Flashcards: Project 2

Part of the Examination Assessment of
ITPE3200 Web Applications at OsloMet

30.11.2023

Documentation

1 Introduction

The group recreated the MVP of the flashcards web application this time using Angular (version 15.2.7) on the front-end and .NET Core 7.0 on the back-end. The front-end utilizes HTML, TypeScript and CSS, while the back-end is written in C#. The group members use different versions of Node.js locally, such as version 18.17.1. The group decided to focus on the Angular framework and SPA features in project 2, because these were not part of project 1 and unlocks many new capabilities. The public subjects and decks from project 1 were included in project 2 as well. Two of these decks of cards had questions and answers generated by ChatGPT (OpenAI, 2023).

1.1 TypeScript

TypeScript adds syntax for types to JavaScript (*JavaScript With Syntax for Types.* , n.d.). A main feature is static typing, which means that the compiler checks for data types of variables, parameters, and return values, and catches potential errors in the code that may be present at compile time. Furthermore, it facilitates concepts of Object-Oriented Programming, such as classes and inheritance (Gillis, n.d.).

1.2 Angular

Angular is an open-source web application framework created by Google. It is used for developing single-page web applications (SPAs) that require dynamic content handling. It has modular features to enhance maintainability of the code. Key features include: Component-Based Architecture, Two-Way Data Binding, Directives, Dependency Injection, Services, Routing, Forms, and RxJS Integration (*Angular Features* , 2023).

Component-Based Architecture refers to independent components responsible for a certain part of the user interface. RxJS Integration describes how Angular uses Reactive Extensions for JavaScript (RxJS) to manage asynchronous tasks through Observables (*RxJS* , n.d.).

1.3 Single-Page Application

Single-page applications (SPAs) are web applications or websites that dynamically rewrite one page rather than loading different pages from a server. Content is fetched as JSON and the DOM structure is updated with JavaScript without reloading the page. Two main features include client-side rendering and routing. Client-side rendering means that the HTML is rendered primarily on the client side, by the browser, through libraries and frameworks, rather than by the server. SPAs utilizes client-side routing to navigate the application. Even though the browser's URL is updated, the page does not get reloaded.

1.4 JSON

JavaScript Object Notation (JSON) is a text format that is language-independent and used to read, store, and write data. JSON data is structured into key-value pairs with a few universal

types. Due to the lightweight-nature of text-formatted data, JSON is used extensively through the Angular framework.

For instance, Model classes are serialized into JsonObjects that contain multiple JsonProperty items. These JsonObjects can then be fetched using HTTP requests.

1.5 CRUD

Our project has a generic CRUD (Create, Read, Update, Delete) base service for API interaction. The purpose is to provide a structure for CRUD operations in our application using RxJS and HttpClient. RxJS operators such as 'Observable' are imported for reactive programming.

This class is designed to be further developed with additional services tailored to specific data types. We made use of the Typescript language features and created an abstract generic class called crudBase. This class implements basic CRUD operations, as well as some complex logic to mark data as stale to prompt refetching. When extending from CrudBase, a service inherits the fundamental CRUD operations simply by providing the super class with a resource endpoint such as /api/decks, simplifying the implementation of these tasks with a minimal need for additional code.

2 Coding Details

2.1 User Experience and Design

The application uses Bootstrap (*Bootstrap* , n.d.). for styling the templates, and utilizes a basic design. The group made use of Bootstrap's breakpoints to ensure friendly navigation on both mobile and desktop screen sizes. The landing page illustrates the breakpoints in action. The landing page contains a dynamic button with content that changes based on whether the user is logged in or not. On figure 1, the text "Welcome Back" is shown to logged in users, while on figure 2, the text "Get Started" is displayed to unauthorized users. Additionally, the frontpage has a carousel with four slides. The user can navigate by pressing anywhere on the sides of the images by the arrows, or by clicking directly on the dot indicators.

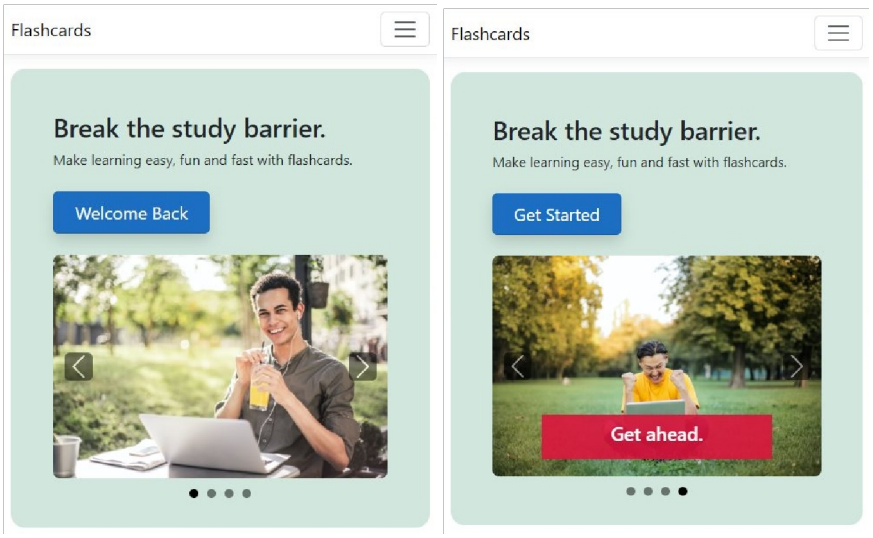


Figure 1 and 2. The landing page on a small screen size (sm).

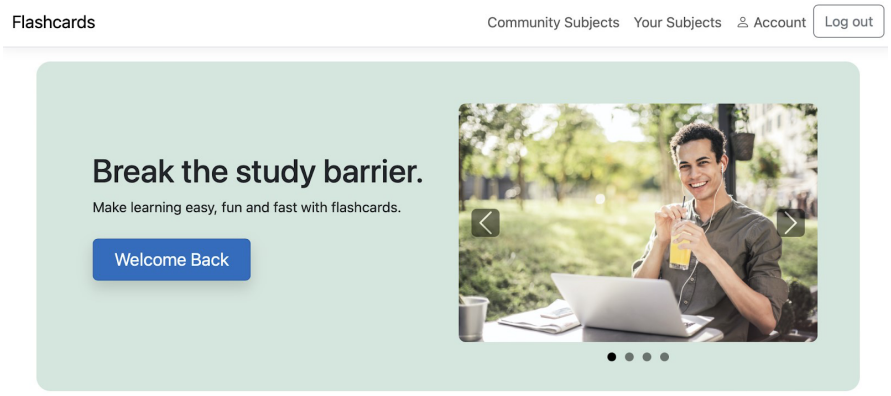


Figure 3. The landing page on a large screen size (lg). The email has been partly cropped.

Users not logged in will be routed to the Login page when pressing the “Get Started” button on the landing page. This page has been auto-generated by Microsoft.AspNetCore.Identity.UI. Duende.IdentityServer is used to generate the identity API, because that was the default when using the dotnet command line interface to scaffold the project using the latest angular template with authentication.

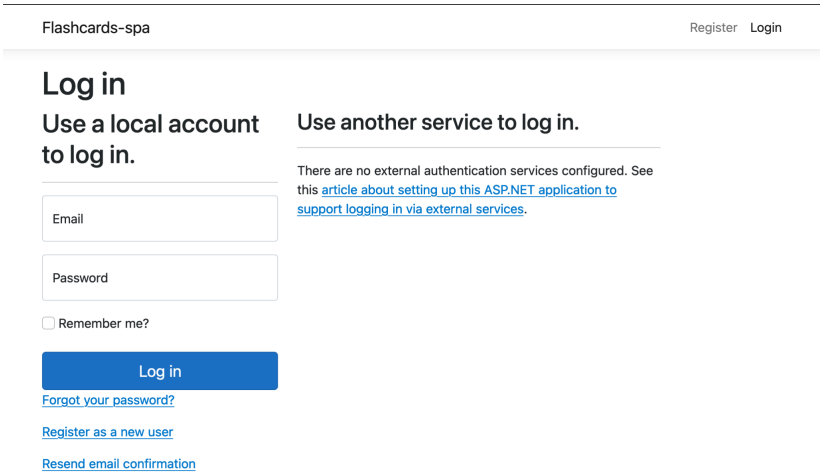


Figure 4. Login page.

2.2 Service Injection

The application uses primarily two types of injected services: services for server communication and services for component communication.. AuthorizeService was generated by the template, and is used to provide components and other classes data on the user currently logged in - or whether the user is unauthorized.

The services that inherit from crudBase have the responsibility of fetching data. They each correspond to one data model that exists on the backend. crudBase also has a method for components to communicate that some data has been changed. The method markStale(id?: number) can be called from any component and tells the service to refetch data. This makes it so that data is re-fetched when the user performs actions such as creating a card or editing a subject description. Enabling this kind of reactivity was one of our main focuses for this assignment as this is the biggest difference between a SPA and a multi page application. Because this is an MVP we have not considered optimizing fetch requests, and we are aware there is a lot of over-fetching to ensure that the UI is responsive.

The services modal.service and practice.service don't make requests to the server but enable component communication and storing data locally. We implemented a modal component that renders different components based on method calls to the modal service. This is used throughout the app, and enables us to quickly add a form to be shown as a dialog without having to manage data-bs properties and id's. It also reduces the amount of markup that is

sent to the client. The practice controller stores a practice session in localstorage and enables the user to practice a deck without having to rely on network requests.

2.3 Forms and Input Validation

The application uses forms and input validation for CRUD operations. For instance, SubjectCreateFormComponent defines a FormGroup that includes several properties with their respective Validators. One such property is Name, which contains four Validators: Required, minimum length, max length, and pattern of accepted symbols. On the template, the validators, or form controls, dynamically provide feedback on invalid inputs, as shown on figure 5 and 6.

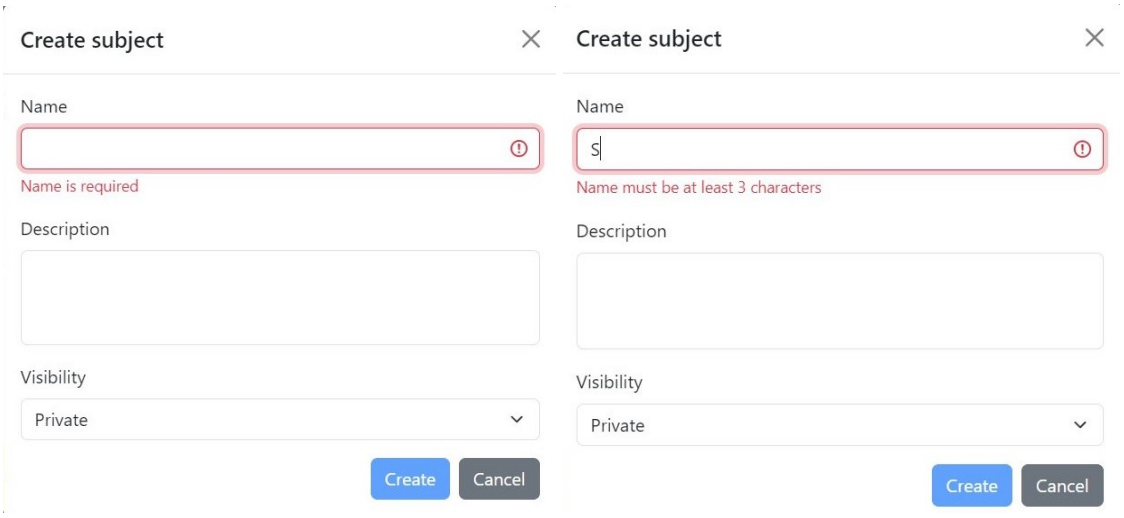


Figure 5 and 6. Input validation in action for the Create Subject form.

2.4 Content Filtering

The application has content filters on the Subject pages that filters on the subject names. For instance, in the case of Public Subjects, subjects in the observable subjects are piped, mapped and filtered by the input value of subjectNameFilter. The result is then passed on to the method filterSubjects in SubjectService, which can be used for both Public and Private Subjects. Finally, if subjectNameFilter is defined, it does an asynchronous for-loop with *ngFor to include all of the filtered subjects on the template.

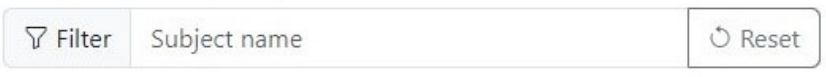


Figure 7. The filter bar on the Subjects pages.

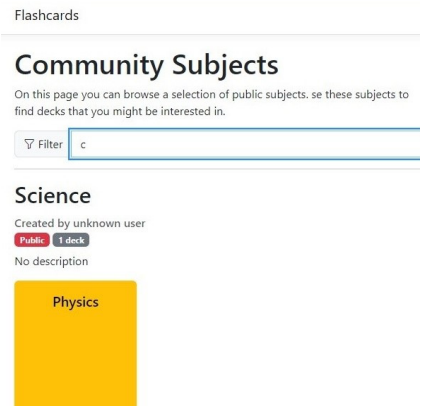


Figure 8. Public subjects filtered by the letter "c".

2.5 Unit Testing

Unit testing is used to verify pieces of code to ensure that the code is implemented correctly. Unit testing can be structured in three phases: arrange, act and assert. Arrange is where the objects, conditions etc. that are needed for the test are set up. This is also where we can create mock instances using moq library, so the mock instances can be called during the test. Act is the phase where the code/method we are testing will be executed. The last phase, assert, is where the result must be as expected so the unit testing can pass successfully. In this project we initially created six unit tests - including positive and negative tests for each CRUD method in the SubjectController, but after discovering that none of the tests were working as intended, we only had time to implement two working unit tests for the methods in SubjectController not requiring authorization.

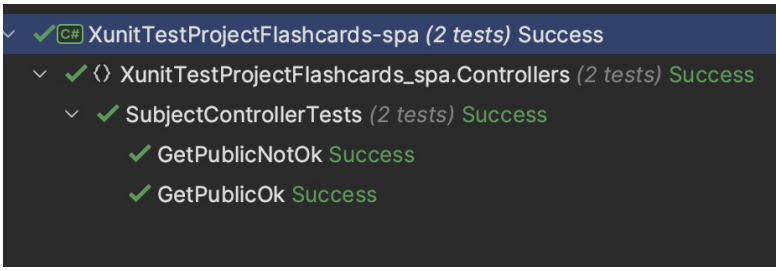


Figure 9. Running tests.

2.6 Custom Pipes

Angular Pipes are designed to modify how the data gets presented to the user; they don't transform the data itself but change how it is displayed. Pipes are simply functions designed to take an input value, process it, and return a converted value as output.

During development we found the need for a custom pipe when counting objects in a list. Instead of having to check for null, get the length, and then write out the wording two times using a ternary operator, we could abstract away this behavior using our custom pipe "count".

We used the PipeTransform interface to perform a transformation into the desired format. Angular calls the transformation model, where the value of a binding is the first argument, and any parameters follow as the second argument in the form of a list.

We also make heavy use of the async pipe. This enables us to automatically subscribe and unsubscribe to observables in the templates thus reducing the need to manually manage subscriptions.

2.7 RESTful API

We developed the server application as a Representational State Transfer API (REST API) to enable simple data exchange between the client and server sides. The guiding principles of REST API are used to ensure a robust and uniform interface (*What Is REST* , 2023).

2.7.1 Uniform interface

Four types of standardized methods are used to interact with resources identified via URIs (Uniform Resource Identifiers):

- GET: Utilized for retrieving information.
- POST: Utilized for data preservation.
- PUT: Utilized for replacing pre-existing data.
- DELETE: Utilized for removing pre-existing data

2.7.2 Client-Server Architecture

The client and the server entities are distinctly separated. The server responds with data or actions to requests sent from the client side and handles them. We created Typescript interfaces corresponding to the data types returned from the server. This bridges the gap between the client and the server and makes it feel like a typesafe layer even though it's not. Ideally we would have some sort of contract between the server and client deciding the communication interface, but for this project we maintained this manually.

2.7.3 Layered System

The architecture is structured into multiple layers, each assigned a specific role or responsibility, which leads to a better separation of concerns and improves scalability. We have not made many changes to the original backend of our project and we still have a data access layer, one controller for each entity and an authorization layer.

2.8 Swagger

Swagger is commonly used for documenting REST APIs. We activated Swagger and SwaggerUI for development mode to test API requests directly through the browser. We used the instructions provided in a guide by (Nguyen, 2023) as a reference to implement bearer authentication on the swagger page.

Run the project and go to page https://localhost:44437/swagger . You will now see the application APIS and the authorize button on the right side. To authorize you need the bearer token from one of the requests made after you log in. Log in, and then use the web inspector to find your Token. You can now authorize your user on the swagger page.

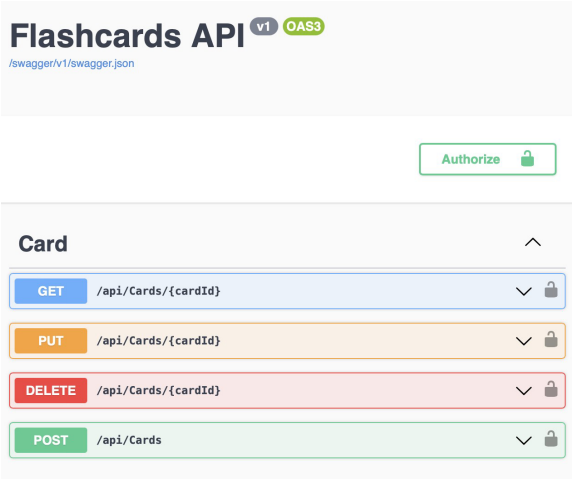
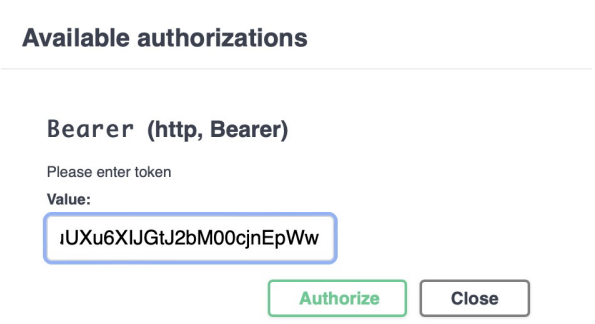


Figure 10. Swagger UI.



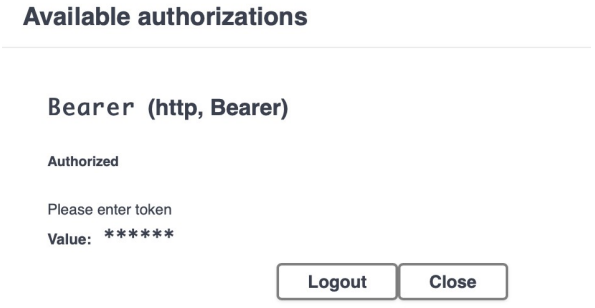


Figure 11 and 12. Authorization.

After authorization, you should be able to test even the protected Swagger endpoints. The swagger interface is helpful when manually testing the backend api before the frontend UI is finished.

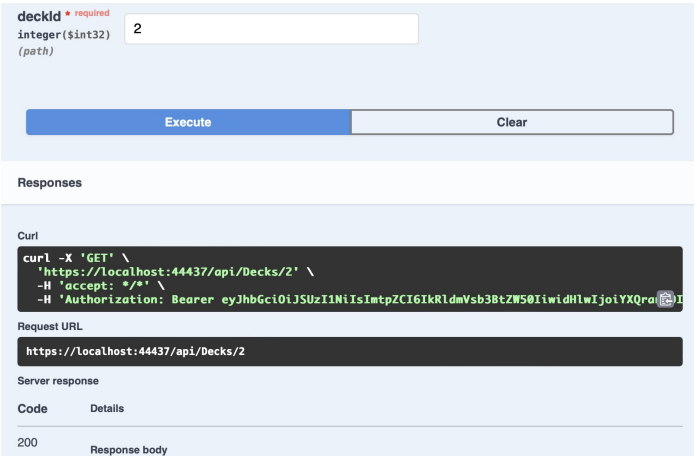


Figure 13. Making a request.

2.9 Error Handling and Logging

We used the same error method as in project 1 with small changes, and focused on having API-friendly error handling. Our application primarily manages error handling using HTTP status codes such as NotFound, Forbid, and various StatusCode responses. This method is most seen in RESTful API services, where direct HTTP responses are essential for client-server communication. When comparing this with Project 1, we notice an additional layer in error handling. Project 1, apart from utilizing HTTP status codes, also employs ViewData and TempData to convey error messages and codes.

We have implemented a try-catch block to detect and identify any exceptions under code execution in a try block in our controllers. Once an exception happens, a HTTP status 500 is provided, which indicates an error that occurred on the server side. Formatting and logging the exception message for the specific controller action, also takes place. The exception message represents errors during code execution in a try block.

We have also implemented various checks for null statements and errors that can occur during try block in a controller, and correctly log them.

3 Conclusion

The purpose of project 2 was to create a MVP of an Angular/.NET single-page application. The group has demonstrated the usage of multiple Angular features, and dynamic content handling.

The group has learned about the Angular framework and how to utilize JSON data to build a SPA. This knowledge is of great value to the group for future opportunities in web development.

4 Reference List

(n.d.). TypeScript: JavaScript With Syntax For Types. Retrieved November 30, 2023, from https://www.typescriptlang.org/

Angular Features . (2023, August 13). Coding Ninjas. Retrieved November 30, 2023, from https://www.codingninjas.com/studio/library/angular-features

Bootstrap . (n.d.). Bootstrap · The most popular HTML, CSS, and JS library in the world. Retrieved November 30, 2023, from https://getbootstrap.com/

Gillis, A. S. (n.d.). *What is Object-Oriented Programming (OOP)?* TechTarget. Retrieved November 30, 2023, from https://www.techtarget.com/searchapparchitecture/definition/object-oriented-programming-OOP

Nguyen, D. (2023, January 5). *OAuth Bearer Token with Swagger UI — .NET 6.0 | by Dee Nguyen* . Medium. Retrieved November 30, 2023, from https://medium.com/@deidra108/oauth-bearer-token-with-swagger-ui-net-6-0-86835e616deb

OpenAI. (2023). *ChatGPT* (GPT-3.5) [Large Language Model].

RxJS . (n.d.). RxJS. Retrieved November 30, 2023, from https://rxjs.dev/

What is REST . (2023, November 4). REST API Tutorial: What is REST. Retrieved November 30, 2023, from https://restfulapi.net/