Spring 2024

31297 Application Development with .NET

Assignment 2 Report

Ria Narai - 14326957
Andreas Skotadis - 14183370
Ramon Tovar - 12918761

Project Introduction and Summary

The goal of this project is to create a Pokemon-themed flashcards desktop application (akin to Anki) to assist users in remembering key information for whatever topics they're studying. With this desktop application, users will be able to create their own flashcard decks that are catered to their individual learning objectives and organise them into categories that make sense to them, whether that is by language, topics of discussion, and so on.

For example, if a user wants to learn how to greet other people in Spanish, they can use our application to create a flashcard deck with the name Greetings and the category Spanish. Within that deck they can then create a flashcard with the question "How do you ask someone how they are?" on one side, and then the answer "Como estas?" on the other side. When they have created as many flashcards as they want, they can then quiz themselves using our application's quiz functionality. If a user successfully remembers how to answer a question, they can mark it as successfully remembered. If unsuccessful, they can mark it as such and our application will reveal the total score at the end of the quiz.

<u>The main functionalities of our application are:</u>

- **The Deck Library** - which serves as the Home screen for users. This shows users their created flashcard decks and allows them to navigate to the Add New Deck, View Deck, and Quiz screens.
- **Add Deck** - which allows the user to create a deck by inputting a name, category, and selecting a deck theme from a number of Pokemon types (which determines the deck and card colours as well as the Pokemon images featured on the cards).
- **Modify Deck** - which works similarly to Add New Deck, but allows the user to modify an existing deck's name, category or deck theme.
- **View Deck** - which shows the user all of the cards within their selected deck (and also allows them to change the selected deck), with buttons to Modify the deck, Delete a deck, Modify a card, Delete a / multiple card(s), and Add a new card.
- **Add Card** - which allows a user to create a new card in the deck with a question and an answer. The user can then either save and exit back to the View Deck screen, or can save and create another card.
- **Modify Card** - which works similarly to Add Card, but allows the user to modify an existing card in case of errors.
- **Quiz** - which shows each card in the deck in a randomly shuffled order. The user can click on the text in the centre of the card to see the answer, and must click on either the Don't Remember or Remember buttons. When all cards have been shown, the user is automatically taken to the Quiz Results screen.
- **Quiz Results** - which shows the user's score of successfully remembered cards out of the total cards in that deck, along with an encouraging Pokemon themed message determined by the ratio of cards remembered to not remembered.

Development Approach

Frameworks and Tools

C# and .NET:

We developed our project using C# and the .NET framework, leveraging best practices for clean, efficient, and maintainable code. Our design focused on high cohesion and low coupling across classes and methods to promote modularity and reusability. We implemented constructor overloading to enhance flexibility in object initialisation, ensuring that each class serves a well-defined purpose. Additionally, robust error handling and input validation mechanisms were integrated to ensure reliability and stability throughout the application.

AvaloniaUI:

We selected AvaloniaUI as our user interface framework because of its extensive feature set and strong community support, making it an ideal choice for developing a modern, responsive application. AvaloniaUI's open-source nature and robust documentation streamlined our development process, allowing us to leverage best practices and find solutions effectively.

In our project, we developed several distinct GUI interfaces, each with unique features and functionalities, all designed to be resizable and responsive to ensure a seamless user experience across different devices and screen sizes.

We incorporated several different types of UI elements to enhance the usability and interactivity of the application, such as buttons, text fields, labels, listboxes, scroll views, and data grids. The use of AXAML for UI development enabled us to naturally align our code with the MVVM (Model-View-ViewModel) architecture, making data binding and interface updates efficient and consistent throughout the application. This approach allowed us to build a cohesive, visually appealing, and functionally rich user experience.

MVVM:

We decided to use the Model-View-ViewModel (MVVM) architectural pattern in our application to keep things organised and make it easier to maintain and test. With MVVM, we clearly divided the business logic (Model), the user interface (View), and the code that connects them (ViewModel). This setup allowed our team to work on different parts of the app at the same time, speeding up development and making everything more modular.

Using the MVVM pattern also helped us reuse code and stay flexible. Once we figured out how the View and ViewModel should interact in AvaloniaUI, we could use the same ViewModels across different Views and situations, which made managing the code simpler and cut down on repetition.

In addition, AvaloniaUI's XAML-based data binding worked perfectly with MVVM, making it easy to sync the ViewModel's data with the View. This reduced the amount of boilerplate code we had to write and kept the UI responsive to any data changes.

Draw.io:

The Draw.io software was used for initial wireframing to help us visualise the app's layout at the beginning of our development process. Visualising the app's structure early in the process allowed us to brainstorm and iterate on our design ideas, ensuring that we were all on the same page and knew what we wanted to achieve.

Draw.io enabled us to experiment with different layouts and interface structures, ensuring optimal placement of key elements and a logical flow that aligned with user needs. Through this initial wireframing, we identified potential design and usability issues, allowing for rapid adjustments and refinements before investing time in more detailed prototyping.

Figma:

A more robust interactive prototype was then developed in Figma, allowing us to refine the user interface and plan the user flow between screens before we began coding. Figma's collaboration features allowed us to work on the same prototype remotely, providing each other with feedback and then making adjustments in response. Developing an interactive prototype allowed us to simulate the user experience and identify gaps in our design or issues with our implementation before writing a single line of code.

Design and Implementation

Our development process was iterative and collaborative throughout the entire project. We followed an agile methodology, allowing us to adapt to changes and incorporate feedback continuously.

Wireframing and Prototyping:

We began by mapping out the main screens into quick wireframes using Draw.io, which allowed us to outline the basic structure and layout of our application's key components. These wireframes served as a blueprint, helping us define the placement of core UI elements and establish an initial user flow. This stage provided a clear starting point for discussing navigation patterns, screen hierarchy, and the general look/feel of the app.

Once we had a good foundational structure, we transitioned to Figma to develop a more detailed, interactive prototype. This allowed us to improve our initial wireframes by defining the interactions between screens to simulate the actual user experience.

Collaborating in Figma enabled us to provide real-time feedback and iterate on our designs, ensuring alignment within the team and addressing any usability concerns early on. This iterative design process helped us agree on a well-rounded, user-centred design before we began coding, laying the groundwork for a smoother and more efficient development phase.

Defining the Model:

With a solid prototype in place, we then defined the Model by identifying the core data structures required for our application. This included identifying and implementing classes for Decks, Cards, the DeckLibrary, the Quiz, and the PokemonTypes (used in the visual design of the Decks and Cards). By establishing a clear data model, we ensured that our application would have a strong foundation for managing data and business logic.

Screen Design and Implementation:

We designed the screens based on our Figma prototype using AvaloniaUI. Each screen, including the Deck Library, View Deck, Create/Modify Deck, Create/Modify Card, Quiz and QuizResults screens, was implemented with a focus on usability and visual appeal.

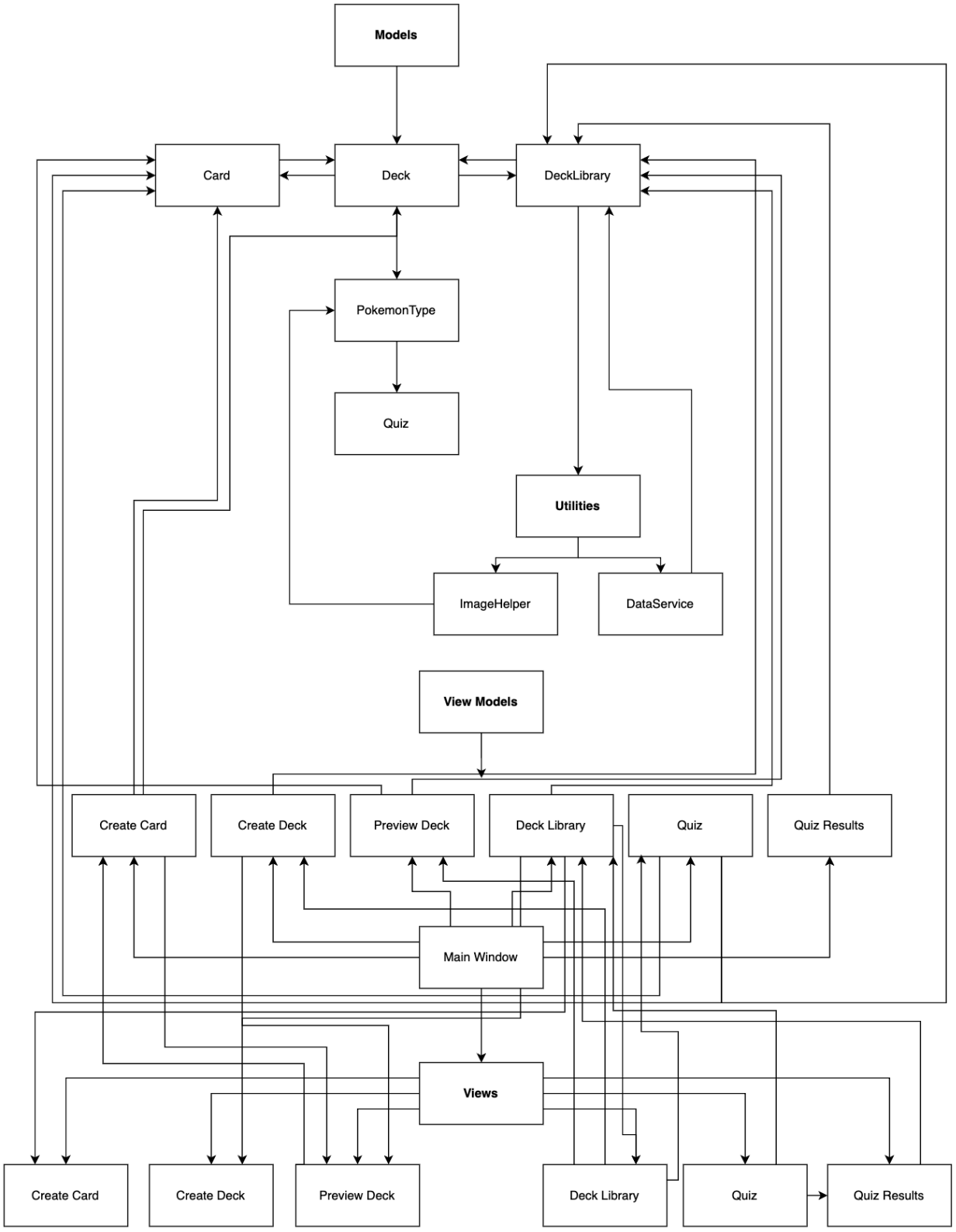
ViewModel Development:

Following the MVVM pattern, we created corresponding ViewModels to handle the logic and data flow for each View. These ViewModels manage the state and behaviour of UI components, handling tasks such as data binding, input processing, and command execution, which ensures that the user interface remains responsive and interactive. This separation allowed us to decouple the UI from the business logic, making it easier to maintain, test, and scale the application. Additionally, leveraging ViewModels enabled efficient data updates across the UI, as changes in the Model were automatically reflected in the View through data binding, enhancing the overall user experience.

Navigation and Data Persistence:

Navigation logic was then implemented between the ViewModels and the MainWindowViewModel, to allow the user to switch between the views based on button clicks. A DataService was created to ensure data persisted between views, using an instance of DeckLibrary to store the Decks and their corresponding Cards. This approach allowed us to manage data centrally and ensure consistency across the application.

Flowchart


Models
Card
Deck
DeckLibrary
PokemonType
Quiz
Utilities
ImageHelper
DataService
View Models
Create Card
Create Deck
Preview Deck
Deck Library
Quiz
Quiz Results
Main Window
Views
Create Card
Create Deck
Preview Deck
Deck Library
Quiz
Quiz Results

Role of Team Members

Team Member	Role
Andreas	- Created final Figma prototype. - Created “Create Deck” screen (View and ViewModel) and “Modify Deck” and “Delete Deck” functionality. - Performed UI enhancements for “Create Card”, “Create Deck”, “View/Preview Deck” and “Deck Library” screens. - Contributed to the Report by adding more detail to the Development Approach & Design and Implementation sections and implementing flowchart diagram.
Ramon	- Created draw.io wireframes. - Created “Create Card” screen (View and View Model) and “Modify Card” and “Delete Card” functionality. - Implemented basic navigation functionality. - Performed bug fixes and enhancements for the “Deck Library”, “View/Preview/Deck”, “Create Deck”, “Modify Deck”, “Delete Deck”, “Quiz” and “Quiz Results” screens. - Contributed to the Report by writing Project Introduction and Summary.
Ria	- Created initial Figma prototype. - Setup Model with “Deck”, “Card”, “DeckLibrary”, “Quiz” and “PokemonType” classes. - Created “Deck Library”, “View/Preview Deck”, “Quiz” and “Quiz Results” screens (Views and ViewModels). - Implemented basic navigation functionality. - Created “ImageHelper” and “DataService” utilities. - Performed bug fixes and enhancements for the “Create Deck” screens. - Polished final UI for “Deck Library”, “View/Preview Deck”, “Quiz” and “Quiz Results” screens. - Contributed to the Report by writing initial Development Approach section (Framework & Tools and Design and Implementation)

References

- Avalonia UI. (n.d.). *Avalonia UI documentation*. Avalonia UI. https://docs.avaloniaui.net/
- Microsoft. (n.d.). *Windows Presentation Foundation (WPF): .NET desktop applications*. Microsoft Learn. Retrieved October 26, 2024, from https://learn.microsoft.com/en-us/dotnet/desktop/wpf/?view=netdesktop-8.0