21.1.2025

Lucas Berresheim

KBBZ-Halberg

Finanzmanagementsystem

Lasten- und Pflichtenheft

Inhalt

[1. Einführung 2](#_Toc188347335)

[2. Ist-Situation 2](#_Toc188347336)

[3. Soll-Situation 2](#_Toc188347337)

[3.1 Soll-Zustand 2](#_Toc188347338)

[3.2 Funktionale Anforderungen 3](#_Toc188347339)

[3.3 Nicht-Funktionale Anforderungen 4](#_Toc188347340)

[3.4 Schnittstellen 4](#_Toc188347341)

[3.5 Risiken 4](#_Toc188347342)

[4. Abnahmekriterien 5](#_Toc188347343)

[5. Use-Case-Diagramme 5](#_Toc188347344)

[6. Projektplan 7](#_Toc188347345)

[7. Produktumgebung 7](#_Toc188347346)

[8. Skizze von GUI 8](#_Toc188347347)

[9. DB-Entwurf 9](#_Toc188347348)

[10. Git-Repository 9](#_Toc188347349)

[11. Testplan 9](#_Toc188347350)

# 1. Einführung

**Projektbezeichnung**: Entwicklung eines Finanzmanagementsystems ähnlich wie Finanzguru.

**Kurzvorstellung des Projektumfelds**: Dieses Projekt simuliert ein reales Szenario zur Entwicklung eines Finanzmanagementsystems, das Privatpersonen dabei hilft, ihre Finanzen effizient zu verwalten. Ziel ist es, eine Software bereitzustellen, die sowohl funktional als auch benutzerfreundlich ist. Diese Software wird durch innovative Funktionen und eine moderne Benutzeroberfläche einen echten Mehrwert bieten.

**Branche und Kontext**: Das Projekt zielt auf den Fintech-Markt ab, der innovative Technologien für Finanzdienstleistungen bietet. Die Zielgruppe umfasst sowohl technisch versierte als auch weniger erfahrene Nutzer, die ein intuitives Werkzeug zur Finanzüberwachung suchen.

# 2. Ist-Situation

**Ausgangslage**: Viele Privatpersonen nutzen rudimentäre Werkzeuge wie Excel oder papierbasierte Systeme, um ihre Finanzen zu verfolgen. Diese Methoden sind zeitaufwändig und fehleranfällig, was zu einem mangelnden Überblick über die eigene finanzielle Situation führen kann.

**Problemstellung**: Der Mangel an einer intuitiven und automatisierten Lösung erschwert es den Nutzern, ihre finanzielle Situation genau zu überblicken und fundierte Entscheidungen zu treffen. Bestehende Software ist oft unübersichtlich, überladen oder benötigt tiefere technische Kenntnisse, die nicht jeder Nutzer mitbringt.

# 3. Soll-Situation

3.1 Soll-Zustand

Die entwickelte Software wird eine zentrale Plattform bieten, um Einnahmen, Ausgaben und Budgets zu verwalten. Funktionen wie automatische Synchronisation mit Bankkonten, KI-gestützte Kategorisierung und dynamische Berichte sollen die Benutzerfreundlichkeit erhöhen. Der Mehrwert für die Nutzer liegt in Zeitersparnis, besserer finanzieller Kontrolle und personalisierten Analysen.

## 3.2 Funktionale Anforderungen

1. **Transaktionsmanagement** (30 Stunden):
   * Import und Verwaltung von Banktransaktionen.
   * Hinzufügen manueller Einträge.
   * Anpassung und Löschung bestehender Transaktionen.
   * Speicherung von Transaktionen in einer Datenbank.
2. **Kategorisierung** (20 Stunden):
   * Automatische Zuordnung von Kategorien durch KI.
   * Erstellung benutzerdefinierter Kategorien.
   * Farbliche Hervorhebung der wichtigsten Kategorien.
   * Anzeige von Statistiken pro Kategorie.
3. **Budgetplanung** (40 Stunden):
   * Erstellung monatlicher und jährlicher Budgets.
   * Warnungen bei Budgetüberschreitungen.
   * Historische Analyse von Budgetdaten.
   * Verknüpfung von Budgets mit Kategorien.
4. **Berichte und Visualisierungen** (35 Stunden):
   * Erstellung dynamischer Diagramme (z. B. Cashflow, Ausgabenkategorien).
   * Exportmöglichkeiten in PDF und Excel.
   * Benutzerdefinierte Berichte nach Zeitraum oder Kategorie.
   * Vergleich von Monats- und Jahresausgaben.
5. **Synchronisation** (30 Stunden):
   * Integration von Open-Banking-APIs.
   * CSV-Import und Export für offline Daten.
   * Echtzeit-Updates von Transaktionsdaten.

## 3.3 Nicht-Funktionale Anforderungen

1. **Design** (25 Stunden):
   * Einhaltung moderner UI/UX-Prinzipien.
   * Anpassbares Layout (z. B. Dark Mode, Farbschema).
2. **Barrierefreiheit** (20 Stunden):
   * Unterstützung für Screenreader.
   * Tastaturnavigation und kontrastreiche Darstellung.
3. **Sicherheit** (30 Stunden):
   * End-to-End-Verschlüsselung sensibler Daten.
   * Zwei-Faktor-Authentifizierung und Passwort-Hashing.
   * Regelmäßige Sicherheitsüberprüfungen.

## 3.4 Schnittstellen

* Open-Banking-API für Echtzeit-Datenabgleich.
* Export in Steuer-Software kompatiblen Formaten.
* Integration mit Kalender-Apps für Zahlungserinnerungen.

## 3.5 Risiken

* **Technische Risiken**:
  + API-Verfügbarkeit und Latenz.
  + Datenintegrität bei großen Imports.
  + Mögliche Kompatibilitätsprobleme bei unterschiedlichen Banken.
* **Organisatorische Risiken**:
  + Unklare Anforderungen.
  + Zeitmangel bei Testphasen.
  + Ressourcenengpässe während der Implementierung.

# 4. Abnahmekriterien

**Muss-Kriterien**:

* Alle Kernfunktionen (Transaktionen, Berichte, Budgets) sind implementiert und getestet.
* Login-Service
* Synchronisation mit einer Banking-API erfolgreich.
* Export von Berichten in PDF und Excel.
* Sicherstellung der Datensicherheit und -integrität.

**Kann-Kriterien**:

* Erweiterung der Analyse durch Prognosen.
* Vorschläge zur Optimierung der Finanzen basierend auf den Daten.

# 5. Use-Case-Diagramme

Ein Bild, das Text, Screenshot, Diagramm, Reihe enthält.

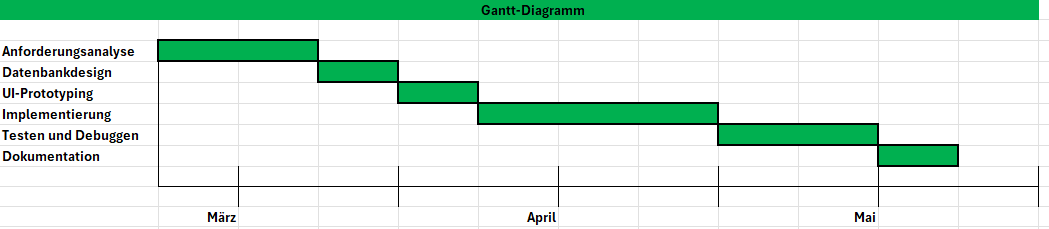
Automatisch generierte Beschreibung

|  |  |
| --- | --- |
| **Use-Case-Name** | **Einnahmen kategorisieren** |
| **Akteur** | Nutzer |
| **Ziel** | Der Nutzer kategorisiert die importierten Einnahmen in definierte Kategorien |
| **Beschreibung** | Nach dem Import von Transaktionen kann der Nutzer manuell Kategorien zuweisen oder automatisch durch das System Vorschläge erhalten |
| **Vorbedingung** | Die Transaktionsdaten wurden erfolgreich importiert |
| **Nachbedingung** | Kategorien werden gespeichert und stehen für Analysen und Berichte bereit |
| **Alternative Szenarien** | Automatische Kategorisierung durch das System bei vorheriger Konfiguration |

|  |  |
| --- | --- |
| **Use-Case-Name** | **Transaktionen importieren** |
| **Akteur** | Nutzer |
| **Ziel** | Der Nutzer importiert Banktransaktionen über eine API oder CSV-Datei |
| **Beschreibung** | Das System ruft Transaktionsdaten entweder von der Bank-API oder einer hochgeladenen CSV-Datei ab |
| **Vorbedingung** | Der Nutzer hat eine gültige Verbindung zur Bank oder die CSV-Datei hochgeladen |
| **Nachbedingung** | Die Transaktionen sind im System gespeichert und können weiter verarbeitet werden |
| **Alternative Szenarien** | Falls der Import fehlschlägt, wird der Nutzer informiert und ein erneuter Versuch ermöglicht |

|  |  |
| --- | --- |
| **Use-Case-Name** | **Automatische Kategorisierung** |
| **Akteur** | System |
| **Ziel** | Transaktionen automatisch basierend auf Stichwörtern oder Regeln kategorisieren |
| **Beschreibung** | Nach dem Import durchsucht das System die Transaktionsbeschreibung und weist basierend auf vorher definierte Regeln Kategorien zu |
| **Vorbedingung** | Die Transaktionsdaten sind vorhanden und Regeln für die Kategorien sind definiert |
| **Nachbedingung** | Kategorien sind zugewiesen und können manuell angepasst werden |
| **Alternative Szenarien** | Falls keine passende Kategorie gefunden wird, bleibt die Transaktion unklassifiziert |

# 6. Projektplan



# 7. Produktumgebung

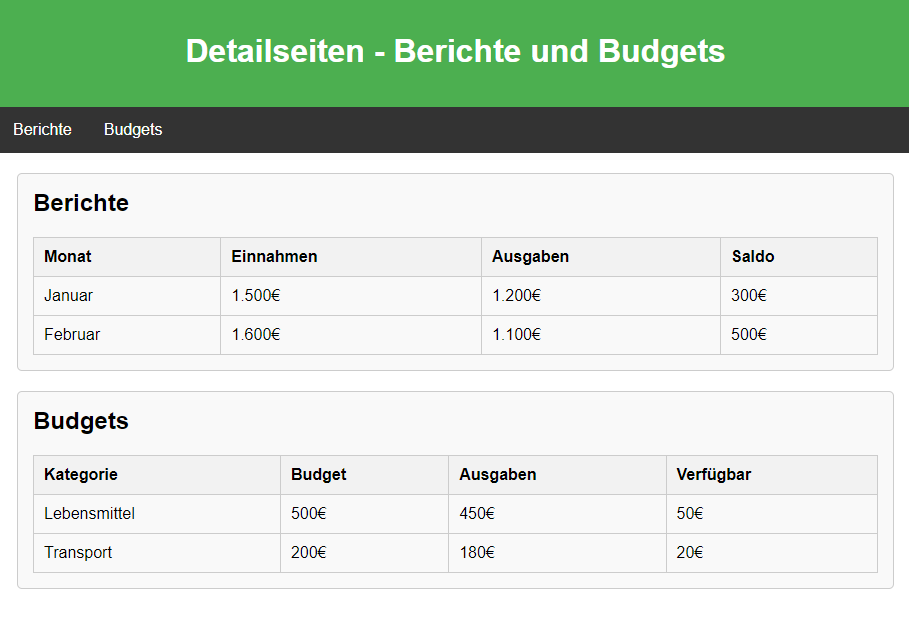
**Technologien**:

* Programmiersprache: C#
* Framework: .NET 6, WPF
* Datenbank: MSSQL
* Externe Schnittstellen: Open-Banking-API
* Tools: Git, (Postman für API-Tests), Visual Studio für den Code

# 8. Skizze von GUI

Ein Bild, das Text, Screenshot, Schrift, Software enthält.

Automatisch generierte Beschreibung



# 9. DB-Entwurf

Ein Bild, das Text, Screenshot, Schrift, Reihe enthält.

Automatisch generierte Beschreibung

# 10. Git-Repository

<https://github.com/lucasberry13/Finanzmanagementsystem>

# 11. Testplan

|  |  |
| --- | --- |
| **ID** | **T01** |
| **Beschreibung** | Hinzufügen, Bearbeiten und Löschen von Transaktionen |
| **Voraussetzung** | Eine bestehende Datenbank mit mindestens einem Nutzer und Transaktionen |
| **Test-Schritte** | * 1. Eine neue Transaktion wird mit gültigen Daten hinzugefügt   2. Eine bestehende Transaktion wird bearbeitet und die Änderungen gespeichert   3. Eine bestehende Transaktion wird gelöscht |
| **Erwartetes Resultat** | Die Transaktionen werden korrekt in der Datenbank hinzugefügt, bearbeitet oder gelöscht |
|  | Änderungen sind in der Transaktionsübersicht sichtbar |

|  |  |
| --- | --- |
| **ID** | **T02** |
| **Beschreibung** | Validierung von Kategorisierungen |
| **Voraussetzung** | Kategorien sind im System definiert. Eine Transaktion ohne Kategorie existiert |
| **Test-Schritte** | 1. Eine Kategorie wird einer Transaktion manuell zugewiesen 2. Eine neue Transaktion wird hinzugefügt und automatisch kategorisiert |
| **Erwartetes Resultat** | Transaktionen werden korrekt manuell und automatisch kategorisiert |
|  | Automatische Kategorien basieren auf definierten Regeln |

|  |  |
| --- | --- |
| **ID** | **T03** |
| **Beschreibung** | Benachrichtigungen bei Budgetüberschreitung |
| **Voraussetzung** | Ein Budget mit einem definierten Grenzwert ist aktiv |
| **Test-Schritte** | Eine Transaktion wird hinzugefügt, die das Budget überschreitet |
| **Erwartetes Resultat** | Eine Benachrichtigung wird ausgelöst, die den Benutzer über die Budgetüberschreitungen informiert |
|  |  |

|  |  |
| --- | --- |
| **ID** | **T04** |
| **Beschreibung** | Validierung der Datenkonsistenz und Visualisierung in Diagrammen |
| **Voraussetzung** | Transaktionen sind im System erfasst und korrekt kategorisiert |
| **Test-Schritte** | 1. Ein Diagramm wird erstellt, das die Transaktionskategorien visualisiert 2. Ein Diagramm wird erstellt, das den Cashflow darstellt |
| **Erwartetes Resultat** | Die Diagramme stellen die Daten korrekt dar und sind konsistent mit den gespeicherten Transaktionen |
|  |  |

|  |  |
| --- | --- |
| **ID** | **T05** |
| **Beschreibung** | Export der Diagramme in verschiedenen Formaten |
| **Voraussetzung** | Diagramme sind erstellt |
| **Test-Schritte** | 1. Ein Diagramm wird als PDF exportiert 2. Ein Diagramm wird als Excel-Datei exportiert |
| **Erwartetes Resultat** | Exportierte Dateien enthalten die korrekten Diagrammdaten und sind lesbar |
|  |  |

**Unit-Tests**:

* Fokus auf API-Datenabruf und Datenbankintegrität.
* Tests für Sicherheitsfunktionen wie Authentifizierung.