

**Faculty of Information Technology I-degree studies**

**Chair of Databases**

**Database specialisation**

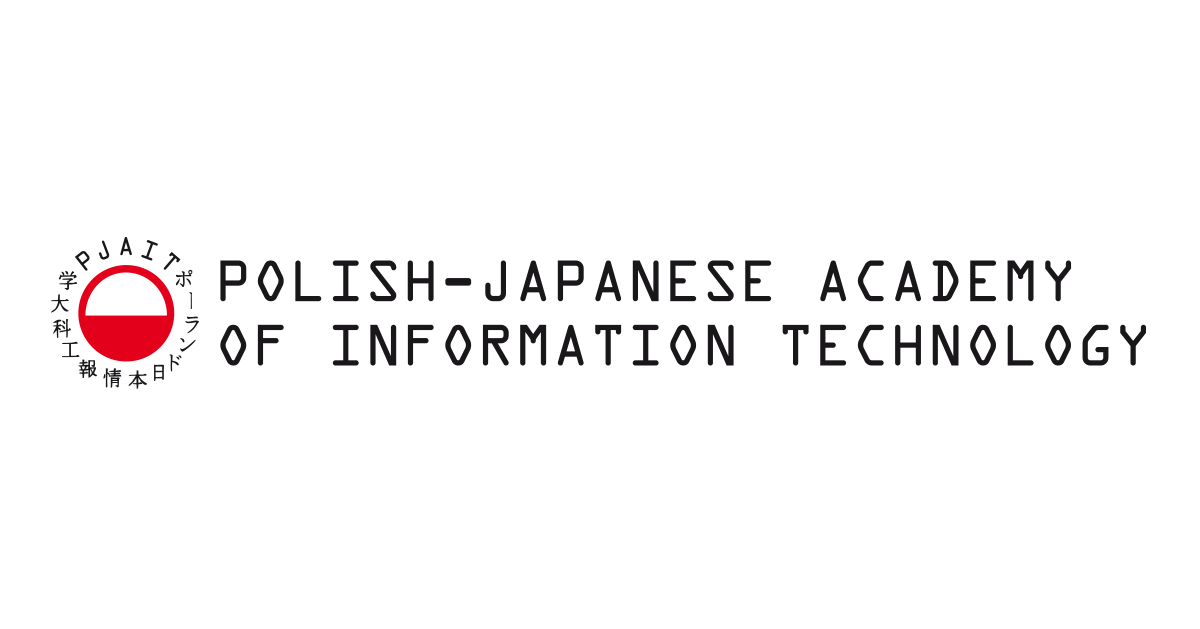
**Daryna Drabysheuskaya s25580 Leonid Kim s24900**

**The Mnemonics App**

**Engineering Thesis**

Supervisor: **Ida Jokisz PhD**

Warsaw, January 2025



**Wydział Informatyki**

**Katedra Baz Danych**

**Specjalizacja Baz Danych**

**Daryna Drabysheuskaya s25580 Leonid Kim s24900**

**The Mnemonics App**

**Praca inżynierska**

Promotor: **Dr. Ida Jokisz**

Warszawa, Styczeń 2025

**Abstract**

The Mnemonics application is an educational web application that supports a learning system based on mnemonic techniques. The platform was developed using .**NET**, **Blazor Web App** and **MSSQL**. Based on an analysis of the detailed user requirements, students, teachers, and administrators' roles in the application were defined.

Students have access to an extensive library of mnemonics prepared independently and created or reviewed by teachers. They can use them in their self-learning. In addition, they can collaborate through the collective development of learning flashcards.

Teachers can add learning cards based on mnemonic techniques or review and publish cards created by students. It ensures high-quality learning cards. Other functionalities developed for teachers include managing student groups and assigning homework as flashcards. Thus, the application can be used to organize interesting group classes on-site or online.

The functionalities developed for administrators allow them to manage users, organize groups, and moderate user content seamlessly.

The "Mnemonics" project will become a user-friendly and easily accessible educational platform.

**Abstrakt**

Aplikacja "Mnemonika" to edukacyjna aplikacja internetowa wspierająca system nauki oparty na technikach mnemonicznych.

Platforma została stworzona w technologii .**NET**, **Blazor Web App** oraz **MSSQL**. Na podstawie analizy szczegółowych potrzeb użytkowników: studentów, nauczycieli i administratorów zdefiniowano ich role w aplikacji.

Studenci mają dostęp do obszernej biblioteki mnemoników opracowanych samodzielnie oraz tworzonych lub recenzowanych przez nauczycieli. Mogą wykorzystywać je w samodzielnej nauce. Mają także możliwość grupowej nauki poprzez wspólne opracowywanie kart edukacyjnych.

Nauczyciele mogą dodawać karty edukacyjne oparte o techniki mnemoniczne oraz recenzować i publikować karty tworzone przez studentów. Zapewnia to wysoką jakość kart edukacyjnych. Inne funkcjonalności opracowane dla nauczycieli to zarządzanie grupami studenckimi oraz przypisywanie zadań domowych w formie kart. Aplikacja może być zatem wykorzystana do organizowania ciekawych ćwiczeń grupowych w formie stacjonarnej lub zdalnej.

Funkcjonalności opracowane dla administratorów umożliwiają łatwe zarządzanie użytkownikami, organizowanie grup i moderowanie treściami tworzonymi przez użytkowników.

Projekt "Mnemonika" stanie się przyjazną dla użytkownika i łatwo dostępną platformą edukacyjną.

The completion of this thesis would not be possible without the support of **Professor Joanna Michalik** of Medical University of Warsaw. Her guidance and insights were of great value and importance. Collaboration with Dr. Michalik also helped us gain experience in working with specific requirements and expectations.

We would also like to extend our thanks to our supervisor, **Dr. Ida Jokisz**. Her constructive feedback and attention to detail were essential in enhancing this project.

**Keywords**

Mnemonics, Flashcards, Education, Project, Web Application, .NET, SQL, Microsoft SQL Server, Blazor, Entity Framework Core, ASP.NET Core Identity, Azure, .NET SDK, Cloud Deployment, Manual Testing, Visual Studio.

**Słowa kluczowe**

Mnemoniki, Fiszki, Edukacja, Projekt, Aplikacja Webowa, Bazy Danych, .NET, SQL, Microsoft SQL Server, Blazor, Entity Framework Core, ASP.NET Core Identity, Azure, .NET SDK, Wdrożenie w chmurze, testy ręczne, Visual Studio.

Table of Contents

[Glossary 9](#_Toc188972076)

[1. Introduction 11](#_Toc188972077)

[1.1. Description 11](#_Toc188972078)

[1.2. Mnemonics App Objectives 11](#_Toc188972079)

[2. Requirements 12](#_Toc188972080)

[2.1. Functional Requirements 12](#_Toc188972081)

[2.2. Non-Functional Requirements 14](#_Toc188972082)

[3. Application Design Decisions 16](#_Toc188972083)

[3.1. Mnemonics App Use Case Diagram 16](#_Toc188972084)

[3.2. Use-Case Scenarios 17](#_Toc188972085)

[3.3. Mnemonics App Entity Diagram 24](#_Toc188972086)

[3.4. Description of the schema 27](#_Toc188972087)

[4. Selected Technologies 28](#_Toc188972088)

[4.1. Backend 28](#_Toc188972089)

[4.2. Frontend 29](#_Toc188972090)

[4.3. ORM 31](#_Toc188972091)

[4.4. Database 32](#_Toc188972092)

[4.5. User Identity 33](#_Toc188972093)

[4.6. Other tools 34](#_Toc188972094)

[5. Implementation 35](#_Toc188972095)

[5.1. Context 35](#_Toc188972096)

[5.2. Overview 36](#_Toc188972097)

[5.3. Details 38](#_Toc188972098)

[6. Typical Usage Scenario 47](#_Toc188972099)

[7. Testing Phase 51](#_Toc188972101)

[7.1. Overview 51](#_Toc188972102)

[7.2. Test Cases 52](#_Toc188972103)

[7.3. Feedback 56](#_Toc188972104)

[8. Possible Improvements 57](#_Toc188972105)

[9. Responsibilities Of Team Memebers 58](#_Toc188972106)

[Bibliography 59](#_Toc188972107)

[List Of Indexes 62](#_Toc188972108)

[Table Of Figures 63](#_Toc188972109)

[Annex I 64](#_Toc188972110)

[User Manual 64](#_Toc188972111)

[Annex II 75](#_Toc188972114)

[Application screenshots 75](#_Toc188972115)

Glossary

**.NET** – it is a cross-platform development framework provided by Microsoft for developing applications, whether on the web, desktop, or mobile.

**.NET SDK** – a set of libraries and tools that developers use to create .NET applications and libraries.

**Angular** – a comprehensive JavaScript framework for building dynamic web applications.

**ASP.NET Core Identity** – a membership system for ASP.NET Core applications that provides authentication, authorisation, and user management.

**AVX-512** – ***Advanced Vector Extensions 512,*** a CPU instruction set designed for high-performance computing.

**AWS Cognito** – a user identity and access management service that provides secure authentication, authorisation, and user management.

**Azure** – a cloud computing platform developed by Microsoft.

**Blazor** – a web framework for building interactive web applications using C# and .NET.

**Blazor Pipeline** – the lifecycle process in Blazor for handling requests and rendering components.

**CPU** – ***Central Processing Unit*,** the primary component of a computer that executes instructions, performs calculations, and processes data.

**CSR** – ***Client-Side Rendering,*** a rendering method where the browser downloads a JavaScript bundle and generates the HTML dynamically on the client side, offering more interactive and seamless experiences but requiring more initial resources.

**Dapper** – a lightweight and high-performance micro-ORM for .NET that simplifies data access by mapping SQL queries to .NET objects.

**DOM** – Document Object Model. A cross-platform and language-independent interface that treats an HTML or XML document as a tree structure. Each node is an object representing a part of the document.

**EF Core** – ***Entity Framework Core*** is a powerful ORM (Object-Relational Mapper) for .NET, enabling developers to interact with databases using .NET objects instead of raw SQL.

**Flashcard** – an interactive learning tool that displays a mnemonic along with its explanation or related details.

**IDE** – ***integrated development environment,*** a software application that provides facilities for software development

**JavaScript** – a versatile programming language primarily used to create dynamic and interactive content on websites.

**JSX** – **JSX *(JavaScript XML)*** allows developers to write HTML-like syntax within JavaScript, making UI development intuitive.

**JVM** – ***Java Virtual Machine*,** a runtime environment that executes Java bytecode, enabling platform-independent application execution.

**Lucidchart** – web-based diagramming application.

**Mnemonic** – a learning tool that uses patterns, phrases, or associations to help in retaining and recalling information.

**MSSQL - *Microsoft SQL Server,*** a robust relational database management system used for storing and managing structured data.

**MudBlazor** – a modern, material design-based UI component library for Blazor applications.

**MVC** – ***Model-View-Controller*,** a design pattern that separates an application into three interconnected components: Model, View, Controller.

**React -** a popular JavaScript library for building user interfaces, known for its component-based architecture and efficient Virtual DOM.

**SSR - *Server-Side Rendering*,** a rendering method where HTML is generated on the server and sent to the client, enabling faster initial load times and better SEO performance, as the content is available immediately to users and search engines.

**Vertabelo** – online database modeler.

**Virtual DOM** – in-memory representation of the actual DOM used by frameworks like React.

**Visual Studio** – IDE for .NET and C++ developers on Windows.

**WebAssembly** – a binary instruction format that allows code written in languages like C, C++, C#, or Rust to run in web browsers at near-native speed.

**Windows** –operating systems developed and marketed by Microsoft.

1. Introduction

## 1.1. Description

The Mnemonics App is an educational platform designed to enhance learning through mnemonic techniques.

It supports diverse functionalities for different user roles:

**Students:**

Students can join groups, browse the library of mnemonics and learn quick facts about them, solve flashcard sets related to mnemonics.

**Teachers:**

Teachers have the capability to contribute new mnemonic content and manage existing mnemonics.

They also create, oversee student groups and assign flashcard sets for groups, which supports structured educational activities and content management.

**Administrators**:

Administrators have access to functionalities available to teachers and also can manage users.

The Mnemonics App effectively combines educational content management with user collaboration and administrative oversight, positioning itself as a solution for educational institutions.

## 1.2. Mnemonics App Objectives

The primary objective of the Mnemonics App project is to develop a user-friendly educational platform that enhances learning and memory retention through the use of mnemonic techniques. The specific objectives include:

1. **Facilitate Learning Through Mnemonics:**

* Offer a collection of mnemonic resources across medical disciplines to assist students in enhancing their memorization skills.

1. **Enable Collaborative Learning:**

* Enable creation and management of groups, allowing students to engage in collaborative learning experiences.

1. **Support Educational Content Management:**

* Allow educators to contribute to, manage, and organize mnemonic resources effectively.

1. **Ensure Usability and Accessibility:**

* Create a user-friendly and accessible interface that adheres to usability standards, making the application easy to navigate for users of all ages and abilities.

1. **Provide Administrative Control:**

* Equip administrators with the necessary tools to oversee users, groups, and content, ensuring the platform operates efficiently.

# Requirements

## 2.1. Functional Requirements

User Functional Requirements

1. **Register**

* The system shall allow users to create a new account by providing necessary information such as email, and password.

1. **Login**

* The system shall authenticate users by verifying their credentials (email and password) and grant access to the application.

1. **Change Credentials for Login**

* The system shall allow users to update their password.

1. **View Departments and Categories**

* The system shall allow users to browse different categories of mnemonics for specific departments.

1. **View Mnemonics**

* The system shall allow users to view mnemonics within each category.

1. **Add Mnemonic to Favorites**

* The system shall allow users to mark mnemonics as favorites for easy access in the future.

Student Functional Requirements

1. **Join Group**
   * The system shall allow students to join existing groups created by teachers or admins.
2. **Solve Flashcard Sets**
   * The system shall allow students to complete sets assigned by their teachers within their groups.
3. **Add New Mnemonics as a Student**
   * The system shall allow students to add new mnemonics and send them to teachers for approval.

Teacher And Admin Functional Requirements

1. **Add New Mnemonics**

* The system shall allow teachers to add new mnemonics to the app’s database.

1. **Manage Mnemonics**

* The system shall allow teachers to edit or delete mnemonics they have created or have access to.

1. **Create Groups**

* The system shall allow teachers to create groups for their students.

1. **Manage Groups**

* The system shall allow teachers to manage the groups they have created, including adding or removing students and modifying group details.

1. **Assign Flashcard Sets**

* The system shall allow teachers to assign flashcards related to mnemonics to their students within groups.

1. **Manage Groups**

* The system shall allow admins to manage all groups within the app, including viewing and modifying group details.

1. **Manage Mnemonics**

* The system shall allow admins to manage all mnemonics in the app, including adding, editing, and deleting mnemonics.

1. **Manage Flashcard Sets**

* The system shall allow admins to manage all flashcard sets within the app, including creating, assigning, and deleting flashcards.

Admin Functional Requirements

1. **Manage Users**

* The system shall allow admins to manage user accounts, including deleting accounts.

## 2.2. Non-Functional Requirements

Performance Requirements

1. **Response Time**

* The system is required to achieve a response time of under 3 seconds for 95% of user requests during standard operating conditions.

1. **Scalability**

* The system must possess the capability to accommodate up to 2000 users while maintaining optimal performance levels. At least 100 users should be able to use the system simultaneously.

Reliability Requirements

1. **Uptime**

* The system shall have an uptime of 99.9%, ensuring that it is available 24/7 except during scheduled maintenance.

1. **Error Handling**

* The system shall gracefully handle errors and provide meaningful error messages to users.

Security Requirements

1. **Authentication**

* The system shall require all users to authenticate using a secure login mechanism (e.g., email and password).

1. **Authorization**

* The system shall ensure that users have access only to the resources and actions that their roles permit.

1. **Data Encryption**

* The system shall encrypt sensitive data.

Usability Requirements

1. **User Interface**

* The system shall have a user-friendly interface that adheres to common usability standards and guidelines.

1. **Documentation**

* The system shall provide comprehensive user documentation and help resources, including a user manual.

Maintainability Requirements

1. **Modularity**

* The system shall be designed in a modular fashion to facilitate easy updates and maintenance.

1. **Code Quality**

* The system's codebase shall follow established coding standards and best practices to ensure readability and maintainability.

1. **Manual Testing**

* The system should include a testing environment in the cloud with error logging in order to ensure that functionalities to be added to production work properly.

Portability Requirements

1. **Platform Compatibility**

* The system shall be compatible with popular operating systems, including iOS, Android, and Windows.

1. **Browser Compatibility**

* The system shall be compatible with major web browsers.

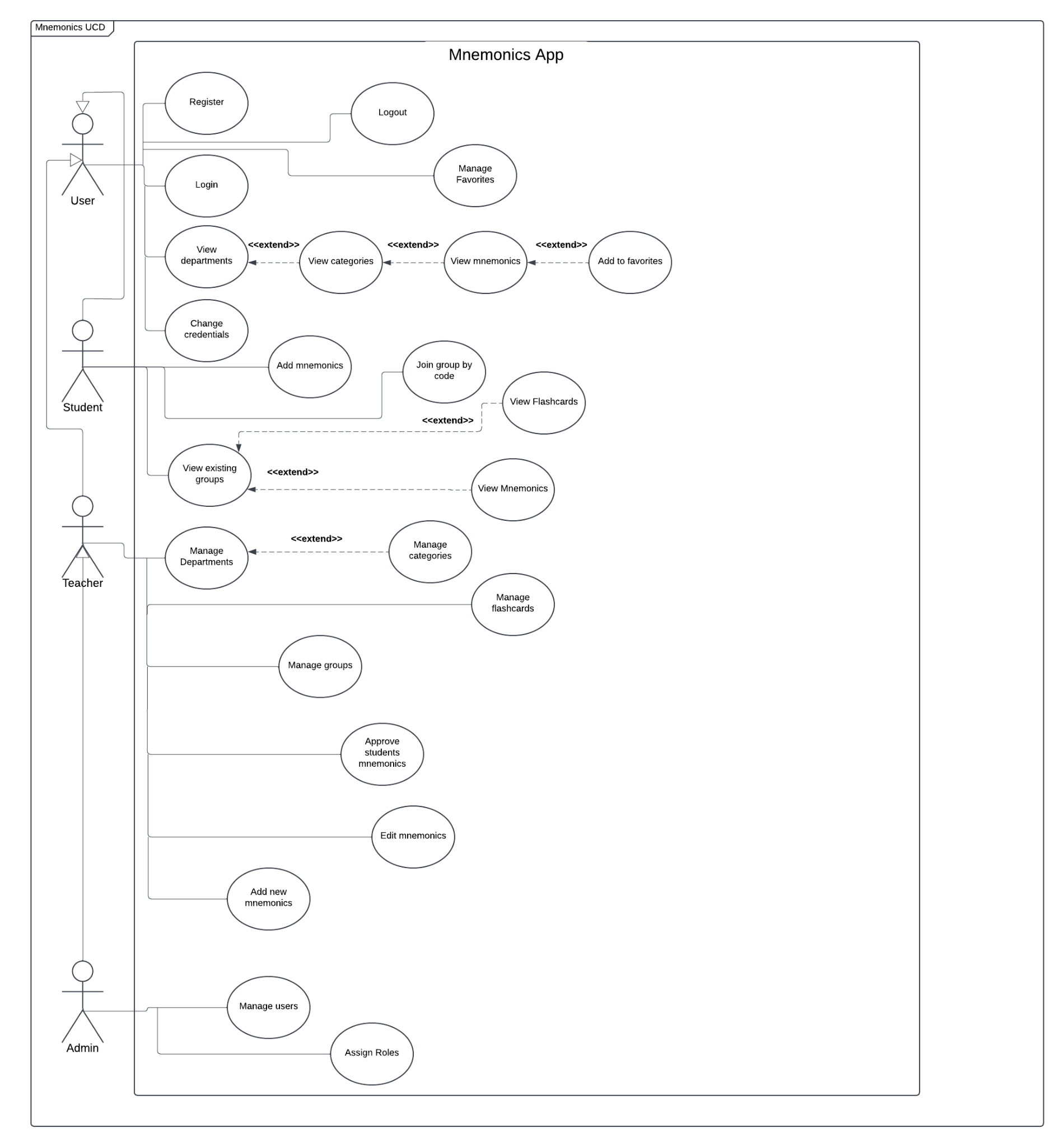
Capacity Requirements

1. **Data Storage**

* The system shall have sufficient data storage capacity to store user data and mnemonic content for at least 2 years.

3. Application Design Decisions

## 3.1. Mnemonics App Use Case Diagram



**Figure 1** Use Case Diagram

## 3.2. Use-Case Scenarios

To make implementation easier we developed use case scenarios according to the specified requirements.

**Use-Case Scenario 1: Registering a New User**

**Actors**: User

**Preconditions**: The user has access to the internet and the Mnemonics App.

**Postconditions**: The user has successfully created an account.

**Main Success Scenario**:

1. The user navigates to the registration page of the Mnemonics App.
2. The user enters required details, including his email, password, and confirmation password.
3. The user submits the registration form.
4. The system validates the input and creates a new user account.
5. The user is redirected to the login page.

**Extensions**:

* If the user does not fill in all required fields, the system prompts the user to complete the missing information.
* If the username or email is already taken, the system prompts the user to choose a different username or email.

**Use-Case Scenario 2: Logging In**

**Actors**: User   
 **Preconditions**: The user has a registered account and access to the internet.  
 **Postconditions**: The user is logged in or notified of login issues.

**Main Success Scenario**:

1. The user navigates to the **Login** page.
2. He enters his registered email and password.
3. The system validates the credentials.
4. On successful validation, the user or admin redirected to the dashboard.

**Extensions**:

* If the password is incorrect, the system displays an error message.
* If the use or admin has forgotten their password, they can reset it via the "Forgot Password" option.

**Use-Case Scenario 3: Viewing Mnemonics**

**Actors**: User  
**Preconditions**:

* The user is logged into their account.
* Mnemonics have been added to the system.

**Postconditions**:

* The user views mnemonics from the selected department or group.

**Main Success Scenario**:

1. The user navigates to the **View Mnemonics** section.
2. They select a department or group.
3. The system displays a list of available mnemonics.
4. The user clicks on a mnemonic to view its details.
5. The system shows the mnemonic's title, phrase, and description.

**Extensions**:

* The user can click "Add to Favorites" to save a mnemonic for later.

**Use-Case Scenario 4: Joining a Group**

**Actors**: Student  
**Preconditions**:

* The user is logged into their account.
* The user has a valid group code.

**Postconditions**:

* The student successfully joins the specified group.

**Main Success Scenario**:

1. The student navigates to the **Join Group** section.
2. The system displays a field to enter the group code.
3. The student enters the code and submits the form.
4. The system validates the code.
5. The student is added to the group and notified of their successful join.

**Extensions**:

* If the code is invalid, the system displays an error message and prompts the user to try again.

**Use-Case Scenario 5: Adding Mnemonics**

**Actors**: Teacher  
**Preconditions**:

* The teacher is logged into account.

**Postconditions**:

* A new mnemonic is added to the system for review.

**Main Success Scenario**:

1. The teacher navigates to the **Add Mnemonics** section.
2. The system displays a form to input mnemonic details.
3. The teacher fills in the name, description, phrase, and selects a category.
4. An image is uploaded for the mnemonic.
5. The teacher submits the form.
6. The system validates the input and saves the mnemonic for review.

**Use-Case Scenario 6: Logging Out**

**Actors**: User  
**Preconditions**:

* The user is logged into their account.

**Postconditions**:

* The user is logged out and redirected to the login screen.

**Main Success Scenario**:

1. The user clicks the "Logout" button in the app.
2. The system ends the user’s or admin’s session.
3. The user is redirected to the login page.

**Use-Case Scenario 7: Managing Favorites**

**Actors**: User  
**Preconditions**:

* The user is logged into their account.
* The user has favorited mnemonics.

**Postconditions**:

* The user views, adds, or removes mnemonics from their favorites list.

**Main Success Scenario**:

1. The user navigates to the "Favorites" section.
2. The system displays a list of favorite mnemonics.
3. The user selects a mnemonic to view its details or removes it from favorites.

**Extensions**:

* If there are no favorites, the system displays a message.

**Use-Case Scenario 8: Changing Credentials**

**Actors**: User  
**Preconditions**:

* The user is logged into their account.

**Postconditions**:

* The user’s credentials (email, password, etc.) are successfully updated.

**Main Success Scenario**:

1. The user navigates to the mail - Profile" section.
2. They update their email or password in the provided fields.
3. The user submits the changes.
4. The system validates the input and updates the credentials.
5. The system confirms the changes.

**Extensions**:

* If the password is weak, the system prompts the user to choose a stronger password.

**Use-Case Scenario 9: Viewing Existing Groups**

**Actors**: User  
**Preconditions**:

* The user is logged into their account.

**Postconditions**:

* The user views a list of existing groups they are a member of or have access to.

**Main Success Scenario**:

1. The user navigates to the "Groups" section.
2. The system displays a list of available groups.
3. The user selects a group to view its details.

**Extensions**:

* If the user is not part of any group, the system displays a message.

**Use-Case Scenario 10: Viewing Flashcards**

**Actors**: User  
**Preconditions**:

* The user is logged into their account.
* Flashcards have been created for a mnemonic group.

**Postconditions**:

* The user views and interacts with mnemonic flashcards.

**Main Success Scenario**:

1. The user navigates to the "Flashcards" section.
2. The system displays a flashcard with the mnemonic image and phrase.
3. The user interacts with the flashcard using options like "I Knew That" or "I Didn’t Know That."
4. The user completes the session and views their performance summary.

**Extensions**:

* If there are no flashcards, the system displays a message.

**Use-Case Scenario 11: Managing Departments**

**Actors**: Admin  
**Preconditions**:

* The admin is logged into their account.

**Postconditions**:

* Departments are successfully added, edited, or deleted.

**Main Success Scenario**:

1. The admin navigates to the "Manage Departments" section.
2. They view a list of existing departments.
3. The admin adds a new department, edits an existing one, or deletes a department.
4. The system updates the department list accordingly.

**Extensions**:

* If the department name is already taken, the system prompts the admin to choose a unique name.

**Use-Case Scenario 12: Managing Groups**

**Actors**: Admin  
**Preconditions**:

* The admin is logged into their account.

**Postconditions**:

* Groups are successfully managed (added, edited, or deleted).

**Main Success Scenario**:

1. The admin navigates to the "Manage Groups" section.
2. They view a list of existing groups.
3. The admin adds a new group, edits an existing group, or deletes a group.
4. The system updates the group list.

**Use-Case Scenario 13: Approving Mnemonics**

**Actors**: Admin  
**Preconditions**:

* The admin is logged into their account.

**Postconditions**:

* Submitted mnemonics are approved or rejected.

**Main Success Scenario**:

1. The admin navigates to the "Approve Mnemonics" section.
2. The system displays a list of pending mnemonics.
3. The admin reviews the details of a mnemonic.
4. The admin approves or rejects the mnemonic.

**Extensions**:

* If the mnemonic lacks required information, the admin can send it back to the teacher for revision.

**Use-Case Scenario 14: Assigning Roles**

**Actors**: Admin  
**Preconditions**:

* The admin is logged into their account.

**Postconditions**:

* User roles are successfully assigned or updated.

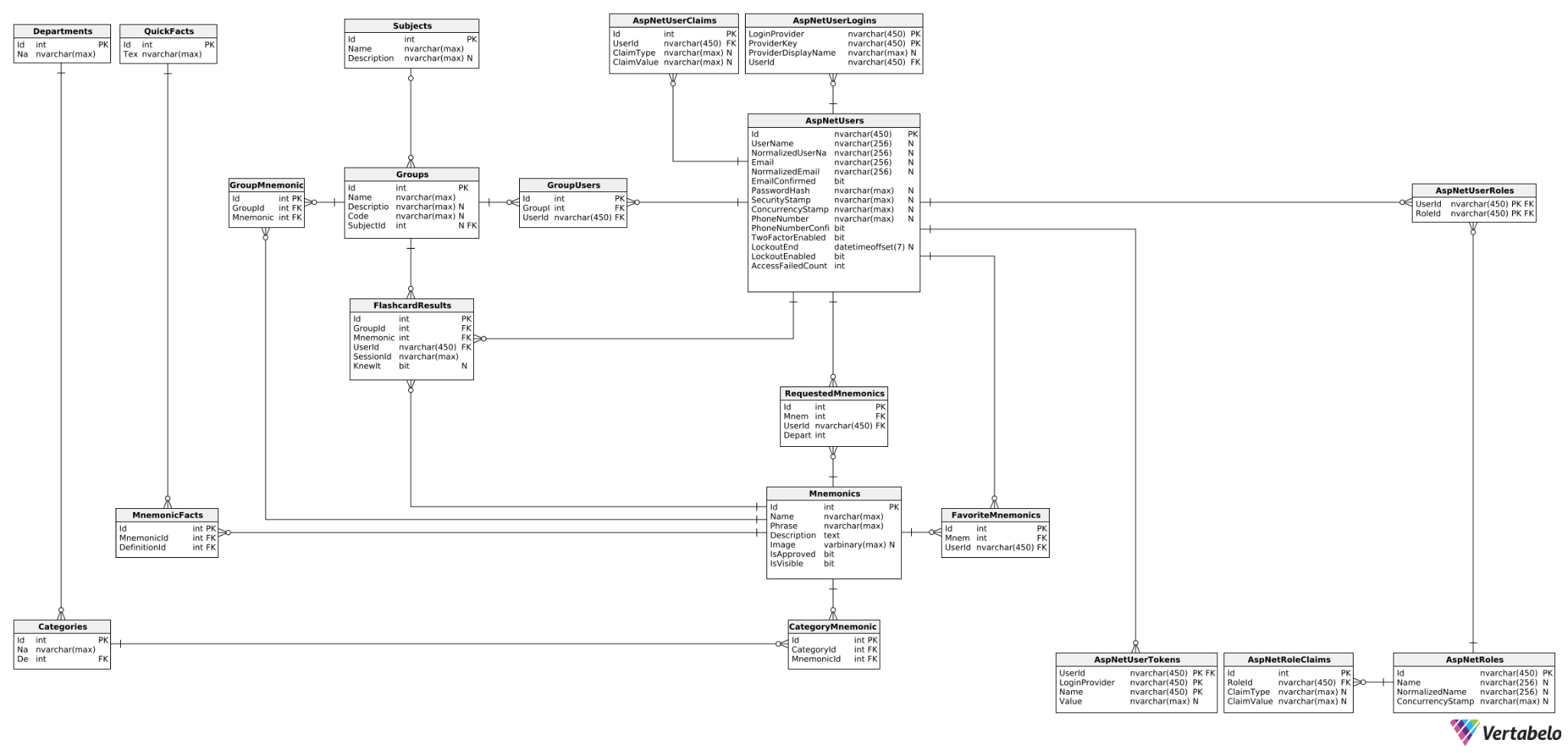
**Main Success Scenario**:

1. The admin navigates to the "Assign Roles" section.
2. They view a list of all registered users.
3. The admin selects a user and assigns a role (e.g., student, teacher).
4. The system saves the changes and confirms success.

**Extensions**:

* If the admin tries to assign a conflicting role, the system provides a warning.

## 3.3. Mnemonics App Entity Diagram

****

**Figure 2** Database Schema

A diagram of a computer flowchart

Description automatically generated

**Figure 3** Zoomed In Schema I

A diagram of a computer flowchart

Description automatically generated

**Figure 4** Zoomed In Schema II

3.4. Description of the schema

This schema describes a database that combines application entities with Microsoft User Authentication. There are essential tables such as **AspNetUsers**, **AspNetRoles**, **AspNetUserRoles**, **AspNetUserClaims**, **AspNetUserLogins**, and **AspNetUserTokens**. These tables are used to manage user accounts, role assignments, claims, and login methods, thereby providing strong authentication and authorization functionalities. Important fields include Id (as the primary key), **UserName**, **Email**, **RoleId**, and **ClaimType**, all supported by suitable primary and foreign key relationships to ensure data integrity.

Additionally, the application includes tables dedicated to the management of mnemonics, definitions, categories etc. and their interconnections. The **Mnemonics** table holds educational mnemonics along with relevant data, including **Phrase**, **Description**, and **Image**. Related tables such as **Categories**, **Groups**, and **Departments** categorize mnemonics into organized structures, while **MnemonicDefinitions**, **CategoryMnemonics**, and **GroupMnemonics** facilitate many-to-many relationships.

Furthermore, tables like **FlashcardResults** and **FavoriteMnemonics** track user interactions and preferences.

In summary, the schema is designed for optimal scalability, balancing between normalization and performance.

4. Selected Technologies

## 4.1. Backend

**Selected Technology: .NET 8**

**Advantages**:

**Performance**: Dynamic Profile-Guided Optimization and ***AVX-512*** support enhance execution speed **1**

**Container Enhancements**: Integrated tooling for containerization **1**

**Cross-Platform**: Unified development across Windows, macOS, and Linux **2**

**Disadvantages**:

**Vendor Lock-In**: Heavy reliance on Microsoft's ecosystem **3**

**Complexity**: Advanced features may increase the learning curve **4**

**Alternative 1: *Spring Boot (Java)***

**Advantages**:

**Performance**: Mature ***JVM***-based optimizations **5**

**Cloud-Native**: Built-in support for microservices with Spring Cloud **6**

**Wide Adoption**: A large ecosystem with enterprise-grade applications **7**

**Disadvantages**:

**Setup Complexity**: Requires more configuration than .NET 8 **7**

**Java Dependency**: May require Java knowledge for effective use **7**

**Alternative 2: *Node.js***

**Advantages**:

**Event-Driven**: Asynchronous I/O for high-performance, non-blocking servers **8**

**Scalability**: Excellent for lightweight microservices **8**

**Disadvantages**:

**Callback Hell**: Can result in hard-to-maintain code if not handled properly **8**

**Single-Threaded**: May struggle with ***CPU***-intensive operations **8**

## 4.2. Frontend

**Selected Technology: Blazor Web App**

**Advantages**:

**Full-Stack C#**: Unified language across client and server **9**

**Seamless Integration with .NET Ecosystem:** Considering the selected framework, it is a good choice for frontend development **9**

**Skills of Developers**: Since we are familiar with .NET, Blazor Web App is great within Microsoft’s ecosystem.

**Disadvantages**:

**Initial Load Time**: Slower load times compared to ***React*** or ***Angular* 10**

**Runtime Environment**:Compared to React, the latter uses browser’s ***JavaScript*** engine (which is an industry standard), while Blazor relies on ***WebAssembly*** for C# execution **11**

**Alternative 1: React.js**

**Advantages**:

**Component-Based**: Encourages modular and reusable components **12**

***Virtual DOM***: Optimized for fast UI rendering.

**Disadvantages**:

***JSX* Complexity**: Can be challenging for beginners.

**Large Ecosystem**: Steep learning curve for beginners.

**Alternative 2: Angular**

**Advantages**:

**Full-Featured Framework**: Built-in solutions for routing and state management **13**

**TypeScript Support**: Enhances maintainability and code quality **13**

**Disadvantages**:

**Learning Curve**: More complex compared to Blazor or React **13**

**Bundle Size**: Larger bundle sizes compared to React **13**

## 4.3. ORM

**Selected Technology: Entity Framework Core**

**Advantages**:

**Simplified Data Access**: Allows interaction with databases using .NET objects, eliminating the need for extensive SQL queries **14**

**Cross-Platform Compatibility**: Supports Windows, macOS, and Linux environments **14**

**Code First Approach**: Enables schema generation from .NET classes, promoting flexibility in model design **15**

**Automatic Migrations**: Simplifies schema changes and version control alongside application code **16**

**Asynchronous Methods**: Built-in async query and save support improves app responsiveness **17**

**Disadvantages**:

**Performance Overhead**: Slightly slower for complex queries compared to raw SQL or micro ORMs like Dapper **18**

**Complex Learning Curve**: Advanced features like custom mappings can be challenging **19**

**Alternative 1: *Dapper* (Micro ORM)**

**Advantages**:

**High Performance**: Faster than ***EF Core*** due to minimal abstraction overhead. **20**

**Flexibility**: Supports raw SQL queries, offering more control for complex database operations **20**

**Lightweight**: Simple to integrate with projects, especially for apps requiring optimized database access. **20**

**Disadvantages**:

**Manual Mapping**: Developers need to handle object mapping explicitly.

**No Built-In Migrations**: Schema management is not included, requiring external tools or manual scripts.

## 4.4. Database

**Selected Technology: Microsoft SQL Server**

**Advantages**:

**Scalability and Performance:** SQL Server supports large-scale database operations with ease.

**Security**: Its security features have robust encryption techniques and advanced forms of authentication so that sensitive information is protected and safe.

**Integration**: Offers strong integration with various Microsoft products including Azure.

**Various Editions Available**: suitable for different needs and budgets **21**

**Disadvantages**:

**Cost**: Although many options are available, enterprise edition is priced quite steeply **22**

**Configuration Complexity:**: Setting up MSSQL may be challenging for beginners **23**

**Alternative 2: MongoDB (NoSQL)**

**Advantages**:

**Flexible Schema**: Ideal for apps where data structures frequently evolve **24**

**Horizontal Scalability**: Suited for handling large volumes of unstructured or semi-structured data **24**

**Document-Oriented Storage**: Efficient for applications with hierarchical or nested data **25**

**Disadvantages**:

**Transaction Support**: Transactions are less robust compared to relational databases **25**

**Indexing Challenges**: Queries without proper indexes can degrade performance **25**

## 4.5. User Identity

**Selected Technology: *ASP.NET Core Identity***

**Advantages**:

**Built-in Security Features**: provides fundamental security features that include password hashing, token-based authentication **26**

**Scalability:** By design, ASP.NET Core Identity works effectively from the smallest application all the way into large enterprise applications **26**

**Ecosystem**: Visual Studio 2022 provides simple integration with ASP.NET Core Identity

**Disadvantages**:

**Learning Curve**: According to the feedback provided by .NET developers, ASP.NET Core Identity functionalities **27**

**Documentation and support**: Some developers have noted that the documentation and support for ASP.NET Core Identity can be lacking **28**

**Alternative 1: Auth0**

**Advantages**:  
 Easily integrates with your Blazor app for authentication, offering robust security and scalability.

**Advantages**:

**Ease of Setup**: Straightforward implementation with Blazor WebAssembly **29**

**Comprehensive Features**: Provides social login and multi-factor authentication **29**

**Scalability**: Supports large user bases effectively **29**

**Disadvantages**:

**Cost**: Can become expensive as user count grows **30**

**Third-Party Dependency**: Adds reliance on external services for critical functionality **29**

**Alternative 2: Firebase Authentication**

**Advantages**:

**Ease of Integration**: SDKs for Blazor Web App available **30**

**Scalability**: Works seamlessly with Firebase services for authentication and database **30**

**Flexibility**: Supports passwordless and social logins **30**

**Disadvantages**:

**Vendor Lock-In**: Dependency on Firebase ecosystem **31**

**Limited Customization**: Less flexible ***than AWS Cognito* 30**

## 4.6. Other tools

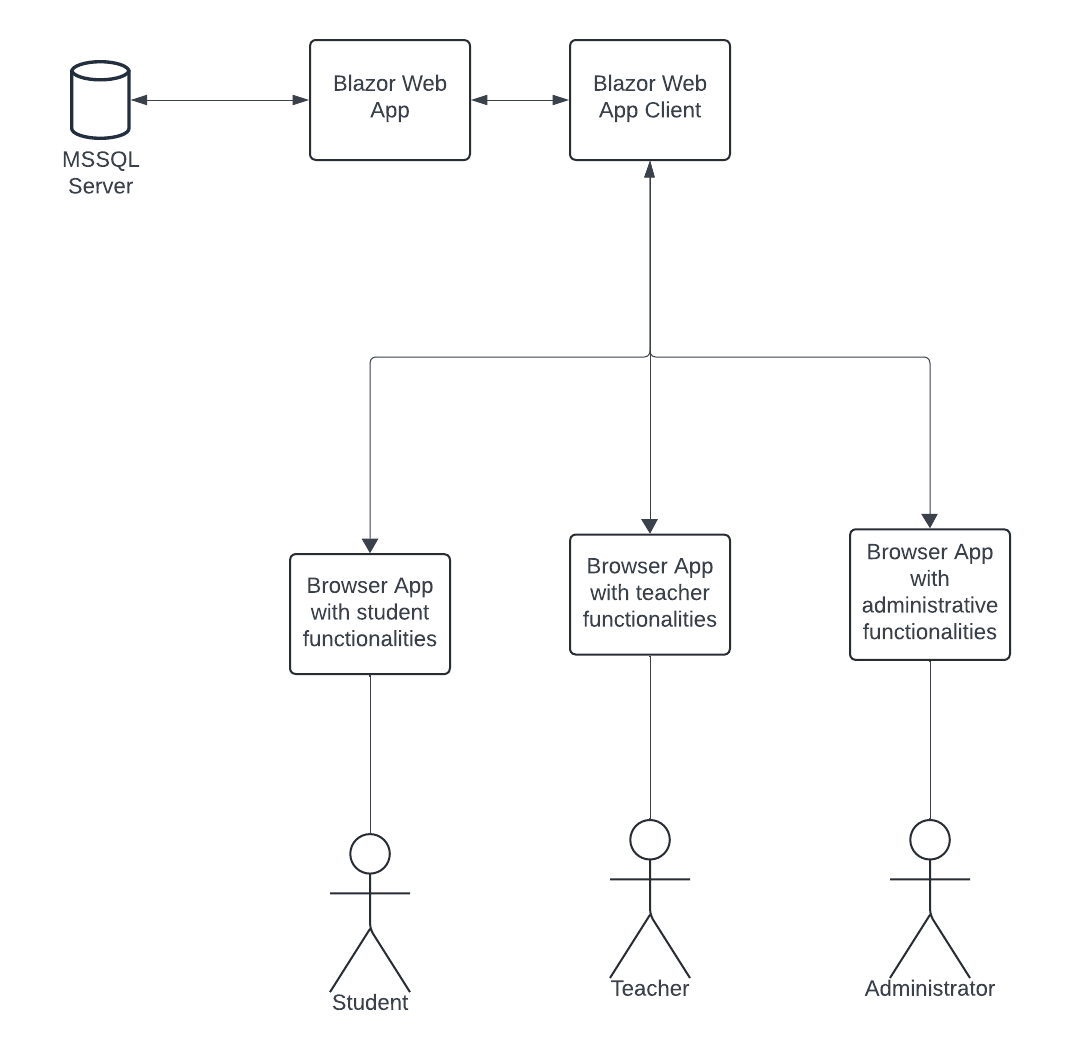
**IDE:** Visual Studio 2022

**Text Editor**: Microsoft Word

**Cloud Deployment for Testing**: Microsoft Azure Container Apps

5. Implementation

## 5.1. Context



**Figure 5** Context Diagram

## 5.2. Overview

This section describes the methodology used for creating our app. It includes design choices, aspects of implementation and goes further into details on how chosen frameworks and technologies work for our goals.

Blazor Pipeline

There are two main ways to process requests and return responses:

* Blazor Pipeline
* MVC

For the needs of our application using Blazor Pipeline is more than enough. Blazor allows to select out of several render options. In most cases we used Interactive Auto Mode. Interactive Auto Mode is an interactive SSR using Blazor Server initially and then CSR on subsequent visits after the Blazor bundle is downloaded.

This approach allows the application to download .NET runtime and app bundle to the client, cache them and use them on future visits, which would potentially increase server performance.

Project Structure  
  
The main project structure is as follows:

**wwwroot**: Contains static files such as CSS, JavaScript, and images.  
**Components**: Contains reusable UI components and pages. Components are then split into packages based on the data models that are required in them.  
**Data**: Contains the data models and the DbContext for Entity Framework Core. EF Core utilizes a metadata model to define how the entity types in the application are mapped to the corresponding database structure.  
**Services**: Contains the service classes used for business logic and data access.  
**Program.cs**: The entry point of the application, where services are configured, and the application is built and run.

**appsettings.json**: Contains configuration settings for the application, including connection strings and logging settings.

Blazor apps are built out of Razor components, more often called Blazor components. A component is a self-contained piece of UI, such as a page, dialog, or form, that includes the processing logic for the functionality that it implements.

When a component is rendered, an in-memory representation of the browser's Document Object Model (DOM) is updated to reflect the structure and state of the component. This in-memory representation is known as a render tree.

Components are implemented using a combination of C# code and HTML markup in Razor component files, which have the.razor file extension.

ComponentBase is the base class for components defined in Razor files. ComponentBase implements the essential abstraction of components as declared by the IComponent interface. ComponentBase provides basic properties and methods that allow basic component functionality, including support for a set of built-in lifecycle events.

A screenshot of a computer

Description automatically generated

**Figure 6** Project Structure

## 5.3. Details

In this section we will discuss the details of implementation with some of the noteworthy examples.

Data

An integral part of the application are mnemonics. The “Mnemonic” entity is the one that uses more datatypes and relations in typical scenarios.

A screenshot of a computer program

Description automatically generated

**Figure 7** Data Model (Mnemonic)

Let us explain each line in the model.

**Id** represents the primary key (EF allows us to use attribute [**Key**] for that purpose) of the table, uniquely identifying each mnemonic. **Name** represents the name of a mnemonic and is mandatory, which is why the attribute [**Required**] is used. **Phrase** represents the actual mnemonic phrase. **Description** provides more details on a mnemonic. **Image** storesbinary data of an image, in case the image is provided by the user. Image size is limited to 512 kilobytes. Flags **IsApproved** and **IsVisible** serve is a filter mechanism, which allows teachers to have control over data that is accessible to students. The last three fields serve as navigation properties in EF, used for data retrieval.

A screenshot of a computer program

Description automatically generated

**Figure 8** Data Model (RequestedMnemonic)

**RequestedMnemonic** is an example on how the models could be used together. The purpose of that entity is to allow students to create their own mnemonics and then assign a teacher (referenced by **User** and **UserId**). The teacher is able to approve the mnemonic and make it accessible to other students. **MnemonicId** serves as a Foreign Key to mnemonic and object **Mnemonic** is used as a back-reference. **DepartmentId** allows the student to assign a department for the mnemonic.

Other entities are described in the same fashion. Models are then added to the database context.

A screenshot of a computer

Description automatically generated

**Figure 9** Application Database Context

Then the command “Add-Migration” is applied, which would allow us to capture model changes. Then we can run “Update-Database” to apply those changes to the database.

A screenshot of a computer code

Description automatically generated

Services

Classes that are put in **Services** package handle data access and business logic.

In a complex project hard-coded dependencies are problematic and may introduce unnecessary problems for developers. For that reason, we address those problems through the usage of dependency injections.

Each service uses an interface to abstract the dependency implementation.

A screenshot of a computer program

Description automatically generated

**Figure 10** Mnemonic Service Interface

The interface is then implemented by a concrete type.

A screenshot of a computer program

Description automatically generated

**Figure 11** Mnemonic Service Concrete Class

In the concrete class database context is provided, methods to create, read, update and delete data are implemented along with other methods which manipulate data.

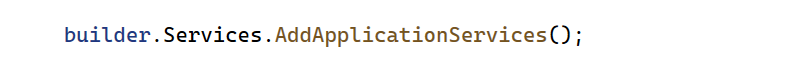
Class **ServiceCollectionExtensions** collects the required services for convenient use.

A screenshot of a computer program

Description automatically generated

**Figure 12** Service Collection Extensions

And then they are registered in **Program**.**cs**.



**Figure 13** Registering of Services in the Application

**Scoped** lifetime means that a service is created once per request and is disposed afterwards.

Components

Components (Blazor pages) are where everything comes together.

Each component class is written in the form of a Razor markup page. It allows us to use HTML markup with C# code.

Firstly, we need to select a render mode for the component, specify a route to the page, import necessary services and models, inject services, and in some cases specify the roles, for which this page is available.

A screen shot of a computer program

Description automatically generated

**Figure 14** Imports and injections in a Component

Then, we create Razor markup. The provided example shows how the form to add mnemonics as a teacher was developed.

A screenshot of a computer

Description automatically generated

**Figure 15** Razor markup I

A screenshot of a computer

Description automatically generated

**Figure 16** Razor markup II

The imported model allows us to bind specific values of a mnemonic entry to corresponding parts of the page, as well as to bind methods to buttons.

Underneath, the code that handles the entire logic of the page is written.

A computer screen shot of a program

Description automatically generated

**Figure 17** Code example in a Component I

A computer screen shot of text

Description automatically generated

**Figure 18** Code example in a component II

This is how it looks like in the browser:

A white screen with black text

Description automatically generated

**Figure 19** Rendered page example

A teacher provides data on the mnemonic, may upload an image from the device on which the browser runs. Before submitting, the system checks that the required fields are not empty and then the mnemonic is saved.

**MudBlazor** makes the process of development a lot easier with its rich library. Another noteworthy example would be the implementation of lists of categories and mnemonics displayed to the user. With **MudBlazor,** pagination and proper layout of the page were relatively trivial to achieve. Moreover, **Blazor** itself handles scaling of the application for different displays. Here is an example of a list of mnemonics on a smartphone.

A screenshot of a medical list

Description automatically generated

**Figure 20** Rendered list example

6. Typical Usage Scenario

1. The user logs in as a **Teacher**.

A screenshot of a login form

Description automatically generated

**Figure 21** Typical Usage Step 1

2. Then adds his/her **department** and necessary **categories** of mnemonics.  
A screenshot of a computer

Description automatically generated

**Figure 22** Typical Usage Step 2

A screenshot of a computer

Description automatically generated

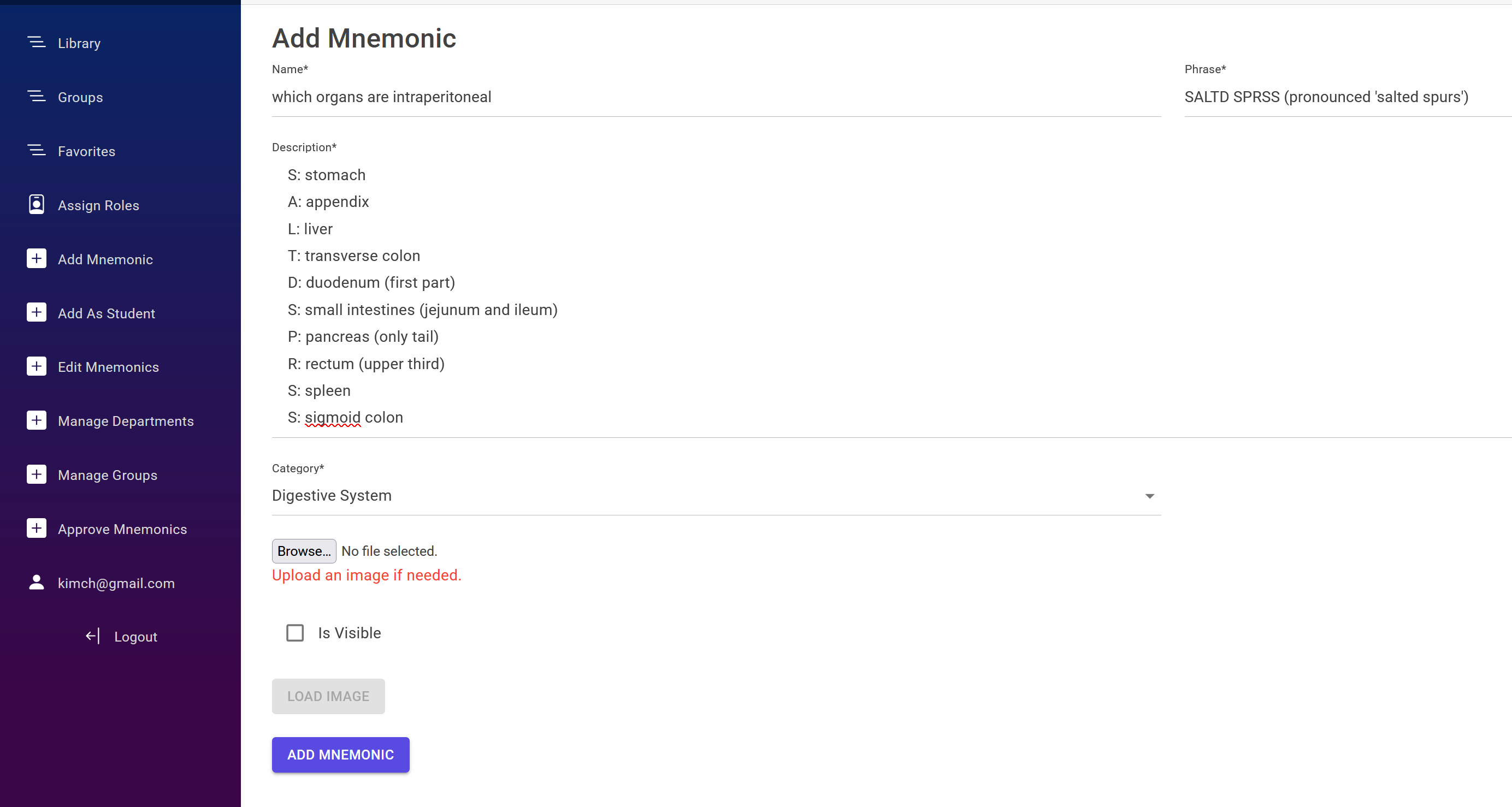
**Figure 23** Typical Usage Step 2

A white rectangular object with a blue line

Description automatically generated

**Figure 24** Typical Usage Step 2

3. Add a **mnemonic** to the **category**.



**Figure 25** Typical Usage Step 3

4. Add a **group** of students.

A screenshot of a computer

Description automatically generated

**Figure 26** Typical Usage Step 4

5. Add **mnemonics** to the group.

A screenshot of a computer

Description automatically generated

**Figure 27** Typical Usage Step 5

6. Join the **group**

A white rectangular object with purple lines

Description automatically generated

**Figure 28** Typical Usage Step 6

7. Select the **group** and revise mnemonics if necessary.

A close-up of a white background

Description automatically generated

**Figure 29** Typical Usage Step 7

8. Solve a set of **flashcards.**

A screenshot of a computer

Description automatically generated

**Figure 30** Typical Usage Step 8

9. View the **results.**

A white background with black and orange text

Description automatically generated

**Figure 31** Typical Usage Step 9

# 7. Testing Phase

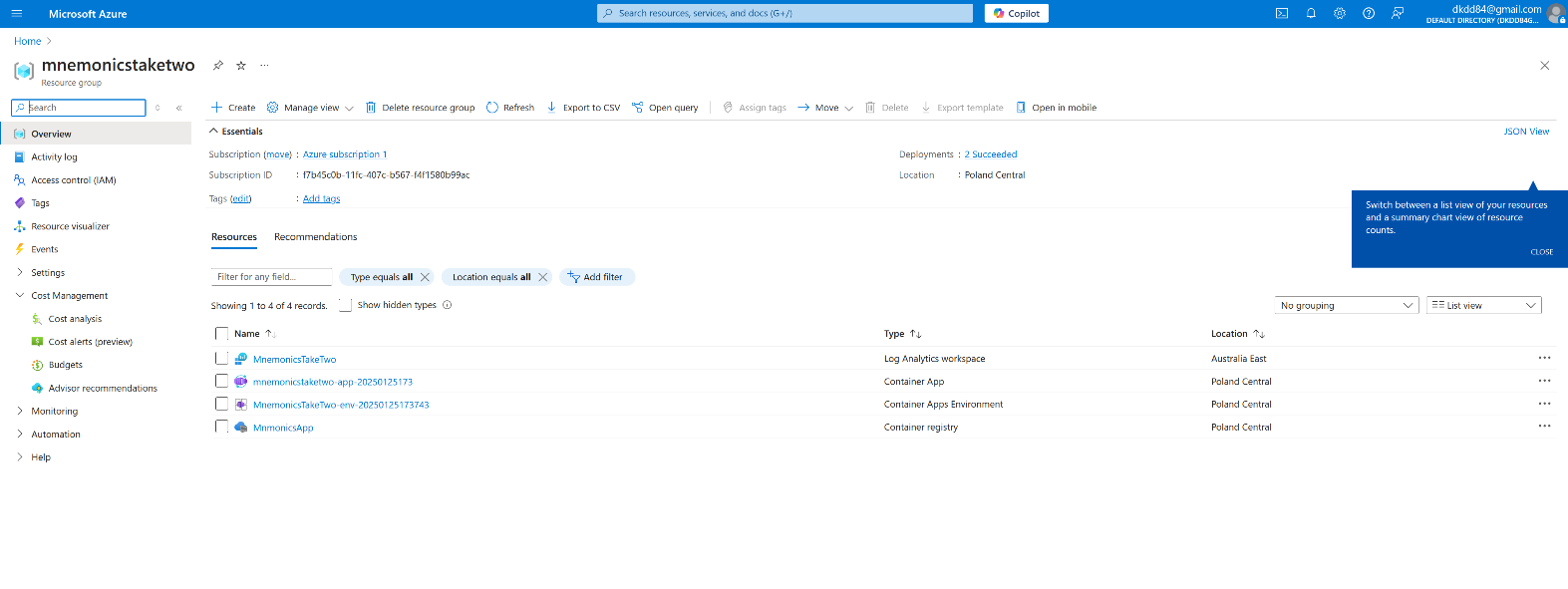
7.1. Overview  
  
In order to test the application, it was deployed to **Azure**, using built-in features of **Visual Studio** for Publishing. A dedicated MSSQL database server has also been deployed using **Azure SQL**.

The container image was built using .**NET SDK**.

A screenshot of a computer

Description automatically generated

**Figure 32** Cloud Deployment Configuration



**Figure 33** Cloud Deployment Registry

The test included 6 volunteers chosen among classmates.

After successful deployment a link to the website was sent to them.

The participants were given sets of instructions, which are written down in “Test Steps” column of test cases.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Test Case ID** | **Scenario** | **Steps** | **Prerequisites** | **Data** | **Expected Result** | **Actual Result** | **Status** |
| TC001 | Login into dashboard | 1. Navigate to Login Section. 2. Enter Email and Password. 3. Optionally, check "Remember Me". 4. Click "Log In". | User is registered. | Email: adminuser@gmail.com Password: UserPassword1! | Successful login navigates the user to the Dashboard. | 6/6 correct outcomes | Passed |
| TC002 | Login Failure with Incorrect Credentials | 1. Navigate to Login Section. 2. Enter incorrect email or password. 3. Click "Log In". | One of the used accounts is registered | Email: wrongemail@gmail.com Password: WrongPassword!  Email: adminuser@gmail.com Password: WrongPassword! | Error message, indicating that credentials are incorrect | 6/6 expected outcomes | Passed |
| TC003 | Registering New User | 1. Navigate to Register Section. 2. Fill out registration form. 3. Submit the form by clicking "Register". 4. Confirm registration | None. | Email: newuser@gmail.com Password: NewUser123! Confirm Password: NewUser123! | User account created. User can Login. | 6/6 correct outcomes | Passed |

## 7.2. Test Cases

Test cases were developed with main functionalities in mind. Each tester did them both in a desktop browser and a mobile browser.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| TC004 | Join group by code | 1. Navigate to Groups Section. 2. Click "Join Group". 3. Enter valid group code. 4. Click "Join". | User is logged in, group exists, groups code is assigned to the group | Group Code: Q8UDWZT3 | User successfully joins the group | 6/6 correct outcomes | Passed |
| TC005 | View Mnemonics in Group | 1. Navigate to Groups Section. 2. Click "View Mnemonics". 3. Select a mnemonic and click "View". | User is a group member, and mnemonics exist in the group. | Mnemonic: Title: "Mnemonic 1", Description: "Details about Mnemonic 1" | Mnemonic is displayed | 6/6 correct outcomes | Passed |
| TC006 | Solve Flashcards in Group | 1. Navigate to Groups Section. 2. Click "Flashcards". 3. Interact with flashcards using “Show description”, “I Knew That” or “I Didn’t Know That”. 4. Click “Finish”. | User is a group member, and flashcards exist in the group. |  | Flashcards are displayed with Name, Phrase, Image. Description Is shown when “Show description” is pressed. Displays a results screen showing a summary of responses and mnemonic details. | 6/6 correct outcomes | Passed |
| TC007 | Manage Roles as Admin | 1. Navigate to Assign Roles section. 2. Click "Manage Roles" for a user. 3. Assign, remove roles, or delete the user. | Users have admin privileges. | Email: adminuser@gmail.com Password: UserPassword1! | User's roles are updated, or user is deleted as expected. | 6/6 correct outcomes | Passed |
| TC008 | Add a New Department as Admin | 1. Navigate to Manage Departments. 2. Click "Add Department". 3. Enter department name and save. | Users have admin ore teacher privileges. | Department Name: New Department | New departments are created and displayed in the list of departments | 6/6 correct outcomes | Passed |
| TC009 | Add a Mnemonic as A Student | 1. Navigate to Add Mnemonic as Student section. 2. Select department. 3. Select a teacher 4. Fill out details and submit. | User is logged in | Name: New Mnemonic, Description: Details, Phrase: Example Phrase | Proposed mnemonic is displayed on the list of requested mnemonics in the account of the teacher. | 6/6 correct outcomes | Passed |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| TC010 | Approve Mnemonic | 1. Navigate to Approve Mnemonics section. 2. Open a mnemonic request. 3. Review, edit, set “is visible” to true and click "Approve" or "Reject". | Users have admin or teacher privileges, and mnemonic requests exist. | Mnemonic Name: Mnemonic 1, Description: Sample Description. | Mnemonic is approved, becomes visible, or is rejected. | 6/6 expected outcomes | Passed |
| TC011 | Add mnemonic to Favorites | 1. Navigate to Mnemonics in a selected Group. 2 Click “View Mnemonic”. 3. When the mnemonic is displayed, click “Add to favorites”. 4. Navigate to “Favorites” section. | User is logged in; the mnemonic and group exist | Mnemonic Name: Mnemonic 1  Group Name: Group 1 | The mnemonic is in the favorites list. | 6/6 correct outcomes | Passed |

## 7.3. Feedback

Although every written test case was passed successfully, testers were asked which improvements they would like to see in the future as users. A set of questions was asked when the testing was over.

|  |  |
| --- | --- |
| **Question** | **Summary** |
| How did you like the general esthetics of the application? | Testers noted that minimalistic layout, chosen colors, shapes of interface elements, but to make a better impression the app should have some animations when it comes to flashcards and mnemonics. One of the testers stated that he would like to have the ability to apply “Dark theme”. |
| In your opinion, which other functionalities should be present in the application? | Three out of six testers stated that they would like to have the ability to group favorite mnemonics and sort the mnemonics list in the library. One of the testers noted that “a search bar would be nice to have”. Another tester stated that sometimes a concept requires explanation, which is why video support could prove to be useful. |
| Would you use that kind of application in your studies? | Most of the testers stated that the application could be very useful to them when it comes to subjects that require a lot of reading and memorizing. |
| Did you find using the application via web browser convenient? | Four out of six testers stated that it seems very convenient since it is available on any device that has a web browser, however two testers noted that they would like to have an ability to view mnemonics without having to access the internet. |

To avoid possible confusions in the navigation, a **User Manual** was developed, which you may find in the **annex** to the thesis.

# 8. Possible Improvements

Based on the feedback we concluded that several steps could be taken to make the application more available and convenient:

1. **Improve esthetics**: add animations to flashcards, improve layouts so the navigation is more intuitive.
2. **Improve navigation**: add sorting and grouping options for mnemonics, add a search bar for the lists of Departments, Categories, Mnemonics and Groups.
3. **Add customization options**: allow the educational organizations to choose from a variety of options that would make the application more suitable for their needs. Such options might include adding a logo of the organization on the main page and navigation bar, choosing the color scheme, allowing the organization to organize data in a more flexible way.
4. **Add video playback:** allow the teachers to upload videos with explanations of concepts that are harder to understand.
5. **Switch to cloud file storge:** storing large files in a database would affect performance, which is why changing the way images and videos are stored is a necessity.
6. **Develop a standalone application:** it would allow the users to use the application without an internet connection, browse mnemonics whenever possible and download new data only if needed.

9. Responsibilities Of Team Memebers

|  |  |  |
| --- | --- | --- |
| **Responsibilty** | **Tools used** | **Team member** |
| Thesis Introduction | Microsoft Word | Leonid Kim, Daryna Drabysheuskaya |
| Collecting requirements |  | Leonid Kim, Daryna Drabysheuskaya |
| Specifying functional requirements | Microsoft Word | Daryna Drabysheuskaya, Leonid Kim |
| Specifying non-functional requirements | Microsoft Word | Leonid Kim |
| Use Case Diagram | Lucidchart | Daryna Drabysheuskaya, Leonid Kim |
| Use Case Scenarios | Microsoft Word | Daryna Drabysheuskaya |
| Initial mock-up | Figma | Daryna Drabysheuskaya |
| Database Schema | Vertabelo | Leonid Kim |
| Schema description | Microsoft Word | Leonid Kim, Daryna Drabysheuskaya |
| Selecting technologies |  | Leonid Kim, Daryna Drabysheuskaya |
| Comparing technologies |  | Daryna Drabysheuskaya, Leonid Kim |
| Data models development | Entity Framework, .NET, Visual Studio | Leonid Kim, Daryna Drabysheuskaya |
| Services development | .NET | Leonid Kim |
| Markup development | .NET, Blazor, MudBlazor | Daryna Drabysheuskaya |
| Component code development | .NET, Blazor | Leonid Kim, Daryna Drabysheuskaya |
| Cloud Deployment for testing | Microsoft Azure | Leonid Kim |
| Test cases | Microsoft Word | Leonid Kim |
| Testing overseeing |  | Daryna Drabysheuskaya |
| Feedback collection | Microsoft Word | Daryna Drabysheuskaya |
| Possible improvements | Microsoft Word | Leonid Kim |
| User manual | Microsoft Word | Daryna Drabysheuskaya |

# Bibliography

**1 Microsoft DevBlogs. (2023).** "Performance: Dynamic Profile-Guided Optimization and AVX-512 support enhance execution speed." Retrieved on 24.01.2025, from <https://devblogs.microsoft.com/dotnet/announcing-dotnet-8/>

**2 Microsoft .NET Platform. (2023).** "Cross-Platform: Unified development across Windows, macOS, and Linux." Retrieved on 24.01.2025, from <https://dotnet.microsoft.com/en-us/platform/why-choose-dotnet>

**3 Vaughan-Nichols, S. J.** **(2023).** "Vendor Lock-In: Heavy reliance on Microsoft's ecosystem." Retrieved on 24.01.2025, from <https://www.infoworld.com/article/2335395/whats-new-and-improved-in-microsofts-net-8.html>

**4 Microsoft Learn. (2023).** "Complexity: Advanced features may increase the learning curve." Retrieved on 24.01.2025, from <https://learn.microsoft.com/en-us/dotnet/fundamentals/>.

**5 Spring.io. (2023).** "Performance: Mature JVM-based optimizations." Retrieved on 24.01.2025, from <https://spring.io/projects/spring-boot>.

**6 Spring.io. (2023)**. "Cloud-Native: Built-in support for microservices with Spring Cloud." Retrieved on 24.01.2025, from <https://spring.io/projects/spring-cloud>.

**7 Spring.io. (2023).** "Wide Adoption: A large ecosystem with enterprise-grade applications." Retrieved on 24.01.2025, from <https://spring.io/>.

**8 Node.js. (2023).** "Event-Driven: Asynchronous I/O for high-performance, non-blocking servers; Scalability: Excellent for lightweight microservices; Callback Hell: Can result in hard-to-maintain code if not handled properly; Single-Threaded: May struggle with CPU-intensive operations." Retrieved on 24.01.2025, from <https://nodejs.org/en>.

**9 Microsoft .NET. (2023).** "Full-Stack C#: Unified language across client and server; Seamless Integration with .NET Ecosystem: Considering the selected framework, it is a good choice for frontend development." Retrieved on 24.01.2025, from <https://dotnet.microsoft.com/en-us/apps/aspnet/web-apps/blazor>.

**10 McAllister, N. (2023).** "Initial Load Time: Slower load times compared to React or Angular." Retrieved on 24.01.2025, from <https://www.infoworld.com/article/2268533/how-to-run-net-apps-in-the-browser-with-blazor.html>.

**11 Flatirons. (2023).** "Runtime Environment: Compared to React, the latter uses the browser’s JavaScript engine (which is an industry standard), while Blazor relies on WebAssembly for C# execution." Retrieved on 24.01.2025, from <https://flatirons.com/blog/blazor-vs-react/>.

**12 React.dev. (2023).** "Component-Based: Encourages modular and reusable components." Retrieved on 24.01.2025, from <https://react.dev/>.

**13 Angular.dev. (2023).** "Full-Featured Framework: Built-in solutions for routing and state management; TypeScript Support: Enhances maintainability and code quality; Learning Curve: More complex compared to Blazor or React; Bundle Size: Larger bundle sizes compared to React." Retrieved on 24.01.2025, from <https://angular.dev/>.

**14 Microsoft Learn. (2023).** "Simplified Data Access: Allows interaction with databases using .NET objects, eliminating the need for extensive SQL queries; Cross-Platform Compatibility: Supports Windows, macOS, and Linux environments." Retrieved on 24.01.2025, from <https://learn.microsoft.com/en-us/ef/core/>.

**15 Microsoft Learn. (2023).** "Code First Approach: Enables schema generation from .NET classes, promoting flexibility in model design." Retrieved on 24.01.2025, from <https://learn.microsoft.com/en-us/ef/core/modeling/>.

**16 Microsoft Learn. (2023).** "Automatic Migrations: Simplifies schema changes and version control alongside application code." Retrieved on 24.01.2025, from <https://learn.microsoft.com/en-us/ef/core/managing-schemas/migrations/?tabs=dotnet-core-cli>.

**17 Microsoft Learn. (2023).** "Asynchronous Methods: Built-in async query and save support improves app responsiveness." Retrieved on 24.01.2025, from <https://learn.microsoft.com/en-us/ef/core/miscellaneous/async>.

**18 Luxwisp. (2023).** "Performance Overhead: Slightly slower for complex queries compared to raw SQL or micro ORMs like Dapper." Retrieved on 24.01.2025, from <https://www.luxwisp.com/pros-and-cons-of-entity-framework/>.

**19 C# Corner. (2023).** "Complex Learning Curve: Advanced features like custom mappings can be challenging." Retrieved on 24.01.2025, from <https://www.c-sharpcorner.com/blogs/understanding-entity-framework-and-entity-framework-core>.

**20 Dapper Tutorial. (2023).** "High Performance: Faster than EF Core due to minimal abstraction overhead; Flexibility: Supports raw SQL queries, offering more control for complex database operations; Lightweight: Simple to integrate with projects, especially for apps requiring optimized database access." Retrieved on 24.01.2025, from <https://dapper-tutorial.net/>.

**21 Virtual DBA. (2023).** "Advantages: Scalability and Performance; Security; Integration; Various Editions Available." Retrieved on 24.01.2025, from <https://virtual-dba.com/blog/microsoft-sql-server-advantages-and-best-practices/>.

**22 LearnSQL. (2023).** "Disadvantages: Cost." Retrieved on 24.01.2025, from <https://learnsql.com/blog/microsoft-sql-server-pros-and-cons/>.

**23 Licendi. (2023).** "Configuration Complexity: Setting up MSSQL may be challenging for beginners." Retrieved on 24.01.2025, from [https://licendi.com/en/blog/microsoft-sql-server-advantages-and-disadvantages%20/](https://licendi.com/en/blog/microsoft-sql-server-advantages-and-disadvantages%2520/).

**24 MongoDB. (2023).** "Advantages: Flexible Schema; Horizontal Scalability." Retrieved on 24.01.2025, from <https://www.mongodb.com/>.

**25** **MongoDB. (2023).** "Document-Oriented Storage; Disadvantages: Transaction Support; Indexing Challenges." Retrieved on 24.01.2025, from <https://www.mongodb.com/resources/basics/databases/nosql-explained>.

**26 Patel, R. (2023).** "A Beginner's Guide to ASP.NET Core Identity for Authentication and Authorization with .NET 8." Retrieved on 24.01.2025, from <https://medium.com/@ravipatel.it/a-beginners-guide-to-asp-net-core-identity-for-authentication-and-authorization-with-net-8-e6c8deb612f4>

**27 Reddit Users. (2020).** "Learning Curve: According to the feedback provided by .NET developers, ASP.NET Core Identity functionalities." Retrieved on 24.01.2025, from <https://www.reddit.com/r/dotnet/comments/huu40f/anyone_using_identity_on_aspnet_core_if_so/>.

**28 Stack Overflow Users. (2014).** "Documentation and support: Some developers have noted that the documentation and support for ASP.NET Core Identity can be lacking." Retrieved on 24.01.2025, from <https://stackoverflow.com/questions/21207246/asp-net-identity-vs-simple-membership-pros-and-cons>.

**29 Auth0. (2023).** "Ease of Setup: Straightforward implementation with Blazor WebAssembly; Comprehensive Features: Provides social login and multi-factor authentication; Scalability: Supports large user bases effectively; Third-Party Dependency: Adds reliance on external services for critical functionality." Retrieved on 24.01.2025, from <https://auth0.com/>.

**30 Firebase. (2023).** "Ease of Integration: SDKs for Blazor Web App available; Scalability: Works seamlessly with Firebase services for authentication and database; Flexibility: Supports passwordless and social logins; Limited Customization: Less flexible than AWS Cognito." Retrieved on 24.01.2025, from <https://firebase.google.com/docs/auth>.

**31 Firebase. (2023).** "Vendor Lock-In: Dependency on Firebase ecosystem." Retrieved on 24.01.2025, from <https://firebase.google.com/>.

# List Of Indexes

**.NET, 17**

**.NET SDK, 51**

**Angular, 29**

**ASP.NET Core Identity, 33**

**AVX-512, 28**

**AWS Cognito, 34**

**Azure, 34**

**Blazor, 29**

**Blazor Pipeline, 36**

**CPU, 29**

**CSR, 36**

**Dapper, 31**

**DOM, 30**

**EF Core, 31**

**Flashcard, 11**

**JavaScript, 29**

**JSX, 30**

**JVM, 28**

**Mnemonic, 11**

**MSSQL, 32**

**MudBlazor, 46**

**MVC, 36**

**React, 29**

**SSR, 36**

**Virtual DOM, 30**

**WebAssembly, 29**

# Table Of Figures

[**Figure 1** Use Case Diagram 16](#_Toc188971281)

[**Figure 2** Database Schema 24](#_Toc188971282)

[**Figure 3** Zoomed In Schema I 25](#_Toc188971283)

[**Figure 4** Zoomed In Schema II3.4. Description of the schema 26](#_Toc188971284)

[**Figure 5** Context Diagram 35](#_Toc188971285)

[**Figure 6** Project Structure 37](#_Toc188971286)

[**Figure 7** Data Model (Mnemonic) 38](#_Toc188971287)

[**Figure 8** Data Model (RequestedMnemonic) 39](#_Toc188971288)

[**Figure 9** Application Database Context 40](#_Toc188971289)

[**Figure 10** Mnemonic Service Interface 41](#_Toc188971290)

[**Figure 11** Mnemonic Service Concrete Class 41](#_Toc188971291)

[**Figure 12** Service Collection Extensions 42](#_Toc188971292)

[**Figure 13** Registering of Services in the Application 42](#_Toc188971293)

[**Figure 14** Imports and injections in a Component 43](#_Toc188971294)

[**Figure 15** Razor markup I 43](#_Toc188971295)

[**Figure 16** Razor markup II 44](#_Toc188971296)

[**Figure 17** Code example in a Component I 44](#_Toc188971297)

[**Figure 18** Code example in a component II 45](#_Toc188971298)

[**Figure 19** Rendered page example 45](#_Toc188971299)

[**Figure 20** Rendered list example 46](#_Toc188971300)

[**Figure 21** Typical Usage Step 1 47](#_Toc188971301)

[**Figure 22** Typical Usage Step 2 47](#_Toc188971302)

[**Figure 23** Typical Usage Step 2 47](#_Toc188971303)

[**Figure 24** Typical Usage Step 2 48](#_Toc188971304)

[**Figure 25** Typical Usage Step 3 48](#_Toc188971305)

[**Figure 26** Typical Usage Step 4 48](#_Toc188971306)

[**Figure 27** Typical Usage Step 5 49](#_Toc188971307)

[**Figure 28** Typical Usage Step 6 49](#_Toc188971308)

[**Figure 29** Typical Usage Step 7 49](#_Toc188971309)

[**Figure 30** Typical Usage Step 8 50](#_Toc188971310)

[**Figure 31** Typical Usage Step 9 50](#_Toc188971311)

[**Figure 32** Cloud Deployment Configuration 51](#_Toc188971312)

[**Figure 33** Cloud Deployment Registry 51](#_Toc188971313)

# Annex I

## User Manual

**- Login into Dashboard:**

**Main Flow**

1. **Click on Login Section → Navigation to Login Section**
2. **Enter Your Credentials:**

* Locate the **"Email"** field and enter your registered email address (e.g., [**adminuser@gmail.com**](mailto:adminuser@gmail.com) ).
* In the **"Password"** field, type your secure password (e.g., **UserPassword1!**)

***Optional: Remember Your Login:***

Check the **"Remember me"** box if you want to stay logged in on this device.

1. **Submit Your Login:**

* Click the **"Log in"** button to proceed.

***Successful Login  → Navigation to Dashboard***

***Incorrect Login  → Error Message***

**Optional Flow**

1. **Forgot Your Password?**

* Click button **"Reset password"** **→** Navigation to Forgot password screen
* Entering email  → Following received steps
* Follow the steps to reset your password

1. **Email Confirmation Issues?**

If you didn’t receive the email confirmation during registration, click **"Resend"** to trigger a new confirmation email.

1. **Clicking “Register as a new user” → Navigation to Register  Section**

**- Register:**

**Main Flow**

1. **Click on Register  Section → Navigation to Register Section** .
2. **Fill Out the Registration Form:**

The form prompts the user to enter the following:

* **Email:** Providing a valid email address .
* **Password:** Creating a secure password.
* **Confirm Password:** Re-entering the same password.

1. **Submit the Registration Form:**

* After filling in the required fields, clicking  the **"Register"** button to submit information.

***Successful Registration  → Creating account with saving it into db***

***Incorrect Registration  → Error Message***

1. **Click on Login Section → Navigation to Login Section**

**- Dashboard:**

User flow for main functionalities:

**- Groups Section**

*Available to users.*

**Key Features in the Groups Section:**

1. **Join Group by Code:**

Enables users to join an existing group using a unique group code shared by the group administrator.

1. Click on "**Join Group"**
2. Enter the provided group code (e.g., Q8UDWZT3).
3. Click **"Join"** to become a member of the group.

**- View existing groups with extended functionalities:**

**2.    View Mnemonics:**

Provides access to mnemonics  shared within the group.

1. Clicking on button **“View Mnemonics”**
2. View mnemonics with title and description
3. Clicking **“View”** button to see full description
4. See full mnemonics description with **“Add to favorites”** button
5. **Add to favorites**  →  Corresponding mnemonic will add to **Favorites section**
6. Handling Back Functionality to previous screen

**3.    View Flashcards:**

Allows users to access flashcards shared within the group.

* **Access Flashcards:**

The user clicks on the **"Flashcards"** option in the group interface, directing them to the flashcards for the selected group.

* **View Flashcard:**

A flashcard appears with image only.

***Buttons:***

* **Previous**: To return to the previous flashcard.
* **Next**: To proceed to the next flashcard.
* **Show Description**: To reveal the mnemonic's short description above the picture.

1. **Respond to the Flashcard:**

The user interacts with the flashcard by choosing:

* **"I Knew That"**: Indicating user  knows the content of the mnemonic.
* **"I Didn’t Know That"**: Indicating user do not know the content.

1. **Complete the Session:**

After going through all the flashcards, the user clicks **"Finish"** to conclude the flashcard session.

**- Navigation to Results Screen after clicking buttons Finish, I knew that, i did not know that:**

Upon completing the flashcard session, the user is directed to a summary screen showing:

***A table with columns for:***

***Name***: The title of the mnemonic.

***Phrase:*** The mnemonic phrase shared in the flashcard.

***Description:*** Additional details or explanations of the mnemonics.

A Ratio of Answers displaying the number of correct or incorrect responses.

**- Favorites**

*Available to users*

**Key Features in Favorites  Section:**

1. **View Favorite Mnemonics:**

Provides users with a list of mnemonics they have marked as favorites for quick access and reference.

* **View a table** listing all favorited mnemonics with the following details:
* For each mnemonic, **two action buttons** are available:

**2.     View:**

Allows users to access mnemonic, which added to favorites,  full details.

**3.     Remove Favorites:**

Allows users to remove their favorite mnemonics.

**- Assign Roles**

*Available to administrators only.*

**Key Features in Assign Roles:**

1. **View Users:**

Provides an overview of all registered users:

**View the table containing:**

* Email Addresses: Each user's email for identification.
* Manage Roles Button: A button next to each email that allows administrators to modify the user's roles.

1. **Perform Role Actions:**
2. Clicks **“Manage Roles button”**
3. On the detailed role management page:

* Assign or remove roles by clicking the respective buttons.
* Delete a user if necessary.

1. Handling Back Functionality to previous screen.

**- Add As Student**

This form allows students to contribute new mnemonics to the system by filling in required details and uploading a corresponding image.

*Available to users.*

**Key Features in Add As Student:**

1. **Form Fields in Assign :**

* Name (Required):
* Title or identifier for the mnemonic.
* A detailed explanation or description of the mnemonic's purpose.
* The mnemonic phrase.
* The department or group the mnemonic is associated with.
* Assign an admin user responsible for reviewing the submission.

.

1. **File Upload (Required):**

Purpose: Attach a visual aid or representation of the mnemonic (e.g., a diagram or illustrative image).

* Select an image from the local system, and upload it.

***No image is uploaded →   A warning message is displayed***

**Buttons:**

* Load Image

Previews the uploaded image before submission.

* Add Mnemonic

Submits the completed form for review to the teacher selected.

**- Manage Departments**

This form allows students to add, edit and delete departments and also view existing categories for them with add, delete, edit functionality.

*Available to administrators and teachers only*

**Key Features in Manage Departments:**

1. **View Departments:**

*Displays a list of all departments within the system.*

Each department is shown in a table with the following:

* **Name:** The title of the department (e.g., "New Department," "Test Dep").
* **Actions:** Buttons for editing, deleting, and managing categories.

1. **Add Department:**

* Click the "Add Department" button at the bottom of the screen.
* Enter the department name in the prompted form.
* Save

1. **Edit Department:**

* Click the "Edit" button next to the desired department.
* Update the department name in the provided form.
* Save the changes.

1. **Delete Department:**

* Click the "Delete" button next to the department.
* Confirm the deletion in the pop-up or confirmation prompt.

1. **Manage Categories for selected department:**

* Click the "View Categories" button next to the department.
* **Buttons for Edit, Delete, Add Categories** are implemented with the logic and view, as for departments.

1. **Handling Back Functionality to previous screen.**

**- Manage Groups**

*Available to administrators and teachers only*

**Key Features in Manage Groups :**

1. **View Groups:**

Each group is shown in a table with:

* **Name:** The name of the group.
* **Description:** A brief description of the group.
* **Actions:** Buttons for editing and deleting the group.

1. **Add Group:**

* Click the **"Add Group"** button at the bottom of the screen.
* Enter name and description for the group and generate unique code for it  in the prompted form.
* Save the new group

*Students can add this group by entering selected code in Groups code.*

1. **Edit Group:**

* Click the "Edit" button next to the desired group.
* Update the group name, description, regenerate code in the form provided.
* Save the changes.

**Button “Add mnemonics”:**

You can add existing mnemonics to the selected group.

1. Choose the corresponding department.
2. Select existing mnemonics from the list to add them to this group and save changes.

**Button “Remove mnemonics”:**

You can remove existing mnemonics from the selected group.

1. Select existing mnemonics in this group from the list to remove them.
2. Save changes.

1. **Delete Group:**

* Click the "Delete" button next to the group.
* Confirm the deletion in the pop-up or confirmation prompt.

1. **Handling Back Functionality to previous screen.**

**- Approve Mnemonics**

*Available to administrators and teachers only*

This section allows administrators to review and approve mnemonics submitted by students. The approval process ensures that only high-quality and relevant mnemonics are added to the system.

**Key Features in Approve Mnemonics:**

1. **View Requested Mnemonics:**

* Each requested mnemonic is shown in a table with:
* Name: The title of the mnemonic.
* Phrase: The mnemonic phrase.
* Description: A short description explaining the mnemonic.
* Actions: An Open button to view the full details of the mnemonic.

1. **Open and Review Mnemonic:**

Allows administrators to review the mnemonic in detail before approving or rejecting it.

* Click the "Open" button next to a mnemonic in the list.
* The system navigates to the detailed view, which includes:

Name: The title of the mnemonic.

Phrase: The mnemonic phrase.

Description: A full explanation of the mnemonic.

Category: The category associated with the mnemonic.

Visibility Option: A checkbox to mark whether the mnemonic should be visible after approval.

Image: A visual aid uploaded with the mnemonic (if applicable).

1. **Edit Requested Mnemonic:**

Allows administrators to make changes to the mnemonic before approval.

* Update fields such as the name, description, phrase, or category.
* Modify the image if necessary by uploading a new file.
* Mark the "Is Visible" checkbox if the mnemonic is ready for display.

1. **Approve or Reject Mnemonic:**

* Review the mnemonic details and make edits if needed.

**Click:**

* "**Approve**" (green button): To accept the mnemonic and make it available in the system.
* "**Reject"** (red button): To decline the mnemonic, removing it from the pending list.

1. **Handling Back Functionality to previous screen.**

**- Edit Mnemonics**

*Available to administrators and teachers only*

This section allows administrators to manage mnemonics associated with specific departments, including editing their details and adjusting visibility settings.

**Key Features in Edit Mnemonics:**

1. **Select Department:**

* View a list of departments displayed as cards (e.g., "New Department," "Test Dep").
* Click the **"Select"** button on the department card to access the mnemonics specific to that department.

1. **View Mnemonics for Department:**

* A table lists the mnemonics with:

**Name:** Title of the mnemonic.

**Phrase:** Mnemonic phrase.

**Description:** A brief explanation of the mnemonic.

**Actions:** An **Edit** button for each mnemonic.

* Review the list of mnemonics for the department.
* Click the **"Edit"** button to modify a specific mnemonic.

1. **Edit Mnemonic:**

* The **Edit Mnemonic** form opens, showing:

**Name:** The title of the mnemonic.

**Phrase:** The mnemonic phrase.

**Description:** Detailed explanation of the mnemonic.

**Visibility:** A checkbox to mark whether the mnemonic should be visible in the system.

**Image:** An option to upload or change the associated image.

* Update any fields as needed.
* To replace the image:

Click **Choose File** to upload a new image.

Click **Load Image** to preview the image.

* Click **Save Changes** to confirm updates.

1. **Handling Back Functionality to previous screen.**

**- Add Mnemonics**

*Available to administrators and teachers only*

This section allows administrators to create new mnemonics for specific departments, ensuring that content is well-organized and accessible to users.

**Key Features in Add Mnemonics:**

1. **Select Department:**

* View a list of departments displayed as cards (e.g., "New Department," "Test Dep").
* Click the **"Select"** button on a department card to proceed with adding mnemonics to that department.

1. **Add Mnemonic Form:**

Allows administrators to input details for a new mnemonic.

* **Fields:**

**Name (Required):** Title of the mnemonic (e.g., "Order of Operations").

**Description (Required):** A detailed explanation or purpose of the mnemonic.

**Phrase (Required):** The mnemonic phrase (e.g., "PEMDAS: Please Excuse My Dear Aunt Sally").

**Category (Required):** A dropdown to select the appropriate category for the mnemonic.

**Image Upload (Required):**

**Functionality:** Use the **Choose File** button to upload an image, and **Load Image** to preview it.

**Visibility Option:** A checkbox to determine whether the mnemonic should be immediately visible to users.

* **Action Buttons:**

**Load Image:** Previews the uploaded image to ensure correctness before submission.

**Add Mnemonic:** Submits the form and saves the mnemonic to the system.

1. **Handling Back Functionality to previous screen.**

**- Library**

*Available to users.*

The Library section provides a centralized area for users and administrators to access and manage mnemonics, categories, and departments.

**Key Features in Library:**

1. **Departments:**

* View a list of departments displayed as buttons (e.g., "New Department," "Test Dep").
* Click a department button to access its associated categories and mnemonics.

1. **Categories:**

Manage and view categories within the selected department.

* View a list of categories  displayed as buttons.
* Click to the name of the chosen category to view mnemonics associated with the selected category.

1. **Mnemonics:**

Provide detailed access to mnemonics within a category or department.

* View a list of mnemonics displayed as buttons.
* View mnemonic details by clicking on a specific mnemonic title.

1. **Mnemonic Details:**

Display full details of a mnemonic.

* Features:

**Details:** Name, phrase, description, and an image associated with the mnemonic.

**Add to Favorites Button:** Add the mnemonic to the user's favorites list for quick access.

***When clicking “Add to Favorites”  → Mnemonic will display in Favorite Section***

1. **Handling Back Functionality to previous screen.**

**- Account (displaying as your logged mail , example : adminuser@gmail.com)**

*Available to users.*

The Profile Section allows users and admins to manage their account settings, update personal information, and enhance account security.

**Key Features in Account:**

**1. Profile**

Update the user’s basic account details.

* Navigate to the Profile tab.
* View the username (displaying as logged email, could not be edited ).
* Update or enter a phone number in the respective field.
* Click Save to apply changes.

**2. Email**

Allow users to update their email address.

* Navigate to the Email tab.
* Enter the new email address in the designated field.
* Click Change Email to update the account email.

**3. Password**

Enable users to update their account password securely.

* Navigate to the Password tab.
* Enter the old password, then type the new password and confirm it.
* Click Update Password to save the changes.

# Annex II

## Application screenshots

A screenshot of a computer

Description automatically generated

