**Initial Concepts and Research Exploration:**

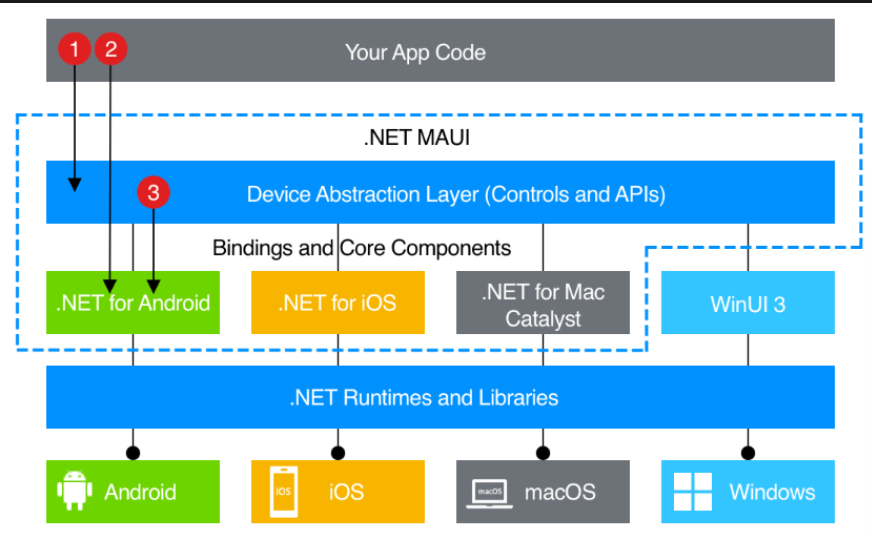
.NET MAUI is Microsoft’s open-source cross platform framework that is used to create native mobile and desktop apps with C# and XAML (Britch et al, 2025). It is an evolution of “Xamarin.forms”, allowing users to build apps that can run on different operating systems such as Android, IOS, macOS & Windows from a single codebase. Through .NET MAUI, users can create multi-platform apps using a single project and one of its key aims is to allow the implementation of logic and UI layout from single shared codebase.   
  
**How it works?**In most important ways, .NET MAUI combines four operating systems; Android, iOS, macOS, and Windows, into a single API that allows a "write-once, run anywhere" developer experience (Tyagi, 2024). This increases efficiency and enables cross-platform usage because the single API is written from a shared codebase that is compatible with each operating system. .NET 6 or later provides platform-specific frameworks for creating apps: Android uses .NET Android, iOS uses .NET iOS, macOS uses Mac Catalyst, while Windows uses the WinUI 3 library. All these platform-specific frameworks have access to the same .NET Base Class Library (BCL) (Britch et al, 2025). The BCL depends on the .NET runtime to provide the execution environment for your code. Each time developers write code from a single shared codebase in Visual Studio, that code primarily interacts with the .NET MAUI Controls and API. See figure below:  


Figure 1: Architecture of a .NET MAUI App (Britch et al, 2025)

rom Figure 1, we can deduce how the code written for the app directly interacts with each platform's specific controls and API. The Base Class Library that all these frameworks access depends on the .NET runtime of each platform to provide an execution environment. This enables support for data binding, creating more elegant and maintainable development patterns (Pryor, 2025). The cross-platform efficiency provided by .NET MAUI also allows for well-defined graphics functionality. These graphics and UI implementation are made possible through the usage of XAML and managed source code while the app is running (hot reload). With hot reload, developers don't need to recompile their edits into the app, as these can be synced with a concurrent running instance of the application.  
**Key Concepts for Financial Savvy**

1. Single Project Architecture:

Leveraging .NET MAUI's ability to support a single shared codebase, the project targets multiple platforms through shared business logic and UI. This is particularly compelling for a financial literacy app such as "Financial Savvy," where a consistent experience across different devices is essential to maintain accessibility and user experience consistency.

1. Hot Reload and Native Performance:  
   UI feedback is a crucial aspect of the user experience in this project. Users' motivation to return to the app depends on the feedback they receive about their progress in understanding key financial concepts. Through hot reload, live UI updates during development and iterations of interactive financial simulations will help users receive timely feedback upon challenge completion, enabling them to learn from the financial simulations they encounter (Spada, 2024).
2. 2D-3D Graphics Integration in MAUI:  
   For graphics implementation, SkiaSharp within MAUI will be utilized to generate interactive charts and graphs for financial data visualization.

**Financial Savvy Project’s Architecture**

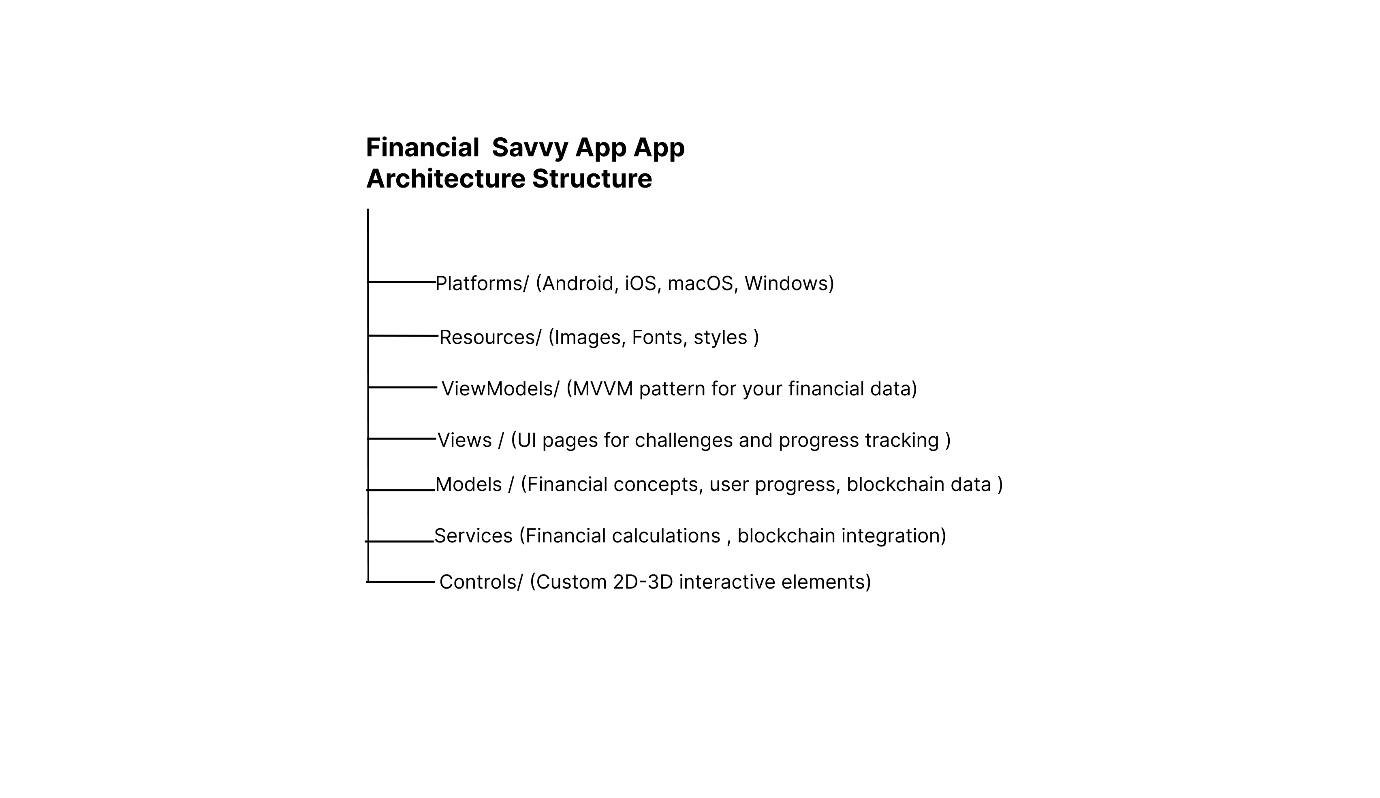


Figure 2: Financial Savvy App Architecture (Mofolo, 2025).

Within the project, I’ve ensured that each specific category has its own folder to ensure that the overall folder structure remains organized and maintainable.  
For the MVVM (Model-View-ViewModel) pattern folder, I’ve decided to break it down into three pattern benefits, which include:  
- ViewModels: Contains all the business logic for calculations, progress tracking and rewards systems.  
-Models: Houses all financial concepts , user profiles and blockchain data will be.  
-Views : Contains all UI components for challenges, SkiaSharp rendering, and progress dashboards.  
**The Key Features for Financial Savvy:**

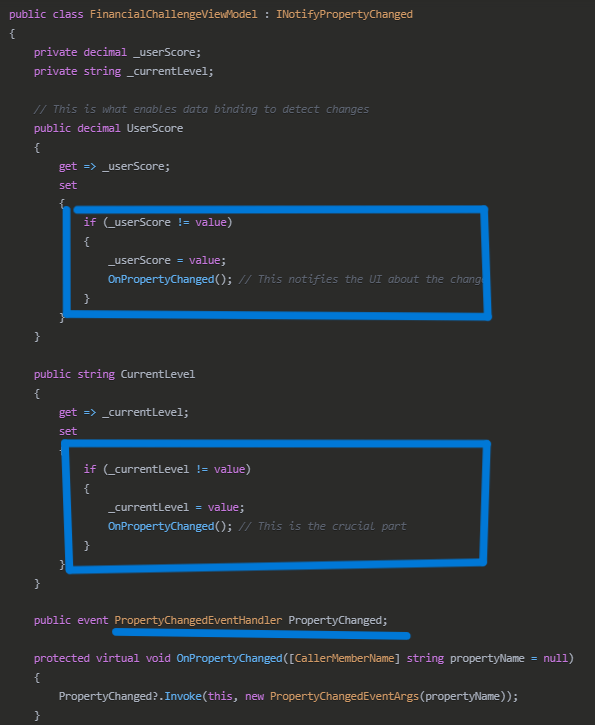
1. Data Binding & MVVM:  
   Data Binding refers to a mechanism where UI elements relate to underlying data sources (your ViewModel), automatically updating the UI when the data changes (Aung, 2024). This leads to a more maintainable and declarative approach to handle data and enables concepts such as Hot reloads as well. Whenever the observed data changes, the data binding engine automatically updates as well with the corresponding newly added UI elements in your view with the new values. For example, the following is a code snippet of how I’m going to use data biding within the project:  
   

Figure 3: Data-Binding Code Snippet from the Financial Savvy Project (Mofolo, 2025)

The code snippet in figure 3 above is one of the effective ways in which data binding is enabled in the project using the C# class “INotifyPropertyChanged”. When the data binding engine connects to the Financial Challenge View Model”, it uses the “OnPropertyChanged” method that is called in the setter method each time the UI values such as the user score and the current level the user is on update. This in turn fires an event that notifies all bound UI elements to refresh and display the newly updated values. This creates a system where the changes in the ViewModel such as the progress tracking are synced with the UI without the need to constantly manually change the value each time it changes.

1. Navigation System:

Shell navigation and a tab-based interface are planned to allow easy access to different financial topics through specific section selection. Shell Navigation in MAUI refers to the application container that describes the app's overall visual hierarchy and navigation structure. It provides URI-based navigation using a system like web navigation, along with route registration, a process that links page pathways in the app so that when users click specific navigation icons, they are redirected to the corresponding pages. See figure below:  
A screenshot of a computer

AI-generated content may be incorrect.

Figure 4: Routing Set Up in Financial Savvy App Project (Mofolo, 2025)

As shown in figure 4, the Routing system allows for the effective pathway to the specific page in the application, making it easier to control and maintain the overall navigation.

The following sitemap is meant to give an indication of how the navigation will be set up in each page:  
A diagram of a company

AI-generated content may be incorrect.

(P.s: If the sitemap is not visible enough upon zooming, you can also access it here:

(<https://www.figma.com/design/AhAPGoSz9jgTgvL1qosej6/Financial-Savvy-App-Project?node-id=0-1&t=T0GWltLoerG6jDvL-1> )  
  
**Analysis of Current Progress and Next Steps:**Currently (as of 01 June 2024), the first sprint on my agile spreadsheet is 80% complete. This section had to do with the core frameworks set up. This included a lot of the concepts I explored in this document that relate to the .Net MAUI terminology. These are:  
**Phase 1: Core Framework Setup**

1. **Project Structure**: Set up MAUI project with architecture.
2. **Navigation**: Implemented Shell navigation for different financial pages or section.
3. **Data Layer**: Created components for financial concepts, user progress, challenges
4. **MVVM Setup**: Implemented ViewModels for each gameplay challenge.
5. **Game mechanics outline**: Designed 3 out 9 gameplay challenge mechanics as displayed in the sitemap. The actual execution is yet to occur once I begin the 2nd sprint.  
     
   While the full app development is far from 100% completion, the foundation establishes the application as a promising and impactful tool. Current achievements include core framework setup and middle-fidelity wireframe designs accessible on Figma: <https://www.figma.com/design/AhAPGoSz9jgTgvL1qosej6/Financial-Savvy-App-Project?node-id=0-1&t=T0GWltLoerG6jDvL-1>.

Learning .NET MAUI presents a steep learning curve as it falls outside my major field of study. Development progress depends on how quickly I can teach myself and understand the framework aspects I'm working with. The stages leading to first sprint completion involved extensive learning, as I intentionally spent time grasping concepts and code functionality to understand not only what the code does, but how it works and how to adapt it for this specific project. The latter in turn becomes a testament of my own understanding which then helps me to explain it better. While there is still a lot of work to be done as far the next steps are concerned (see Spreadsheet), the act of learning anew skill is a challenging but a rewarding journey.  
  
As far as the next steps of the project are concerned, I have tried to create a detailed sprint of each section on the spreadsheet, but a breakdown of that sprint can only be seen below :  
**Phase 2: Educational Content System**

1. Content Management: Create flexible system for financial lessons
2. Progress Tracking: Implement user progress persistence
3. Challenge Framework: Build reusable challenge/quiz system as games
4. Reward System: Integrate blockchain-based rewards

**Phase 3: Interactive Elements**

1. **2D Visualizations**: Implement 2D financial concept demonstrations using SkiaSharp
2. **Gamification**: Add achievement systems, leaderboards tracking.
3. **Simulations**: Create real-world financial scenario simulations.
4. **Personalization**: AI-driven personalized learning paths

**Phase 4: Advanced Features (Extra: if time allows)**

1. **Blockchain Integration**: Connect to blockchain for rewards/certificates
2. **Real-world Data**: Integrate live financial market data
3. **Social Features**: Community challenges and peer learning
4. **Advanced Analytics**: Detailed progress and learning analytics.

# References

Aung, P. (2024). *Medium.* [Online]   
Available at: https://medium.com/@phyothinzaraung/how-does-data-binding-work-in-mvvm-and-what-are-its-advantages-c74d960429f0  
[Accessed 28 May 2025].

Britch, D. &. M. R. (2025). *Microsoft .NET MAUI.* [Online]   
Available at: https://learn.microsoft.com/en-us/dotnet/maui/what-is-maui?view=net-maui-8.0  
[Accessed 25 April 2025].

Pryor, J. (2025). *Microsoft .NET MAUI.* [Online]   
Available at: https://learn.microsoft.com/en-us/dotnet/maui/what-is-maui?view=net-maui-8.0  
[Accessed 23 April 2025].

Spada, P. (2024). *Microsoft .NET MAUI.* [Online]   
Available at: https://learn.microsoft.com/en-us/dotnet/maui/xaml/hot-reload?view=net-maui-8.0  
[Accessed 28 May 2025].

Tyagi, H. (2024). *Microsoft .NET MAUI.* [Online]   
Available at: https://learn.microsoft.com/en-us/dotnet/maui/supported-platforms?view=net-maui-8.0  
[Accessed 24 April 2025].