

Software Engineering

**Abschlussbericht Projekt: Budget Tracker
Team 07**

Noah-Josua Hartmann, 2376662

Michelle Koops, 2651042

Avery Lönker, 2645331

Laura Siekierski, 2618388

Hochschule für Angewandte Wissenschaften Hamburg

09.07.2024

Inhaltsverzeichnis

1. Einleitung	3
2. Anforderungen	4
2.1 Anforderungsanalyse	4
2.1.1 Benutzeranforderungen	5
2.1.2 Systemanforderungen	6
3. Projektmanagement & Vorgehensmodell	6
3.1 Vorgehensmodell "Scrumban"	7
3.2 Arbeitsweise im Team	7
3.3 Versionierung	8
4. Entwurf	8
4.1 Softwarearchitektur	9
4.1.1 Drei-Schichten-Architektur	9
4.1.2 Publisher-Subscriber-Architektur	9
4.1.3 Client-Server-Architektur	10
4.1.4 Komponentenbasierte Architektur	10
4.2 Entwurfsmuster	10
4.2.1 Beobachtermuster	10
4.2.2 Model-View-ViewModel Muster	10
4.2.3 Singleton Muster	11
4.2.4 Dependency Injection und Inversion of Control	11
4.3 Datenbankstruktur	12
4.4 UML-Diagramme	12
4.4.1 Use-Case-Diagramme	12
4.4.2 Klassendiagramm	12
4.4.3 Komponentendiagramm	13
4.4.4 Sequenzdiagramm	13
4.4.5 Datenbank: Entity-Relationship-Modell	14
4.4.6 Datenbank: Relationenmodell	14
5. Implementierung	14
5.1 Team	15
5.2 Michelle	15
5.3 Laura	16
5.4 Avery	16
5.5 Noah	16
5.6 Versionierung	17
5.7 Dokumentation	17
6. Testing	18
6.1 Tools	18
6.2 Implementierung	18
7. CI/CD	19
8. Fazit	19
8.1 Endprodukt	19

8.2 Learnings	20
9. Anhang	21
9.1 Screenshots der Anwendung	21
Startansicht ohne Einträge	21
Erstellung eines neuen Eintrags	21
Ansicht nach Erstellung mehrerer Einnahmen und Ausnahmen	21
Ansicht nach Klicken auf Button "Einnahmen"	21
Ansicht nach Klicken auf Button "Ausgaben"	21
Ansicht nach Klicken auf Button "Gesamt"	21
9.2 Diagramme	22
9.2.1 UML Klassendiagramm	22
9.2.2 UML Komponentendiagramm	23
9.3 Zeiterfassung	25
9.4 Gantt-Diagramm	26
9.5 Quellen	26

1. Einleitung

Im Rahmen des Kurses Software Engineering wurde der Entwicklungsprozess von Software, von der Anforderungsanalyse bis zum Deployment, anhand eines Projektes nachvollzogen. Das Projekt bestand in der Entwicklung einer **Webanwendung zur Erfassung von Einnahmen und Ausgaben**. Die Ergebnisse werden in dem folgenden Bericht dargestellt.

Das Backend unserer Anwendung wurde mit Java [Spring Boot](#) entwickelt. Zum Testen wurden unter anderem die Frameworks [JUnit 5](#) und [Mockito](#) verwendet. Das Logging wurde mit [Log4J](#) umgesetzt und die Dokumentation wurde mit [Springdoc OpenAPI](#) erzeugt. Im Frontend setzten wir [Angular](#) mit [PrimeNG](#) und [PrimeIcons](#) ein. Als Testing-Framework verwendeten wir [Jasmine](#) zusammen mit [Karma](#). Die Dokumentation wurde mit [Compodoc](#) erzeugt.

GitHub-Repositories

- Frontend: https://github.com/noahjosua/BudgetTracker_FRONTEND/tree/rel/1.0
- Backend: https://github.com/noahjosua/BudgetTracker_BACKEND/tree/rel/1.0

Deployment (online bis 30.07.2024):

- [Anwendung](#)
- [Dokumentation Frontend](#)
- [Dokumentation Backend](#)

2. Anforderungen

Egal ob Industrie- oder Hochschulkontext: Projekte entstehen aus Ideen bzw. Aufträgen. In unserem Fall sind wir unsere eigenen Auftraggeber und haben uns entschieden, eine Webanwendung zur Erfassung von Einnahmen und Ausgaben zu entwickeln. Die Anforderungen seitens des Kurses bestehen darin, den Prozess des Software Engineerings anhand der Entwicklung einer Software nachzuvollziehen und zu verstehen. Außerdem gibt es Deadlines, zu denen die Software bzw. dieser Bericht fertiggestellt sein muss. Um die Anforderungen an unsere Software klar definieren zu können, beschäftigen wir uns zunächst mit der Anforderungsanalyse.

2.1 Anforderungsanalyse

Unter einer Anforderungsanalyse wird das systematische Vorgehen bei der Definition, Erstellungen und Überprüfung von Anforderungen an ein Softwaresystem verstanden. Um die effektive Erstellung eines Softwareprodukts zu gewährleisten, umfasst der Prozess einer Anforderungsanalyse mehrere Aufgaben, die dabei helfen, die Anforderungen der Stakeholder - in unserem Fall sind das wir - zu verstehen, aufzuzeichnen und zu verwalten. (vgl. GeeksforGeeks 2024).

Anforderungsermittlung

Der Prozess der Anforderungsanalyse beginnt mit der Anforderungsermittlung. Konkret bedeutet das in unserem Fall ein Brainstorming, in dem wir unsere jeweiligen Vorstellungen

von und Anforderungen an eine Software zur Erfassung von Einnahmen und Ausgaben frei zusammengetragen haben. In Hinblick auf den, vom Kurs vorgegebenen, zeitlichen Rahmen haben wir unsere Ideen bereits an dieser Stelle in drei Kategorien priorisiert: "Must have", "Nice to have", "Bonus". Dieser Schritt gehört eigentlich zur Spezifikation.

Analyse & Spezifikation

Aufgrund der, wie unter [Punkt 3.1 Vorgehensmodell: Scrumban](#) beschrieben, von uns gewählten agilen Arbeitsweise, haben wir die ermittelten Anforderungen anschließend mit Hilfe von User Stories spezifiziert und festgehalten. Bei einer User Story handelt es sich um "eine informelle, allgemeine Erklärung eines Software-Features, die aus Sicht des End-Nutzers verfasst wurde. Der Zweck einer User Story ist es, zu formulieren, wie eine einzelne Aufgabe dem Kunden einen bestimmten Wert liefert" (Rehkopf o. D.).

Jede User Story besteht dabei aus einem Titel, einer Beschreibung, Akzeptanzkriterien und einer Definition of Done.

Bei der Verfassung der User Stories wurde darauf geachtet, dass diese einen realistischen Umfang für die Dauer eines Sprints (eine Woche) haben. Nach der initialen Verfassung der User Stories in einem Google Drive Dokument überführten wir jede Story als Ticket in unser, unter [Punkt 3.2. Arbeitsweise im Team](#) beschriebenes Kanban-Board.

Die Phase der Spezifikation dient der Erstellung formaler Softwareanforderungsmodelle, d.h. "die im Analyseschritt ermittelten Anforderungen werden klar, konsistent und eindeutig dokumentiert" (GeeksforGeeks 2024).

Unsere funktionalen und nicht-funktionalen Anforderungen haben wir bereits in Form von User Stories verfasst. Zusätzlich wurden in dieser Phase verschiedene Diagramme (siehe [Kapitel 4.4 UML-Diagramme](#)) als visuelle Hilfsmittel angefertigt.

Validierung

Bei dem Arbeitsschritt der Validierung wird sichergestellt, dass die erstellte Software den Kundenanforderungen entspricht. Durch unser gewähltes Vorgehensmodell und die Definition of Done an jeder User Story war sehr klar definiert, welche Anforderungen jeweils erfüllt sein müssen, um das Ticket in den Review schieben zu dürfen. Unterstützt wurde die Validierung zusätzlich durch Testing und unsere wöchentlichen Sprint Termine.

2.1.1 Benutzeranforderungen

Wie unter [Punkt 2.1 Anforderungsanalyse](#) bereits beschrieben, ermittelten wir unsere Benutzeranforderungen durch ein Brainstorming, gefolgt vom Verfassen der User Stories inklusive Akzeptanzkriterien und einer Definition of Done. Unterstützend wurden in diesem Schritt außerdem Mockups angefertigt, um eine visuelle Diskussionsgrundlage zu haben. Während des Prozesses wurde schnell klar, dass beispielsweise zwischen einmaligen und wiederkehrenden Einnahmen bzw. Ausgaben unterschieden werden muss. Auch stellte sich heraus, dass wir unterschiedliche Vorstellungen davon hatten, ob unsere Anwendung über eine Userverwaltung verfügen soll oder nicht. Letzten Endes wurde sich dafür entschieden, die User Stories zu einmaligen Einnahmen/Ausgaben in die Kategorie "Must Have" und die Stories zu wiederkehrenden Einnahmen/Ausgaben in die Kategorie "Nice to Have" zu schieben. Die Userverwaltung kategorisierten wir als "Bonus". Diese Priorisierung liegt darin begründet, dass es eine Anforderung des Kurses ist, den Prozess des Software Engineerings zu verstehen. Um uns darauf konzentrieren zu können, erschien es uns

sinnvoll, zunächst eine Anwendung mit minimaler Funktionalität zu erstellen und alle weiteren Features erstmal außen vor zu lassen.

2.1.2 Systemanforderungen

Die Systemanforderungen festzulegen, war zu Beginn etwas schwierig, da es in der Gruppe den Wunsch nach einer Android-Anwendung, einer Desktop-Anwendung und einer Webanwendung gab.

Wir waren uns vorerst einig, dass wir die Entwicklung einer Anwendung für Android oder den Desktop interessanter finden würden, als die einer Webanwendung. Relativ schnell wurde jedoch klar, dass wir uns gegen Android entscheiden, da wir das Ziel, den Prozess des Software Engineerings zu verstehen, nicht aus den Augen verlieren wollen, indem wir uns erst noch eine für uns alle neue Plattform einarbeiten müssen.

Somit standen noch Webanwendung oder Desktop-Anwendung zur Auswahl und wir einigten uns schon mal darauf, das Backend mit Java zu programmieren, da wir uns mit der Sprache alle sicher fühlen. Hier wurde sich für JDK 21 entschieden, da dies, zum Startzeitpunkt des Projektes, der letzte LTS Release der Java SE Platform ist.

Für das Frontend stellte die Verwendung des TypeScript basierten Frameworks [Angular](#) eine Option dar, da ein Gruppenmitglied bereits viel Angular Erfahrung mitbringt. Da es sich bei Angular um ein Framework für Webapplikationen handelt, wurde zunächst nochmal recherchiert, welche Möglichkeiten es für die Implementierung des Frontends einer Desktop App gibt. Letztlich wurde sich dann für eine Webanwendung unter der Verwendung von Angular entschieden, damit bestehende Ressourcen genutzt werden und sich nicht in ein neues Framework eingearbeitet werden muss. Hier wurde sich für Angular 17 entschieden (aktuellste Version zum Startzeitpunkt des Projektes). Außerdem wurde die [PrimeNG](#) Library (ebenfalls Version 17) über den Paketmanager npm als Dependency verwendet. PrimeNG bietet eine Vielzahl von UI Komponenten wie Button, Tabellen etc.

Für das Backend fiel die Wahl auf Java [Spring Boot](#), da auch für dieses Framework bereits Expertise in der Gruppe vorhanden ist. Weitere Gründe sind, dass Spring Boot über einen eingebetteten Tomcat-Server verfügt, sodass die Anwendung als eigenständige Java-Anwendung ausgeführt werden kann. Des Weiteren erlaubt Spring Boot eine einfache Implementierung von RESTful APIs, die wir für die Kommunikation zwischen Backend und Frontend benötigen. Auch das Schreiben von Unit- und Integrationstests wird durch die Integration von Test-Frameworks wie JUnit und Mockito erleichtert. Das Projekt wurde per [Spring Initializr](#) mit Version 3.2.5 erstellt.

Bezüglich eines Datenbankmanagementsystems gab es mehrere Möglichkeiten, mit der Einschränkung, dass es sich um eine relationale Datenbank handeln sollte, deren Einrichtung möglichst unkompliziert ist. Somit fiel die Wahl für die Zeit der Entwicklung auf die [H2 Database Engine](#) (Version 2.2.224) im embedded connection mode.

Für das Deployment auf [Render](#) wurde eine PostgreSQL Datenbank verwendet.

3. Projektmanagement & Vorgehensmodell

Das folgende Kapitel beleuchtet die Wahl des Vorgehensmodells sowie verschiedene Aspekte unserer Arbeitsweise im Team.

3.1 Vorgehensmodell “Scrumban”

Eine der ersten Entscheidungen, die wir im Team getroffen haben, war die Entscheidung für das Vorgehensmodell Scrumban. Wie der Name bereits vermuten lässt, handelt es sich dabei um eine Vorgehensweise, die das agile Projektmanagement Framework Scrum mit dem ebenfalls agilen Projektmanagement Tool Kanban vereint.

Für Scrumban haben wir uns aus verschiedenen Gründen entschieden. Als Gruppe haben wir wenig Erfahrung im Software Engineering und Software Development, weshalb sich eine agile Arbeitsweise aus unserer Sicht sehr gut eignet, weil wir uns im Laufe des Projektes neue Fähigkeiten erarbeiten müssen. So müssen sich Teile der Gruppe beispielsweise in Angular einarbeiten, andere in Spring Boot und alle in Test-Frameworks wie JUnit und Mockito. Außerdem können wir, aufgrund unserer fehlenden Erfahrung, davon ausgehen, dass wir auf Probleme stoßen werden, weil wir bei der Planung etwas übersehen haben oder eine User Story doch nicht so umsetzbar ist, wie von uns geplant. Mit einer agilen Arbeitsweise ist es uns möglich, flexibel und schnell auf auftretende Probleme und Änderungen zu reagieren.

Wichtig zu erwähnen ist, dass wir uns frei an den Werkzeugen von Scrum und Kanban bedient haben, um eine für uns und das Projekt passende Arbeitsweise zu implementieren. Was das konkret bedeutet, wird im nächsten Abschnitt erläutert.

3.2 Arbeitsweise im Team

Dadurch, dass wir uns in einem künstlich geschaffenen Rahmen bewegen und nicht Vollzeit an dem Projekt arbeiten, ergibt sich für uns die Notwendigkeit, die gewählte Methode Scrumban auf unsere Bedürfnisse anzupassen.

Zunächst bedeutet das, dass es innerhalb des Teams keine, wie von Scrum (vgl. Atlassian o. D.) vorgesehen, Rollen gibt. Stattdessen füllt das gesamte Team die Rolle des Scrum-Masters, des Product Owners, des Entwicklerteams, des Planungsteams und die des Kunden. Außerdem begrenzen wir die Dauer der Sprints auf eine Woche. Eine längere Sprintdauer erscheint uns angesichts des kurzen Semesters nicht sinnvoll.

Die einzelnen Sprint Termine erstrecken sich meist über mindestens zwei Stunden, da wir, anders als traditionell von Scrum vorgegeben, die Review, die Retrospektive und das Planning zusammengefasst haben. Die Dailys wurden komplett gestrichen.

Für kurze Absprachen zwischendurch wurde eine WhatsApp Gruppe verwendet, die Sprint Termine fanden über MS Teams statt.

Zur Organisation und Visualisierung wurde ein Kanban-Board (vgl. Atlassian o. D.) und ein Gantt-Diagramm eingesetzt. Das Gantt-Diagramm dient uns als Visualisierung unserer gesetzten Meilensteine bzw. Deadlines, die wir, ebenfalls aufgrund der extrem kurzen Projektzeit, für sinnvoll halten. Das Kanban-Board verwenden wir zur Verwaltung der Tickets und damit auch zur Visualisierung des Arbeitsflusses. Wir wollen in der Lage sein, Tickets neu zu priorisieren, zuzuweisen und zu aktualisieren. Außerdem verteilen wir zwar Tickets in

unseren Sprint-Terminen (Scrum), jedoch ist es auch möglich, sich bei Bedarf neue Tickets zuzuweisen (Pull Prinzip, Kanban) oder im Backlog anzulegen. Durch diese Kombination aus Scrum und Kanban erreichen wir eine für uns und unser Projekt geeignete Arbeitsweise.

Für unser Kanban-Board und Gantt-Diagramm verwenden wir die Software [YouTrack](#). Voneinander abhängige User Stories können dort als "ist abhängig von" oder "ist erforderlich für" definiert werden. Auch die Aufwandsschätzung sowie der tatsächliche Zeitaufwand können direkt am Ticket festgehalten werden und am Ende als Zeittabelle ausgegeben werden. Für ticketunabhängige Zeiterfassung, wie beispielsweise Sprint-Termine, nutzen wir eine Excel-Tabelle.

3.3 Versionierung

Als Versionierungstool verwenden wir Git, dessen Verwendung in der Softwareentwicklung weit verbreitet ist. Als verteiltes Versionskontrollsystem bietet es den Vorteil, dass jeder Benutzer zu jedem Zeitpunkt den vollständigen Code zur Verfügung hat, was für eine effektive Zusammenarbeit von verschiedenen Standorten von Nutzen ist.

Außerdem bietet Git uns die Möglichkeit, verschiedene Zweige anzulegen und diese nach Bedarf durch konkrete Regeln zu schützen. Pro User Story wird eine Feature Branch angelegt, welche nach Abschluss (Akzeptanzkriterien und Definition of Done sind erfüllt), über einen Pull Requests und zwei Reviews in den Main gemerged werden kann. So stellen wir sicher, dass sich auf dem Main stets eine lauffähige Version unserer Software befindet und auf den Feature Branches experimentiert werden kann.

Die Benennung der Feature Branches folgt stets dem gleichen Muster:
feat/<Ticketnummer>/<Teammitglied>

Ein weiterer Vorteil von Git ist, dass es sich nahtlos in viele CI/CD Systeme und IDEs integrieren lässt. Am Ende des Projektes wird auf beiden Repositories eine Release Branch "rel/1.0" angelegt, welche mit der PaaS-Plattform [Render](#) bzw. der Hosting Plattform [Netlify](#) verbunden ist.

4. Entwurf

"Fußend auf der Anforderungsanalyse- und definition erfolgt mit dem Software-Entwurf eine Projektphase, die sich gezielt mit der Übersetzung der Anforderung in verschiedene Systemkomponenten beschäftigt. [...] Ein Software-Entwurf besteht dabei aus verschiedenen Teilen, die genau definieren, wie die Programmierung letztlich erfolgt [...]" und am Ende "steht ein möglichst spezifischer Überblick darüber, wie die Anforderungsanalyse technisch erfüllt werden soll, sowie ein kompletter Abriss über das gesamte System." (Rauch 2023).

Auf die verschiedenen Teile, aus denen ein Entwurf besteht, sind wir bereits in Abschnitt 2.1.2 *Systemanforderungen* eingegangen. Hier noch einmal im Überblick:

- Entwicklungsumgebung: IntelliJ IDEA und Webstorm bzw. Visual Studio Code
- Programmiersprache: Java (Backend) und Typescript (Frontend)
- Libraries (Backend): [Gson](#), [Springdoc OpenAPI](#)
- Libraries (Frontend): [PrimeNG](#), [PrimeIcons](#), [Compodoc](#), [ngx-translate](#), [rxjs](#)

- Frameworks (Backend): [Java Spring Boot](#), [JUnit 5](#), [Mockito](#), [Spring Test](#), [Log4J](#)
- Framework (Frontend): [Angular](#), [Jasmine](#)
- APIs: RESTful API für Management von Einnahmen und Ausgaben

Um die Zusammenhänge zwischen den Anforderungen und dem zu konstruierenden System zu beschreiben, müssen sich zunächst Gedanken über die Softwarearchitektur gemacht werden (vgl. <https://gi.de/informatiklexikon/software-architektur>).

4.1 Softwarearchitektur

“Die Architektur eines Softwaresystems beschreibt dieses als Komponenten zusammen mit den Verbindungen, die zwischen den Komponenten bestehen.” (Informatik o. D.).

In der Anforderungsanalyse war die Frage nach dem “Was” zu klären - die Planung der Architektur beantwortet die Frage nach dem “Wie”. Dazu haben wir uns mit verschiedenen Architekturmustern beschäftigt, diese diskutiert und am Ende eine Entscheidung getroffen. Im Folgenden gehen wir auf die für unser Projekt relevantesten drei Muster genauer ein.

4.1.1 Drei-Schichten-Architektur

Die Drei-Schichten-Architektur unterteilt eine Software in folgende Schichten: die Darstellungsschicht (Benutzerschnittstelle), Anwendungsschicht und Datenschicht (vgl. Was ist eine dreischichtige Architektur? | IBM o. D.) .

Da wir eine dialogorientierte Software entwickeln, welche Benutzereingaben entgegennehmen, verarbeiten, speichern, bearbeiten, löschen und wieder anzeigen können soll, bietet sich die Verwendung dieses Architekturmusters an.

Die Darstellungsschicht ist in unserem Fall das mit Angular implementierte Web-Frontend. Die Anwendungsschicht stellt unser mit Java Spring Boot umgesetzte Backend Server dar und die Datenhaltungsschicht besteht in der Entwicklungsphase aus einer H2 Datenbank - später wird diese durch eine PostgreSQL Datenbank ausgetauscht.

Durch die Verwendung von Angular verwenden wir zur Strukturierung unserer Darstellungsschicht das unter [Punkt 4.2.2](#) beschriebene Model-View-ViewModel (MVVM) Entwurfsmuster.

Unser Backend folgt selbst wieder einer Schichtenarchitektur, um die Geschäftslogik zu strukturieren. Die mit `@Controller` annotierten Klassen stellen Präsentationsschichten dar, die mit `@Service` annotierten Klassen die Business-Logik-Schichten und die mit `@Repository` annotieren die Persistenzschichten (vgl. Augsten/Koller 2021).

4.1.2 Publisher-Subscriber-Architektur

Bei einer Publisher-Subscriber-Architektur handelt es sich um ein Muster, welches Datenproduzenten von Datenkonsumenten entkoppelt (vgl. IBM Integration Bus 10.0.0 o. D.).

Wir werden diese Architektur in unserem Angular-Frontend an unterschiedlichen Stellen verwenden.

In Angular findet sich das Muster vor allem im Umgang mit Subjects (Observables) und Subscriptions aus der RxJS Bibliothek (Reactive Extensions for JavaScript) und mit Event-Emittern. Ein anschauliches Beispiel wären unsere Services. Sie verfügen über Subjects (Observables), welche als Broker fungieren und auf welche verschiedene Komponenten mittels einer Subscription abonnieren können. Zum Beispiel abonniert unsere `app.component.ts` auf das `expensesUpdated` Subject unseres `ExpenseService`, updated mit den Daten aus dem `ExpenseService` ihre eigene Liste von Expenses und gibt diese dann an verschiedene andere Komponenten weiter, die ebenfalls Subscriber sind.

4.1.3 Client-Server-Architektur

Mit der Verwendung einer Client-Server-Architektur existiert eine klare Verteilung von Aufgaben zwischen Client und Server: Der Server ist für die Bereitstellung von Diensten und Informationen verantwortlich, während der Client die bereitgestellten Dienste und Informationen abfragt und nutzt. Die Regeln, nach denen die Kommunikation zwischen Clients und Server stattfindet, sind in Form von Protokollen definiert (vgl. Client-Server-Architektur o. D.).

In unserem Fall ist die Kommunikation für Clients mit unserem mit Java Spring Boot implementierten Server per HTTP-Protokoll möglich.

4.1.4 Komponentenbasierte Architektur

Das Framework Angular verwendet eine komponentenbasierte Architektur. Dies bedeutet, dass jede Komponente ein eigenständiges Teil mit bestimmten Funktionen darstellt und erst wenn alle Komponenten miteinander kombiniert werden, schaffen sie ein vollständiges Software-System.

Durch die Verwendung dieser Architektur wird u.a. ein hoher Grad an Modularität und Wiederverwendbarkeit erreicht (vgl. Tectrain, 2024).

4.2 Entwurfsmuster

“Ein Entwurfsmuster beschreibt eine formale Lösung für ein Teilproblem eines Entwurfs. Es geht dabei nicht um die konkrete Implementierung von Logik sondern explizit um den Entwurf objektorientierter Strukturen.” (IT-Talents 2024).

Während ein Architekturmuster sich auf die gesamte Softwarearchitektur bezieht, liefern Entwurfsmuster Lösungen für spezifische Aufgaben innerhalb des Systems. Sie lassen sich in drei Bereiche einteilen: Strukturmuster, Verhaltensmuster und Erzeugungsmuster.

Auch hier haben wir zu einer Vielzahl an Mustern recherchiert, diskutiert und wollen im Folgenden auf die für unsere Anwendung relevantesten Muster genauer eingehen.

4.2.1 Beobachtermuster

Das Beobachtermuster gehört zur Familie der Verhaltensmuster und definiert eine One-to-Many Abhängigkeit zwischen Objekten. Alle abhängigen Objekte, auch Beobachter genannt, werden automatisch benachrichtigt und aktualisiert, wenn ein Objekt, auch Subjekt genannt, seinen Zustand ändert (vgl. GeeksforGeeks 2024). Es fällt auf, dass dieses Muster sich hervorragend für die Umsetzung der Publisher-Subscriber-Architektur eignet und entsprechend in unserer Implementierung des Frontends Platz finden wird.

4.2.2 Model-View-ViewModel Muster

Das Model-View-ViewModel-Muster (MVVM-Muster) besteht aus drei Kernkomponenten: der View, dem ViewModel und dem Model. Dadurch wird die Geschäftslogik- und Präsentationslogik klar von der Benutzeroberfläche getrennt. "Eine klare Trennung zwischen Anwendungslogik und Benutzeroberfläche trägt zur Behebung zahlreicher Entwicklungsprobleme bei und erleichtert das Testen, Verwalten und Weiterentwickeln von Anwendungen. Des Weiteren kann sie die Wiederverwendungsmöglichkeiten von Code erheblich verbessern und die Zusammenarbeit von Entwicklern und Benutzeroberflächendesignern bei der Entwicklung ihrer jeweiligen App-Komponenten vereinfachen." (Michaelstonis 2024).

Unter der Verwendung von Angular findet sich das MVVM-Muster innerhalb jeder Komponente wieder. Das Model repräsentiert eine Datenstruktur - in Angular sind Models meist TypeScript Klassen oder Interfaces. In unserer Anwendung haben wir ein Model namens "Entry", welches eine Einnahme bzw. Ausgabe repräsentiert. Die View ist die Benutzeroberfläche der Anwendung, die in HTML definiert wird. Das ViewModel ist in Angular die Komponente. Diese verbindet das Model mit der View, indem sie Daten aus dem Model bereitstellt und die Interaktionen der Benutzeroberfläche steuert. Angular bietet eine starke Unterstützung für Datenbindung zwischen View und ViewModel, was bedeutet, dass Änderungen an Daten im ViewModel automatisch in der View reflektiert werden können und umgekehrt. Services werden verwendet, um Geschäftslogik und Datenzugriff von der Komponente zu trennen (vgl. Angular and MVVM: Model-View-ViewModel Pattern 2023).

4.2.3 Singleton Muster

"Das Singleton Muster gehört zur Kategorie der Erzeugungsmuster [...]. Seine Aufgabe besteht darin, zu verhindern, dass von einer Klasse mehr als ein Objekt erstellt werden kann. Das wird dadurch erreicht, dass das gewünschte Objekt in einer Klasse selbst erzeugt und dann als statische Instanz abgerufen wird." (Redaktion 2020).

In Angular wird dieses Muster in Bezug auf Services auf zwei Weisen implementiert: Entweder die *providedIn* Eigenschaft im `@Injectable`-Decorator wird auf 'root' gesetzt (empfohlen) oder der Service wird im AppModule im `@NgModule` providers Array registriert (vgl. Angular o. D.).

In unserer Anwendung folgen die Services, der `MessageService` von PrimeNG sowie das `TranslateModule` aus der `@ngx-translate` Bibliothek dem Singleton Muster.

4.2.4 Dependency Injection und Inversion of Control

Inversion of Control (IoC) ist ein Prinzip, welches einem Framework ermöglicht, die Kontrolle über den Programmfluss zu übernehmen und Aufrufe an unseren benutzerdefinierten Code zu machen. IoC kann durch verschiedene Mechanismen, darunter Dependency Injection (DI), erreicht werden (vgl. Crusoveanu/Crusoveanu 2024).

Der Spring IoC Container ist der Kern des Spring Frameworks. Er erstellt die Objekte (auch Beans genannt), konfiguriert und stellt ihre Abhängigkeiten zusammen und verwaltet ihren gesamten Lebenszyklus. Der Container verwendet Dependency Injection, um die Komponenten zu verwalten, aus denen die Anwendung besteht. Für die Implementierung von DI gibt es verschiedene Möglichkeiten (vgl. Kumar 2023).

In unserer Anwendung verwenden wir unter anderem Constructor-Injection (beispielsweise in unseren Controller Klassen) sowie Field-Injection (beispielsweise in den Testklassen für unsere Controller). Dabei verwenden wir jeweils die `@Autowired` Annotation, welche dafür sorgt, dass Spring bei der Initialisierung einer Bean automatisch nach den benötigten Dependencies sucht und diese injiziert.

4.3 Datenbankstruktur

Da wir zum Speichern der Einnahmen und Ausgaben eine Datenbank verwenden, muss über die Struktur dieser nachgedacht werden.

Unsere Datenbankstruktur ist wenig komplex, da wir das Speichern von wiederkehrenden Einnahmen bzw. Ausgaben nicht unterstützen und die Anwendung auch über keine Benutzerverwaltung verfügt.

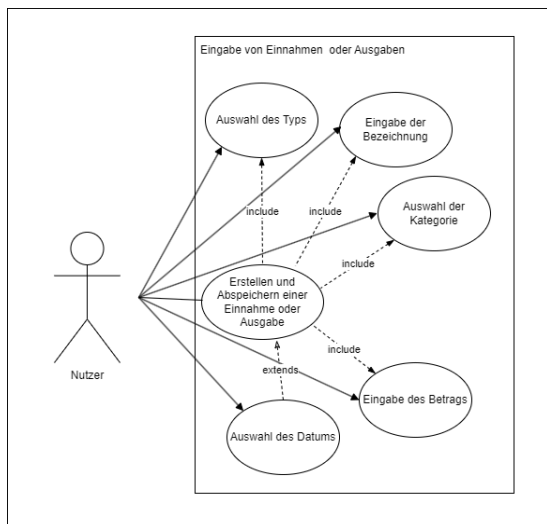
Wir benötigen also lediglich zwei Tabellen (eine für Einnahmen und eine für Ausgaben) mit identischem Aufbau. Diagramme zur Veranschaulichung finden sich im folgenden Abschnitt.

4.4 UML-Diagramme

Um unseren Softwareentwurf visuell zu stützen und eine gute Diskussionsgrundlage zu haben, wurden verschiedene UML-Diagramme entworfen, die im Folgenden vorgestellt werden. **Größere und detailliertere Ansichten aller Diagramme sind im Anhang unter [Punkt 9.2 Diagramme](#) zu finden.**

4.4.1 Use-Case-Diagramme

“Das UML Use Case Diagramm [...] ist ein UML-Verhaltensdiagrammtyp und wird häufig zur Analyse verschiedener Systeme verwendet. Es ermöglichen Ihnen, die verschiedenen



Rollentypen in einem System zu visualisieren und zu sehen, wie diese Rollen mit dem System interagieren.” (Siriwardhana 2021)

Abbildung 1 zeigt das Use-Case-Diagramm für den Aufgabenbereich des Erstellens neuer Einträge. Für die Anwendung gibt es nur die Rolle des Nutzers, welcher bei dem Erstellen eines neuen Eintrags Typ, Bezeichnung, Kategorie und Betrag angeben muss, damit diese erfolgreich gespeichert werden kann. Ein Datum muss nicht zwangsläufig gewählt werden, da der aktuelle Tag automatisch als Standard angegeben wird.

Abbildung 1: UML Use-Case-Diagramm zur Eingabe von einer Einnahme bzw. Ausgabe

4.4.2 Klassendiagramm

Klassendiagramme zeigen die statische Struktur eines Systems durch Klassen, deren Attribute, Methoden und Beziehungen zueinander. Da Angular einer komponentenbasierten Architektur folgt, wurde lediglich für das Backend ein Klassendiagramm erstellt

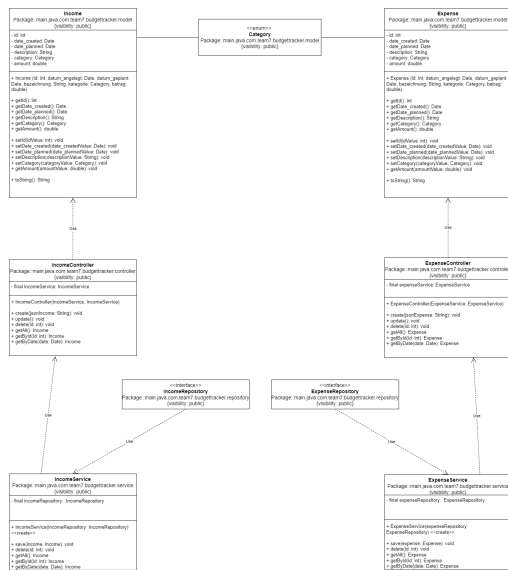


Abbildung 2 zeigt die Struktur unserer Software mit zwei Hauptkomponenten: Income (Einkommen) und Expense (Ausgaben). Für beide gibt es ähnliche Strukturen: **Modellklassen** mit Attributen wie ID, Datum, Beschreibung, Kategorie und Betrag, **Controller-Klassen** zur Verwaltung der Geschäftslogik, **Service-Klassen** für die Geschäftslogik und Datenzugriff und **Repository-Interfaces** für den Datenbankzugriff.

Abbildung 2: Darstellung unseres Backends als UML Klassendiagramm

4.4.3 Komponentendiagramm

“Komponentendiagramme werden verwendet, um die Organisation von Systemkomponenten und die Abhängigkeitsbeziehungen zwischen ihnen zu visualisieren. Sie bieten einen Überblick über die Komponenten innerhalb eines Systems auf hoher Ebene.” (Siriwardhana 2022).

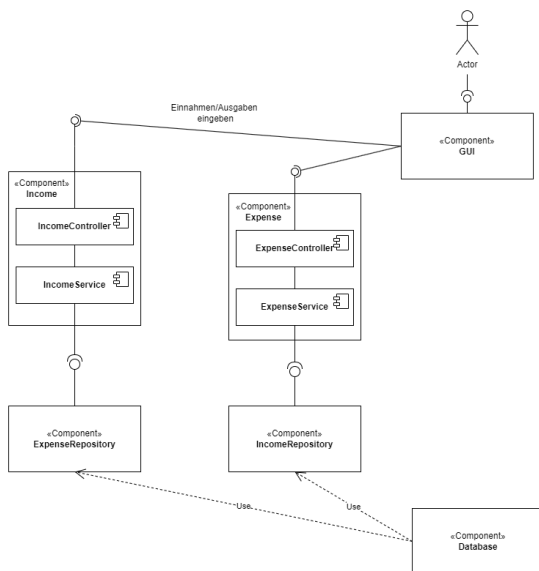


Abbildung 3 zeigt das Komponentendiagramm für unser Backend. Der User greift über unser GUI auf die Einkommen-Komponente zu, welche dem User die Möglichkeit (Schnittstelle) bietet, neue Einnahmen zu erstellen und später (durch die bereitgestellte Schnittstelle des Repositories) zu speichern. Innerhalb dieser Komponente befinden sich also der jeweilige Service sowie der Controller. Über die Repositories werden die eingegebenen Daten in der Datenbank gespeichert.

Abbildung 3: Darstellung unseres Backends als Komponentendiagramm

4.4.4 Sequenzdiagramm

“Ein Sequenzdiagramm ist eine Art von Interaktionsdiagramm [...] und bildet die Interaktion zwischen einer Gruppe von Objekten sowie die Reihenfolge ab.” (UML sequenzdiagramm o. D.).

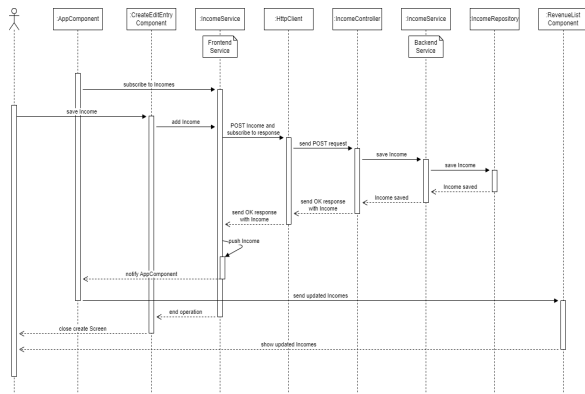


Abbildung 4: Prozess der Erstellung einer neuen Einnahme als Sequenzdiagramm

Abbildung 4 zeigt das Sequenzdiagramm für den Fall, dass eine neue Einnahme erstellt, in der Datenbank gespeichert und auf der Website angezeigt wird. Dabei schickt zuerst das Frontend die Daten der Einnahme an das Backend, welches diese in der Datenbank speichert und Rückmeldung gibt, dass der Prozess erfolgreich abgeschlossen ist. Die AppComponent, die auf eine Liste, die alle Einnahmen beinhaltet, registriert ist, benachrichtigt daraufhin die RevenueList Component, die nun alle Einnahmen einschließlich der eben erstellten anzeigt.

4.4.5 Datenbank: Entity-Relationship-Modell

“Das Entity-Relationship-Modell - abgekürzt mit ER-Modell oder ERM - dient als Grundlage für einen Datenbankentwurf.” (Datenbanken - für Anfänger und Profis 2023).

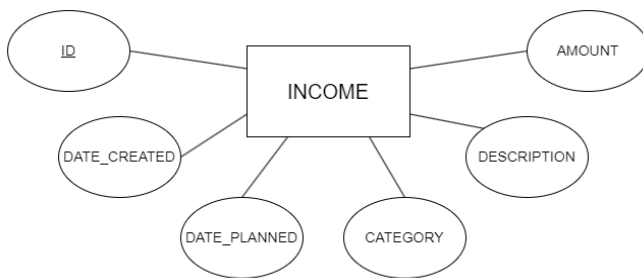


Abbildung 5 zeigt ein ERM für Einnahmen.

Da wir in unserer Implementierung keine Beziehung zwischen den beiden Tabellen haben, besteht unser ERM lediglich aus Entitäten (Rechteck) und Attributen (Oval).

Abbildung 5: Entität “Einnahme” als ERM

4.4.6 Datenbank: Relationenmodell

Ein Relationenmodell stellt die “Beziehungen zwischen verschiedenartigen Daten in einer strukturierten Art und Weise” dar. “Es ist ein wichtiges Werkzeug für die Organisation, das Design und die Verwaltung von Datenbanken.” (AlleAktien GmbH 2024).

Abbildung 6 zeigt das Relationenmodell unserer Datenbank basierend auf dem Entity-Relationship-Modell aus [Punkt 4.4.5](#). Dies war bereits sehr einfach, da wir kaum Beziehungen innerhalb unserer Datenbank haben. Dementsprechend wurde aus der Income-Entity aus Abbildung 5 eine Datenbankobjekt in Form einer Tabelle mit mehreren Zeilen (entsprechend den Attributen sowie deren Datentypen).

Income		
id	INT	PK
date_created	DATE	
date_planned	DATE	
description	VARCHAR(255)	
category	VARCHAR(255)	
amount	DECIMAL(10, 2)	

Abbildung 6: Darstellung unseres Einnahme-Objektes als Relationenmodell

5. Implementierung

Im Nachfolgenden gehen wir kurz auf unsere Arbeit im Team ein. Anschließend erläutert jedes Teammitglied seine eigene Arbeit.

5.1 Team

Zu Beginn haben wir uns über Projektmanagement und die Wahl des Vorgehensmodells Gedanken gemacht. Wie bereits unter [Punkt 3, Projektmanagement & Vorgehensmodell](#) beschrieben, fiel die Wahl auf Scrumban als Vorgehensmodell und ein Kanban-Board zur Organisation. Für unabhängige Zeiterfassungen (wie Sprinttermine) nutzten wir eine Excel Tabelle.

Anschließend starteten wir unser Projekt mit der Anforderungsanalyse, gefolgt von dem Schreiben der User Stories inklusive Akzeptanzkriterien und Definition of Done.

Wir recherchierten zahlreiche Entwurfs- und Architekturmuster, stellten uns die Ergebnisse im darauffolgenden Sprint einander vor und diskutierten, ob das entsprechende Muster für uns sinnvoll ist und wenn ja, warum es dies ist.

Der nächste Schritt war die Entwurfsphase, innerhalb welcher wir auch mit der Erstellung verschiedener UML-Diagramme begannen, die bis zum Ende immer wieder überarbeitet und angepasst werden mussten.

Kurz nachdem wir in die Entwurfsphase gestartet sind, begannen wir parallel mit dem Aufsetzen der Repositories und der Implementierung. Dabei trennten wir zunächst Frontend und Backend. Das Frontend, welches auf einem von Noah entworfenen PoC (Proof of Concept) aufgebaut wurde, funktionierte eigenständig und zeigte Mock Daten an. Das Backend wurde soweit vorbereitet, dass es in der Lage war, Daten in die Datenbank zu schreiben und aus ihr zu lesen. Anschließend wurde sichergestellt, dass das Frontend Daten von unserer API anfragen und diese auch verarbeiten und anzeigen kann. Am Ende wurde sich mit Tests beschäftigt, es war Zeit für kleinere Bugfixes und die Dokumentation des Quellcodes. Außerdem wurden sowohl das Frontend, das Backend und die Datenbank deployed.

5.2 Michelle

Ich habe mich zusätzlich zu den Aufgaben, denen sich alle im Team gewidmet haben, zu Beginn mit dem Design unserer Webanwendung beschäftigt und verschiedene Entwürfe erstellt, die zuletzt aus Zeitgründen jedoch nicht umgesetzt werden konnten.

Das Komponentendiagramm für das Backend sowie das Entity-Relationship-Modell für die Datenbank wurden ebenfalls von mir erstellt.

Weiterhin war meine Aufgabe, die Datenbankverbindung zwischen dem Frontend und Backend herzustellen, wo ich zunächst unerklärliche Probleme traf, die selbst mit kompletter Neuaufsetzung des Projekts nicht zu lösen waren und eine Weile von der Arbeit am Projekt abhielten. (Es stellte sich heraus, dass ich eine Cache-Datei der Datenbank manuell löschen musste.) Hierbei konnte ich mich jedoch stets auf die Hilfe und Unterstützung der anderen Teammitglieder verlassen.

Zusätzlich habe ich mich um Service-Funktionen gekümmert, die das Frontend mit dem Backend verbinden, wie etwa die Verarbeitung von Save-, Update- oder Delete-Funktionen. Danach wurde die Funktionalität der Buttons, um Alles oder nur Einkommen oder Ausgaben in der Übersicht zu toggeln, von mir und Avery implementiert. Auch hier wurde ich vor allem von Noah unterstützt, wann immer es nötig war.

Ich habe mich außerdem in das Testing Framework Jasmine eingelesen und anschließend zusammen mit Laura Tests für das Frontend geschrieben.

Zum Schluss habe ich die READ-ME's für die GitHub-Repositories angelegt und die Abschlusspräsentation vorbereitet.

5.3 Laura

Neben den im Team bearbeiteten Aufgaben habe ich noch verschiedene Aufgaben innerhalb des Front- und Backends erledigt.

Zu den Aufgaben, die ich übernommen habe, zählt unter anderem die Validierung der Erstellung von Einträgen im Frontend. Das stellte sich für mich zunächst als schwieriger als erwartet heraus. Dank Noahs Unterstützung hat das letztendlich aber gut geklappt.

Zusammenhängend damit, habe ich später dann auch das Verhalten der Validierung bei Updates von Einträgen implementiert.

Außerdem habe ich mit Michelle Tests für das Frontend geschrieben, hierbei insbesondere für Create-Edit-Entry-Component, IncomeService sowie ExpenseService.

Im Backend habe ich ebenfalls Tests für den IncomeController, den IncomeService sowie das IncomeRepository geschrieben. Zusätzlich habe ich mich im Backend auch mit der Verbindung zum Frontend beschäftigt, insbesondere mit dem Error-Handling der fetchByld und fetchAll Methode. Die fetchByld Methode haben wir aber letztendlich doch nicht im finalen Projekt implementiert.

Desweiteren habe ich das Backend-Klassendiagramm erstellt und mich ebenfalls dem Frontend-Komponentendiagramm gewidmet, was sich aufgrund unseres konkreten Projekts auf diese Weise aber doch als ungeeignet herausstellte, weswegen wir es schlussendlich rausgelassen haben. Zusätzlich habe ich das Icon für unsere Anwendung entworfen und eingebaut.

5.4 Avery

Zusätzlich zu den im Team bearbeiteten Aufgaben habe ich mich zu Beginn des Semesters, mit der Umsetzung von festen Einnahmen und Ausgaben beschäftigt, die es aus zeitlichen Gründen nicht mehr in die Anwendung geschafft haben. Ich habe mich mit den ersten Entwürfen der Architektur und deren Darstellung beschäftigt und das Use-Case-Diagramm sowie das Sequenzdiagramm entworfen.

Sobald es mit der Umsetzung des Entwurfs losging, habe ich verschiedene Aufgaben in Frontend und Backend übernommen. Zu den Aufgaben im Backend zählen die Implementation der Income- und Expense-Services und Repository Schnittstellen, sowie der Verbindung zum Frontend in den Income- und Expense-Controllern einschließlich des Errorhandlings für die create, update und delete Methoden. Im Frontend habe ich zu Beginn einen Mock für die Umsatzliste erstellt, mit dem wir die Anwendung fortlaufend testen konnten. Am Ende habe ich noch gemeinsam mit Michelle die Implementierung der Frontend-Buttons übernommen, die das Umschalten zwischen der Anzeige von Eingaben, Ausgaben oder beidem ermöglichen und dank Noahs Unterstützung zeitnah fertiggestellt werden konnten.

5.5 Noah

Neben den Aufgaben, die wir als Team bewältigt haben, bestand meine erste Tätigkeit darin, ein Gantt-Diagramm (siehe [Anhang 9.4 Gantt-Diagramm](#)) zu erstellen. Dazu legte ich eine Übersicht über die verbleibende Anzahl an Sprints sowie über alle zu bewältigenden Aufgaben an und überführte diese im Zuge der Erstellungs des Diagramms in Tickets. Somit hatten wir zu jedem Zeitpunkt des Projekts einen Überblick darüber, wie gut oder schlecht wir in der Zeit sind und was die nächsten Schritte sind.

Innerhalb der Entwurfsphase fertigte ich ein UI Mockup mit der Software Figma an und baute darauf basierend, wie bereits unter [Punkt 5.1 Team](#) erwähnt, ein PoC für unser Frontend.

Die Implementierungsphase begann für mich im Backend mit der Erstellung unserer Controller. Später arbeitete ich mich in die Testing-Frameworks JUnit 5 und Mockito ein und schrieb Integrationstests für den ExpenseController, den ExpenseService sowie das ExpenseRepository. Außerdem las ich mich in das Logging-Framework Log4J ein und integrierte dies in unsere Anwendung. Um ein einheitliches und übersichtliches Exception Handling zu gewährleisten, recherchierte ich außerdem, wie dies in einer Spring Boot Application am besten umzusetzen ist und implementierte dieses schließlich.

Im Frontend implementierte ich die GET-Requests in den Services, band den MessageService von PrimeNG an, um visuelles Feedback auf eine Nutzerinteraktion geben zu können und beschäftigte mich mit der ngx-translate Bibliothek für die Internationalisierung (i18n) in Angular-Anwendungen. Sie ermöglicht die Übersetzung und Sprachumschaltung zur Laufzeit. Wir benötigen diese Bibliothek, da wir sowohl unser Backend als auch unser Frontend in englischer Sprache geschrieben haben und das User Interface (UI) in deutscher Sprache sein soll.

Am Ende der Implementierungsphase übernahm ich sowohl im Backend als auch im Frontend das Refactoring sowie die Dokumentation (mehr unter [Punkt 5.7 Dokumentation](#)) des Codes. Außerdem beschäftigte ich mich mit dem Aufbau einer CI/CD Pipeline. Mehr dazu unter [Punkt 7 CI/CD](#).

5.6 Versionierung

Die Versionierung mit Git hat gut funktioniert. Durch die Verwendung von Feature-Branches war eine parallele Arbeit möglich und es kam nur selten zu geringfügigen Merger-Konflikten. Bis auf seltene Ausnahmen wurde mit jedem Sprintende jede Feature-Branch, sowohl im Backend als auch im Frontend, über einen Pull-Request in den main-Branch gemerged,

sodass wir stets eine lauffähige Version unseres Programms hatten. Um unsere Commit-Historie übersichtlich zu halten, haben wir Squash Merges verwendet.

5.7 Dokumentation

Zur Dokumentation unserer RESTful API wurde die Springdoc OpenAPI Bibliothek verwendet. Die OpenAPI Specification (OAS) definiert eine standardisierte, programmunabhängige Schnittstellenbeschreibung für HTTP-APIs und hilft dabei, die Generierung von auf Spring basierenden REST-APIs im OpenAPI 3.0-Format zu automatisieren. Sofern die Swagger UI aktiviert ist, verwendet die Bibliothek die Swagger-API, um die Dokumentation auch im HTML-Format zu erstellen. Mittels Swagger Annotationen kann diese außerdem individuell angepasst werden (vgl. Gupta/Gupta 2024).

Zur Dokumentation unseres Angular Frontends wurde Compodoc verwendet. Dabei handelt es sich um ein Open Source Dokumentations Tool für Angular Anwendungen, welches eine statische Dokumentation erzeugt (vgl. Compodoc - The missing documentation tool for your Angular application o. D.).

6. Testing

Im Backend wurden für die Controller- und Serviceklassen sowie für die Repository-Interfaces Integrationstests geschrieben.

Beispielsweise testet die Klasse `ExpenseControllerTests` die Interaktion zwischen dem `ExpenseController` und dem `ExpenseService`. Die Tests verifizieren, dass die verschiedenen Endpunkte des Controllers die erwarteten Ergebnisse liefern, wenn sie mit dem Service interagieren.

Isolierte Unit Tests wurden nicht geschrieben, da uns dies für die entsprechenden Methoden aufgrund ihrer geringen Komplexität nicht sinnvoll erschien.

Für die Komponenten im Frontend wurden Unit-Tests angelegt. Außerdem wurde zusätzlich mittels Whitebox-Testing auf Funktionalität geprüft. Dazu wurde zum einen auf jeder Komponente der "happy path" getestet, bei dem alles wie gewünscht funktioniert und zum anderen wurde versucht, durch verschiedene Kombinationen von Fehleingaben, mögliche Ausnahmezustände zu provozieren.

6.1 Tools

Für die Tests im Backend wurden die zwei Frameworks JUnit 5 und Mockito verwendet. JUnit 5 ist ein Framework für das Schreiben und Ausführen von Tests in Java und Mockito ist ein Framework zum Erstellen von Mock-Objekten für Unit-Tests. Auch die Open-Source-Bibliothek [AssertJ](#) wurde verwendet, welche dem Schreiben flüssiger und umfassender Assertion-Tests in Java dient.

Im Frontend wurde Jasmine, ein Behavior-Driven-Development (BDD) Framework benutzt. Die einfache Syntax, bestehend aus *describe* und *it-should-do*-Blöcken sorgt für gute Lesbarkeit. Beim Schreiben der Tests nutzten wir ebenfalls Karma, ein Test-Running-Tool für JavaScript, das Echtzeit-Feedback ermöglicht sowie die Angular-Testwerkzeuge TestBed und ComponentFixture.

6.2 Implementierung

Die selbsterklärenden Methodennamen der Testfälle im Backend weichen von der ansonsten verwendeten CamelCase-Notation ab, um die Lesbarkeit zu verbessern und sind wie folgt aufgebaut:

- Controller-Testklassen: `Methodenname_Status_StatusCode`
- Service-Testklassen: `Methodenname_Returns<Typ>`, `Methodenname_Success` und `Methodenname_Throws<ExceptionTyp>`
- Repository-Testklassen: `Methodenname_Returns<Typ>`

Im Frontend nutzten wir die von Jasmine bereits vorgegebene, jedoch durchaus übersichtliche Syntax. Es gibt keine Testmethoden – die einzelnen Tests bestehen vielmehr aus mehreren Blöcken:

- “Describe” Blöcke, die Tests (beispielsweise nach Funktionsweise) gruppieren und somit mehrere “it” Blöcke zusammenfassen.
- “It” Blöcke, die ähnlich wie sonst Testmethoden eine bestimmte Funktion testen.

7. CI/CD

Die Einrichtung von Continuous Integration und Continuous Delivery hat nicht stattgefunden, da wir uns mit dem gesamten Thema CI/CD erst am Ende des Projektes beschäftigt haben. Unsere Codeänderungen haben wir, wie bereits beschrieben, nach jedem Sprint manuell in dem main-Branch zusammengeführt.

Was wir jedoch eingerichtet haben, ist das Continuous Deployment. Dafür wurde eine automatische Übertragung (ausgelöst durch einen Push auf die entsprechende Branch `rel/1.0`) unseres Frontends auf die Hosting-Plattform Netlify eingerichtet. Danach beschäftigte ich (Noah) mich, leider erfolglos, mit dem Deployment unseres Backends auf [AWS \(Amazon Web Services\) Elastic Beanstalk](#). Die Lösung bestand schließlich darin, einen Dockerfile zu schreiben, welcher einen Multi-Stage-Build-Prozess für unsere Anwendung definiert. Es wird ein Gradle-Image verwendet, um die Anwendung zu bauen und ein OpenJDK-Image, um die gebaute Jar-Datei auszuführen. Durch einen Push auf die entsprechende Branch (`rel/1.0`) klonst sich die mit dem Repository verbundene PaaS-Plattform Render das Repository und baut basierend auf dem Dockerfile ein Image. Aus dem Image wird schließlich ein Container gestartet und die Anwendung darin ausgeführt. Anschließend erstellte ich noch eine PostgreSQL Instanz auf Render. Um das Backend mit der Datenbank und das Frontend mit dem Backend zu verbinden modifizierte ich schließlich noch die `application.properties`, stellte sicher, dass die Konstante “`ALLOWED_ORIGIN`” im Backend sowie das Attribut “`baseUrl`” der “`environment`” Konstante im Frontend die richtigen Werte enthalten und pushte alles erneut.

8. Fazit

Im folgenden Abschnitt wollen wir unser Endergebnis sowie unsere Zusammenarbeit im Team kurz reflektieren und zusammenfassen.

8.1 Endprodukt

Mit der entstandenen Software sind wir zufrieden. Wir konnten alle User Stories aus der Kategorie “Must Have” vollständig (in Hinblick auf Akzeptanzkriterien und DoD) implementieren und haben sogar noch ein paar aus der Sparte “Nice to Have” umgesetzt. Abbildung 7 zeigt einen Mockup, der zu Beginn des Projektes entstanden ist. Abbildung 8 zeigt unser Endprodukt.

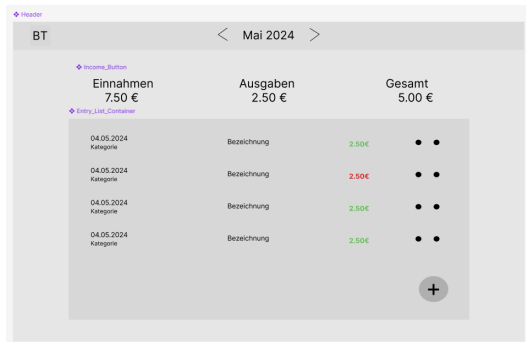


Abbildung 7: Mockup der Software

Datum	Kategorie	Bezeichnung	Summe		
01.07.2024	Gehalt	Test 1	2,5		
02.07.2024	Taschengeld	Test 1	2,5		
03.07.2024	Unterhalt	Test 3	3,5		
02.07.2024	Kapitalvermögen	Test 4	2,5		
02.07.2024	Vermietung	Test 5	2,5		
02.07.2024	Lohnersatzmittel	Buch 1	2,5		
02.07.2024	Drogerie	Buch 2	2,5		

Abbildung 8: Endprodukt der Software

8.2 Learnings

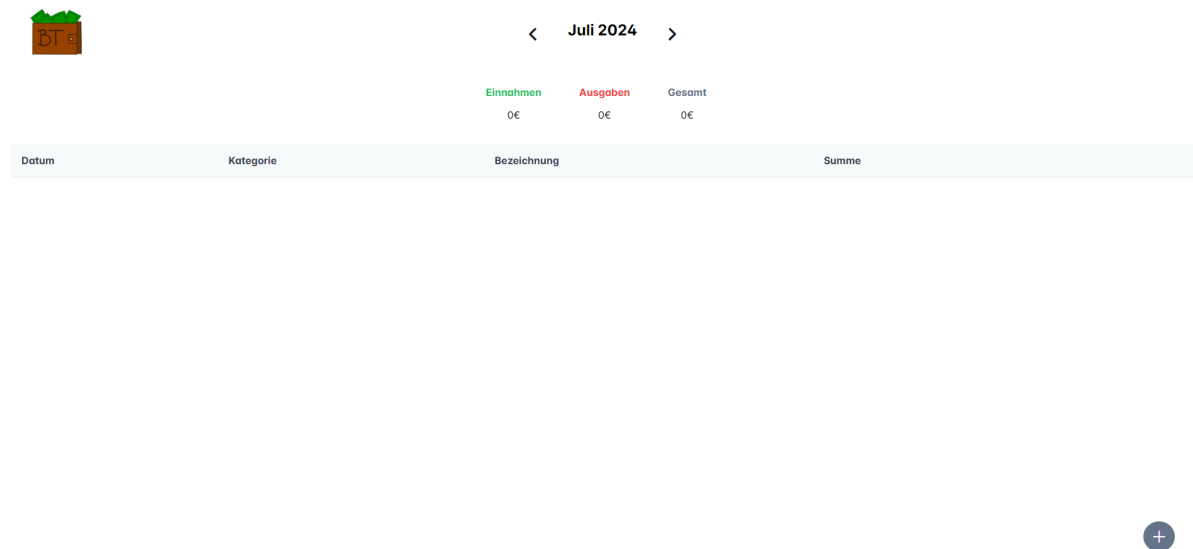
Im Folgenden wollen wir übersichtlich festhalten, was in diesem Projekt gut funktioniert hat und wo es für das nächste Projekt Verbesserungsbedarf gibt.

- Die gewählte Arbeitsweise mit Scrumban funktionierte einwandfrei und dank unseres Gantt-Charts haben wir es über die gesamte Projektzeit hinweg geschafft, in unserem Zeitplan zu bleiben und so in keine extremen zeitlichen Schwierigkeiten zu geraten.
- Die Kommunikation über WhatsApp zwischen den Sprintterminen war eine sehr gute Unterstützung, um sich schnell und unkompliziert austauschen zu können.
- [Planning Poker](#) zur Aufwandsschätzung in unseren Sprintterminen haben wir leider regelmäßig vergessen zu nutzen
- Uns allen bereitete die Erstellung der UML Diagramme Schwierigkeiten - das war der Aufgabenteil, der am meisten Iterationszyklen benötigte.
- Für Themen wie Anwendungssicherheit blieb am Ende leider keine Zeit mehr übrig.
- Über die Form der Commit-Messages haben wir uns leider keine Gedanken gemacht, sodass diese unterschiedlich ausführlich und präzise sind.
- Die Benennung der Branches nach dem, wie unter [Punkt 3.3](#) Versionierung beschrieben, zu Beginn festgelegten Schema wurde nicht immer eingehalten.
- Wir alle haben nach diesem Projekt einen besseren Überblick darüber, welche Schritte Software Engineering umfasst.
- Wir haben neue Frameworks und Tools kennengelernt.
- Architekturmuster und Entwurfsmuster sind weniger abstrakt.

9. Anhang

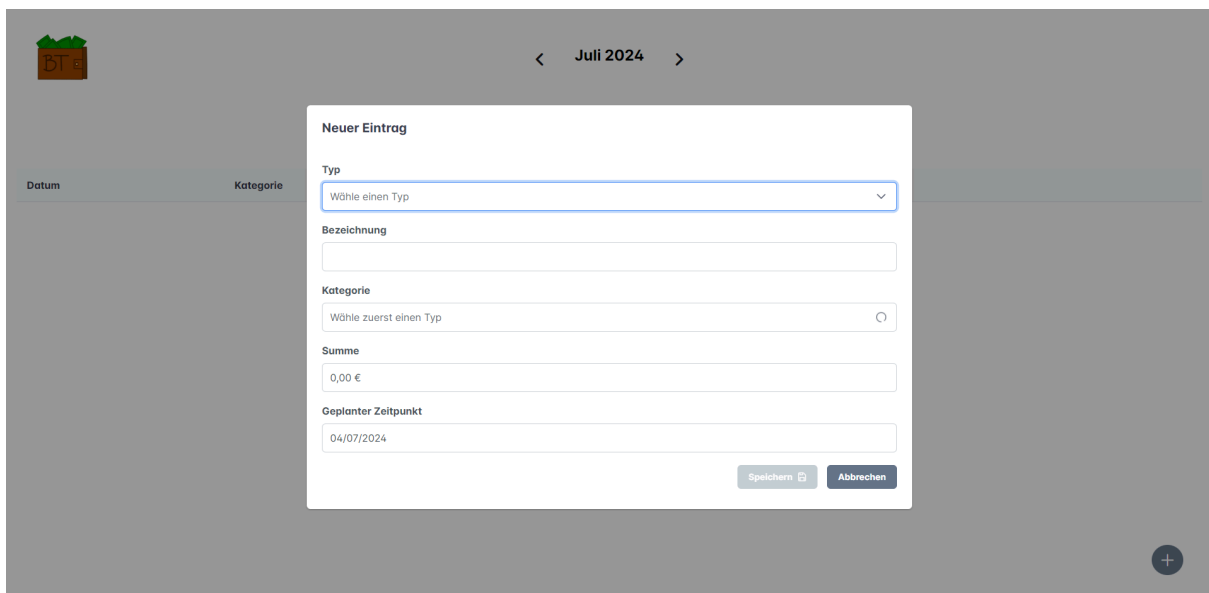
9.1 Screenshots der Anwendung

Startansicht ohne Einträge




The screenshot shows the application's start view. At the top left is a logo with a green plant and the text 'BT-8'. To its right is a navigation bar with '< Juli 2024 >'. Below this are three columns: 'Einnahmen' (green), 'Ausgaben' (red), and 'Gesamt' (grey), each with '0€' underneath. A table with the following headers is visible: 'Datum', 'Kategorie', 'Bezeichnung', and 'Summe'. On the right side, there is a circular button with a plus sign.

Erstellung eines neuen Eintrags



The screenshot shows the application with the 'Neuer Eintrag' (New Entry) form open. The form has the following fields: 'Typ' (a dropdown menu with 'Wähle einen Typ'), 'Bezeichnung' (a text input field), 'Kategorie' (a dropdown menu with 'Wähle zuerst einen Typ'), 'Summe' (a text input field with '0,00 €'), and 'Geplanter Zeitpunkt' (a text input field with '04/07/2024'). At the bottom right of the form are two buttons: 'Speichern' (Save) and 'Abbrechen' (Cancel). The background shows the same navigation bar and table headers as the previous screenshot, but they are dimmed.

Ansicht nach Erstellung mehrerer Einnahmen und Ausnahmen



< Juli 2024 >

Einnahmen

12.5€

Ausgaben


12.5€

Gesamt

0€

Datum	Kategorie	Bezeichnung	Summe		
04.07.2024	Gehalt	Test 1	2.5		
04.07.2024	Taschengeld	Test 1	2.5		
04.07.2024	Unterhalt	Test 3	2.5		
04.07.2024	Kapitalvermögen	Test 4	2.5		
04.07.2024	Vermietung	Test 5	2.5		
04.07.2024	Lebensmittel	Buch 1	2.5		
04.07.2024	Drogerie	Buch 2	2.5		
04.07.2024	Freizeit	Buch 3	2.5		

Ansicht nach Klicken auf Button “Einnahmen”



< Juli 2024 >

Einnahmen

12.5€

Ausgaben


12.5€

Gesamt

0€

Datum	Kategorie	Bezeichnung	Summe		
04.07.2024	Gehalt	Test 1	2.5		
04.07.2024	Taschengeld	Test 1	2.5		
04.07.2024	Unterhalt	Test 3	2.5		
04.07.2024	Kapitalvermögen	Test 4	2.5		
04.07.2024	Vermietung	Test 5	2.5		

Ansicht nach Klicken auf Button “Ausgaben”














< Juli 2024 >

Einnahmen
12.5€


Ausgaben
12.5€

Gesamt
0€

Datum	Kategorie	Bezeichnung	Summe		
04.07.2024	Lebensmittel	Buch 1	2.5		
04.07.2024	Drogerie	Buch 2	2.5		
04.07.2024	Freizeit	Buch 3	2.5		
04.07.2024	Miete	Buch 4	2.5		
04.07.2024	Bildung	Buch 5	2.5		



Ansicht nach Klicken auf Button “Gesamt”




















< Juli 2024 >

Einnahmen
12.5€

Ausgaben
12.5€

Gesamt
0€

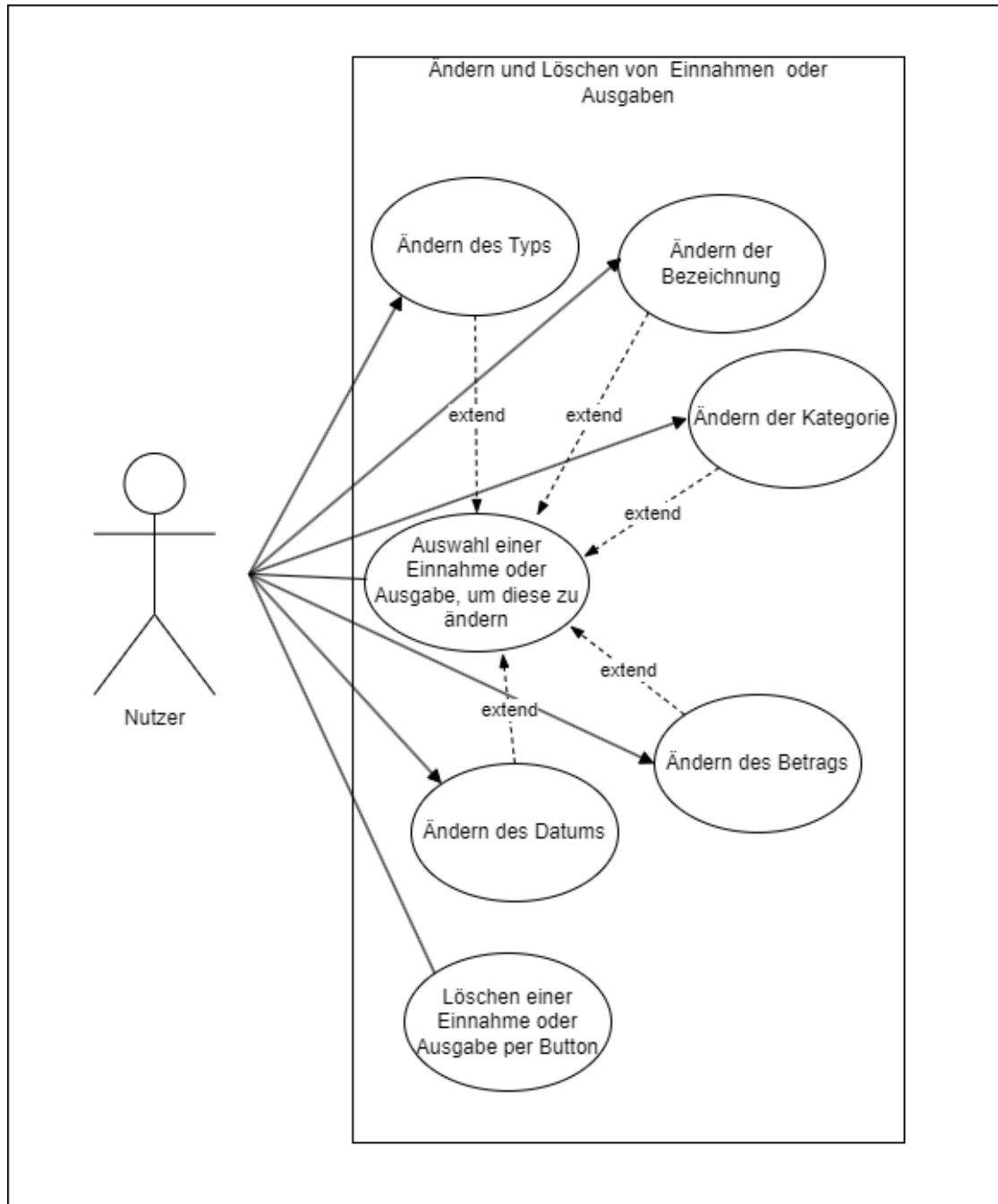
Datum	Kategorie	Bezeichnung	Summe		
04.07.2024	Gehalt	Test 1	2.5		
04.07.2024	Taschengeld	Test 1	2.5		
04.07.2024	Unterhalt	Test 3	2.5		
04.07.2024	Kapitalvermögen	Test 4	2.5		
04.07.2024	Vermietung	Test 5	2.5		
04.07.2024	Lebensmittel	Buch 1	2.5		
04.07.2024	Drogerie	Buch 2	2.5		
04.07.2024	Freizeit	Buch 3	2.5		

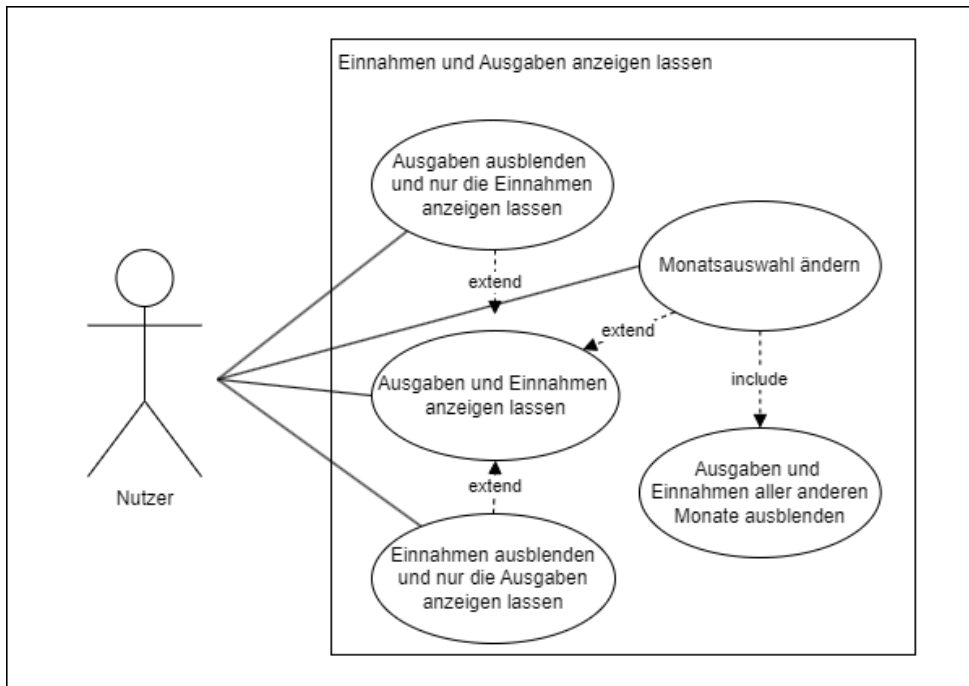


9.2 Diagramme

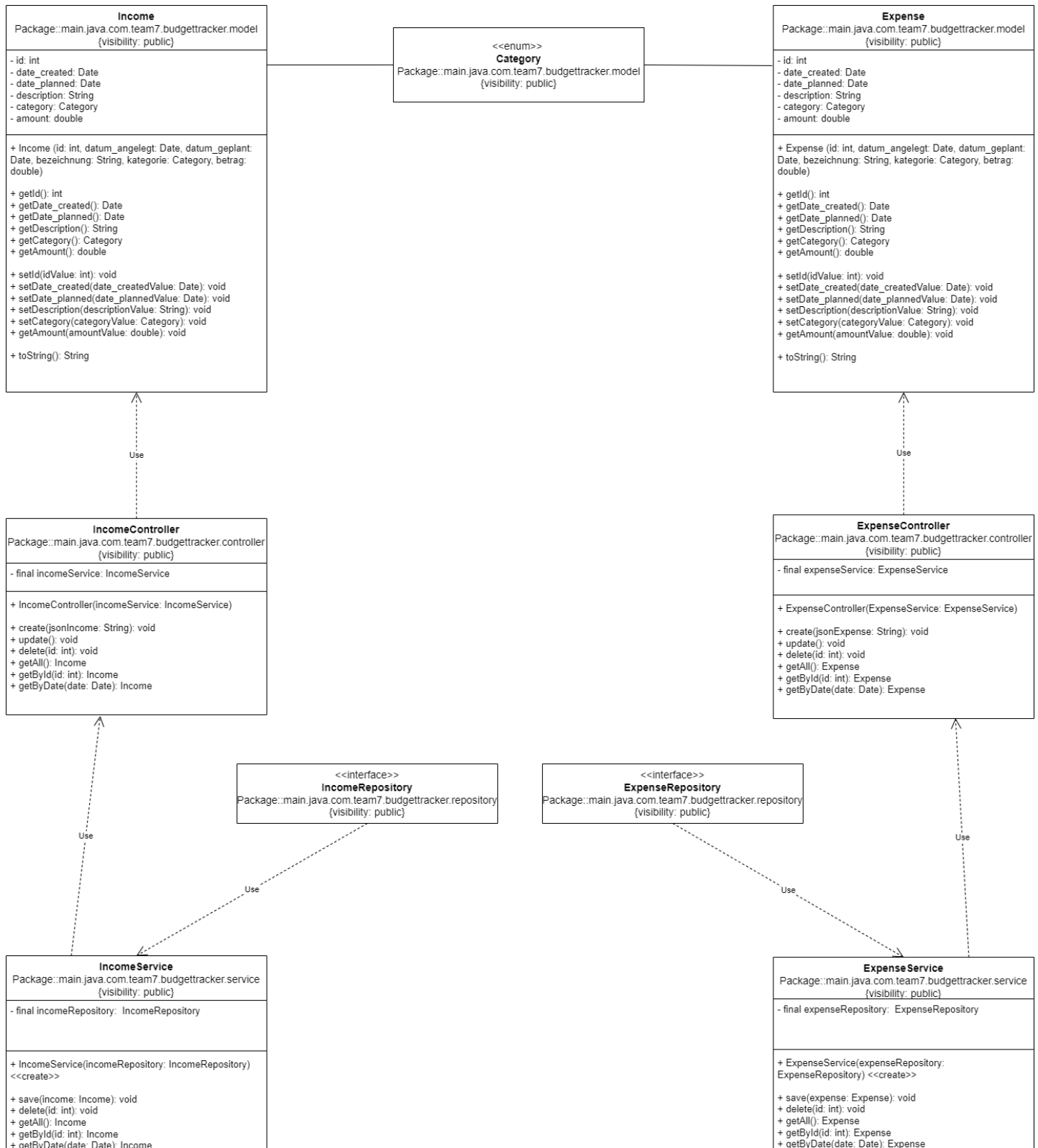
9.2.1 Use-Case Diagramm

Die abgebildeten Diagramme zeigen zwei weitere Fälle für die Benutzung unserer Website. Zusammen mit dem im [Punkt 4.4.1](#) vorgestellten Aufgabenbereich des Erstellens neuer Einträge sind damit alle Szenarien abgedeckt.

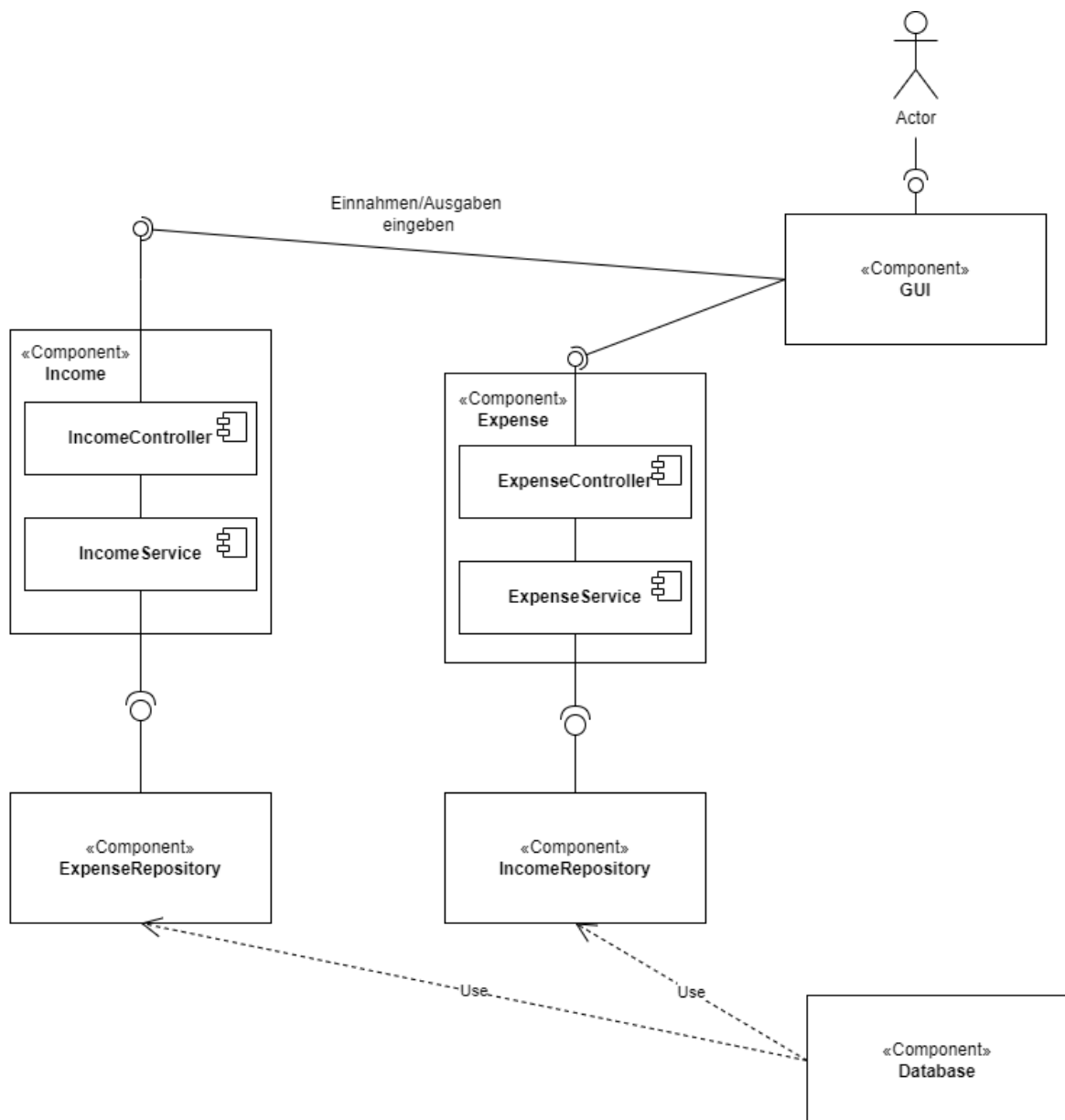




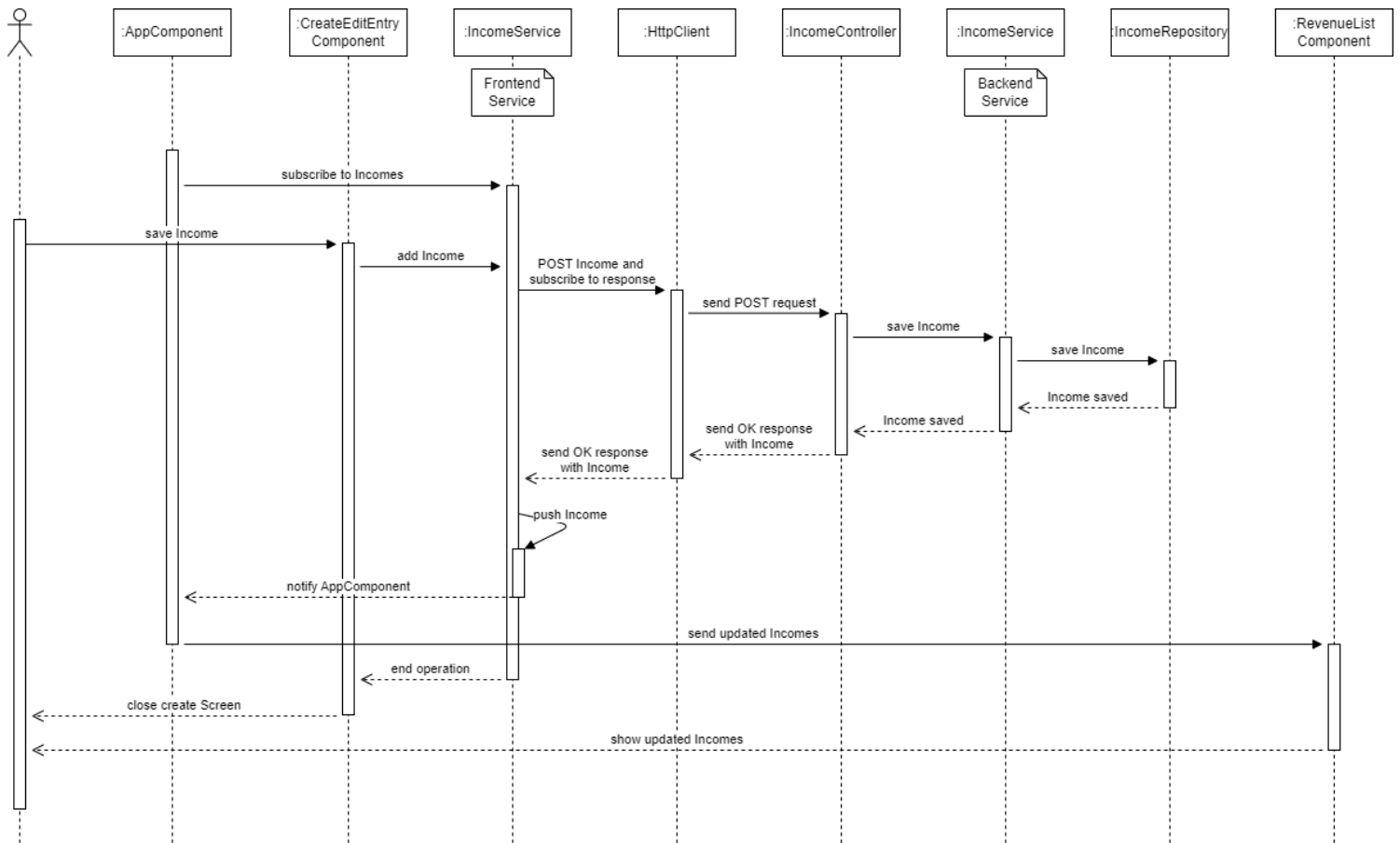
9.2.2 Klassendiagramm



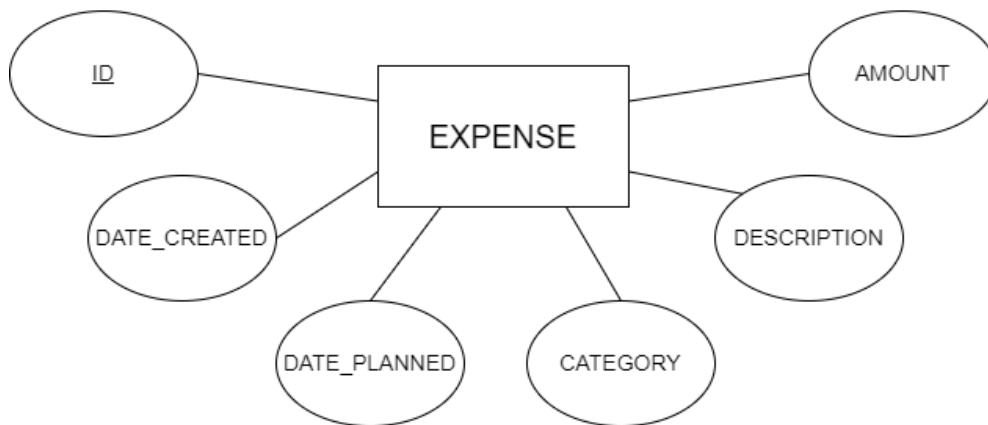
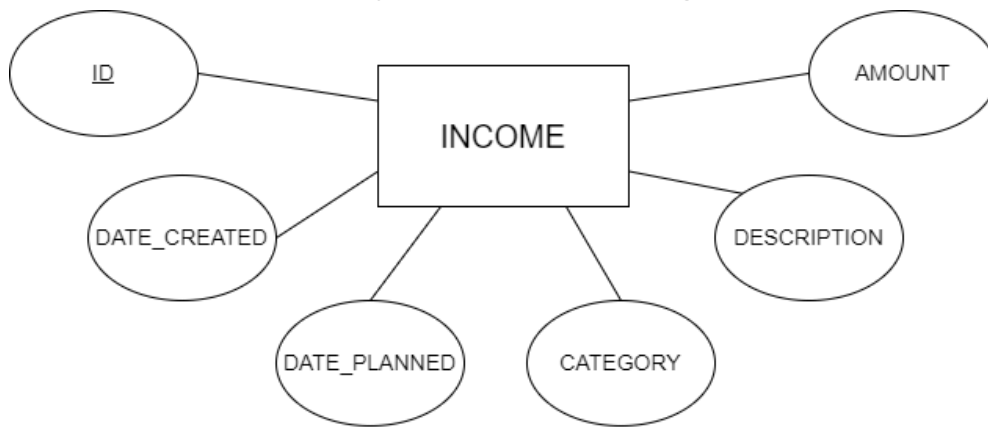
9.2.3 Komponentendiagramm



9.2.4 Sequenzdiagramm



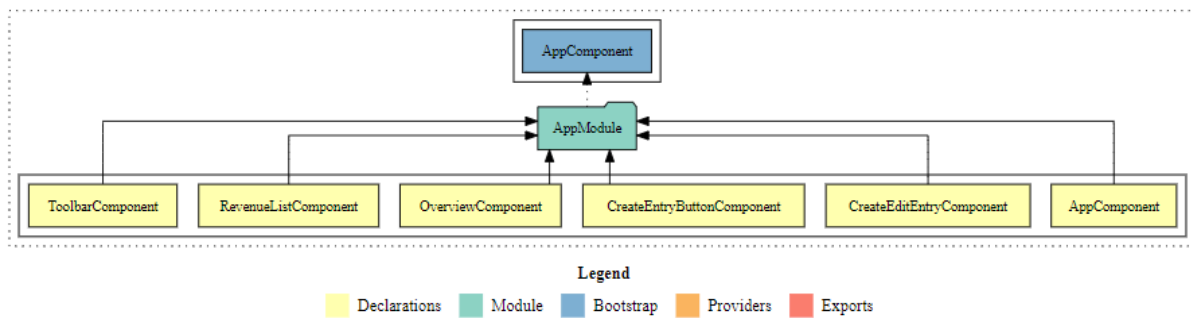
9.2.5 Datenbank: Entity-Relationship Diagramm



9.2.6 Datenbank: Relationenmodell

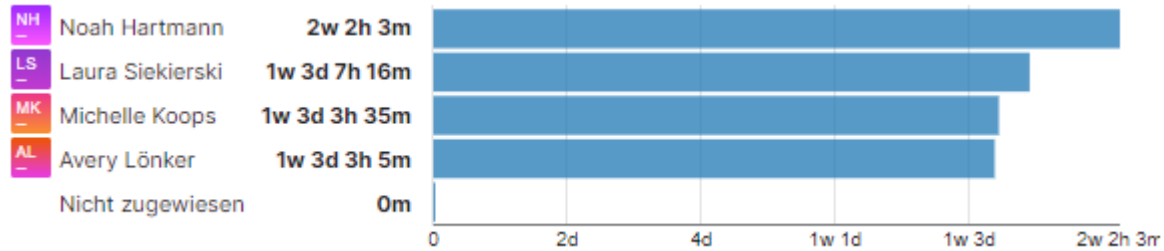
Income		
id	INT	PK
date_created	DATE	
date_planned	DATE	
description	VARCHAR(255)	
category	VARCHAR(255)	
amount	DECIMAL(10, 2)	

9.2.7 Strukturdiagramm Frontend



Die obige Abbildung zeigt den Aufbau unseres Frontends. Bei den gelben Rechtecken handelt es sich um unsere Komponenten, welche alle über das AppModule in der App Komponente zusammenlaufen. Das Diagramm wurde mit Compodoc erzeugt.

9.3 Zeiterfassung



Datum	Person	Von	Bis	Pause	Was	Beschreibung	Dauer	Stunden
dd.mm.yyyy	Dropdown	hh:mm	hh:mm	hh:mm	Dropdown	Kommentar	hh:mm	
13.04.2024	Noah	12:00	13:00		Projektmanagement	Welches Vorgehensmodell? Warum? Wie sollten wir das explizit umsetzen, ...	01:00	1
13.04.2024	Noah	14:30	16:00		Projektmanagement	Weitere Recherche, Aufwandsschätzung --> Story Points, Planning Poker To	01:30	1,5
17.04.2024	Team	13:00	13:45		Projektmanagement	Besprechung von Kanban vs. Scrum, Entscheidung für Scrumban, generelle	00:45	3
16.04.2024	Michelle	17:30	19:00		Anforderungsanalyse	Überlegungen Anforderungen	01:30	1,5
23.04.2024	Team	10:00	11:00		Anforderungsanalyse	Besprechen der geschriebenen User Stories und des UML Case Diagramm	01:00	4
23.04.2024	Team	11:00	11:30		Projektmanagement	Nächsten Sprint besprechen	00:30	2
23.04.2024	Noah	11:30	11:45		Projektmanagement	Protokoll schreiben	00:15	0,25
23.04.2024	Noah	11:45	12:30		Projektmanagement	Dokumente und Tickets für Architekturmuster, Entwurfsmuster und Entwurf	00:45	0,75
30.04.2024	Team	08:30	10:00		Recherche	Architekturmuster, Entwurfsmuster besprechen	01:30	6
30.04.2024	Noah	13:00	15:00		Entwurf	Übersicht für Klassendiagramm als Tabelle erstellen, Relationenmodell erste	02:00	2
01.05.2024	Team	09:00	10:00		Recherche	Entwurfsmuster besprechen, Todos für den nächsten Sprint besprechen	01:00	4
04.05.2024	Noah	11:45	15:30	00:30	Projektmanagement	Zeitplanung: Wie viele Sprints haben wir noch? Welche Aufgaben gibt es zu	03:15	3,25
04.05.2024	Noah	15:30	16:30		Entwurf	UI MockUp Figma	01:00	1
05.05.2024	Avery	10:30	12:00		Entwurf	Zusammentragen von Entwurfsmustern in erste Ideen für ein Klassendiagra	01:30	1,5
05.05.2024	Michelle	17:45	18:05		Projektmanagement	Protokoll schreiben	00:20	0,3333333333
05.05.2024	Michelle	18:10	20:30:00	00:20	Recherche	Recherche Komponentendiagramme, Relationenmodelle	02:00	2
08.05.2024	Michelle	18:00	18:30		Projektmanagement	Protokoll schreiben	00:30	0,5
12.05.2024	Michelle	17:30	18:30		Projektmanagement	Komponentendiagramm erstellen	01:00	1
06.05.2024	Team	17:45	19:30		Projektmanagement	Besprechung der Umsetzung des Klassendiagramms, Zeitplanung, nächster	01:45	7
14.05.2024	Team	08:30	10:00		Entwurf	entworfen Diagramme besprechen, Angular Einführung	01:30	6
18.05.2024	Avery	10:00	11:30		Sonstiges	technische Einrichtung	01:30	1,5
18.05.2024	Noah	10:00	11:30		Sonstiges	technische Einrichtung	01:30	1,5
15.05.2024	Michelle	08:30	12:00	01:00	Entwurf	Entwürfe für grafisches Design der Anwendung	02:30	2,5
18.05.2024	Michelle	20:40	20:50		Projektmanagement	Protokoll schreiben	00:10	0,1666666667
20.05.2024	Avery	14:45	15:00		Sonstiges	technische Einrichtung	00:15	0,25
20.05.2024	Laura	14:30	15:00		Sonstiges	technische Einrichtung	00:30	0,5
21.05.2024	Team	09:00	10:40		Projektmanagement	Besprechung des Klassen- und Komponentendiagramms und Aufgabenver	01:40	6,666666667
28.05.2024	Team	09:30	10:40		Projektmanagement	Problemgespräch, Klassendiagramm	01:10	3,5
29.05.2024	Team	16:30	20:25		Implementierung	Bugfixing und Aufgabenverteilung für den nächsten Sprint	03:55	11,75
04.06.2024	Team	09:00	11:40		Projektmanagement	Code Review für Error Handling und Backend-Frontend-Verbindungen, Aufg	02:40	10,66666667
11.06.2024	Team	08:00	09:55		Projektmanagement	Sprint, Besprechung der Pull Requests, Sequenzdiagram, Aufgabenverteilu	01:55	7,666666667
16.06.2024	Noah	10:00	12:00		Testing	Einlesen in Jasmine Framework	02:00	2
16.06.2024	Noah	12:00	18:00		Sonstiges	Einlesen und versuchen Backend auf AWS zu deployen	06:00	6
16.06.2024	Noah	18:00	20:00		Sonstiges	Deployment: Frontend auf Netlify, Backend und DB auf Render, alles verbint	02:00	2
18.06.2024	Team	08:30	10:00		Projektmanagement	Sprint Review und Aufgabenverteilung	01:30	6
25.06.2024	Team	08:30	10:50	00:10	Projektmanagement	Sprint Review und Aufgabenverteilung	02:10	8,666666667
02.07.2024	Team	08:00	10:00		Projektmanagement	Sprint Review	02:00	8
04.07.2024	Team	08:30	11:00		Projektmanagement	Fertigstellung des Berichts	02:30	10
								137,9166667

9.4 Gantt-Diagramm



9.5 Quellen

Alle Quellen wurden zuletzt am 01.07.2024 abgerufen.

- AlleAktien GmbH (2024): Relationenmodell definition | AlleAktien, AlleAktien, [online]
<https://www.alleaktien.com/lexikon/relationenmodell>.
- Angular (o. D.): [online] <https://docs.angular.lat/guide/singleton-services>.
- Angular and MVVM: Model-View-ViewModel Pattern (2023): Dopebase, [online]
<https://dopebase.com/angular-mvvm-pattern>.
- Atlassian (o. D.): Informationen zu Scrum und Tipps für den Einstieg, [online]
<https://www.atlassian.com/de/agile/scrum>.
- Atlassian (o. D.): Kanban, [online] <https://www.atlassian.com/de/agile/kanban>.
- Augsten, Stephan/Dirk Koller (2021): Services und DTOs in Spring-Boot-Anwendungen, in: *Dev-Insider*, 16.12.2021, [online]
<https://www.dev-insider.de/services-und-dtos-in-spring-boot-anwendungen-a-991082/>
- Client-Server-Architektur (o. D.): [online]
<https://www.elektronik-kompodium.de/sites/net/2101151.htm>.
- Compodoc - The missing documentation tool for your Angular application (o. D.): [online] <https://compodoc.app/guides/getting-started.html>.
- Crusoveanu, Loredana/Loredana Crusoveanu (2024): Inversion of Control and Dependency Injection with Spring | Baeldung, Baeldung, [online]
<https://www.baeldung.com/inversion-control-and-dependency-injection-in-spring>.
- Datenbanken - für Anfänger und Profis (2023): Entity-Relationship-Modell (ER-Modell / ERM) | Datenmodellierung Grundlagen, Datenbanken - für Anfänger und Profis, [online]
<https://www.datenbanken-verstehen.de/datenmodellierung/entity-relationship-modell/>.
- GeeksforGeeks (2024): Observer design pattern, GeeksforGeeks, [online]
<https://www.geeksforgeeks.org/observer-pattern-set-1-introduction/>.

- GeeksforGeeks (2024): Requirements Engineering Process in Software Engineering, GeeksforGeeks, [online]
<https://www.geeksforgeeks.org/software-engineering-requirements-engineering-process/>.
- Gupta, Lokesh/Lokesh Gupta (2024): Spring Boot 3 OpenAPI Docs with Springdoc and Swagger, HowToDoInJava, [online]
<https://howtodoinjava.com/spring-boot/springdoc-openapi-rest-documentation/>.
- IBM Integration Bus 10.0.0 (o. D.): [online]
<https://www.ibm.com/docs/de/integration-bus/10.0?topic=applications-publishsubscribe-overview>.
- Informatik, Gesellschaft Für (o. D.): Software-Architektur, Gesellschaft für Informatik e.V. (GI), [online] <https://gi.de/informatiklexikon/software-architektur>.
- IT-Talents (2024): Was sind Entwurfsmuster und wozu brauche ich so etwas?, IT-Talents.de, [online] <https://it-talents.de/it-ratgeber/entwurfsmuster-it/>.
- Kumar, Reetesh (2023): Spring Boot — Dependency Injection - Reetesh Kumar - Medium, in: *Medium*, 18.12.2023, [online]
<https://medium.com/@reetesh043/spring-boot-dependency-injection-137f85f84590#:~:text=Field%20injection%20involves%20injecting%20dependencies,directly%20set%20the%20field%20values>.
- Michaelstonis (2024): Model View ViewModel - .NET, Microsoft Learn, [online]
<https://learn.microsoft.com/de-de/dotnet/architecture/maui/mvvm>.
- Rauch, Gedeon (2023): Was ist ein Software-Entwurf?, in: *Dev-Insider*, 11.02.2023, [online]
<https://www.dev-insider.de/was-ist-ein-software-entwurf-a-ac9e018da98039bb8e4744beaa9a126f/>.
- Redaktion, Ionos (2020): Das Singleton Pattern – eine Klasse für sich, IONOS Digital Guide, [online]
<https://www.ionos.de/digitalguide/websites/web-entwicklung/was-ist-das-singleton-pattern>

[tern/#:~:text=Das%20Singleton%20Pattern%20geh%C3%B6rt%20zur.ein%20Objekt%20erstellt%20werden%20kann.](#)

- Rehkopf, Von Max (o. D.): User Storys | Beispiele und Vorlage | Atlassian, Atlassian, [online]
<https://www.atlassian.com/de/agile/project-management/user-stories#:~:text=Perspektive%20des%20Softwarebenutzers.,Eine%20User%20Story%20ist%20eine%20informelle%2C%20allgemeine%20Erkl%C3%A4rung%20eines%20Software,Kunden%20einen%20bestimmten%20Wert%20liefert.>
- Siriwardhana, Shalin (2022): Die einfache Anleitung für Komponentendiagramme, Creately Blog, [online]
<https://creately.com/blog/de/diagramme/komponentendiagramm-uml/#:~:text=UML%20Komponentendiagramme%20werden%20verwendet%2C%20um,eines%20Systems%20auf%20hoher%20Ebene.>
- Siriwardhana, Shalin (2021): Use Case Diagramm (UML Anwendungsfalldiagramm) mit Beispiele, Templates, Erstellung, Creately Blog, [online]
[https://creately.com/blog/de/diagramme/use-case-diagramm/.](https://creately.com/blog/de/diagramme/use-case-diagramm/)
- Tectrain. (2024, February 19). Was ist komponentenbasierte Softwarearchitektur? - tectrain. *tectrain*. <https://tectrain.ch/de/komponenten-software-architektur-leitfaden>
- Types of Testing (2024): Selenium, [online]
[https://www.selenium.dev/documentation/test_practices/testing_types/.](https://www.selenium.dev/documentation/test_practices/testing_types/)
- UML sequenzdiagramm (o. D.): Lucidchart, [online]
<https://www.lucidchart.com/pages/de/uml-sequenzdiagramme.>
- Was ist CI/CD? Konzepte und CI/CD Tools im Überblick (o. D.): [online]
<https://www.redhat.com/de/topics/devops/what-is-ci-cd#:~:text=CI%2FCD%2DTools%20k%C3%B6nnen%20ein,kontinuierliches%20Testen%20oder%20%C3%A4hnliche%20Funktionen.>

- Was ist eine dreischichtige Architektur? | IBM (o. D.): [online]
<https://www.ibm.com/de-de/topics/three-tier-architecture>.

Verwendete Frameworks und Libraries

- <https://spring.io/projects/spring-boot>
- <https://github.com/google/gson>
- <https://springdoc.org/>
- <https://logging.apache.org/log4j/2.x/>
- <https://junit.org/junit5/>
- <https://site.mockito.org/>
- <https://assertj.github.io/doc/>
- <https://angular.io/>
- <https://primeng.org/>
- <https://primeng.org/icons>
- <https://compodoc.app/index.html>
- <https://github.com/ngx-translate/core>
- <https://rxjs.dev/>
- <https://jasmine.github.io/>
- <https://karma-runner.github.io/6.4/index.html>

Sonstiges

- <https://www.jetbrains.com/de-de/youtrack/>
- <https://render.com/>
- <https://www.netlify.com/>
- <https://www.h2database.com/html/main.html>