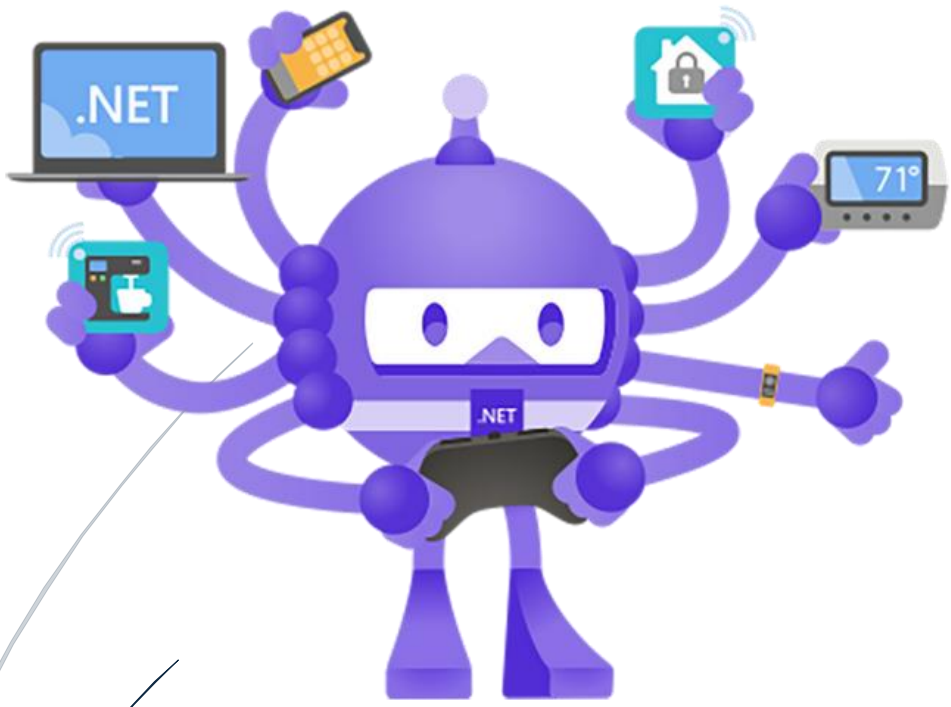
12.5.2025

SaveUp-App Dokumentation

ICT-Modul 322



Akshai Arulananthan und Even Stuck

Inhaltsverzeichnis

1 Informieren

Im Rahmen des ICT Moduls 322 – Benutzerschnittstellen entwerfen und implementieren – war es unsere Aufgabe, eine mobile App zu entwickeln, mit der Benutzer ihre Spareinträge verwalten können. Die offizielle Prüfungsaufgabe wurde uns über ein GitHub-Repository zur Verfügung gestellt. Wir haben sie gründlich gelesen und analysiert, wobei wir von Anfang an darauf geachtet haben, welche Bewertungskriterien erfüllt werden müssen, um die maximale Punktzahl zu erreichen.

1.1 Ausgangslage

Die Ausgangslage sah vor, eine plattformübergreifende Anwendung mit .NET MAUI zu erstellen. Die App sollte in der Lage sein, Benutzereingaben entgegenzunehmen, sie zu verarbeiten, lokal zu speichern und in einer benutzerfreundlichen Oberfläche darzustellen. Zusätzlich wurde eine sprachliche Umstellung der Benutzeroberfläche zwischen Deutsch und Englisch verlangt. All dies war Teil einer praktischen Modulprüfung, die sowohl die technische Umsetzung als auch die Dokumentation und Präsentation des Projekts umfasste.

1.2 Ziele

Unser Ziel war es, eine einfach gehaltene, aber funktionale Anwendung zu entwickeln, die durch ein klares und übersichtliches Design überzeugt. Die App sollte Eingaben wie Beschreibung, Betrag und Datum erfassen können, die Gesamtersparnis automatisch berechnen und die Daten in einer lokalen JSON-Datei speichern. Besonderen Wert legten wir auch auf die Umsetzung der Mehrsprachigkeit, die über einen einfachen Sprachumschalt-Button realisiert wurde. Die technische Struktur des Projekts richteten wir konsequent nach dem MVVM-Entwurfsmuster aus, um eine saubere Trennung zwischen Benutzeroberfläche, Logik und Datenmodell sicherzustellen. Abschliessend war es uns wichtig, die gesamte Entwicklung nach dem IPERKA-Modell zu dokumentieren und das Projekt in einer strukturierten Präsentation vorzustellen.

2 Planen

Wir haben uns als Zweierteam organisiert und die Aufgabe in einzelne Arbeitspakete aufgeteilt. Dabei haben wir zuerst grob festgelegt, was alles gemacht werden muss (z. B. GUI, Logik, Sprache umschalten, Daten speichern).

2.1 Wireframes

Dann haben wir **Wireframes** gezeichnet, um das Aussehen und den Aufbau der App zu planen. So konnten wir uns vorstellen, wie die Benutzeroberfläche später aussehen soll.

Abbildung 1 Mockups

2.2 Gantt Planung

Danach haben wir die Aufgaben mit realistischen Zeiten in einem **Gantt-Diagramm** eingeplant. So war klar, wer was macht und bis wann. Für die Dokumentation, das Testen und die Präsentation haben wir ebenfalls genug Zeit eingeplant.

AUFGABE	ZUGEWIESEN AN	SOLL (Min)	IST (Min)	FORTSCHRITT	START	ENDE
Aufgabe durchlesen	Beide	10	10	100%	9.5.25	9.5.25
GANTT Plan erstellen	Akshai	45	60	100%	9.5.25	9.5.25
GANTT anschauen und evt. anpassen	Even	5	10	100%	9.5.25	9.5.25
Mockups zeichnen	Akshai	30	30	100%	9.5.25	9.5.25
Neues Projekt in .NET MAUI erstellen	Even	5	5	100%	10.5.25	10.5.25
GitHub Repository erstellen	Even	5	4	100%	10.5.25	10.5.25
MVVM Struktur aufbauen	Even	30	30	100%	10.5.25	10.5.25
Navigation zwischen Content Pages umsetzen	Even	30	10	100%	10.5.25	10.5.25
Eingabemaske erstellen (Kurzbeschreibung, Preis)	Even	30	10	100%	10.5.25	10.5.25
Datenbindung zwischen UI und ViewModel einricht	Even	20	30	100%	10.5.25	10.5.25
Datum/Zeit automatisch setzen	Even	20	20	100%	10.5.25	10.5.25
Produkt speichern (Speichern-Button)	Even	15	15	100%	10.5.25	10.5.25
Liste anzeigen mit allen erfassten Produkten	Akshai	30	30	100%	11.5.25	11.5.25
Listendesign / Layout gestalten	Akshai	15	15	100%	11.5.25	11.5.25
Gesamtsumme der Ersparnisse berechnen und anz	Even	30	20	100%	11.5.25	11.5.25
Einträge lokal speichern (JSON oder XML)	Even	30	20	100%	11.5.25	11.5.25
Daten beim Start laden	Even	20	30	100%	11.5.25	11.5.25
Eingaben überprüfen und Speichern absichern	Even	10	20	100%	11.5.25	11.5.25
1. Optionale Anforderung	Akshai	40	20	100%	11.5.25	11.5.25
2. Optionale Anforderung	Akshai	40	100	100%	11.5.25	11.5.25
Kontrollieren ob wir alles haben	Beide	20	20	100%	11.5.25	11.5.25
Testing der eigenen Funktionen und aufschreiben	Beide	30	30	100%	11.5.25	11.5.25
Dokumentation schreiben	Akshai	60	120	100%	11.5.25	11.5.25
Dokumentation anschauen + ergänzen	Even	15	15	100%	11.5.25	11.5.25
Präsentation vorbereiten	Akshai	30	15	100%	11.5.25	11.5.25
Präsentation evt. ergänzen	Even	5	5	100%	12.5.25	12.5.25
Präsentation besprechen	Beide	5	2	100%	12.5.25	12.5.25
Projektabgaben	Beide	15	15	100%	12.5.25	12.5.25
END ZEIT		640	711			

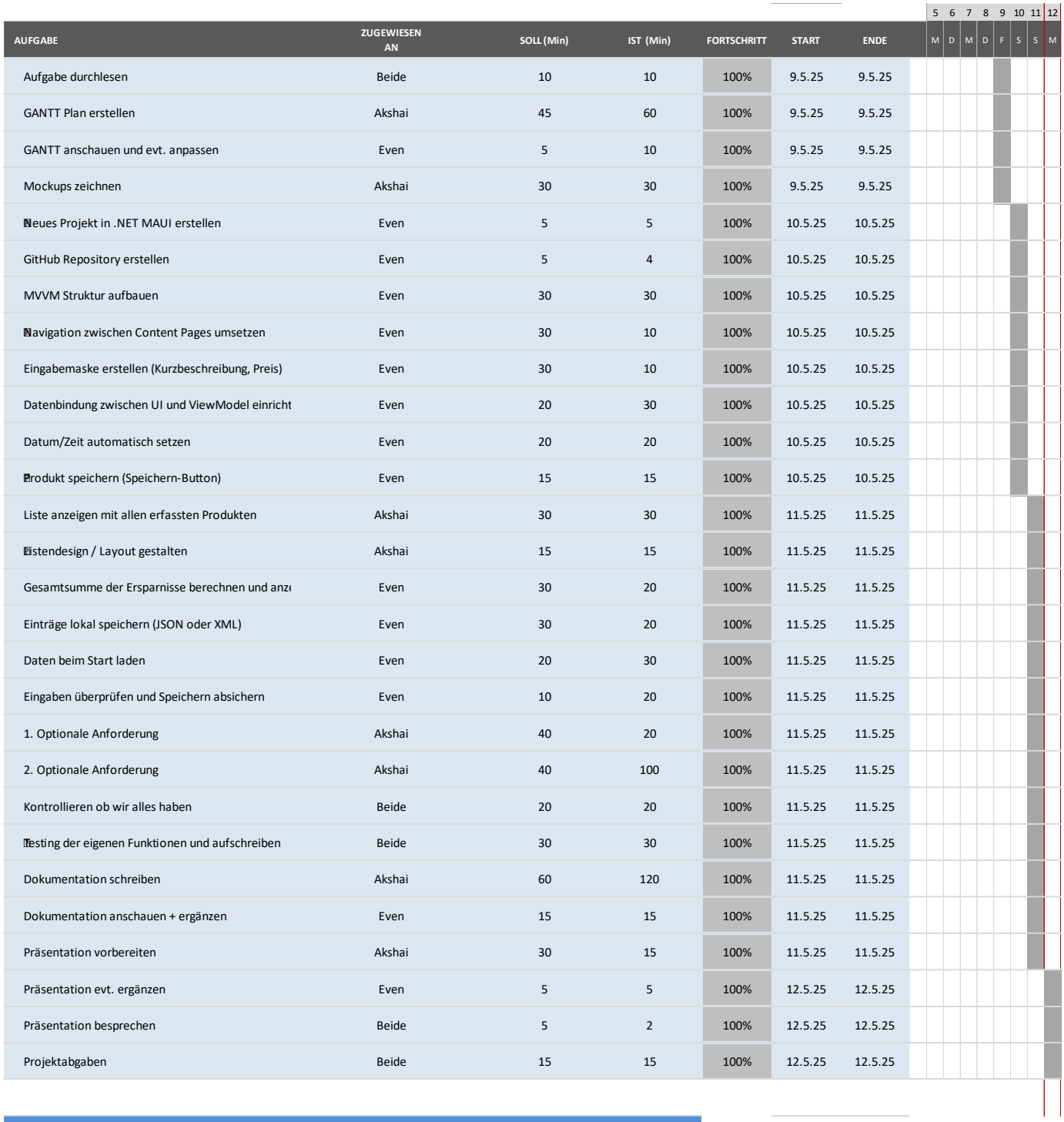
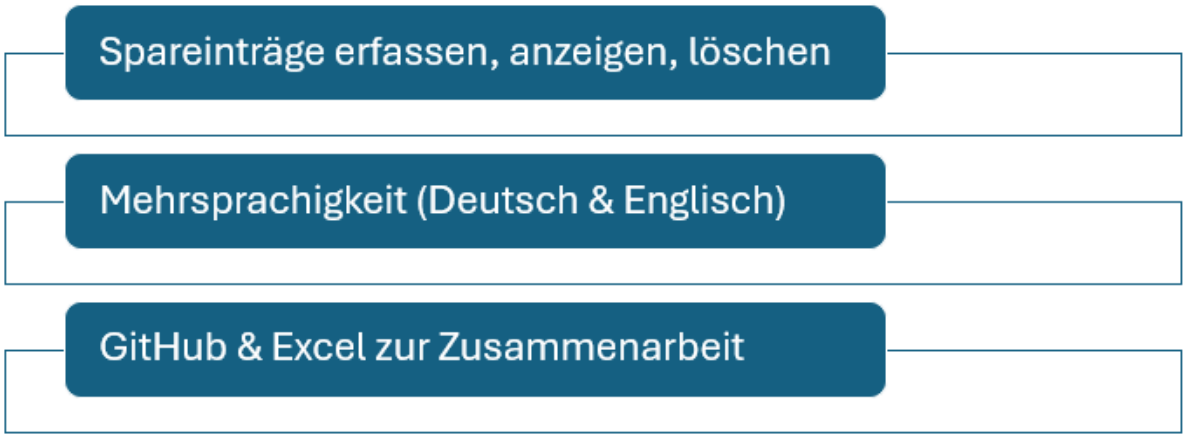


Abbildung 2 Gantt-Diagramm

3 Entscheiden

Nach dem gemeinsamen Durchlesen und Verständnis der Aufgabenstellung auf GitHub entschieden wir uns, eine mobile App mit **.NET MAUI** zu entwickeln, da wir bereits erste Erfahrungen damit hatten und sie uns eine plattformübergreifende Entwicklung (Android, Windows) ermöglichte.

Wir legten gemeinsam fest:



4 Realisieren

Zu Beginn der Umsetzungsphase wurde ein neues .NET MAUI-Projekt in Visual Studio erstellt. Parallel dazu wurde ein zentrales GitHub-Repository eingerichtet, um den Code strukturiert zu versionieren und kollaborativ daran zu arbeiten.

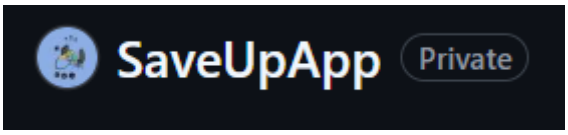


Abbildung 3 GitHub-Repository

Die Projektstruktur wurde strikt nach dem MVVM-Entwurfsmuster aufgebaut:

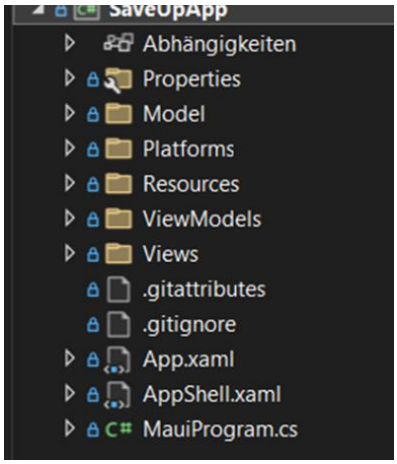


Abbildung 4 Projektstruktur

- **Model:** Die Datei SaveItem.cs enthält die Datenstruktur für jeden Spareintrag mit Beschreibung, Betrag und Datum.
- **ViewModels:** Enthält die zentrale Logik für das Programm. Hier wurden z. B. SaveUpViewModel.cs für die Verarbeitung der Eingaben und ShellViewModel.cs für dynamische Tab-Benennungen erstellt.
- **Views:** Die Benutzeroberfläche wurde in drei Seiten aufgeteilt: Übersicht, Eintrag hinzufügen und Liste. Jede View ist per DataTemplate an einen Tab im AppShell.xaml gebunden.
- **Resources:** Es wurden Ressourcen-Dateien für Deutsch und Englisch (.resx) erstellt, um eine mehrsprachige App zu ermöglichen. Ausserdem wurden Fonts, Styles und Icons eingebunden.

4.1 Schrittweise Umsetzung

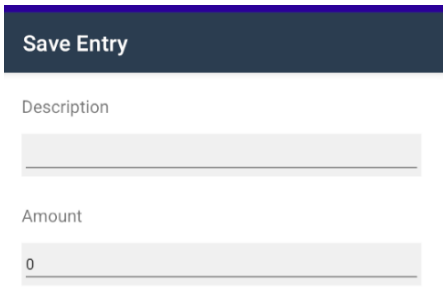


Abbildung 5 Eingabemaske

Eingabemaske (ContentPage):
Eine Seite wurde erstellt, auf der Benutzer eine Beschreibung und einen Betrag eingeben können.

Speicherfunktion:
Ein Klick auf „Eintrag speichern“ erzeugt ein neues SaveItem, das automatisch in eine ObservableCollection (Items) eingefügt wird. Diese Sammlung ist direkt mit der Liste in der Oberfläche verbunden.

Abbildung 6 Speicherbutton

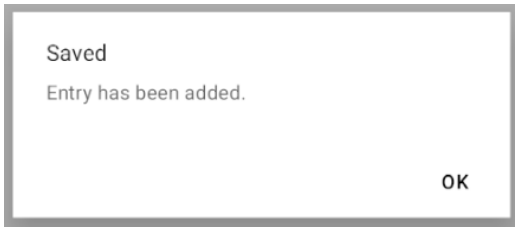


Abbildung 7 Speicher-Alert

Gesamtsumme berechnen:
Eine Methode summiert alle erfassten Beträge in der Liste und zeigt sie zentral an. Diese Funktion wird automatisch aufgerufen, wenn sich die Liste verändert.

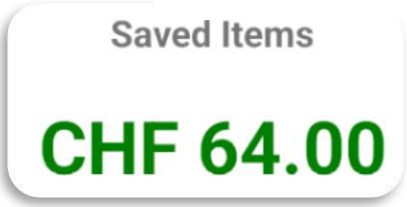


Abbildung 8 Gesamtsumme

Automatische Datumszuweisung:
Jeder Eintrag erhält beim Speichern automatisch das aktuelle Datum (DateTime.Now). Es gibt kein separates Eingabefeld dafür.

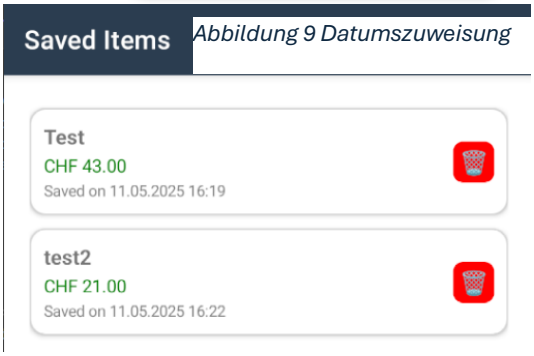


Abbildung 9 Datumszuweisung

Datenpersistenz (lokal):
Alle Einträge werden automatisch in einer .json-Datei gespeichert, sobald ein Eintrag hinzugefügt oder gelöscht wird. Beim App-Start wird die Datei gelesen und der Inhalt wieder in die Liste geladen.

Mehrsprachigkeit:
Alle Texte wie Button-Beschriftungen, Labels und Hinweise wurden in Ressourcen-Dateien ausgelagert. Beim Klick auf „Deutsch“ oder „Englisch“ wird die Sprache dynamisch umgeschaltet. Dazu werden die Texte im ViewModel und in der AppShell aktualisiert.

Abbildung 10 Mehrsprachigkeit

Struktur & Stil:
Es wurden einfache, aber einheitliche Styles für Buttons, Hintergründe und Texte verwendet. Die Navigation zwischen den Seiten erfolgt über Tabs mit eigenen Icons.

Abbildung 11 Icon

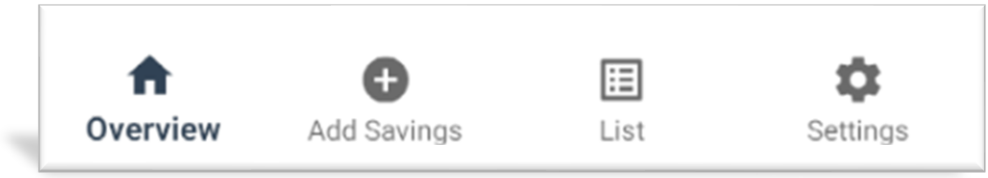


Abbildung 12 Navigation

5 Kontrollieren

Nach der Realisierung der Anwendung wurden alle Funktionen umfassend getestet. Dabei ging es darum sicherzustellen, dass sowohl Pflichtanforderungen als auch optionale Anforderungen korrekt umgesetzt wurden und stabil funktionieren.

Zuerst haben wir überprüft, ob **alle Aufgaben aus unserem Gantt-Plan** erledigt wurden. Die Aufgabenliste wurde Punkt für Punkt kontrolliert. So stellten wir sicher, dass kein Teil vergessen wurde – weder funktionale Features noch visuelle Details wie Styles oder Icons.

5.1 Testplan und Protokoll

Zusätzlich haben wir **einen strukturierten Testplan** erstellt, um die Funktionen systematisch zu überprüfen. Dabei haben wir die App auf dem **Android-Emulator** getestet. Alle Tests wurden **manuell** durchgeführt.

Tabelle 1 Testplan und Protokoll

Test-Nr.	Was wird getestet?	Erwartetes Verhalten	Tatsächliches Ergebnis	Ergebnis
T1	Neues Produkt erfassen (Beschreibung + Betrag)	Produkt erscheint in der Liste	Funktioniert wie erwartet	Bestanden
T2	Gesamtsumme wird korrekt berechnet	Neuer Betrag wird zur Gesamtsumme hinzugefügt	Wird korrekt berechnet	Bestanden
T3	Eingaben sind leer → Fehleranzeige	Alert "Description cannot be empty" erscheint	Alert erscheint	Bestanden
T4	Betrag ist 0 oder negativ → Fehleranzeige	Alert "Amount must be greater than 0" erscheint	Alert erscheint	Bestanden
T5	Liste löschen über "Clear All"-Button	Liste wird geleert, Gesamtsumme wird 0	Funktioniert	Bestanden
T6	Sprache wechseln zu Englisch	Texte auf Buttons und Tabs ändern sich auf Englisch	Funktioniert	Bestanden
T7	Sprache wechseln zu Deutsch	Texte auf Buttons und Tabs ändern sich auf Deutsch	Funktioniert	Bestanden

Test-Nr.	Was wird getestet?	Erwartetes Verhalten	Tatsächliches Ergebnis	Ergebnis
T8	App schliessen und erneut öffnen	Daten aus letzter Sitzung sind noch da	Produkte bleiben gespeichert	Bestanden
T9	Anzeige: Übersicht, Sparen, Liste	Alle Tabs funktionieren korrekt	Navigation funktioniert	Bestanden
T10	Falsche Eingaben (nur Leerzeichen etc.)	Alert erscheint	Wird abgefangen	Bestanden

Zusätzlich kontrollierten wir, ob alle Ressourcen korrekt eingebunden waren, und ob bei einem Sprachwechsel auch die UI-Daten entsprechend aktualisiert wurden. Die Tab-Titel in der AppShell stellten dabei eine besondere technische Herausforderung dar, konnten aber abschliessend auch lokalisiert werden.

6 Auswerten

6.1 Was gut geklappt hat

- **Teamarbeit:**
  Wir haben gut zusammengearbeitet, uns regelmässig abgesprochen und Aufgaben klar verteilt. Bei Problemen haben wir uns gegenseitig unterstützt.
- **GitHub-Nutzung:**
  Das Projekt wurde von Anfang an über GitHub verwaltet. So konnten wir gemeinsam am Code arbeiten, Änderungen verfolgen und Konflikte vermeiden.
- **Mehrsprachigkeit:**
  Die Sprache der App kann gewechselt werden. Texte in der Benutzeroberfläche ändern sich dynamisch je nach gewählter Sprache. Auch die Tab-Titel in der AppShell konnten lokalisiert werden.
- **Funktionalität:**
  Die App funktioniert wie geplant: Einträge können hinzugefügt, gelöscht und gespeichert werden. Auch das Löschen einzelner oder aller Einträge ist möglich.
- **Dokumentation:**
  Die Dokumentation wurde vollständig nach dem IPERKA-Modell erstellt. Sie beschreibt den Ablauf und die Umsetzung unseres Projekts klar und nachvollziehbar.
- **Präsentation:**
  In der Präsentation konnten wir das Projekt verständlich vorstellen. Die Funktionen wurden gezeigt, und wir konnten Fragen beantworten und unsere Entscheidungen begründen.

6.2 Was weniger gut lief

- Obwohl wir eine Planung mit Aufgabenverteilung hatten, wurden Aufgaben manchmal auch von anderen übernommen. Dadurch gab es Verwirrung, wer was schon erledigt hat.
- Die Sprachumschaltung mit dynamischen Tab-Titeln war technisch deutlich schwieriger als erwartet, was uns am Ende wichtige Zeit für Feinschliff gekostet hat.
- Die Komplexität von .NET MAUI, insbesondere AppShell und dynamische Bindings in XAML, führte zu Verständnisproblemen und einigen Fehlern.

6.3 Verbesserungen fürs nächste Mal

- Eine sichtbare Fortschrittskontrolle einführen (z. B. abhaken, sobald eine Aufgabe erledigt ist).
- Komplexere Features zuerst umsetzen, um am Ende genug Zeit für Tests und Fehlerbehebung zu haben.

7 Fazit

Durch die Arbeit an diesem Projekt konnten wir unsere technischen Fähigkeiten im Umgang mit .NET MAUI, MVVM und GitHub vertiefen.
Wir haben gelernt, wie wichtig eine gute Planung, klare Aufgabenverteilung und saubere Kommunikation ist, besonders wenn mehrere Personen gemeinsam an einem Softwareprojekt arbeiten.
Wir sind stolz auf die Umsetzung der Mehrsprachigkeit und die strukturierte Architektur nach dem MVVM-Entwurfsmuster.
Rückblickend würden wir noch stärker auf frühzeitige Fehlererkennung und detaillierte Planung achten.
Insgesamt war das Projekt sehr lehrreich und hat uns sowohl technisch als auch methodisch weitergebracht.

8 Glossar

Tabelle 2 Glossar

Begriff	Erklärung
.NET MAUI	Plattformübergreifendes Framework zur Entwicklung von Apps für Android, iOS, macOS und Windows mit einer Codebasis.
AppShell	Steuert in .NET MAUI die Navigation (z. B. Tabs, Menüs) und das Layout der gesamten App.
Binding	Verknüpft UI-Elemente mit Daten aus dem ViewModel, sodass Änderungen automatisch angezeigt werden.

Begriff	Erklärung
ContentPage	Eine Bildschirmseite in der App mit Benutzeroberfläche und Logik.
DataTemplate	Vorlage, wie Daten visuell dargestellt werden (z. B. Listeneinträge).
GitHub	Plattform zur Versionsverwaltung und Zusammenarbeit an Quellcodeprojekten.
Gantt-Diagramm	Zeitplan zur Darstellung von Aufgaben, deren Dauer und Reihenfolge im Projekt.
IPERKA	Modell zur Projektarbeit: Informieren, Planen, Entscheiden, Realisieren, Kontrollieren, Auswerten.
JSON	Leichtgewichtiges Datenformat, in diesem Projekt für lokale Speicherung verwendet.
MVVM	Architekturmodell: Model (Daten), View (UI), ViewModel (Logik, Bindings) – trennt Darstellung und Funktionalität.
ObservableCollection	Liste, die automatisch UI-Änderungen anzeigt, wenn sich ihre Elemente ändern.
resx-Dateien	Ressourcendateien für Texte – für Mehrsprachigkeit (z. B. Deutsch, Englisch) genutzt.
ShellContent	Tab-Inhaltselement in AppShell – enthält eine ContentPage.
ViewModel	Teil der MVVM-Struktur – enthält Datenlogik und wird mit der Benutzeroberfläche (View) verbunden.

9 Verzeichnisse

9.1 Quellenverzeichnis

Es wurden keine externen Quellen aus dem Internet verwendet. Alle Inhalte stammen aus eigener Arbeit, Aufgabenstellung und Unterrichtsmaterialien des ICT Moduls 322.

9.2 Abbildungsverzeichnis

9.3 Tabellenverzeichnis