1.2.2023

Dokumentation: Save Up

Modul 335 - Mobile-Applikation realisieren

David Mangold
IBZ

Inhalt

Tabelle Verzeichnis

Abbildungsverzeichnis

Abkürzungen

Abkürzung	**Beschreibung**
MAUI	Multi-platform App UI
JSON	JavaScript Object Notation
XAML	Extensible Application Markup Language
XML	Extensible Markup Language

Einleitung

Der Zweck der Projektarbeit "SaveUp-App" ist die Erstellung einer Applikation welche der User die möglichkeit bietet, seine Einsparungen aufzuzeichnen und dadurch gezielt Geld für private Investitionen wie beispielsweise Ferien zu sparen. Dabei soll es dem User ermöglicht werden, auf übliche kleine Ausgaben wie Kaffee oder Süssigkeiten zu verzichten und das gesparte Geld in der SaveUp-App zu erfassen. Diese Applikation soll dem User eine Übersicht über den angesparten Geldbetrag bieten, sodass er laufend informiert ist und motiviert bleibt, sein Sparziel zu erreichen.

Dieses Projekt sollte Folgende Punkte beinhalten:

- Name und Titel der App: **SaveUp**
- App besteht min. aus 2 Content Pages
- GUI Design (Mock-ups)
- Produkterfassung besteht aus min. 2 Eingaben (Kurzbeschreibung u. Preis)
- Bietet zwei Menüfunktionen (Action) zur Speicherung und Aufruf der Listendarstellung an.
- Einfache bzw. intuitive Bedienung, geeignetes Layout (Styles)
- Codestrukturierung nach Model View ViewModel (MVVM) Entwurfsmuster
- Verwendung von XAML-Styles der Steuerelemente
- Eigenes App-Icon
- Dokumentation u. Testing

Optionale Anforderungen

Zusatzpunkte für optionale Erweiterungen. Zur Erreichung der max. Punktzahl müssen zwei optionale Anforderungen umgesetzt werden.

- Speichern (Persistenz) der Einträge in einer lokalen Datei (XML, JSON)
- Speicher der Einträge in Backend Datenbank mit REST Implementierung
- Löschen der erfassten Einträge (Clear), einzeln u. komplett.
- Datum/Uhrzeit als zusätzliches Attribut, wann der Kaufverzicht erfolgte.

Randbedingungen

Es müssen folgende Randbedingungen eingehalten werden:

- Xamarin Forms App oder Maui .NET
- Copy/Paste Lösungen aus dem Internet werden mit Note 1 bewertet

Verwendet Tools & Nuget:

Für die Umsetzung meines Projekts habe ich Visual Studio Code als Texteditor genutzt und das MAUI.NET Webframework implementiert. Zur Überprüfung der Anwendung auf mobilen Geräten habe ich den im Visual Studio integrierten Emulator verwendet.

Zur Verwaltung des Projekts habe ich ein GitHub-Repository erstellt und täglich alle Änderungen hochgeladen. Dadurch konnte ich das Projekt effektiv organisieren und sicherstellen, dass alle Änderungen dokumentiert sind.

Für dieses Projekt wurden folgende Nuggets installiert, um die vorhandenen Komponenten zu ergänzen::

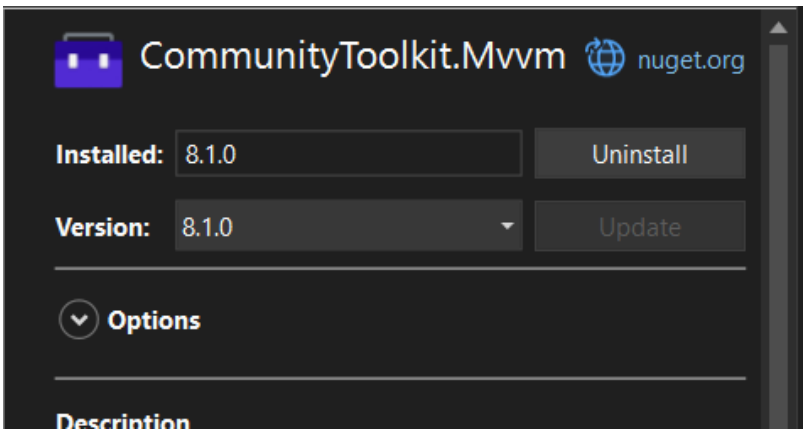


Abbildung 1: NuGet CommunityToolkit

Das CommunityToolkit.Mvvm NuGet-Paket bietet nützliche Klassen und Methoden, um die Implementierung des MVVM-Musters zu erleichtern. Dazu gehören beispielsweise die "ObservableObject"-Klasse, die eine Datenbindung zwischen View und ViewModel ermöglicht, sowie die "RelayCommand"-Klasse, die das Definieren von Befehlen im ViewModel vereinfacht.

Dokumentation nach dem IPERKA-Modell

Die Dokumentation erfolgt unterteilt nach den sechs Phasen des IPERKA-Modells.

Informieren

Während der Informationsphase habe ich umfangreiche Recherchen durchgeführt, um die Anforderungen und Bedürfnisse des Projekts zu verstehen. Dabei bin ich auf MAUI.NET gestossen und habe mich eingehend mit dieser Technologie auseinandergesetzt. Zusätzlich habe ich auch im Internet nach weiterführenden Informationen gesucht, um ein umfassendes Verständnis zu erlangen. Diese Recherchen haben mir geholfen, eine fundierte Entscheidung zu treffen und das beste Framework für unser Projekt zu wählen.

Projekt Informieren

Am ersten Tag des Moduls wurde uns das Projekt vorgestellt. Da es ein komplett neues Auftrag war, begann ich sofort mit der Recherche, wie man eine Solche Applikation realisieren kann. Ich beschäftigte mich mit MAUI .NET und nutzte sowohl die Informationen, die wir in der Schule erhielten, als auch das Internet (die Links finden sich im Quellenverzeichnis). Ein grosser Teil meiner Recherche war es, den Aufbau dieses Framework , da es ein wesentlicher Bestandteil dieses Projekts ist.

Fragen aufschreiben

Ich habe bei meiner Recherche immer wieder Fragen notiert, wenn ich nicht weitergekommen bin. Diese Fragen habe ich entweder aufbewahrt und weiter erforscht (einige haben sich von selbst geklärt), oder ich musste sie meinem Lehrer oder einer Klassenkamerad stellen.

Die Unklarheiten waren unter anderem:

- Was sind die wichtigsten Features von MAUI.NET und wie unterscheidet es sich von anderen Frameworks?
- Welche Programmiersprachen werden von MAUI.NET unterstützt?
- Gibt es spezielle Anforderungen oder Einschränkungen bei der Entwicklung von MAUI.NET-Anwendungen?
- Wie können MAUI.NET-Anwendungen auf verschiedenen Plattformen getestet werden?
- Gibt es bekannte Herausforderungen oder Schwierigkeiten bei der Implementierung von MAUI.NET?
- Wie ist die Performance von MAUI.NET im Vergleich zu anderen Frameworks?
- Sind zusätzliche Bibliotheken oder Pakete erforderlich, um bestimmte Funktionalitäten in MAUI.NET zu implementieren?
- Wie kann die Sicherheit von MAUI.NET-Anwendungen gewährleistet werden?

Nachdem die Unklarheiten ausgeräumt waren, habe ich mich weiter im Internet informiert, um die im Arbeitsauftrag beschriebenen Schritte ausführen zu können. In diesem Modul haben wir auch verschiedene Tools und Methoden kennengelernt, um die MMVM Methode zu implantieren.

Planen

Als Erstes in der Planungsphase habe ich zeitliche Schätzungen für jeden Teil des Projekts vorgenommen und einen Zeitplan erstellt. Dadurch war es möglich, einen konkreten Plan für die Applikation aufzustellen, indem ich das passende Schema ausgewählt habe.

PSP und GANT Plan erstellen

Um das Projekt erfolgreich innerhalb des gegebenen Zeitrahmens abzuschliessen, habe ich mithilfe des WBStools einen Ablaufplan erstellt. Dieser GANT- und PSP-Plan sollte mir helfen, meine Arbeitspakete im Blick zu behalten und stets zu wissen, welche Aufgaben als Nächstes anstehen. Obwohl es mir nicht immer gelungen ist, exakt nach Plan zu arbeiten, habe ich den Zeitrahmen im Grossen und Ganzen eingehalten.

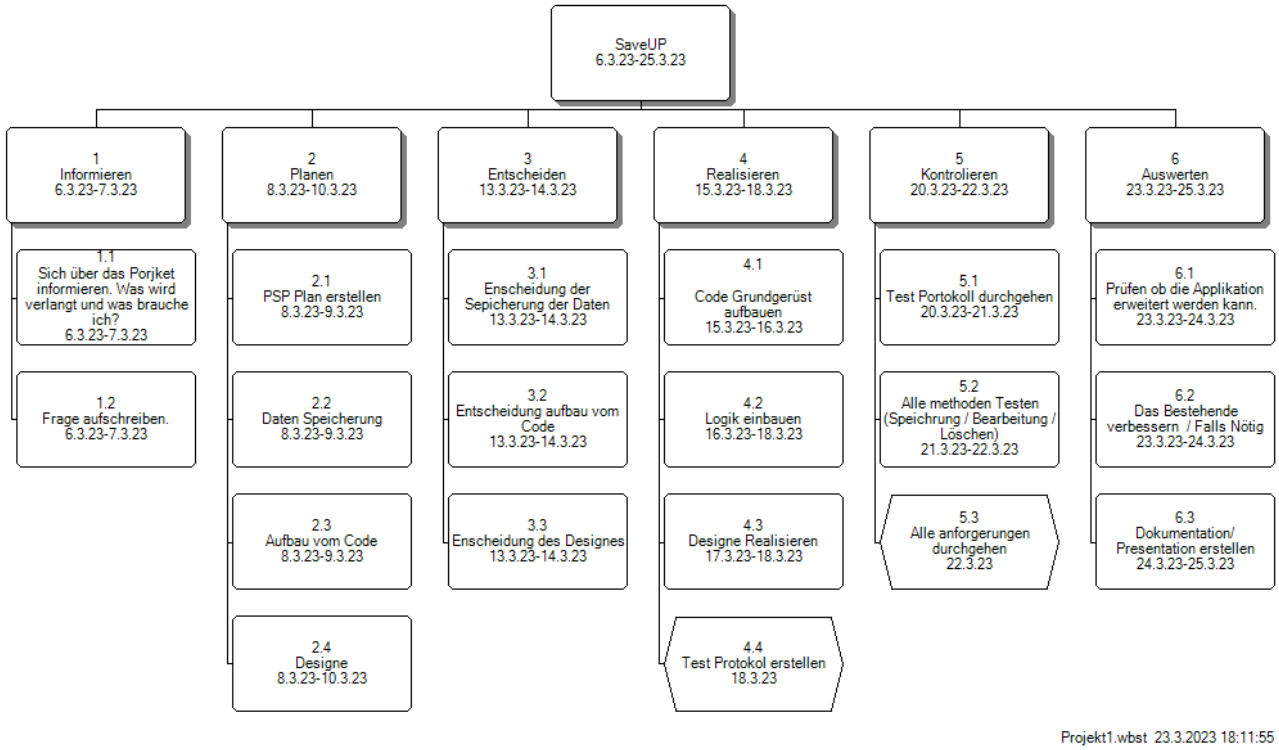


Abbildung 2 GANT und PS

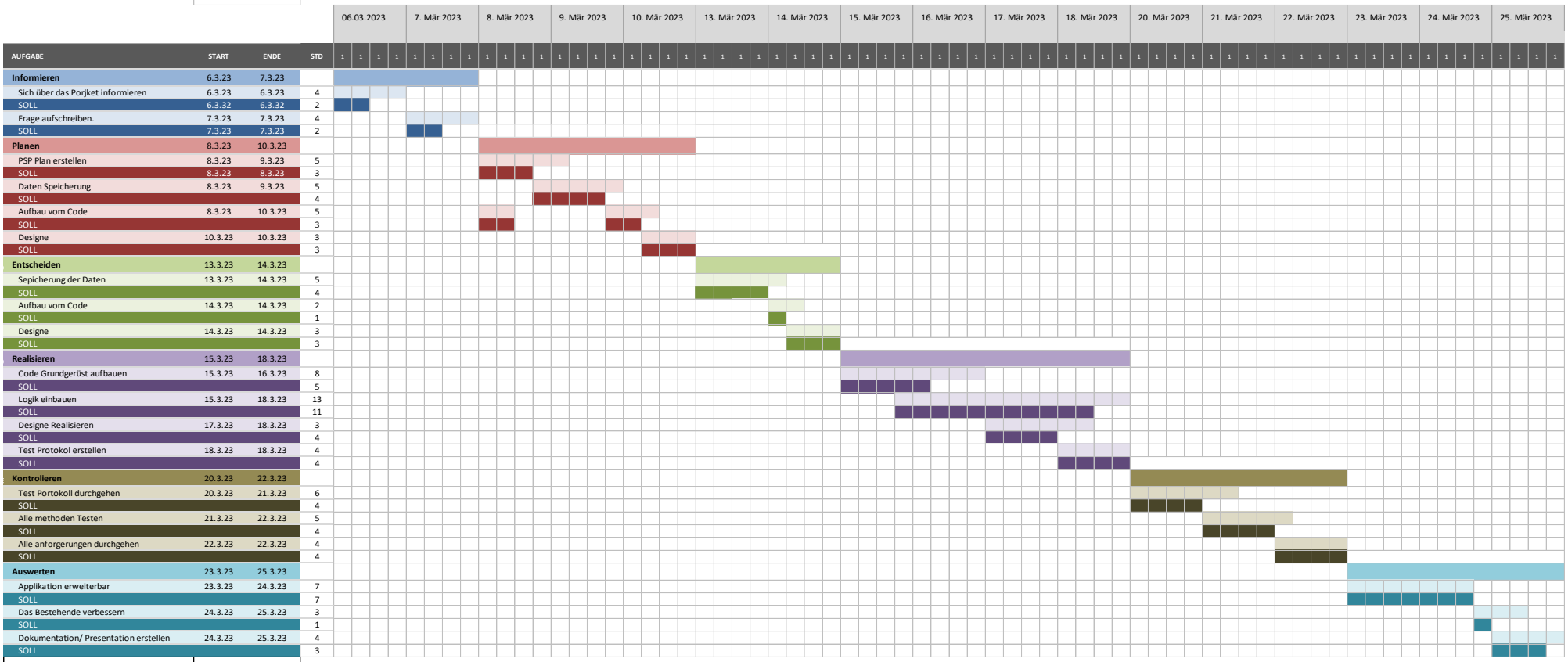
SaveUP

IBZ
David Mangold

Helle Farbe : IST

Duckle Fabe: SOLL

Projektanfang:: 06.03.2023



AUFGABE	START	ENDE	STD
Informieren	6.3.23	7.3.23	
Sich über das Porjket informieren	6.3.23	6.3.23	4
SOLL	6.3.32	6.3.32	2
Frage aufschreiben.	7.3.23	7.3.23	4
SOLL	7.3.23	7.3.23	2
Planen	8.3.23	10.3.23	
PSP Plan erstellen	8.3.23	9.3.23	5
SOLL	8.3.23	8.3.23	3
Daten Speicherung	8.3.23	9.3.23	5
SOLL			4
Aufbau vom Code	8.3.23	10.3.23	5
SOLL			3
Designe	10.3.23	10.3.23	3
SOLL			3
Entscheiden	13.3.23	14.3.23	
Sepicherung der Daten	13.3.23	14.3.23	5
SOLL			4
Aufbau vom Code	14.3.23	14.3.23	2
SOLL			1
Designe	14.3.23	14.3.23	3
SOLL			3
Realisieren	15.3.23	18.3.23	
Code Grundgerüst aufbauen	15.3.23	16.3.23	8
SOLL			5
Logik einbauen	15.3.23	18.3.23	13
SOLL			11
Designe Realisieren	17.3.23	18.3.23	3
SOLL			4
Test Protokol erstellen	18.3.23	18.3.23	4
SOLL			4
Kontrolieren	20.3.23	22.3.23	
Test Portokoll durchgehen	20.3.23	21.3.23	6
SOLL			4
Alle methoden Testen	21.3.23	22.3.23	5
SOLL			4
Alle anforgerungen durchgehen	22.3.23	22.3.23	4
SOLL			4
Auswerten	23.3.23	25.3.23	
Applikation erweiterbar	23.3.23	24.3.23	7
SOLL			7
Das Bestehende verbessern	24.3.23	25.3.23	3
SOLL			1
Dokumentation/ Presentation erstellen	24.3.23	25.3.23	4
SOLL			3

Total Stund SOLL:	72
Total Stund IST	93

Abbildung 3: GANT Ta

Daten Speicherung

In der Planungsphase musste ich eine Entscheidung darüber treffen, wie ich die Daten speichern möchte, damit der Benutzer seine Einträge nach dem Schliessen der App erneut sehen kann. Nach sorgfältiger Abwägung der Vor- und Nachteile der verschiedenen Optionen habe ich mich zwischen der lokalen Speicherung mit JSON oder XML oder der Nutzung einer Datenbank mit API entscheiden müssen.

Die lokale Speicherung mit JSON oder XML bietet den Vorteil, dass sie einfacher zu implementieren ist und keine zusätzlichen Kosten für Server- oder Datenbankzugriffe verursacht. Allerdings hat diese Option auch einige Einschränkungen, z.B. eine begrenzte Grösse der Datenmenge und eine eingeschränkte Möglichkeit zur Suche oder Filterung von Daten.

Die Nutzung einer Datenbank mit API eröffnet dagegen weitere Möglichkeiten, wie z.B. die Speicherung von grösseren Datenmengen und eine erweiterte Suche und Filterung. Allerdings erfordert diese Option auch mehr Aufwand bei der Implementierung und die zusätzlichen Kosten für Server- oder Datenbankzugriffe.

In der Entscheidungsphase werde ich diese Faktoren noch einmal sorgfältig abwägen und mich dann für die beste Option entscheiden.

Aufbau vom Code

In der Planungsphase habe ich den Aufbau des Codes für das Projekt geplant. Dazu habe ich mir zuerst einen Überblick über die Funktionen der App verschafft und diese in kleinere Komponenten und Module aufgeteilt. Neben einem Flussdiagramm, habe ich auch ein Use-Case-Diagramm, ein Sequenzdiagramm und ein UML-Klassendiagramm erstellt, um die Struktur der App besser zu visualisieren und die Abhängigkeiten zwischen den verschiedenen Elementen zu identifizieren. Um sicherzustellen, dass der Code gut strukturiert und einfach zu warten ist, habe ich Entwurfsmuster wie das Model-View-ViewModel-Muster berücksichtigt. Das Flussdiagramm dient als Grundlage für die Implementierung des Codes in der nächsten Phase, während die anderen Diagramme als Referenz während der Implementierung und als Dokumentation dienen.

Flussdiagramm

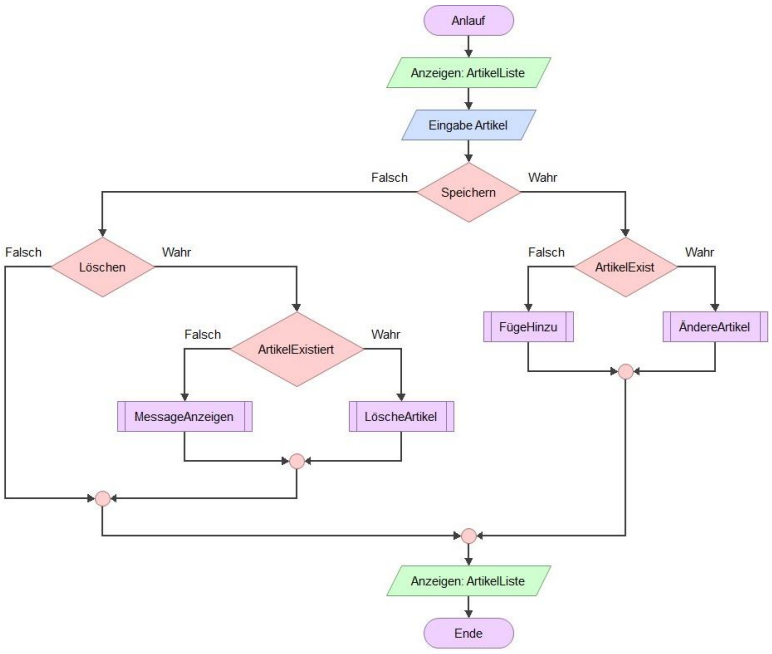


Abbildung 4: Flussdiagram

Das Programm gibt zuerst die Artikel-Liste aus und wartet dann auf eine Benutzereingabe. Wenn der Benutzer "Speichern" wählt, prüft das Programm, ob der Artikel bereits in der Liste vorhanden ist. Wenn ja, wird der Artikel aktualisiert, wenn nein, wird er zur Liste hinzugefügt. Wenn der Benutzer "Löschen" wählt, prüft das Programm, ob der Artikel in der Liste vorhanden ist. Wenn ja, wird der Artikel aus der Liste entfernt, wenn nicht, wird eine Meldung angezeigt. Schliesslich gibt das Programm die aktualisierte Liste aus.

Use-Case-Diagramm 1

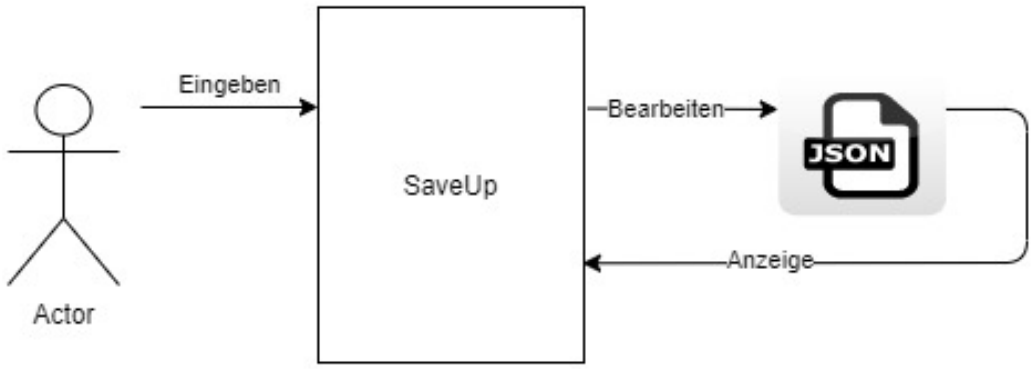


Abbildung 5: Use Case Diagramm 1

Hier wird ein einfacher Anwendungsfall gezeigt, der den Prozess beschreibt. Der Benutzer fügt über die Anwendung einen neuen Eintrag hinzu, der dann in einer JSON-Datei gespeichert wird. Der Eintrag wird anschliessend in der Anwendungsliste angezeigt.

Use-Case-Diagramm 2

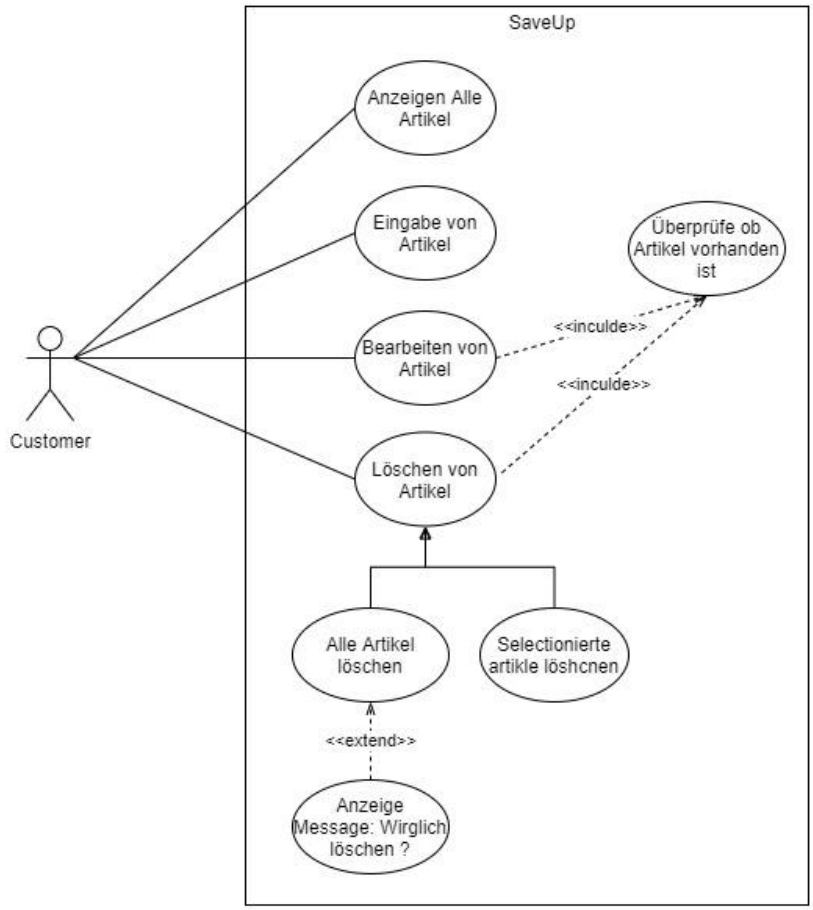


Abbildung 6: Use Case Diagramm 2

Hier wird das interne Prozessdiagramm der Anwendung dargestellt, in dem gezeigt wird, wie der Benutzer (Kunde) innerhalb der Anwendung arbeitet. Der Benutzer kann die Liste der Artikel anzeigen lassen, Artikel bearbeiten oder löschen. Wenn der Benutzer einen Artikel bearbeitet oder löscht, wird ein anderer Anwendungsfall aufgerufen, der überprüft, ob der Artikel vorhanden ist oder nicht.

Sequenz-Diagramm

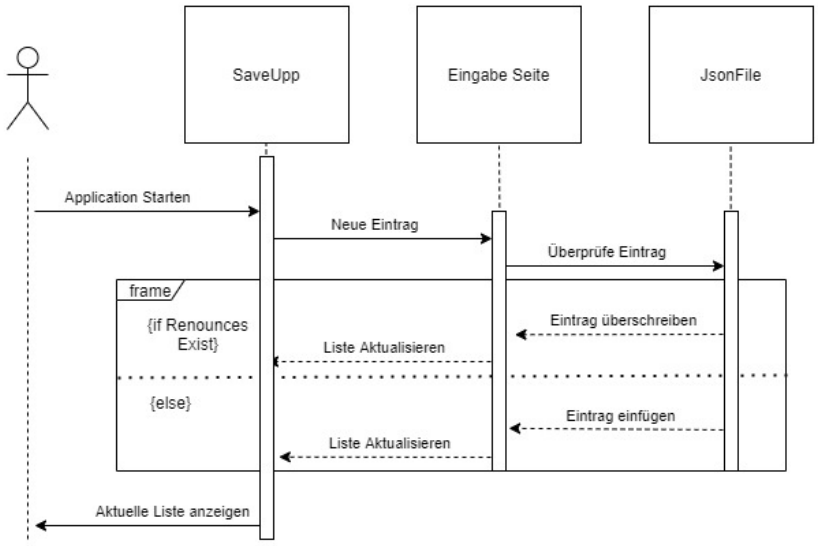


Abbildung 7: Sequenz Diagramm

Das Sequenzdiagramm zeigt den Speicherungsprozess zwischen drei Objekten: Saveup, Eingabeseite und Jsonfile. Der Prozess beginnt, wenn der Benutzer einen neuen Eintrag auf der Eingabeseite erstellt und auf die Schaltfläche "Speichern" klickt. Die Eingabeseite übergibt dann die Daten an das Saveup-Objekt, das den Speicherungsvorgang initiiert. Das Saveup-Objekt ruft dann das Jsonfile-Objekt auf, um die Daten in eine JSON-Datei zu schreiben. Wenn die Speicherung abgeschlossen ist, gibt das Saveup-Objekt eine Bestätigung an die Eingabeseite zurück, dass der Speicherungsvorgang erfolgreich abgeschlossen wurde. Das Sequenzdiagramm zeigt auch mögliche Fehlerzustände, wie z.B. das Fehlschlagen der Speicherung, und wie das System darauf reagiert. Insgesamt zeigt das Sequenzdiagramm, wie der Speicherungsprozess zwischen den verschiedenen Objekten innerhalb des Systems abläuft.

Klassen Diagramm

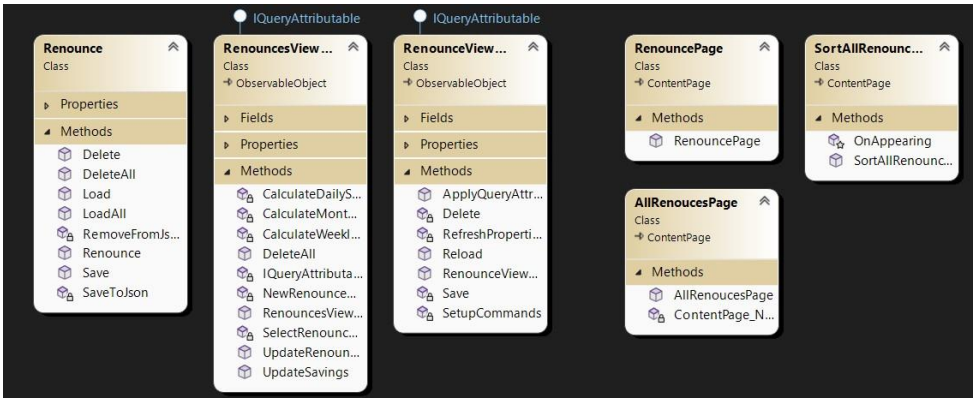


Abbildung 8: Klassen Diagramm

Die Klasse "Renounce" verfügt über eine Reihe von Methoden zum Speichern, Laden und Löschen von "Renounce"-Objekten. "RenouncesViewModel" ist ein ViewModel, das eine ObservableCollection von "RenounceViewModel" enthält und über Methoden zum Hinzufügen und Löschen von "Renounce"-Objekten verfügt. "RenounceViewModel" ist eine ViewModel-Klasse, die ein einzelnes "Renounce"-Objekt darstellt und Methoden zum Aktualisieren und Löschen des Objekts enthält.

Designe

In der Planungsphase musste ich auch das Design der App planen, um eine Vorstellung davon zu haben, wie sie aussehen sollte. Dazu habe ich ein sogenanntes Mockup erstellt, das ist so etwas wie ein Entwurf oder eine Skizze. Ich habe ein Programm namens Just in Mind verwendet, um das Mockup zu erstellen.

Das Mockup zeigt, wie die Benutzeroberfläche der App aussehen könnte. Es enthält die verschiedenen Elemente, wie Buttons, TabBar und Textfelder, die der Benutzer auf dem Bildschirm sehen wird, sowie die Anordnung dieser Elemente. Durch das Mockup konnte ich sehen, wie die App aussehen würde, bevor ich mit der Implementierung begann.

Das Mockup war auch hilfreich bei der Entscheidungsfindung, da ich durch das Visualisieren der App eine bessere Vorstellung davon hatte, welche Funktionen und Elemente ich implementieren sollte. So konnte ich besser entscheiden, welche Entscheidungen in der Implementierungsphase getroffen werden sollten.

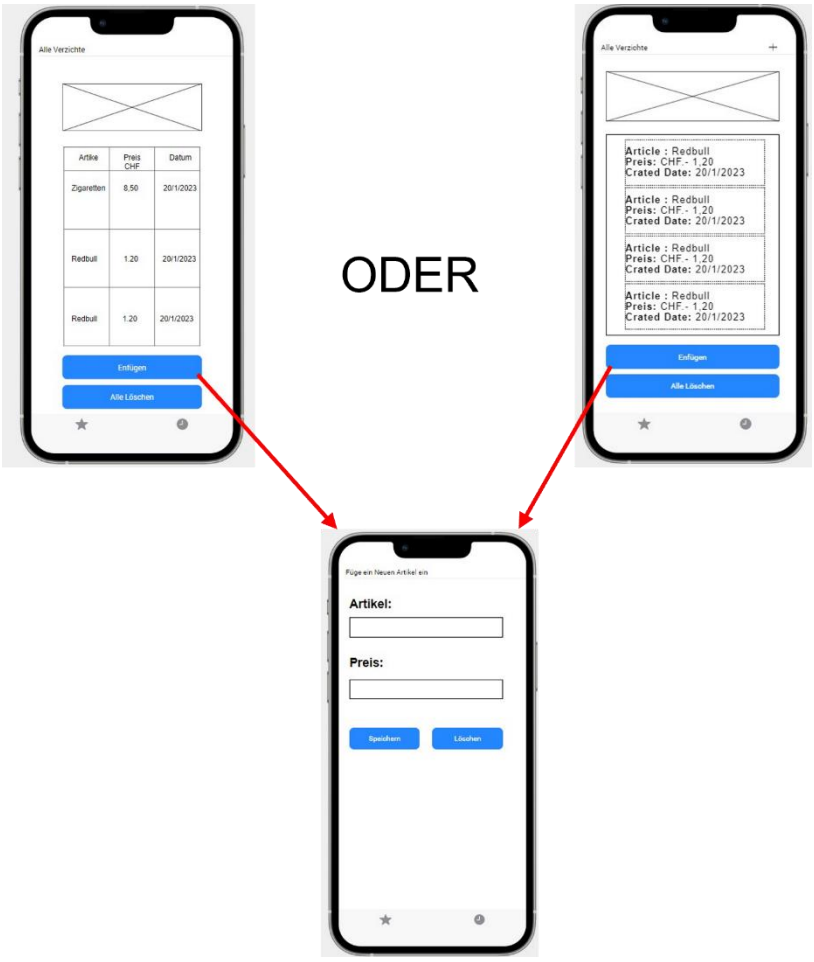


Abbildung 9: Moqup Designe

Entscheiden

In der Entscheidungsphase habe ich verschiedene Entscheidungsmethoden angewendet, um die zuvor identifizierten Optionen zu bewerten und die besten Entscheidungen zu treffen. Eine dieser Methoden waren Entscheidungsmatrizen, die mir halfen, die Optionen anhand von verschiedenen Kriterien zu bewerten und zu vergleichen.

Speicherung der Daten

Für die Entscheidung, wie die Daten gespeichert werden sollten (lokal oder DB mit API), habe ich eine Entscheidungsmatrix erstellt, die die Vor- und Nachteile jeder Option sowie die Gewichtung der Kriterien (z.B. Zuverlässigkeit, Skalierbarkeit, Wartbarkeit) berücksichtigte.

Speicherung der Daten					
Kriterien	Gewichtung	Lokal		DB mit API	
		Bewertung	Total	Bewertung	Total
Zuverlässigkeit	5	5	25	4	20
Skalierbarkeit	3	1	3	3	9
Wartbarkeit	3	3	9	3	9
Einfach umzusetzen	6	5	30	4	24
TOTAL			67		62

Tabelle 1: Speicherung der Daten Entscheidungsmatrix 1

Wie man sehen kann, habe ich mich für die Lokale Speicherung der Daten entschieden, da es in meiner Matrix am meisten Punkte bekommen hat.

XML Vs JSON

Nachdem ich eine Entscheidungsmatrix erstellt hatte, um zu bestimmen, wie die Daten gespeichert werden sollten, konnte ich zwischen zwei Optionen wählen: JSON oder XAML. Um die bestmögliche Wahl zu treffen, wog ich die Vor- und Nachteile jeder Option sowie die Gewichtung der Kriterien wie Zuverlässigkeit, Skalierbarkeit und Wartbarkeit ab. Hier sind die Vor- und Nachteile von JSON und XML.

Datei	**Vorteile**	**Nachteile**
XML	• Integrierte Unterstützung für Datentypen • Unterstützt komplexe Datenstrukturen • Leicht zu validieren und zu sichern	• Grössere Dateigrösse • Schwieriger zu lesen als JSON • Benötigt eine spezielle Serialisierungsbibliothek
JSON	• Einfach zu implementieren und zu lesen • Geringere Dateigrösse • Unterstützt viele Programmiersprachen	• Keine integrierte Unterstützung für Datentypen • nicht für komplexe Datenstrukturen verwendet werden • schwer zu validieren

Tabelle 2: Vor und Nachteil JSON/XML

Basierend auf diesen Vor- und Nachteilen habe ich mich entschieden, JSON zu verwenden, da es einfacher zu implementieren ist und die Dateigrösse geringer ist. Hier ist die Entscheidungsmatrix, die ich für diese Entscheidung verwendet habe.

Speicherung der Daten					
Kriterien	Gewichtung	JSON		XML	
		Bewertung	Total	Bewertung	Total
Zuverlässigkeit	5	5	25	4	20
Skalierbarkeit	3	2	6	2	6
Wartbarkeit	3	3	9	3	9
TOTAL			40		35

Tabelle 3: Speicherung der Daten Entscheidungsmatrix 2

Aufbau der Applikation

In ähnlicher Weise habe ich Entscheidungsmatrizen verwendet, um andere Entscheidungen im Projekt zu treffen, wie z.B. die Auswahl des Frameworks oder der Bibliotheken. Durch diese Methode konnte ich fundierte Entscheidungen treffen und sicherstellen, dass die ausgewählten Optionen den Anforderungen des Projekts entsprechen und die bestmöglichen Ergebnisse liefern.

Hier ist eine Entscheidungsmatrix mit Gewichtung und Bewertung, die ich bei der Wahl zwischen MAUI und Xamarin geholfen hat:

Optionen	Gewichtung	MAUI	Xamarin
Kompatibilität	30%	9/10	7/10
Performance	25%	8/10	7/10
Entwicklungszeit	20%	7/10	6/10
Community Support	15%	7/10	8/10
Dokumentation	10%	8/10	8/10

Tabelle 4: Entscheidungsmatrix Maui / Xamarin

Gewichtungssumme: 100%

Nach Bewertung jeder Option habe ich mich für MAUI entschieden, da es in fast allen Kategorien besser abgeschnitten hat als Xamarin. Insbesondere hat es eine bessere Kompatibilität, eine bessere Performance, und eine ähnliche Dokumentation. Obwohl die Entwicklungszeit etwas länger war, entschied ich mich dafür, da ich der Meinung war, dass die langfristigen Vorteile von MAUI die kurzfristigen Nachteile aufwiegen würden.

Designe

Wie bereits erwähnt, habe ich in der Entscheidungsphase zwei Mockups erstellt, zwischen denen ich mich entscheiden musste.

Das Mittlere Mockup (Siehe Abbildung X) zeigt eine einfache Seite zur Eingabe von Artikeln und Preisen. Obwohl dieses Design simpel ist, fiel es mir schwer, eine Alternative zu finden. Aus diesem Grund habe ich mich darauf konzentriert, die Artikelübersicht zu verbessern und in anderen Designs zusätzliche Funktionen für die Benutzerfreundlichkeit zu implementieren. Im Folgenden habe ich eine Entscheidungsmatrix erstellt, in der ich die Vor- und Nachteile beider Designs aufgelistet habe, um die richtige Entscheidung zu treffen.

Designe Nummer 1:

Designe Nummer 2

Designe					
Kriterien	Gewichtung	Designe 1		Designe 2	
		Bewertung	Total	Bewertung	Total
Übersicht der Artikel	4	4	16	4	16
Änderung eines Artikels	5	3	15	4	20
Intuitiv	5	4	20	4	20
TOTAL			51		56

Tabelle 5. Designe Entscheidungsmatrix

Wie man sehen kann, habe ich mich für Design Nummer 2 entschieden, da es in meiner Matrix am meisten Punkte bekommen hat.

Realisieren

Nach Abschluss der Planung und Beschaffung aller notwendigen Ressourcen für mein Projekt, habe ich die Realisierungsphase begonnen und mit dem Programmieren gestartet. Hierbei habe ich zuerst eine grundlegende Struktur erstellt und anschliessend die Feinheiten programmiert (einschliesslich der Integration von Loic).

Dieses Projekt wurde mit MAUI .NET 7.0 entwickelt und benötigte folgende NuGet-Pakete (inklusive Versionen):

- CommunityToolkit.Mvvm

Die Entwicklung erfolgte mit Visual Studio 17.4.4 auf einem Rechner mit Microsoft Windows 10

Grundgerüst

In der Realisierungsphase habe ich damit begonnen, den Code für das Projekt zu implementieren. Dazu habe ich zuerst drei Ordner erstellt: ein Ordner für das Model mit einer Renounce-Klasse, ein Ordner für die View mit drei XAML-View-Klassen und ein Ordner für das ViewModel mit zwei Klassen RenouncesViewModel und RenounceViewModel

Projekterstellung: Ich habe ein neues MAUI.NET Projekt in Visual Studio erstellt.

Model Ordner: Datenmodelle als Klassen definiert mit den benötigten Eigenschaften und Attributen.

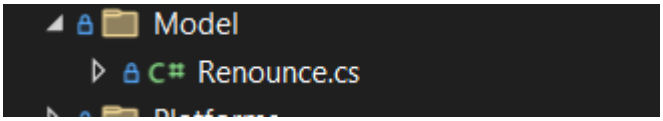


Abbildung 10: Models Ordner

View-Schicht: Die View sind für die Darstellung der Benutzeroberfläche zuständig.

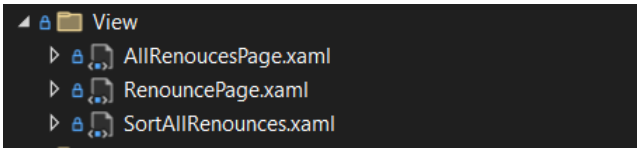


Abbildung 11: View Ordner

ViewModel-Schicht: Das ViewModel verbindet diese beiden Komponenten und dient als Vermittler zwischen ihnen.

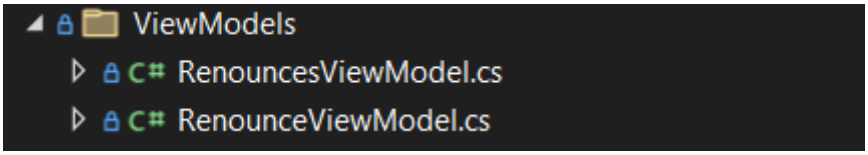


Abbildung 12:ViewModel Ordner

Logik einbauen

Im Einzelnen sieht das Vorgehen folgendermassen aus:

Integration der IQueryAttributable-Schnittstelle: Dieses Interface ermöglicht es, bestimmte Abfragen auf die Daten durchzuführen und entsprechende Änderungen in der Anwendung vorzunehmen. Beispielsweise wird über das "deleted"-Attribut ein Eintrag aus der Liste der Einträge gelöscht und über das "saved"-Attribut ein Eintrag aktualisiert oder hinzugefügt. Das Interface ermöglicht es also, die Anwendung auf einfache Weise zu steuern und zu aktualisieren, ohne den Code jedes Mal ändern zu müssen.

```
void IQueryAttributable.ApplyQueryAttributes(IDictionary<string, object> query)
{
    if (query.ContainsKey("deleted"))
    {
        string renounceId = query["deleted"].ToString();
        RenounceViewModel matchedRenounce = AllRenounce.Where((n) => n.Identifier == renounceId).FirstOrDefault();

        // If resource exists, delete it
        if (matchedRenounce != null)
            AllRenounce.Remove(matchedRenounce);
    }
```

Abbildung 13: IQueryAttributable-Schnittstelle

Speicherung der Daten: In der Realisierungsphase habe ich mich darum gekümmert, wie ich die Daten speichern werde, damit der Benutzer seine Einträge auch nach dem Schliessen der App wiedersehen kann. Wie bei der Entscheidung Phase schon erwähnt habe ich mich für die Speicherung der Daten in einem JSON-Format entschieden.

Dazu habe ich eine entsprechende Klasse erstellt, die die notwendigen Methoden zum Lesen und Schreiben von JSON-Dateien enthält. Die Daten werden dann in einem entsprechenden Ordner auf dem Gerät des Benutzers gespeichert.

```
/// <summary>
/// Speichert eine Liste von Renounce-Objekten in der JSON-Datei
/// </summary>
private void SaveToJson()
{
    List<Renounce> renounceList = LoadAll().ToList();
    Renounce existingRenounce = renounceList.FirstOrDefault(r => r.Filename == this.Filename);
    if (existingRenounce != null)
    {
        // Aktualisiere den vorhandenen Eintrag
        existingRenounce.Date = this.Date;
        existingRenounce.Preis = this.Preis;
    }
    else
    {
        // Füge den neuen Eintrag hinzu
        renounceList.Add(this);
    }
    string json = JsonSerializer.Serialize(renounceList);
    File.WriteAllText(Path.Combine(FileSystem.AppDataDirectory, "renounceList.json"), json);
}
```

Abbildung 14: Methode Speicherung in Json

Durch die Verwendung von JSON als Speicherformat kann die App auch in der Zukunft einfach erweitert werden, da das Format universell lesbar ist und die Daten leicht exportiert und importiert werden können.

BindingContext:

Ich habe das Data Binding im XAML-Code implementiert. Dabei habe ich die Verbindung zwischen der Benutzeroberfläche (View) und den Daten (ViewModel) hergestellt, sodass Änderungen in den Daten automatisch auf der Benutzeroberfläche aktualisiert werden. Das bedeutet, dass wenn ich eine Änderung an den Daten im ViewModel vornehme, diese Änderung automatisch auf der Benutzeroberfläche dargestellt wird. Das vereinfacht das Aktualisieren der Benutzeroberfläche und stellt sicher, dass die Daten immer auf dem neuesten Stand sind.

```
<ContentPage.BindingContext>
    <viewModels:RenouncesViewModel />
</ContentPage.BindingContext>
```

Abbildung 15: BindingContext

Designe

Hier wird ein Codeausschnitt gezeigt und erklärt, wie das Design erstellt wurde und wie es funktioniert. Diese Implementierung von Styles wurde auch auf andere Klassen (Views) angewendet.

```
<ContentPage.Resources>
    <ResourceDictionary>
        <Style TargetType="Button">
            <Setter Property="Margin"
                    Value="5" />
            <Setter Property="BackgroundColor"
                    Value="#0062a3" />
            <Setter Property="TextColor"
                    Value="White" />
            <Setter Property="CornerRadius"
                    Value="20" />
            <Setter Property="FontFamily"
                    Value="Segoe UI" />
            <Setter Property="FontSize"
                    Value="16" />
        </Style>
    </ResourceDictionary>
</ContentPage.Resources>
```

Abbildung 16: Style in XAML

Der Codeausschnitt zeigt, wie ich ein Style-Objekt erstellt haben, das auf alle Buttons in die ContentPage angewendet wird. Mit dem TargetType-Attribut geben ich an, dass dieser Stil auf Buttons angewendet werden soll. Die Setter-Elemente in dem Style-Objekt legen verschiedene Eigenschaften des Buttons fest, wie z.B. die Margin, die Hintergrundfarbe, die Textfarbe, die Eckenradius, die Schriftart und die Schriftgrösse.

Das Hinzufügen dieses Style-Objekts in die ResourceDictionary der ContentPage macht es für alle Buttons auf dieser Seite zugänglich. Ich kann diesen Stil nun auf jeden Button anwenden, indem ich den Style-Attribut verwenden und den Wert auf den Namen des Stils setzen, den ich erstellt habe. Dies erleichtert das Designen meine Anwendung, da ich denselben Stil an mehreren Stellen verwenden kann, ohne ihn immer wieder neu definieren zu müssen.

Test Protokoll erstellt

Use Case	Soll	Ist	Problem	Lösung
User kann Neuen Artikel eingeben	Artikel soll gespeichert werden	Artikel wird gespeichert		-
User kann Preis Eingeben	Preis soll gespeichert werden	Preis wird gespeichert		-
User kann einen Artikel Speichern	Artikel soll nach dem Speichern in der Liste angezeigt werden	Artikel wird nach dem Speichern in der Liste angezeigt		-
Das aktuelle Datum wird automatisch bei eintragen des Artikel gespeichert	Aktuelle Datum soll automatisch eingetragen	Aktuelle Datum wird automatisch eingetragen		-
User kann ein Artikel bearbeiten	Artikel soll bearbeitbar sein und gespeichert	Artikel kann bearbeitet werden und Gespeichert	Wird nicht direkt auf beide Seite Aktualisiert	Gelöst
User kann ein Artikel löschen	Artikel soll in der Liste nichtmehr vorhaben sein	Artikel ist in der Liste nichtmehr vorhanden		
User kann alle Artikel Löschen	Alle Artikel sollen gelöscht werden	Alle Artikel werden gelöscht		
User soll beim wiederöffnen der App alle Artikel widersehen können	Alle Artikel sollen beim wiederöffnen der App wider angezeigt werden	Alle Artikel werden beim Neustart wider angezeigt		-

Tabelle 6: Test Protokoll Tabelle:

In dieser Tabelle wurden die Testergebnisse für verschiedene Anforderungen an ein Maui-Projekt aufgeführt. Die Tabelle besteht aus vier Spalten, die den Use Case, die Soll-Anforderung, die Ist-Anforderung und mögliche Probleme oder Lösungen beschreiben.

Die Anforderungen waren unter anderem, dass der User einen neuen Artikel eingeben, einen Preis eingeben, einen Artikel speichern und bearbeiten sowie Artikel löschen und alle Artikel löschen können sollte. Des Weiteren sollte das aktuelle Datum automatisch beim Eintragen des Artikels gespeichert werden und beim Wiederöffnen der App sollten alle Artikel wieder angezeigt werden.

Die Testergebnisse zeigten, dass alle Anforderungen erfüllt wurden, abgesehen von einem Problem bei der Aktualisierung des bearbeiteten Artikels auf beiden Seiten. Dieses Problem konnte jedoch erfolgreich gelöst werden.

Insgesamt wurden alle Anforderungen erfolgreich umgesetzt und das Maui-Projekt konnte alle wichtigen Funktionen bereitstellen, um den Usern eine reibungslose Nutzung zu ermöglichen.

Kontrollieren

Dieser Prozess beanspruchte die meiste Zeit, da es das Testen der Anwendung beinhaltete. Es gab ständig Fehler, die ich beheben musste. Ich verwendete den Debugger, um die Probleme zu lokalisieren. Ich überprüfte, ob alle Projektanforderungen erfüllt waren, und ob die gesamte Logik funktionierte. Schliesslich das Test Protokoll durchgegangen um sicher zu stellen das alle Anforderungen erfüllt waren.

Alle Methoden Testen

Die SaveUp-App enthält verschiedene Methoden zur Verarbeitung von Daten und zur Interaktion mit der Benutzeroberfläche. Im Folgenden werden die wichtigsten Methoden beschrieben und erklärt, wie sie getestet werden.

LoadAll

Die LoadAll-Methode lädt alle Einträge aus der JSON-Datei und gibt sie als IEnumerable zurück. Um diese Methode zu testen, erstelle ich eine JSON-Datei mit einigen Einträgen und rufe die LoadAll-Methode auf. Dann überprüfe ich, ob die zurückgegebene Liste alle erwarteten Einträge enthält

SaveToJson

Die SaveToJson-Methode fügt einen neuen Eintrag zur JSON-Datei hinzu. Um diese Methode zu testen, rufe ich die SaveToJson-Methode mit einem neuen Eintrag auf und überprüfe, ob dieser in der JSON-Datei gespeichert wurde. Ich teste auch, ob die Methode die Datei korrekt erstellt, wenn sie noch nicht vorhanden ist.

DeleteFromJson

Die DeleteFromJson-Methode entfernt einen Eintrag aus der JSON-Datei. Um diese Methode zu testen, erstelle ich eine JSON-Datei mit einigen Einträgen und rufe die DeleteFromJson-Methode mit einem Eintrag auf, der aus der Liste entfernt werden soll. Dann überprüfe ich, ob dieser Eintrag nicht mehr in der Datei vorhanden ist.

SortByCategory

Die SortByCategory-Methode sortiert die Einträge nach Kategorie. Um diese Methode zu testen, erstelle ich eine Liste mit Einträgen und sortiere sie mit der SortByCategory-Methode. Dann überprüfe ich, ob die Einträge korrekt nach Kategorie sortiert wurden.

SortByDate

Die SortByDate-Methode sortiert die Einträge nach Datum. Um diese Methode zu testen, erstelle ich eine Liste mit Einträgen und sortiere sie mit der SortByDate-Methode. Überprüfe, ob die Einträge korrekt nach Datum sortiert wurden.

AddRenounce

Die AddRenounce-Methode fügt einen neuen Eintrag zur Liste hinzu. Um diese Methode zu testen, rufe ich die AddRenounce-Methode mit einem neuen Eintrag auf und überprüfe, ob dieser korrekt zur Liste hinzugefügt wurde. Dann überprüfe ich auch, ob die Benutzeroberfläche aktualisiert wurde, um den neuen Eintrag anzuzeigen.

Anforderungen erfüllt

Ich habe den Fortschritt des Projekts regelmässig überprüft, um sicherzustellen, dass alle Anforderungen erfüllt werden. Trotzdem habe ich manchmal den Fokus verloren und Zeit für unerforderliche Aufgaben verschwendet. Am Ende konnte ich jedoch alle Anforderungen erfüllen und stellte sogar fest, dass ich zwei zusätzliche Anforderungen erfüllt hatte.

Auswerten

Eines der Dinge, die ich verbessern könnte, ist die Zeitmanagement, um eine bessere Projektplanung zu erreichen und meine Fähigkeiten zu verbessern, um Herausforderungen zu bewältigen. Ich denke auch daran, wie ich mein Code modular gestalten und einfacher lesbar machen kann, um in Zukunft schnellere Fehlerbehebungen und Wartung zu ermöglichen. Eine weitere Möglichkeit wäre, die Benutzerfreundlichkeit der App zu verbessern, indem ich eine intuitive und einfach zu verwendende Schnittstelle bereitstelle, die Benutzer ermöglicht, mit der Applikation effektiver zu arbeiten.

Schlussfolgerung

Ich bin zu dem Schluss gekommen, dass diese Arbeit sehr fordernd war und erfordert viel Geduld und Fachwissen. In Zukunft werde ich früher mit dem Projekten beginnen und eine gezielte Arbeitsweise anstreben. Ein Aspekt, den ich leider nicht wie geplant umsetzen konnte, ist das effektive Dokumentieren meiner Arbeit. Eine Vorlage zu erstellen, um das Dokumentieren von Anfang an in Angriff zu nehmen, wäre ein guter Ansatz gewesen, um meine Arbeit besser zu organisieren.

Zukunft der Applikation

In der Zukunft gibt es einige Verbesserungen, die für die SaveUp-App geplant sind. Eine der ersten Änderungen wird das Aktualisieren des Icons sein, um es moderner und ansprechender zu gestalten.

Ein weiterer wichtiger Punkt ist die Speicherung der Daten auf einer externen Datenbank mit einer API. Dies wird nicht nur die Sicherheit der Daten erhöhen, sondern auch eine einfachere und zuverlässigere Möglichkeit bieten, auf die Daten zuzugreifen, insbesondere wenn mehrere Benutzer auf die App zugreifen.

Zusätzlich soll es dem Benutzer möglich sein, ein Zielbetrag einzugeben und in Echtzeit anzuzeigen, wie viel Prozent er bereits erreicht hat. Dies wird ihm helfen, seine Finanzen besser zu verwalten und motiviert ihn, weiter zu sparen.

Ein weiterer wichtiger Aspekt ist die Möglichkeit, die Liste nach Art des Artikels zu sortieren. Dies wird den Benutzern helfen, ihre Ausgaben besser zu kategorisieren und den Überblick zu behalten.

Insgesamt werden diese Verbesserungen dazu beitragen, die SaveUp-App noch nützlicher und benutzerfreundlicher zu machen und den Benutzern ein noch besseres Erlebnis zu bieten.

Fazit

Ich habe bei diesem Projekt viel Freude empfunden und unglaublich viel gelernt. Das Schreiben von C#-Programmen war für mich begeisternd und die erfolgreiche Umsetzung meines Codes hat mich immer wieder motiviert.

Leider hatte ich nicht genug Zeit, um das Projekt noch weiter zu verbessern und alle zusätzlichen Anforderungen zu erfüllen. Dennoch war es für mich eine grosse Herausforderung, da wir in kurzer Zeit viel lernen und umsetzen mussten. Der Umfang des Projekts war die grösste Herausforderung für mich. Trotz allem liebe ich C# und werde auch weiterhin an diesem Projekt arbeiten.

C# Macht Spass!

Quellenverwies

- https://docs.microsoft.com/en-us/dotnet/multi-platform/maui/
- https://maui-community.github.io/
- https://github.com/dotnet/maui
- https://www.youtube.com/c/dotNETMAUI
- https://www.telerik.com/support/maui-ui
- https://uxwing.com/svg-icon-editor/
- https://stackoverflow.com/questions/65106633/creating-first-net-maui-project