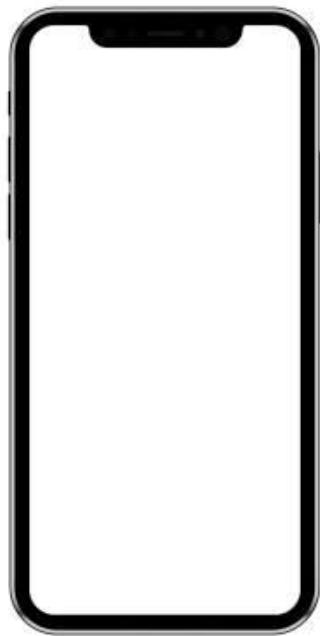




MODUL 322

Benutzerschnittstellen entwerfen und implementieren



MAY 12, 2025
AARAU IPSO IBZ
Von David Lindörfer & Leon Egli

Inhaltsverzeichnis

1. Informieren	2
Ausgangslage	2
Ziele.....	2
Zielgruppe.....	4
Rahmenbedingungen	4
2. Planen.....	5
Mockups/Wireframes	5
Gantt Diagramm	7
3. Entscheiden.....	8
4. Realisieren.....	9
Frontend.....	9
1. Technologie	9
2. Architektur:	9
3. Hauptkomponenten	9
Backend	11
1. Technologie	11
2. Datenbank:.....	11
3. Hauptfunktionen	11
Zusammenarbeit zwischen Frontend und Backend.....	12
Projektstruktur.....	12
Frontend	12
Backend	13
5. Kontrollieren	13
Testplan.....	13
Testfälle: Frontend	13
Testfälle: Backend.....	13
Testprotokoll.....	13
6. Auswerten.....	14
Fazit.....	14
Reflexion.....	14
7. Glossar.....	15
8. Quellen	17

Projekt-Dokumentation: SaveUp App

1. Informieren

Ausgangslage

In unserem Alltag tätigen wir häufig kleine, spontane Ausgaben sei es ein Kaffee unterwegs, eine Packung Süßigkeiten oder ein Snack am Automaten. Diese scheinbar unbedeutenden Beträge summieren sich im Laufe der Zeit zu einer nicht unerheblichen Summe. Oft bemerken wir gar nicht, wie viel Geld wir tatsächlich für diese Gewohnheiten ausgeben.

Aus diesem Grund haben wir uns dazu entschieden, bewusst auf solche kleinen Ausgaben zu verzichten und das eingesparte Geld stattdessen für eine grössere private Investition beispielsweise eine Reise oder einen grösseren Wunsch zu sparen. Um unseren Fortschritt nachvollziehen und sichtbar machen zu können, möchten wir eine eigene App entwickeln, die uns dabei unterstützt.

Unsere Idee ist es, eine Anwendung zu erstellen, mit der wir jeden einzelnen Verzichtskauf dokumentieren können. Das heisst: Jedes Mal, wenn wir uns bewusst gegen den Kauf eines Produktes entscheiden (zum Beispiel gegen einen Kaffee), möchten wir diesen Verzicht in der App festhalten – inklusive einer kurzen Beschreibung, des Preises und des Zeitpunkts. Die App soll daraufhin automatisch alle gesammelten Einträge anzeigen und die Gesamtersparnis berechnen. So behalten wir jederzeit den Überblick über unsere Fortschritte beim Sparen.

Ziele

Das Hauptziel unseres Projekts ist die Entwicklung einer Applikation, mit der eingesparte Ausgaben – also sogenannte Verzichtskäufe – einfach erfasst, gespeichert und ausgewertet werden können. Die App ist in drei ContentPages unterteilt:

1. Dashboard: Hier können Produkte, auf die verzichtet wurde, mit einer Kurzbeschreibung (z. B. „Kaffee beim Bahnhofskiosk“), dem Preis (z. B. CHF 4.50) sowie Datum und Uhrzeit des Verzichts hinzugefügt werden. Alle Einträge werden in einer Liste dargestellt, wobei die Gesamtersparnis gut sichtbar angezeigt wird. Die hinzugefügten Produkte können auch wieder gelöscht werden.
2. Sparbetrag-Seite: Diese Seite zeigt übersichtlich an, wie viel man insgesamt bereits gespart hat und wie weit man vom persönlichen Sparziel entfernt ist.
3. Statistik-Seite: Hier werden statistische Informationen wie die Anzahl der verzichteten Produkte, die bisherige Gesamtersparnis sowie das eingegebene
4. Sparziel angezeigt. Das Sparziel kann direkt auf dieser Seite erfasst und gespeichert werden.

Besonderer Wert wird auf ein übersichtliches Layout und eine intuitive Benutzerführung gelegt, um eine angenehme Bedienung der App zu gewährleisten. Alle Daten werden lokal gespeichert – wahlweise im JSON- oder XML-Format – was die App unabhängig von einer Internetverbindung macht und den Einsatz auch offline ermöglicht.

Zusätzlich zu diesen Kernfunktionen planen wir einige optionale Erweiterungen, darunter etwa eine grafische Darstellung der Ersparnisse über die Zeit sowie eine mehrsprachige Benutzeroberfläche, um die App noch vielseitiger und benutzerfreundlicher zu gestalten.

Kategorie	Eingesetzt	Begründung
Programmiersprache	C#	Hauptsprache für .NET MAUI und moderne App-Entwicklung
Framework	.NET MAUI	Für plattformübergreifende Entwicklung (Android, iOS, Windows)
Entwurfsmuster	MVVM (Model-View-ViewModel)	Ermöglicht saubere Trennung von UI, Logik und Daten
Layoutsprache	XAML	Für deklarative Benutzeroberflächen mit Styles
Datenspeicherung	JSON (lokal)	Offline-Persistenz ohne Datenbankserver
IDE	Visual Studio 2022/2023	Haupt-Entwicklungsumgebung mit MAUI-Unterstützung
Design Tools	Skizzen, Balsamiq	Für GUI-Mockups und visuelle Planung
Versionskontrolle	Git (lokal oder mit GitHub)	Für die Verwaltung und Nachverfolgbarkeit des Codes
Datenbank	MonogoDB mit REST-API	Für Datenspeicherung

Tabelle 1: Technologien

Zielgruppe

Die App richtet sich an Menschen, die bewusster mit Geld umgehen wollen insbesondere an Jugendliche, Lernende, Studierende oder Berufseinsteiger, die ihre Finanzen besser kontrollieren möchten. Auch Personen, die sich einer finanziellen Herausforderung (z. B. einer «No-Spend-Challenge») stellen oder auf ein konkretes Ziel hinsparen, profitieren vom Konzept der App.

Durch die einfache Bedienung, die klare Struktur und das ansprechende Design ist die App auch für Nutzerinnen und Nutzer ohne tiefes technisches Verständnis geeignet.

Rahmenbedingungen

Folgende technische und organisatorische Rahmenbedingungen sind einzuhalten:

- **Technologie:** Die Anwendung muss mit .NET MAUI und C# entwickelt werden.
- **Plattform:** Die App soll auf mobilen Geräten (Android und iOS) sowie auf Windows lauffähig sein.
- **Strukturierung:** Das MVVM-Muster muss eingehalten werden, um eine saubere Codebasis zu gewährleisten.
- **Bedienoberfläche:** Die Benutzeroberfläche wird vollständig in XAML gestaltet und mit Styles versehen.
- **Funktionen:** Mindestens drei Content Pages, lokale Speicherung, Anzeige der Gesamtersparnis, Menüfunktionen zur Navigation.
- **Design:** Die App erhält ein individuelles Icon und ein strukturiertes, benutzerfreundliches GUI.
- **Persistenz:** Alle Daten werden lokal gespeichert, eine Verbindung zu einem Server ist nicht erforderlich (optional erweiterbar).
- **Dokumentation & Test:** Der gesamte Entwicklungsprozess wird dokumentiert, und es werden grundlegende Tests zur Sicherstellung der Funktionalität durchgeführt.

2. Planen

In der Planungsphase wurden die zentralen Grundlagen für den Projektverlauf geschaffen. Ziel war es, eine strukturierte Vorgehensweise festzulegen, um alle Aufgaben effizient umzusetzen. Zunächst wurde der zeitliche Ablauf mithilfe eines Gantt-Diagramms visualisiert, wodurch die einzelnen Arbeitsschritte übersichtlich dargestellt und priorisiert werden konnten. Die Verantwortlichkeiten im Team wurden klar verteilt, um eine reibungslose Zusammenarbeit sicherzustellen.

Darüber hinaus wurden bereits im Vorfeld Wireframes entworfen, um die Benutzeroberfläche der Anwendung visuell zu planen. Diese dienten als wichtige Orientierung für das spätere Design und halfen dabei, die Benutzerführung frühzeitig zu durchdenken und zu optimieren.

Mockups/Wireframes

Die Wireframes wurden bereits im Vorfeld erstellt, um die Struktur und Gestaltung der Benutzeroberfläche systematisch zu planen. Sie dienten als visuelle Entwürfe, anhand derer die Positionierung von Bedienelementen, die Navigation sowie das grundsätzliche Layout der Anwendung festgelegt werden konnten.

Dashboard: Übersicht über alle Produkte und Gesamtersparnisse.

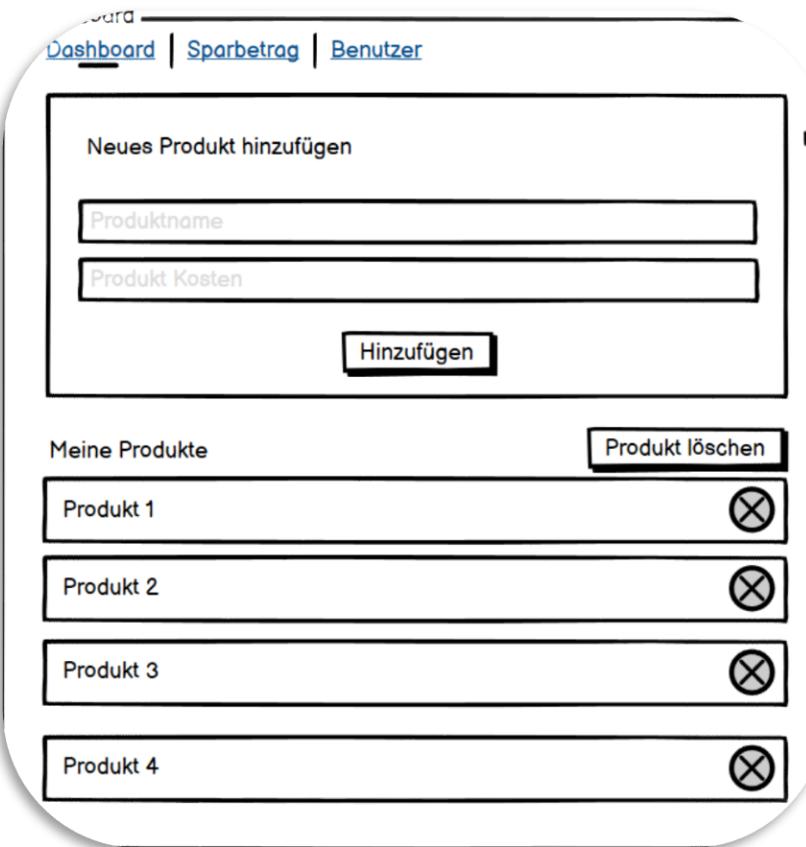


Abbildung 1: Wireframe Dashboard

Sparbetrag Seite: Zeigt an, wie viel bereits gespart wurde und wie nah man dem Sparziel ist.

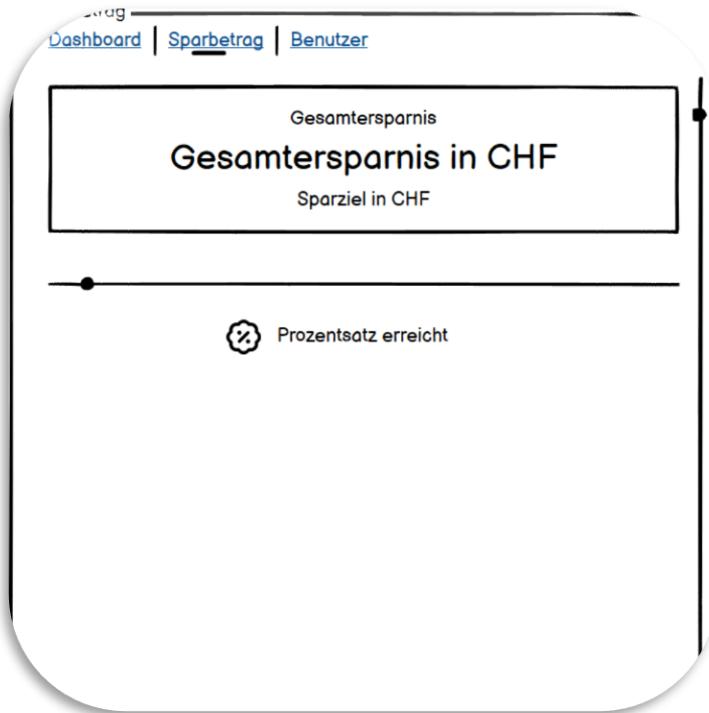


Abbildung 2: Wireframe Sparbetrag

Benutzeroberseite: Einstellungen und Sparzielverwaltung.

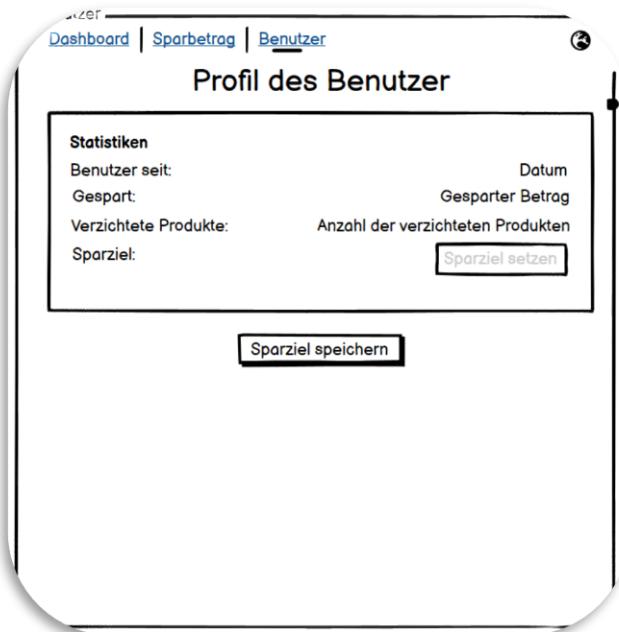


Abbildung 3: Wireframe Benutzer

Gantt Diagramm

Für die Planung des Projekts wurde ein Gantt-Diagramm erstellt, wie unten angezeigt. Somit war der gesamte Projektzeit vollstrukturiert. In dem Diagramm sind Arbeitspakete, die zeitliche Abfolge, und die Abhängigkeiten dargestellt. Die Planung half bei der Strukturierung und realistischen Abläufen des Gesamtprojekts. Die Zeitplanung erleichterte die Beurteilung des Arbeitsaufwands und Zuweisung von Aufgaben zwischen Projektteams. Außerdem half das Diagramm während der Recherche und Implementierung, den Fortschritt zu ermitteln, und Zeitunterbrechungen unmittelbar zu unternehmen. Die strukturierte Planung half die Aufgaben fair aufzuteilen.

Kurzübersicht:

Arbeitspaket	Dauer	Beschreibung
Backend-API erstellen	5 Tage	Entwicklung der RESTful API und Datenbank.
Frontend-Integration	5 Tage	Integration der API ins Frontend.
Offline-Funktionalität	3 Tage	Implementierung der JSON-Dateispeicherung.
Tests und Fehlerbehebung	3 Tage	Durchführung von Tests und Debugging.



Tabelle 2:Gantt

3. Entscheiden

Technologieentscheidungen nach der Anforderungsanalyse

Nach der ausführlichen Analyse der funktionalen und nicht-funktionalen Anforderungen wurden zentrale technologische Entscheidungen für die Umsetzung des Projekts getroffen. Dabei wurde besonders auf Faktoren wie Wartbarkeit, Erweiterbarkeit, Performance und Kompatibilität geachtet.

Für das **Frontend** fiel die Wahl auf **.NET MAUI**, ein modernes Framework von Microsoft, das die Entwicklung plattformübergreifender Anwendungen ermöglicht. Diese Entscheidung basierte auf dem Ziel, eine einheitliche Codebasis für verschiedene Plattformen – insbesondere Android, iOS und Windows – nutzen zu können. Damit konnte Entwicklungsaufwand gespart und gleichzeitig eine konsistente Benutzererfahrung auf mehreren Endgeräten gewährleistet werden. Die Integration in Visual Studio sowie die Unterstützung des MVVM-Patterns erleichterten zusätzlich die Umsetzung komplexer UI-Logik.

Im **Backend** wurde auf **ASP.NET Core Web API** gesetzt. Dieses Framework zeichnet sich durch hohe Performance, modulare Struktur und einfache Integration mit modernen Datenbanksystemen aus. Zudem unterstützt es RESTful-Architekturen, was die Kommunikation zwischen Frontend und Backend klar strukturiert und standardisiert. Die Entscheidung fiel zugunsten von ASP.NET Core, da es eine saubere Trennung von Logik und Präsentation ermöglicht und eine breite Community sowie umfangreiche Dokumentation bietet – ein klarer Vorteil für Entwicklung und langfristige Wartung.

Für die **Datenhaltung** wurde **MongoDB** gewählt, eine dokumentenorientierte NoSQL-Datenbank. Ihre hohe Flexibilität bei der Speicherung unstrukturierter oder sich häufig ändernder Datenstrukturen war ein ausschlaggebender Grund für die Wahl. Im Gegensatz zu relationalen Datenbanken erlaubt MongoDB die Speicherung von JSON-ähnlichen Dokumenten (BSON), was besonders gut mit .NET-Anwendungen harmoniert. Die horizontale Skalierbarkeit und einfache Anbindung an das Backend trugen zusätzlich zur Entscheidung bei.

Insgesamt entstand durch diese Technologieauswahl ein modernes, robustes und zukunftsfähiges Fundament für die Umsetzung des Projekts, das sowohl den funktionalen Anforderungen als auch den technischen Herausforderungen gerecht wird.

4. Realisieren

Frontend

1. Technologie: .NET MAUI

- **Was wurde gemacht?**

- Die Entwicklungsumgebung für .NET MAUI wurde eingerichtet, inklusive der Installation aller benötigten Abhängigkeiten.
- Es wurde sichergestellt, dass die Anwendung auf **Android, iOS, Windows und macOS** ausgeführt werden kann.
- Ein plattformübergreifendes Layout wurde erstellt, um eine konsistente Benutzererfahrung auf allen unterstützten Geräten zu gewährleisten.

2. Architektur: MVVM (Model-View-ViewModel)

- **Was wurde gemacht?**

- MVVM wurde konsequent implementiert, um die Trennung von Benutzeroberfläche (UI) und Logik zu gewährleisten.
- Für jede Hauptfunktion der App (z. B. Dashboard, Benutzerseite) wurde ein eigenes **ViewModel** erstellt.
- Datenbindung wurde in XAML definiert, um die Logik in den ViewModels direkt mit den Views zu verbinden.

3. Hauptkomponenten

- **Views (Benutzeroberfläche in XAML)**

- **Dashboard-Seite:**

- Zeigt eine Übersicht über alle Produkte und deren Sparziele.
 - Enthält eine Fortschrittsanzeige (z. B. Balkendiagramm oder Kreisdiagramm), um den aktuellen Sparstand visuell darzustellen.

- **Produktseite:**

- Ermöglicht Benutzern, neue Produkte hinzuzufügen, bestehende Produkte zu bearbeiten oder zu löschen.
 - Eingabefelder für Produktname, Zielbetrag und aktuelle Ersparnisse wurden entwickelt.

- **Benutzerseite:**
 - Zeigt Benutzerdaten an, einschließlich des Gesamtfortschritts aller Sparziele.
 - Ermöglicht die Einstellung von Benachrichtigungen oder persönlichen Sparzielen.
- **ViewModels (Logik und Datenbindung)**
 - **DashboardViewModel:**
 - Logik zum Laden der Produkte vom Backend oder aus der Offline-Datenbank.
 - Berechnung des Gesamtfortschritts für alle Produkte.
 - **SavingsViewModel:**
 - Spezifische Logik für die Berechnung des Sparfortschritts eines einzelnen Produkts.
 - Verbindet sich mit der JSON-Datenspeicherung für Offline-Modus.
 - **UserViewModel:**
 - Speichert Benutzereinstellungen und kommuniziert mit den Views, um die Daten anzuzeigen.
- **Services (API-Kommunikation und lokale Datenspeicherung)**
 - **ApiService.cs:**
 - Entwickelt, um RESTful API-Anfragen an das Backend zu senden.
 - Unterstützt Endpunkte wie **GET /api/products**, **POST /api/products**, **DELETE /api/products/{id}**.
 - **Lokale Speicherung:**
 - Implementiert, um Produktdaten in JSON-Dateien zu speichern, wenn keine Internetverbindung verfügbar ist.
 - Synchronisation mit dem Backend, sobald die Verbindung wiederhergestellt ist.

Backend

1. Technologie: ASP.NET Core Web API

- **Was wurde gemacht?**
 - Eine RESTful API wurde mit ASP.NET Core entwickelt, die CRUD-Operationen für Produkte unterstützt.
 - Die API wurde mit Sicherheitsmechanismen ausgestattet, um Eingabedaten zu validieren und unbefugten Zugriff zu verhindern.

2. Datenbank: MongoDB

- **Was wurde gemacht?**
 - MongoDB wurde als Datenbank für die SaveUpApp eingerichtet.
 - Die Datenbank speichert alle Produkte, einschließlich Name, Zielbetrag, aktueller Sparstand und Benutzerinformationen.

3. Hauptfunktionen

- **CRUD-Operationen für Produkte:**
 - **Create (POST):** Ein neuer Endpunkt wurde erstellt, um Produkte hinzuzufügen. Die Daten werden direkt in MongoDB gespeichert.
 - **Read (GET):** Ein Endpunkt wurde entwickelt, um alle Produkte oder ein spezifisches Produkt abzurufen.
 - **Update (PUT):** Änderungen an bestehenden Produkten (z. B. Sparziele) können gespeichert werden.
 - **Delete (DELETE):** Ein Endpunkt wurde implementiert, um Produkte aus der Datenbank zu entfernen.
- **Unterstützung von JSON-Datenformaten:**
 - Die API verarbeitet Anfragen und antwortet im JSON-Format, was die Integration mit dem Frontend erleichtert.
- **Fehlerbehandlung und Validierung:**
 - Eingabedaten werden auf Korrektheit geprüft (z. B. kein negativer Sparbetrag).
 - Fehler, wie z. B. Verbindungsprobleme zur Datenbank, werden durch spezifische Fehlermeldungen behandelt.

Zusammenarbeit zwischen Frontend und Backend

- **API-Kommunikation:**
 - Die API-Endpunkte wurden im Frontend integriert.
 - Mithilfe des ApiService.cs werden Daten zwischen Frontend und Backend ausgetauscht.
- **Offline-Unterstützung:**
 - Wenn keine Verbindung zum Backend besteht, speichert das Frontend die Produktdateien lokal in einer JSON-Datei.
 - Beim nächsten Start der App oder bei Wiederherstellung der Verbindung werden die Daten synchronisiert.
- **Test und Debugging:**
 - Alle API-Endpunkte wurden mit Unit-Tests überprüft.
 - Das Zusammenspiel zwischen Frontend und Backend wurde mit Integrationstests getestet.

Projektstruktur

Frontend

```
SaveUpAppFrontend/
├── Views/
│   ├── DashboardPage.xaml
│   ├── SavingsPage.xaml
│   └── UserPage.xaml
├── ViewModels/
│   ├── BaseViewModel.cs
│   ├── DashboardViewModel.cs
│   ├── SavingsViewModel.cs
│   └── UserViewModel.cs
├── Services/
│   ├── ApiService.cs
│   ├── JsonStorageService.cs
│   ├── ProductService.cs
│   └── SavingGoalService.cs
└── Models/
    └── Product.cs
App.xaml
```

Backend

```
SaveUpAppBackend/
├── Controllers/
│   └── ProductController.cs
├── Models/
│   └── Product.cs
├── Services/
│   └── MongoDBService.cs
├── Data/
│   └── MongoDBSettings.cs
├── appsettings.json
└── Program.cs
```

5. Kontrollieren

Testplan

Testfälle: Frontend

Testfall	Erwartetes Ergebnis	Status
Produkte von der API laden	Produkte werden korrekt geladen.	<input checked="" type="checkbox"/> Bestanden
Produkte aus der JSON-Datei laden	Produkte werden korrekt aus der Datei geladen.	<input checked="" type="checkbox"/> Bestanden
Sparziel setzen und speichern	Sparziel wird gespeichert und angezeigt.	<input checked="" type="checkbox"/> Bestanden

Testfälle: Backend

Testfall	Erwartetes Ergebnis	Status
<code>GET /api/products</code>	Gibt alle Produkte zurück.	<input checked="" type="checkbox"/> Bestanden
<code>POST /api/products</code>	Fügt ein neues Produkt hinzu.	<input checked="" type="checkbox"/> Bestanden
<code>DELETE /api/products/{id}</code>	Löscht ein Produkt erfolgreich.	<input checked="" type="checkbox"/> Bestanden

Testprotokoll

Alle Tests wurden erfolgreich durchgeführt. Kleinere Bugs, wie z. B. fehlerhafte UI-Aktualisierungen, wurden behoben.

6. Auswerten

Fazit

Rückblickend sind wir sehr zufrieden mit dem Ergebnis unserer Arbeit. Die Anwendung erfüllt alle ursprünglich definierten Anforderungen und konnte durch eine klare Struktur und eine benutzerfreundliche Oberfläche überzeugen. Besonders stolz sind wir darauf, dass die App auch ohne Internetverbindung funktioniert, das war uns von Anfang an wichtig. Die Synchronisation zwischen Frontend und Backend verlief reibungslos und zeigte, dass unsere Planung und Umsetzung gut aufeinander abgestimmt waren.

Reflexion

Was gut lief:

Ein großer Erfolg war für uns die konsequente Anwendung des MVVM-Musters. Dadurch konnten wir Logik und Oberfläche sauber trennen, was die Wartbarkeit erheblich verbessert. Auch die Implementierung der Offline-Funktionalität hat gut funktioniert und war ein technischer Meilenstein für uns. Die stabile Integration der API war ebenfalls ein Highlight die Kommunikation zwischen App und Server verlief genau wie geplant.

Was wir in Zukunft verbessern würden:

Wenn wir mehr Zeit gehabt hätten, hätten wir gerne eine Benutzeranmeldung (Authentifizierung) eingebaut, um persönliche Daten noch gezielter zu verwalten. Auch die Unterstützung mehrerer Sprachen im Frontend wäre eine tolle Ergänzung, damit die App für ein breiteres Publikum nutzbar wird.

Insgesamt war dieses Projekt für uns beide eine wertvolle Lernerfahrung – sowohl technisch als auch im Hinblick auf die Zusammenarbeit und das Projektmanagement.

7. Glossar

Begriff	Definition
App	Kurzform für Applikation; ein Softwareprogramm, das speziell für mobile Geräte wie Smartphones oder Tablets entwickelt wurde.
.NET MAUI	Plattform zur Entwicklung plattformübergreifender Anwendungen (Mobile/Desktop) mit C# und XAML.
MVVM	Abkürzung für Model-View-ViewModel; ein Entwurfsmuster zur Trennung von Logik, Daten und UI.
XAML	Extensible Application Markup Language; eine XML-basierte Sprache zur Gestaltung von Benutzeroberflächen in .NET-Anwendungen.
JSON	JavaScript Object Notation; ein leichtgewichtiges Datenformat zur Speicherung und Übertragung strukturierter Daten.
XML	Extensible Markup Language; ein Markup-Format zur Beschreibung hierarchischer Datenstrukturen.
UI (User Interface)	Benutzeroberfläche, über die ein Mensch mit einem digitalen System interagiert.
UX (User Experience)	Nutzererfahrung; beschreibt, wie ein Benutzer die App wahrnimmt und erlebt.
Mockup	Ein Entwurf oder eine visuelle Vorschau der Benutzeroberfläche, meist als Bild oder Grafik.
Storyboard	Eine Abfolge von Skizzen oder Screenshots, die die Navigation und Struktur einer App veranschaulichen.
Datenmodell	Eine strukturierte Darstellung, wie Daten gespeichert, verarbeitet und miteinander verknüpft werden.
Code	Programmiertext, der Anweisungen an den Computer enthält.
Debugging	Der Prozess zur Identifikation und Behebung von Fehlern im Programmcode.
Commit	Das Speichern von Änderungen am Quellcode in einem Versionskontrollsystem wie Git.

Begriff	Definition
Git	Ein verteiltes Versionskontrollsysteem zur Nachverfolgung von Änderungen am Quellcode.
GitHub	Eine Plattform zur Verwaltung von Softwareprojekten und Quellcode mithilfe von Git.
Backup	Eine Sicherheitskopie von Daten, um Verlust zu verhindern.
Iterativ	Ein Vorgehen, bei dem ein Projekt in kleinen Schritten (Iterationen) weiterentwickelt wird.
Refactoring	Die strukturierte Überarbeitung von Quellcode zur Verbesserung seiner Lesbarkeit und Wartbarkeit – ohne das Verhalten zu verändern.
UI-Komponente	Einzelne nutzbare Elemente der Benutzeroberfläche wie z. B. Button, Eingabefeld oder Liste.
Usability	Benutzerfreundlichkeit; Maß für die einfache und intuitive Nutzung einer Software.
Feedback	Rückmeldung zur App, z. B. durch Nutzer oder Betreuer, um sie zu verbessern.
Evaluation	Bewertung und Analyse eines Entwicklungsstands oder Ergebnisses.

8. Quellen

Microsoft MAUI Documentation, Available: https://learn.microsoft.com/de-de/dotnet/maui/?view=net-maui-9.0&WT.mc_id=dotnet-35129-website, Date: 11.05.2025

ASP.NET Core Web API Documentation, Available: <https://learn.microsoft.com/en-us/aspnet/core/?view=aspnetcore-9.0>, Date: 11.05.2025

MongoDB Driver for .NET, Available: <https://www.mongodb.com/docs/languages/csharp/>, Date: 11.05.2025

Abbildung 1:Wireframe Dashboard	5
Abbildung 2:Wireframe Sparbetrag.....	6
Abbildung 3:Wireframe Benutzer.....	6
Tabelle 1:Technologien	3
Tabelle 2:Gant	7