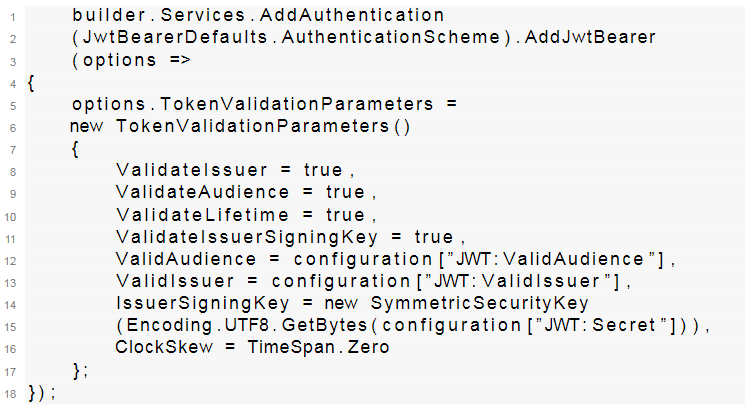
Once we have set up the database driver we may start using it. We can then inject dependencies, in this case the Service classes which we will use for our Controllers when we receive API request on an endpoint.



By injecting dependencies, it makes the component more modular and easier to test, since they are passed in from the outside rather than being hard-coded and makes the code overall easier to maintain. This software design patterns allows us to achieve the Inversion of Control (IOC) between classes and their dependencies. [1]

#### 4.4.2.2 Authentication

In the following section we are setting up token validation for the application. This enables us to authenticate users that are sending requests to endpoints. In ASP.NET, authentication works in such a way that service IAuthenticationService is used by authentication middleware. The authentication service processes any authentication related actions and thereby giving us security OAuth measure over the internet.



After setting up authentication for our application, every request that is trying to access restricted resources must come from authorized users. By providing claims, the application is now able to grant authorization based on whether it is valid or not.

#### 4.4.2.3 Gmail API

In order to use this application one must first create an account by going through the registration process, unless of course they have an account already. To achieve this, we need to use email activation. Certainly, we need not go through the email activation process, however, the purpose of it is to verify the user and to ensure that this email is real. The difference between an active and a non-existent email is, naturally, the fact that non-existent emails cannot activate accounts. As for inactive emails, we do not need to worry because they may or may not be activated.

From the perspective of the application we only need to make sure those accounts that have had a link with activation token sent to their email to be properly activated after visiting the URL. In order to achieve this, we must first set up an email service. In this application, Gmail API was the service of choice.

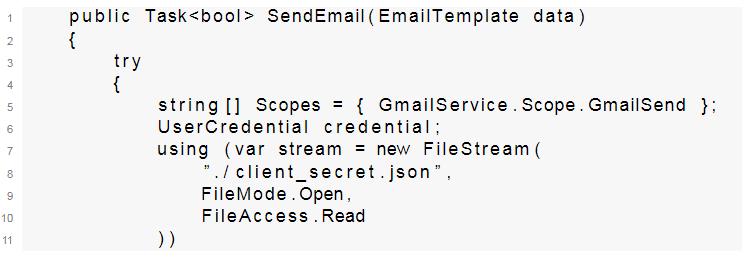
To make Gmail API work we need to complete the following steps as provided on Google’s guide [3]:

1. The application must be registered in the Google API console.
2. Users must grant explicit access permission for the app when the app launches for the first time.
3. Assuming the user agrees, the application will request for credentials to access the Gmail API.
4. Refresh the credentials (if necessary).

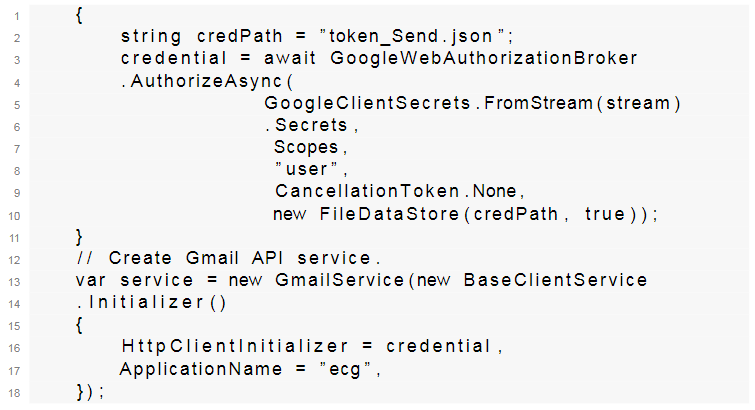
Naturally, we need a Gmail account to register our application in the Google API console. Once we have registered our application, we need to create an OAuth client ID because Gmail API does not use Simple Mail Transfer Protocol (SMTP). Gmail API is a RESTful API that can be used with any programming language that can make HTTP requests and parse JSON responses, it achieves this by using OAuth 2.0 protocol for authorization and authentication.

Once we have created our OAuth client ID, we can download it as a JSON file and use it in our application. In the following code snippet, we have a function that is used for sending emails with EmailTemplate being the source of destination address, subject and body of our email. Notice we are reading from the file ”client\_secret.json”, this is the OAuth client ID we have downloaded from the Google API console, inside it we can find all the relevant credentials.

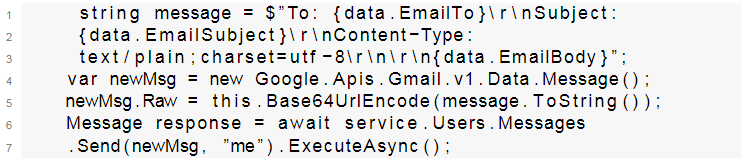
Notice that we have a string array called Scopes. Gmail API allows developers to specify Auth Scopes, the reason for this is to grant as little permission as possible to a user at a given time because some scopes may be used for restricted resources. [6] In this application, we only need to use the ”https://www.googleapis.com/auth/gmail.send” scope, which is GmailSend in the code below.



Next, we prepare the credentials we need to allow our app to verify with a Google account. With our credentials prepared, we can not create our Gmail API service. Notice that we are creating a folder called ”token\_Send.json”, this folder will save, among other important data, the access token and refresh token for the application to use every time it accesses Gmail API.



Lastly, we simply create the message we want to send with the proper headers and send our account activation token or perhaps reset password token depending on which endpoint the user was calling.



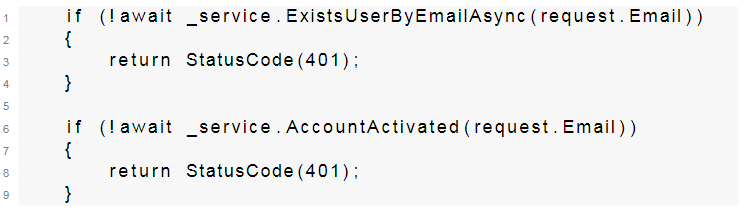
#### 4.4.2.4 Login endpoint

Up to this point we have been looking at how things are set up for this project, let us look at the most basic and perhaps most crucial endpoint for this project, the Login endpoint.

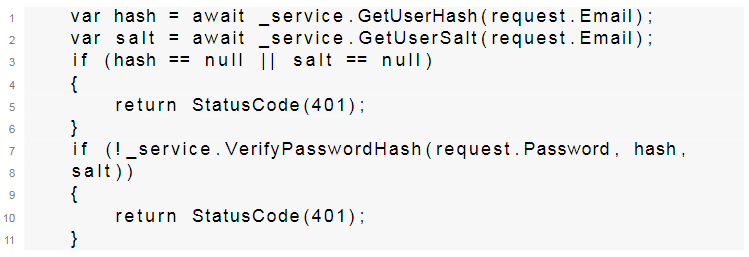
In order for our users to login to use our application they must send a request to the URL ”/Auth/Login” with the payload ”UserLoginDTO” which consists of the user’s email and password.



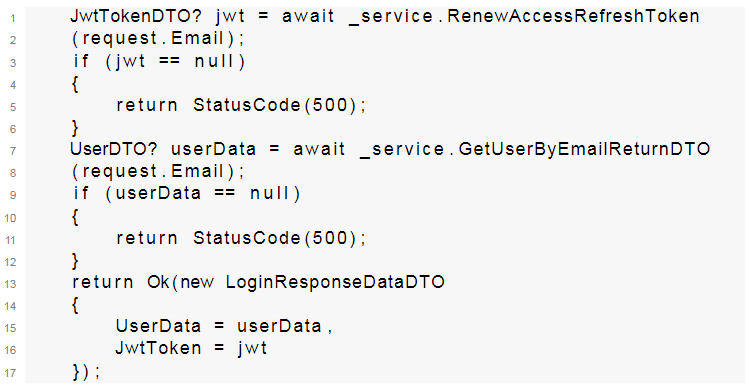
We first and foremost check for whether there exists a user with such an email because if there is not such email, this most likely means the user has not registered yet. Then we check whether the account is activated, activating account is important because theoretically anybody is able to access our website, and we do not want that. The purpose of only allowing accounts that have been activated is so that only users with an email from the medical university’s domain may login.



Next, we check whether the password submitted by the user matches the given email. In order to do this, we will need the submitted password, the hash and the salt in order to compute the correct value. If the value computed checks out, this means our user gave the correct password. Note that passwords are not stored in plain texts whatsoever because doing so is a risk of security, the preferred way is to compute the hash and compare them.



Finally, once all the previous checks have passed, we need only to provide the user its data in order persist information on the client side, and JWT so that every consequent API requests made to access restricted resources will bear it in the Hypertext Transfer Protocol (HTTP) header, thereby allowing us to authenticate and authorize the user.



# 5. Testing and deployment

## 5.1 Unit Tests

## 5.2 Deployment

For deployment, we used Netlify for the frontend, Azure App Service for the backend and Azure Database for MySQL.

In order to make use of Netlify, all we need to do is link the github repository to the website, make sure the build command used during deployment is the correct one and the specified distribution folder is correct. The rest is to make sure all the variables used for local development are changed to accommodate the production version, for example changing the URL to the proper backend URL.

Regarding deployment on Azure, more things had to be done. First and foremost, we need to set up a database where we may persist all important data. When we have a database ready, we may begin to set up the backend.

In Azure, you may choose different tiers for your application. Since our application is 64 bits, we need to choose at minimum the Basic B1 tier plan as it offers a 64 bits platform. All we need to do next is to make sure our application uses the connection string for the database we have just set up previously, proper URLs in the code configuration settings itself and we should be able to access it from the Netlify website after publishing the code onto Azure.

# 6 Documentation for the users

## 6.1 Main pages before login

### 6.1.1 Login page

Login page will be the main page of the application. From here, you are able to login, go to the Registration page or Reset password page.

### 6.1.2 Register page

On the registration page, users are able to register a new account. Once you have sent a request to the server for registering an account, you will receive an email with an activation link. By opening the activation link you received, you will be directed to the activation page. Here, the system will automatically activate your account and once it is activated you will be redirected back to the login page where you may login.

## 6.1.3 Reset password page

The reset password page is where you can reset your password in the event that you forget your password. Similar to the registration page, once you have filled in the form you will receive an email with a link that takes you to a page where you may enter your new password.

## 6.2 Pages after login

### 6.2.1 Page layout

After login, you will notice a sidebar where you can navigate through all the pages. In the bottom left you will find the logout button which logs you out if you click on it. Lastly, in the top right, you may find a button for switching the translation of the page to either Polish or English.

### 6.2.2 Landing page - Dashboard

The landing page, meaning the page you enter after login, will be the dashboard. On this page, you will be able to see your account detail, tasks to be done and statistics of your tasks within the group you have selected.

### 6.2.3 Task section

Within the task section, found on the sidebar, you will notice two subsections, namely "Tasks to do" and "Task History".

#### 6.2.3.1. Tasks to do

Within Tasks to do, you will be able to see all the tasks that are waiting for you to complete, if there is any. Clicking on "Start task" will bring you to the next page where you will begin solving a task. While solving a task, you will be able to view or hide ecg diagrams and once you have filled in your answer or even if you did not, you may submit your task. After submitting, you will enter the summary page in which you will receive details about a certain illness.

#### 6.2.3.2. Task History

As the name suggests, inside task history, you will be able to choose a task to view your choices. Inside a chosen task, you will see all the questions and answers. There are three types of color coding. Green - the answer and your choice matched. Yellow - the answer should have been marked but was not. Red - the choice was incorrect and was not supposed to be marked.

### 6.2.4 Group section

Within the group section, you will find two subsections, "Groups" and "Group management". If you are a student you will see only "Groups".

#### 6.2.4.1. Groups

Inside Groups, you will see all the groups you belong in. You may also enter a group by providing a group code below. Once you submit a group code, you will be automatically added to this group, assuming the code you gave was correct.

#### 6.2.4.2. Group management

Inside the Group management section, teachers will be able to see only the groups which they have created. You may remove a group by clicking the remove button or add a new group by providing a name for your group and clicking the create group button below.

If you instead click on the details button of a group, you will be directed to the group's detail page. Here you will be able to see every user that joined this group and group code. You can remove a user from the group or if you instead click on details you can view this user's details and statistics, similar to the dashboard page.

You will also notice the "Check assigned tasks" button in the group details page. Clicking on the checked assigned tasks button will take you to another page where you can see all the tasks you have assigned. Here you will be able to add new tasks or remove one.

# Bibliography

[1] .NET documentation | Microsoft Learn. https://learn.microsoft.com/en-us/dotnet/. Accessed: 2022-07-12.

[2] ASP.NET documentation | Microsoft Learn. https : / / learn .microsoft . com / en - us / aspnet / core / ?view = aspnetcore - 7 . 0. Accessed: 2022-05-03.

[3] Authorizing Your App with Gmail | Google Developers. https : / /developers.google.com/gmail/api/auth/about-auth. Accessed:

2022-07-06.

[4] C# docs - get started, tutorials, reference. | Microsoft Learn. https:/ / learn . microsoft . com / en - us / dotnet / csharp/. Accessed: 2023-01-03.

[5] Entity Framework documentation | Microsoft Learn. https://learn.microsoft.com/en-us/ef/. Accessed: 2022-05-03.

[6] Gmail API Overview | Google Developers. https : / / developers .google.com/gmail/api/guides. Accessed: 2022-07-07.

[7] Introduction - i18next documentation. https://www.i18next.com/. Accessed: 2022-07-21.

[8] MySQL :: Developer Zone. https://dev.mysql.com/. Accessed: 2022-05-04.

[9] OAuth 2.0 — OAuth. https : / / oauth . net / 2/. Accessed: 2022-05-07.

[10] Aaron Parecki. OAuth 2.0 Simplified. Lulu.com, August 14, 2017.

[11] React – A JavaScript library for building user interfaces. https://reactjs.org/. Accessed: 2022-06-26.

[12] react-chartjs-2 | react-chartjs-2. https : / / react - chartjs - 2 . js .org/. Accessed: 2023-01-03.

[13] Leonard Richardson. RESTful Web APIs: Services for a Changing World. O’Reilly Media; 1st edition, September 30, 2013.

[14] TanStack | High Quality Open-Source Software for Web Developers. https://tanstack.com/. Accessed: 2022-07-22.

[15] The C4 model for visualising software architecture. https : / /c4model.com/. Accessed: 2022-05-04.

[16] TypeScript: The starting point for learning TypeScript. https://www.typescriptlang.org/docs/. Accessed: 2022-06-23.

[17] Janet Mannheimer Zydney and Zachary Warner, ‘Mobile apps for science learning: Review of research. https://www.sciencedirect.com / science / article / abs / pii / S0360131515300737. Accessed: 2022-07-06.

[8] Janet Mannheimer Zydney and Zachary Warner, ‘Mobile apps for science learning: Review of research’

Available: <https://www.sciencedirect.com/science/article/abs/pii/S0360131515300737>

(Citing a website {Title, Link, Accessed date} )

[x] TypeScript: The starting point for learning TypeScript. https://www.typescriptlang.org/docs/. Accessed: 2022-06-23.

(Citing a book {Author, Title, Publisher, Published date} )

[x] Aaron Parecki. OAuth 2.0 Simplified. Lulu.com, August 14, 2017.