Leon Egli & David Lindörfer

[E-Mail-Adresse]

**Ein Bild, das Grafiken, Schrift, Logo, Grafikdesign enthält.

KI-generierte Inhalte können fehlerhaft sein.**

Saveup app

Inhalt

[Projekt-Dokumentation: SaveUp App 2](#_Toc197688098)

[1. Informieren 2](#_Toc197688099)

[Ausgangslage 2](#_Toc197688100)

[Ziel des Projekts 3](#_Toc197688101)

[2. Planen 3](#_Toc197688102)

[Wireframes / Mockups 3](#_Toc197688103)

[Arbeitspakete und Planung 3](#_Toc197688104)

[Gantt-Diagramm 3](#_Toc197688105)

[3. Entscheiden 3](#_Toc197688106)

[4. Realisieren 3](#_Toc197688107)

[Implementierungsschritte 3](#_Toc197688108)

[Highlights der Implementierung 3](#_Toc197688109)

[5. Kontrollieren 4](#_Toc197688110)

[Testplan 4](#_Toc197688111)

[Testprotokoll 4](#_Toc197688112)

[6. Auswerten 4](#_Toc197688113)

[Fazit 4](#_Toc197688114)

[Reflexion 4](#_Toc197688115)

[Anhang 5](#_Toc197688116)

[Quellen 5](#_Toc197688117)

[Screenshots 5](#_Toc197688118)

# Projekt-Dokumentation: SaveUp App

## 1. Informieren

### Ausgangslage

Die SaveUpApp\_Modul322 wurde entwickelt, um Benutzern eine einfache Möglichkeit zu bieten, ihre Ersparnisse zu verwalten, Produkte zu organisieren und Sparziele zu verfolgen. Das Projekt besteht aus einer plattformübergreifenden **Frontend-App** (basierend auf .NET MAUI) und einem **Backend-Server** (basierend auf ASP.NET Core Web API), der als zentrale Datenablage dient.

### Ziele

* Erstellung einer plattformunabhängigen **Frontend-App**, die auf Android, iOS und Desktop läuft.
* Entwicklung eines robusten **Backend-Servers** zur Speicherung und Verwaltung der Daten.
* Offline-Funktionalität im Frontend, um Benutzern auch ohne Internetzugang Zugriff auf ihre Daten zu ermöglichen.
* Sicherstellung der Datenkonsistenz zwischen Frontend und Backend.

## 2. Planen

### Mockups/Wireframes

Wireframes wurden vorab erstellt, um die Benutzeroberfläche zu planen und die Benutzererfahrung zu optimieren.

* **Dashboard**: Übersicht über alle Produkte und Gesamtersparnisse.
* **Produktseite**: Verwaltung von Produkten (Hinzufügen, Bearbeiten, Löschen).
* **Benutzerseite**: Einstellungen und Sparzielverwaltung.

### Gantt-Diagramm

Für die Planung wurde ein Gantt-Diagramm erstellt, das die einzelnen Arbeitspakete und deren Zeitrahmen visualisiert.

| **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. |

## 3. Entscheiden

Nach der Analyse der Anforderungen wurde entschieden: - **Frontend**: .NET MAUI wurde gewählt, um eine plattformübergreifende Entwicklung zu ermöglichen. - **Backend**: ASP.NET Core Web API ermöglicht eine schnelle Entwicklung und einfache Integration mit MongoDB. - **Datenhaltung**: MongoDB wurde aufgrund ihrer Flexibilität und Skalierbarkeit ausgewählt.

## 4. Realisieren

### Technologie-Stack

#### Frontend

* **Technologie**: .NET MAUI
* **Architektur**: MVVM (Model-View-ViewModel)
* **Hauptkomponenten**:
  + Views: Benutzeroberfläche in XAML.
  + ViewModels: Logik und Datenbindung.
  + Services: API-Kommunikation und lokale Datenspeicherung.

#### Backend

* **Technologie**: ASP.NET Core Web API
* **Datenbank**: MongoDB
* **Hauptfunktionen**:
  + CRUD-Operationen für Produkte.
  + Unterstützung von JSON-Datenformaten.

### 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. | ✅ Bestanden |
| Produkte aus der JSON-Datei laden | Produkte werden korrekt aus der Datei geladen. | ✅ Bestanden |
| Sparziel setzen und speichern | Sparziel wird gespeichert und angezeigt. | ✅ Bestanden |

#### Testfälle: Backend

| **Testfall** | **Erwartetes Ergebnis** | **Status** |
| --- | --- | --- |
| GET /api/products | Gibt alle Produkte zurück. | ✅ Bestanden |
| POST /api/products | Fügt ein neues Produkt hinzu. | ✅ Bestanden |
| DELETE /api/products/{id} | Löscht ein Produkt erfolgreich. | ✅ Bestanden |

### Testprotokoll

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

## 6. Auswerten

### Fazit

* Die Anwendung erfüllt alle definierten Anforderungen und bietet eine intuitive Benutzererfahrung.
* Die Offline-Funktionalität gewährleistet, dass Benutzer auch ohne Internetverbindung auf ihre Daten zugreifen können.
* Die Synchronisation zwischen Backend und Frontend funktioniert nahtlos.

### Reflexion

**Erfolge**: - Klare Trennung von Logik und Benutzeroberfläche durch MVVM. - Robuste Offline-Funktionalität. - Stabile API-Integration.

**Verbesserungspotenziale**: - Erweiterung der API um Benutzerauthentifizierung. - Unterstützung für mehrere Sprachen im Frontend.

## 7. Quellen

* [Microsoft MAUI Documentation](https://learn.microsoft.com/en-us/dotnet/maui/)
* [ASP.NET Core Web API Documentation](https://learn.microsoft.com/en-us/aspnet/core/web-api/)
* [MongoDB Driver for .NET](https://www.mongodb.com/docs/drivers/csharp/)