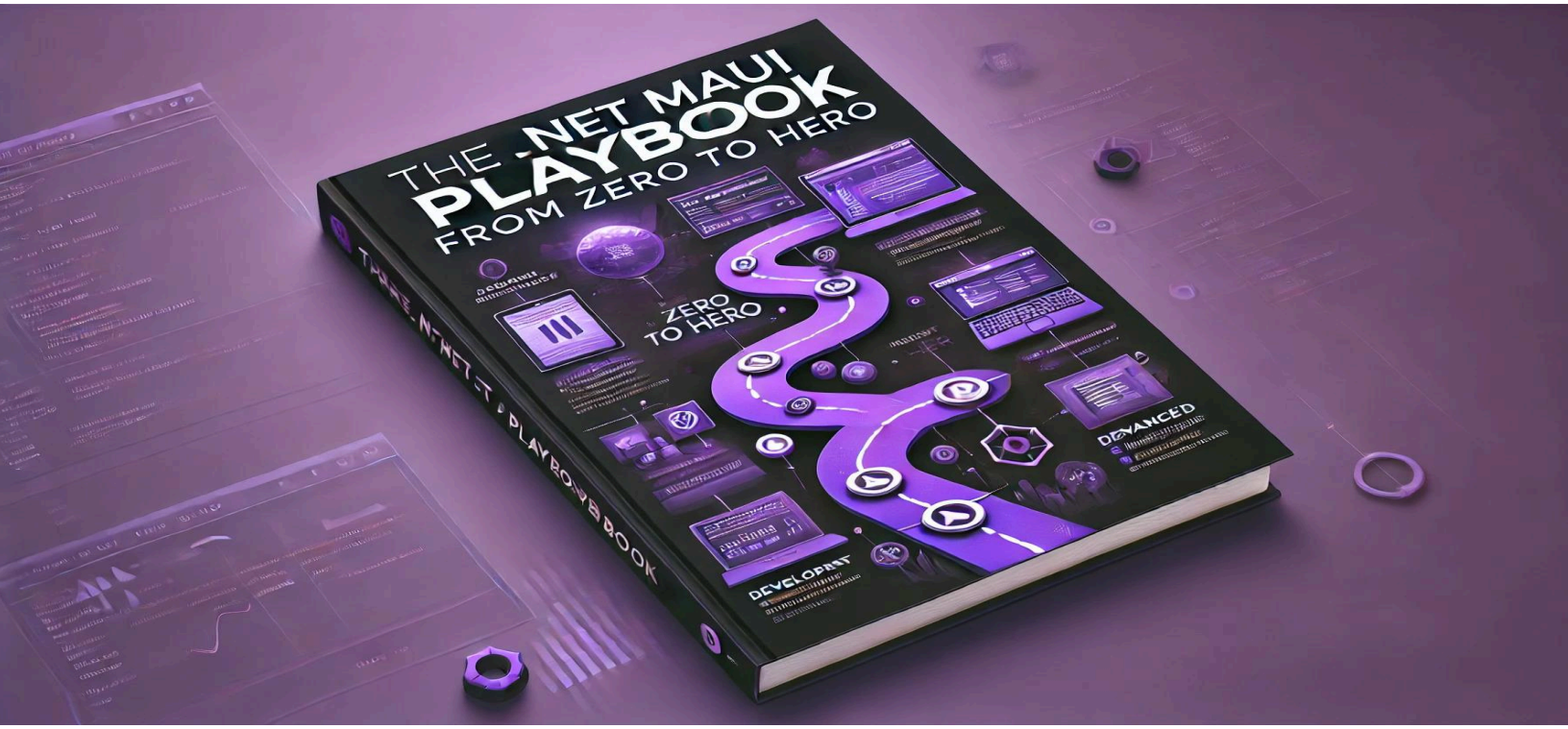
The .NET MAUI Playbook: From Zero to Hero

Version 1: Oktober, 2024

A Self-taught .NET MAUI Developer Roadmap



A big big thank you!

First and foremost, I want to express my sincere gratitude to the excellent programming and .NET MAUI creators, particularly Gerald Versluis and James Montemagno. The fact that they share their knowledge for free is incredible. It is solely through their knowledge and videos, which have accompanied me on my journey over the past 5 years, that my life has developed to where it is today. Thanks to what I learned, I have published several mobile apps and created a software product that I distribute B2B. All of this from just my laptop. What a time to be alive! Now I want to give something back to the community and also share my knowledge! I hope you like it 🙂

What do you get here?

Over the past 5 years, I've taught myself programming, starting with C# and later moving on to Xamarin.Forms and .NET MAUI. I relied entirely on free online tutorials throughout this journey. However, finding truly valuable resources can be quite time-consuming. That's why I've compiled my experiences and the resources I've personally used to learn .NET MAUI here.

You will learn:

1. Your Roadmap to .NET MAUI Success: A 3-Step Framework
2. My favorite .NET MAUI Content Creators
3. Further resources for learning
4. How Do I personally develop my apps using .NET MAUI

Your Roadmap to .NET MAUI Success: A 3-Step Framework

This framework guides you through the essential stages of .NET MAUI app development.

1. **Ignite:** Set up your environment and learn the fundamentals.
2. **Craft:** Develop your app's features and functionality.
3. **Deploy:** Share your app with the world.

I've curated the best resources for each step, ensuring a smooth and successful journey to launching your first .NET MAUI app.

1. Ignite

Ready to dive into .NET MAUI? I've curated a collection of beginner tutorials to help you take your first steps. No prior experience is required! Start exploring these resources and ignite your MAUI app development journey.

Long form beginner tutorials/playlists

- https://youtu.be/02pjHAXYK6s?si=p9ZL_tIEO4lHCFEc
- https://www.youtube.com/watch?v=DuNLR_NJv8U&t=2s

- https://www.youtube.com/watch?v=mgW6xviirQk&list=PLfbOp004UaYVt1En4WW3pVuM-vm66OqZe&pp=iAQB
- If you want to use VS Code as your editor of choice
    - https://youtu.be/1t2zzoW4D98?si=ieSv22Iybj9RCXvP
    - https://youtu.be/zfDCZkITpHU?si=GkpHtJ5yQTqTAQrr
    - https://youtu.be/ZkrUR3A9aM4?si=9pWgKN6GLEoU1CmE
    - https://youtu.be/B-kZ-AgEeO8?si=A_Npokhr4BbDRjJO

If you want to get into Blazor Hybrid development, those are 2 perfect options (I personally don't develop using Blazor Hybrid)

- https://youtu.be/lqLfY9zNKNY?si=Pb6pTI5Di6dnJc9y
- https://youtu.be/Ou0k5XKcIh4?si=h3VL4uJS8nOl9208

2. Craft

Have an idea and want to jump right into building, learning everything else along the way? That's exactly how I did it! Here are some resources that you might find helpful on your journey.

Everything XAML, MVVM, Databinding

- https://youtu.be/GLfR2uosoSw?si=fvPRKbnG-orfLGCJ - XAML 101
- https://youtu.be/B-5e0PJtSDs?si=gM-OQ-90sG9uEyDY - MVVM - Watch that first
- https://youtu.be/sAn4RVsroF4?si=L44hrCCLbdeNNLBx - MVVM - Also a great resource
- https://youtu.be/AXpTeiWtbC8?si=BEHBHNTBBa-3VizM - MVVM - Benefits of that pattern
- https://youtu.be/WQz700h-kKo?si=s61Eos8KC7y-yn4Y - More advanced ⇒ Dependency Injection

Dependency Injection (DI)

- https://youtu.be/9X1TwT9wWCE?si=uF-Us6XsVgngBlnV - Start here
- https://youtu.be/xx1mve2AQr4?si=HCaky9_k2iCJ2-td
- https://youtu.be/Yvu1FUVsOps?si=UVSRD42VUCxEq3g1
- https://youtu.be/wkgbvMlrMhU?si=oP_lS1Ppaq598ROa
- https://youtu.be/rnsjeEAjxXI?si=rI6K5Er77TgHAfyo

- Couple of thoughts from my perspective:
  - DI is a really great thing
  - in the beginning i used to write my code not with MVVM but in XAML codebehind
  - that came with huge problems and quirks ⇒ in short don't do it
  - eventually i switched to MVVM and without DI it was a pain to do
  - then I checked how everything worked and now I have a specific system how I write my MAUI apps
  - more on that later

Local Storage options

- https://youtu.be/exnTL23z4_I?si=POxFSLYeeSQy2-xT
  - Preferences = best option for storing simple data
  - I use it for example for storing information about when I last asked the user for a review
  - No database replacement, only able to store simple data
- https://youtu.be/VziMUc-VQko?si=3IMyVOwLrjlEqC_q
  - SQLite = best overall option for storing data locally in a MAUI App in my opinion
  - I personally use this with every app i create
  - It's a relational database
- https://youtu.be/m03IjT1b2Hs?si=mHj1MTkfjY3GkzIb
  - LiteDB = best option if you want to store data in a NoSQL way
  - has a few problems to in my opinion
    - first: no async options meaning you read and insert data always in a synchronous way
    - second: not aot compatible meaning on iOS you have to enable the interpreter if you want to avoid your app crashing instantly - Explanation here: https://maxmannstein.com/index.php/maui-ios-app-crashing-instantly-in-release-mode/

Options for backend that I use

- Supabase
  - Relational database built on postgres
  - options for Auth, Database, Storage ⇒ everything you need
  - Advantage: has a really good official community c# api
  - Disadvantage: only one free project
  - Resources
    - I plan on doing a tutorial series on my channel: https://www.youtube.com/@maxmannstein
    - until I have done it here are the official docs: https://supabase.com/docs/reference/csharp/introduction
- Firebase
  - json NoSQL database
  - options for Auth, Database, Analytics, Crashlytics (i didn't get it working - just use Sentry) ⇒ everything you need
  - Advantage: endless free projects with limits I personally never reached
  - Disadvantage: sometimes tricky to implement stuff with .NET MAUI because its not officially supported
    - but: Auth and Realtime Database have great unofficial packages
    - https://www.nuget.org/packages/FirebaseDatabase.net/
    - https://www.nuget.org/packages/FirebaseAuthentication.net
    - there is also a plugin called https://www.nuget.org/packages/Plugin.Firebase/ which I personally didn't used tho so I cant say much about it
  - Resources
    - https://www.youtube.com/watch?v=7w2q2D6mR7g&list=PLfbOp004UaYXoq9NLvMqdGSyWeEr_w6fu&pp=iAQB
      - a little bit older but still quite a good resource
    - Single videos, which might be helpful
      - https://youtu.be/dGrh-z1G8lI?si=kxMId9U3LgcATimT
      - https://youtu.be/HvvHNOJ3qMM?si=Hpgs9qYHnLUgp0du
      - This is a blog: https://www.c-sharpcorner.com/article/xamarin-forms-working-with-firebase-realtime-database-crud-operations/
      - https://youtu.be/LYAxYNN8s_s?si=S9GzcAyWmX7OAFyp

- My Perspective
  - If you decide between Firebase and Supabase as a Backend as a Service (BaaS) for your MAUI app i would definetly try out both options
  - if you want to do a mobile app without expecting crazy masses of people using it, just go with firebase and you probably never pay a single $
  - if you want to do a serious project better use supabase because i feel like its better supported with MAUI and you dont have any hassle setting everything up
  - I personally combine even both sometimes because I need google analytics which are only available with Firebase but regarding Database I definetly enjoy Supabase more

Exceptional MAUI Plugins

- https://www.youtube.com/watch?v=ostgj2xB_ok&list=PLfbOp004UaYVgzmTBNVI0qI2qF0LhSEU1&pp=iAQB
  - here you can find everything you need

3. Deploy

Time to share your creation with the world! This stage is about taking your finished .NET MAUI app and making it available to users. I've gathered resources to guide you through the deployment process, from preparing your app for release to distributing it on different platforms. Get ready to celebrate your accomplishment!

- AppStore: https://youtu.be/kpZi5xAvpZA?si=AcbMnlVlEdv92De_
- PlayStore: https://youtu.be/jfSVb_RR7X0?si=CGvpwKBRJjHW203S

- Windows:
  - as msix: https://youtu.be/FNwv_W3TtSU?si=eSo4qVrqL_r1jvG9
  - auto updates for sideloading: https://youtu.be/IzfkpoXtP3o?si=LG1R48NKIcVM55ZZ
  - as zip: https://youtu.be/GgU4ulGYQEk?si=PHG3zFg9X45I39NS

My favorite .NET MAUI Content Creators

- Gerald Versluis: https://www.youtube.com/@jfversluis
    - Discovering latest plugins
    - Great tutorials in general
    - He is THE GUY
- James Montemagno: https://www.youtube.com/@JamesMontemagno
    - great long form tutorials for beginners
    - maintains a really helpful plugin for payments in app and playstore
- Daniel Hendrikes: https://www.youtube.com/c/DanielHindrikes
- Xamarin Guy: https://www.youtube.com/@xamaringuy1951
    - Great tutorials - especially for backend with Firebase
- Javier Suárez: https://www.youtube.com/c/JavierSuárezRuiz
- Naweed Akram: https://www.youtube.com/@naweedakram/
- Singleton Sean: https://www.youtube.com/@SingletonSean
    - Mainly if you want to develop for windows
- Devs School: https://www.youtube.com/@devsschool
    - Mainly for learning UI Design for MAUI
- Programming With Pragnesh: https://www.youtube.com/@ProgrammingWithPragnesh
    - Really underrated channel with many great tutorials
    - Unfortunately he stopped posting about a year ago 😖
- Abhay Prince: https://www.youtube.com/channel/UCoKgtZAEHWP1TI1dO7mRD7Q
- You can decide whether I belong on that list or not

Further resources for learning

- Awesome MAUI Github Repo: https://github.com/jsuarezruiz/awesome-dotnet-maui
    - a resource showing **awesome** .NET MAUI libraries and resources
- MAUI Bug Library: https://maxmannstein.com/index.php/net-maui-bug-library/
    - a list of bugs and quirks of mobile development and maui I personally encountered with an explanation of how I solved them
- Sharpnado Blog: https://www.sharpnado.com/tag/dotnetmaui/
    - really experienced guy running it and good insights

How Do I personally develop my apps using .NET MAUI

Just a quick note before we begin: this is from my blog article, which you can find here:

[How Do I develop .NET MAUI apps - Max Mannstein]

In this blog article I am going to show you, how I personally develop my .NET MAUI apps. Just a quick note before we begin: this is my personal approach to .NET MAUI development. I'm not claiming it's the only way or even the best way, but it's what works for me. So, if you're curious to see how I build, structure, and deploy my apps, read on.

I splitted the article in 5 sections. First we take a look at the common tools that I use to develop my apps, then I will show and explain to you how I typically structure my projects. After that we look at my way of handling the UI, then I give a quick overview over my backend choices and finally I give to you the most valuable nuget packages, that I use for speeding up the development process in every app of mine.

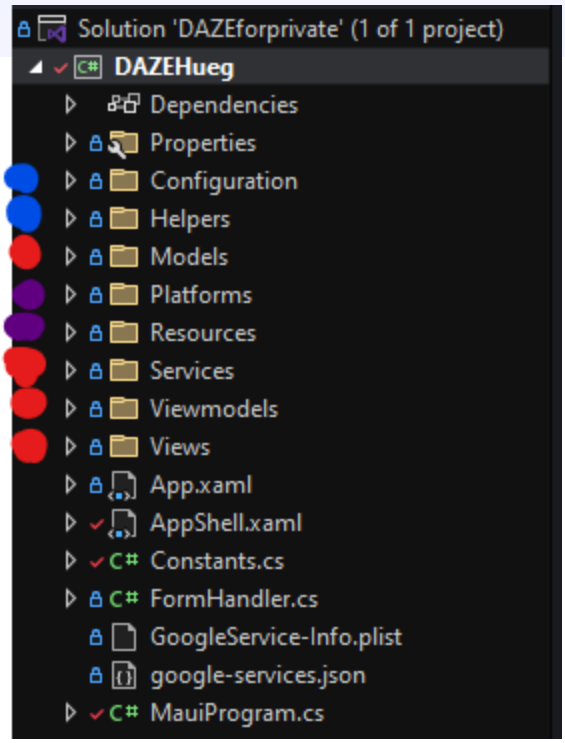
Lets get started 🙂

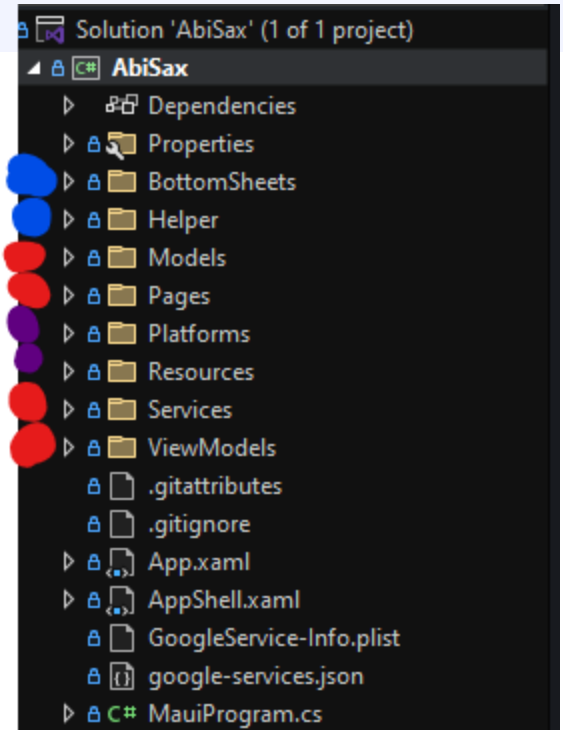
Tools

This is a quick one since you really don't need much to develop apps besides your brain, a laptop and an IDE and preferably an internet connection. So lets take a look.

1. Laptop: I use my 2 year old ThinkPad E14 Gen 4 with a Ryzen 7 (around 1000$ at the time) connected to 2 external monitors to write my code.
2. IDE: My IDE of choice is Visual Studio by Microsoft, which has a perfect integration for MAUI and .NET in general. I also tried VS Code for a while but eventually switched back to Visual Studio as I'm feeling much more comfortable with it.
3. AI Helpers: I personally use GitHub Copilot, which I get for free through my student program and besides that I typically use Claude AI and ChatGPT depending on where I already reached the free limit

Project Structure





As you can see in those images, which showcase the project structure of my 2 latest apps I typically have the same structure in every project. This is what works for me and now I will explain that in further detail.

First all folders marked violet are preexisting for every new MAUI project. The blue folders are additional things i don't need every time. Typically the Helper folder is for storing Converters and stuff like. the others are specific to the apps. One could argue that BottomSheets should be part of the views and he would be right. I can't actually remember why I separated that.

Where I figured out the best way for me is with the red marked folders. Much of that has to do with the architectural pattern that I use, which is called MVVM (Model–view–viewmodel). This explains the 3 main red folders of Models, Views/Pages and ViewModels. The addition I made, which I think is not that common, is the Services folder. But what do those terms even mean in the first place? General definition coming:

MVVM (Model-View-ViewModel)

MVVM is a design pattern used primarily for building user interfaces that separates the concerns of data (Model), user interface (View), and the logic that connects them (ViewModel). This separation offers several benefits like improved testability, maintainability, and code reusability.

Components

- **Model:** Represents the data and business logic of the application. It is responsible for fetching, storing, and manipulating data. It has no direct knowledge of the View or ViewModel.
- **View:** The user interface that displays data from the ViewModel and allows the user to interact with the application. It is typically implemented using XAML or HTML and has no direct knowledge of the Model.
- **ViewModel:** Acts as an intermediary between the Model and the View. It prepares data from the Model in a way that is easily consumable by the View. It also handles user interactions and updates the Model accordingly. It has no direct knowledge of the View.

How it works together

The View observes the ViewModel for changes, and the ViewModel observes the Model. When the Model changes, the ViewModel is updated, which in turn updates the View. User interactions in the View trigger commands in the ViewModel, which then updates the Model. This ensures that the Model, View, and ViewModel remain loosely coupled, making the application easier to develop, test, and maintain.

Example

Imagine a simple application that displays a list of customers.

- **Model:** Would be responsible for fetching the customer data from a database or API.
- **View:** Would display the list of customers in a user-friendly way, perhaps using a list or grid.
- **ViewModel:** Would retrieve the customer data from the Model, format it for display (e.g., sorting, filtering), and expose it to the View.

This separation of concerns allows developers to work on different parts of the application independently, improving productivity and reducing complexity.

Now I'll explain how I use this for developing my apps and which changes/additions I made. Concerning Views and ViewModels I just use them as in the definition explained. Her I have no further things to add.

Concerning the Models tho I have quite a different approach. I personally use Models as pure definitions of the Data they hold meaning they are quite stupid and just hold data instead of for example fetching or manipulating it. Those are the tasks that the Services do in my applications. Below I have two pictures showing exactly that. For the model you can see that it clearly only defines the structure of the data without doing anything. The service on the

other hand is able to talk to the appstore api and use that model to purchase a subscription for example.

```
namespace DAZEHueg.Models
{
    22 references
    public partial class PurchasableItemModel : ObservableObject
    {
        [ObservableProperty]
        bool isSelected;
        [ObservableProperty]
        bool isPopular;
        [ObservableProperty]
        bool isSubscription;
        [ObservableProperty]
        string title;
        [ObservableProperty]
        string priceBefore;
        [ObservableProperty]
        string priceAfter;
        [ObservableProperty]
        DateTime expirationDate;
        [ObservableProperty]
        string storeId;
        [ObservableProperty]
        string nonConsumableKey;
        [ObservableProperty]
        TimeSpan duration;
    }
}
```

Model

```
namespace DAZEHueg.Services
{
    8 references
    public class PaymentsService
    {
        readonly UserService _userService;
        readonly PurchasableItemsService _purchasableItemsService;
        0 references
        public PaymentsService(UserService userService, PurchasableItemsService purchasableItemsService)
        {
            _userService = userService;
            _purchasableItemsService = purchasableItemsService;
        }
        4 references
        private async Task<bool> HasInternetAndConnected()
        {
            if (Connectivity.NetworkAccess != NetworkAccess.Internet)
                return false;
            return await CrossInAppBilling.Current.ConnectAsync();
        }
        1 reference
        public async Task<bool> PurchaseSubscription(PurchasableItemModel item)
        {
            try
            {
                if (!await HasInternetAndConnected())
```

Service

UI

My approach to UI design is relatively basic. I start by defining the color palette for my app and storing it in a dictionary. I tend to stick with the standard MAUI styles and simply modify the colors to match my design. I realize there's room for improvement here, and I'm always looking for ways to enhance my UI skills. To help streamline the process, I utilize these packages:

1. Uranium UI: https://enisn-projects.io/docs/en/uranium/latest/ – Modern, extensible, customizable, and easy to use UI framework for .NET MAUI. Uranium is a Free & Open-Source UI Kit for .NET MAUI. It provides a set of controls and utilities to build modern applications. It is built on top of the .NET MAUI infrastructure and provides a set of controls and layouts to build modern UIs. Really love it!
2. Freaky Controls: https://github.com/FreakyAli/Maui.FreakyControls – Maui.FreakyControls is a UiKit for Maui, you can use to create unique and visually stunning user interfaces for your Maui applications. Each control is highly customizable and comes with a set of parameters that allow you to adjust them to your specific needs. – Not in every project and mostly for really specific things, but really enjoyable using
3. BottomSheets: https://github.com/the49ltd/The49.Maui.BottomSheet
4. Popups: https://github.com/LuckyDucko/Mopups
5. Icons: https://github.com/AathifMahir/MauiIcons

Backend Choices

My backend philosophy is all about efficiency and ease of use. I prefer to avoid the complexities of managing servers and scaling infrastructure. That's why I gravitate towards Backend as a Service (BaaS) solutions like Supabase and Firebase.

Here's a breakdown of my thoughts on each:

- **Firebase:** A great choice for getting started, especially if you're not anticipating a huge influx of users. It's incredibly user-friendly and offers a generous free tier.
- **Supabase:** When you need a robust and scalable backend, Supabase is an excellent option. It integrates seamlessly with MAUI and provides a smooth setup experience. Here I also plan a tutorial series on my YouTube channel, so click the button on the right top of your screen 🙂

In some cases, I find it beneficial to utilize both platforms. I might use Firebase for features like Google Analytics, while relying on Supabase for its powerful database capabilities.

Most valuable nuget packages

Finally, let's take a look at a list of NuGet packages that I incorporate into every app I build:

1. CommunityToolkit.Maui – for a easier time developing in general https://learn.microsoft.com/en-us/dotnet/communitytoolkit/maui/get-started?tabs=CommunityToolkitMaui
2. CommunityToolkit.Mvvm – for less boilerplate regarding UI changes https://learn.microsoft.com/en-us/dotnet/communitytoolkit/mvvm/
3. Plugin.InAppBilling – for monetization https://jamesmontemagno.github.io/InAppBillingPlugin/GettingStarted.html
4. Plugin.Maui.AppRating – for app rating https://github.com/FabriBertani/Plugin.Maui.AppRating
5. Sentry.Maui – for crash reporting https://docs.sentry.io/platforms/dotnet/guides/maui/
6. sqlite-net-pcl – for local storage https://learn.microsoft.com/en-us/dotnet/maui/data-cloud/database-sqlite?view=net-maui-8.0
7. and the UI packages named in the UI part