

# FISCUS

## Personal Budgeting

---



**Eine Dokumentation zur Vordiplomarbeit**

von

**Cyrill Näf & Nicolas Stutz**

07. Februar 2022

Dipl. Techniker HF Informatik

Klasse 20I

HBU Höhere Berufsbildung Uster

# 1. Inhaltsverzeichnis

<b>1. Inhaltsverzeichnis</b>	<b>2</b>
<b>2. Management Summary</b>	<b>4</b>
2.1. Titel der VDA	4
2.2. Kurzbeschreibung des Projekts	4
2.3. Endprodukt	5
2.4. Zielerreichung	5
2.5. Fazit	5
2.6. Ausblick	6
<b>3. Einleitung</b>	<b>6</b>
3.1. Motivation	6
3.2. Aufgabenstellung	6
3.3. Informationen zum Dokument	8
<b>4. Pflichtenheft</b>	<b>8</b>
4.1. Sinn und Zweck	8
4.2. Abgrenzung	8
4.3. Voraussetzung	8
4.4. Erweiterungsmöglichkeiten	9
4.5. Technische Anforderungen	9
4.6. Allgemeine Übersicht der Funktionen	10
4.7. Detaillierte Funktionsbeschreibungen	10
4.8. Schnittstellen	11
4.8.1. Informatikschnittstellen	11
4.9. Leistungsanforderungen	11
4.10. Qualitätsmanagement	11
4.11. Adaptionen	12
4.12. Kosten	12
4.12.1. Personalaufwand	12
4.12.2. Material- und Maschinenkosten	12
<b>5. Projektplanung</b>	<b>12</b>
5.1. Kanban	12
5.2. Termine, Meilensteine	13
5.2.1. Termine	13
5.2.2. Meilensteine	13
5.3. Zeitplanung	14
5.4. Kritischer Pfad	14
<b>6. Systementwurf</b>	<b>15</b>
6.1. Evaluation Backend Entwicklungssoftware	15
6.1.1. Vorgehen	15
6.1.2. Erste Einschränkung	15
6.1.3. Vergleich der engeren Wahl	17
6.1.4. Entscheidung und Begründung	18
6.2. Evaluation Frontend Entwicklungssoftware	18
6.2.1. JavaScript	19
6.2.2. Framework	19
6.2.3. Entscheidung	20
6.3. Software-Architektur	20
6.3.1. System Context	21

6.3.2.	Container Diagram	22
6.3.3.	Component Diagram	23
6.3.4.	Class Diagram	24
<b>7.</b>	<b>Realisierung</b>	<b>26</b>
7.1.	REST API-Schnittstellendefinition	26
7.1.1.	Aufbau	27
7.1.2.	Schemas	27
7.1.3.	Responses	28
7.1.4.	Paths	28
7.2.	Realisierung Backend API	30
7.2.1.	Basis Setup	30
7.2.2.	MVC Pattern	31
7.2.3.	Budgets	31
7.2.4.	Accounts	35
7.2.5.	Tests	37
7.2.6.	Jobs	37
7.2.7.	Labels	39
7.2.8.	Transactions	39
7.3.	Realisierung Frontend	40
7.3.1.	Basis Setup	40
7.3.2.	UI Pattern & Architektur	41
7.3.3.	Beispiel Komponente/Ansicht	43
7.3.4.	Beispiel Store	44
<b>8.</b>	<b>Testing</b>	<b>44</b>
8.1.	Backend Testing	44
8.1.1.	Testdefinition	44
8.1.2.	Typische Testszenarien	45
8.1.3.	Vorbereitung	46
8.1.4.	Relevante Software	46
8.2.	Frontend Testing	47
8.2.1.	Testdefinition	47
8.2.2.	Testszenarien	47
8.2.3.	Vorbereitung	47
8.2.4.	Relevante Software	47
8.3.	Testprotokoll	48
8.3.1.	Unit Testing	48
8.3.2.	Account Anzeigen	49
8.3.3.	Account Erstellen	50
8.3.4.	Account Bearbeiten	51
8.3.5.	Account Löschen	52
<b>9.</b>	<b>Benutzeranleitung</b>	<b>53</b>
9.1.	Installation Backend	53
9.1.1.	Softwarevoraussetzung	53
9.1.2.	Ordnerstruktur	53
9.1.3.	Starten der Entwicklungsumgebung	54
9.1.4.	VM Zugriff	55
9.2.	Installation Frontend	56
9.2.1.	Softwarevoraussetzung	56
9.2.2.	Ordnerstruktur	56
9.2.3.	Frontend Starten	57
9.2.4.	Zugriff	57

9.3. Gebrauchsanweisung des Fiscus Web-UI	58
<b>10. Fazit</b>	<b>61</b>
10.1. Offene Punkte	61
10.1.1. Transaktion Transfer	61
10.1.2. Frontend MVP	61
10.2. Vergleich Zielerfüllung Pflichtenheft	61
10.3. Lernaspekt	62
10.3.1. Nicolas Stutz	62
10.3.2. Cyrill Näf	63
10.4. Ausblick	63
<b>11. Anhang</b>	<b>63</b>
<b>12. Glossar und Verzeichnisse</b>	<b>64</b>
12.1. Glossar	64
12.2. Literaturverzeichnis	65
12.3. Abbildungsverzeichnis	66
12.4. Tabellenverzeichnis	67

## 2. Management Summary

### 2.1. Titel der VDA

FISCUS Personal Budgeting

### 2.2. Kurzbeschreibung des Projekts

Mit der Budgetierungslösung FISCUS soll es für den Nutzer möglich sein die persönliche finanzielle Lage abbilden zu können.

Die Besonderheit gegenüber den diversen anderen Budgetierungstools bildet die Herangehensweise «jeder Franken hat eine Aufgabe». Dies bedeutet, dass optimalerweise das Budget des Nutzers bis auf den letzten Franken aufgeteilt wird in Aufgaben und Sparziele diverser Arten.

## 2.3. Endprodukt

Entstanden ist eine umfangreiche Backend Softwarelösung mit einem Frontend, welches sich sehen lässt.

Date	Charge Account	Favour Account	Charge Job	Value	Action
1986-10-05	Wolff-Ziemann	Gutkowski and Sons	itaque	-90.86	
1991-10-28	Botsford, Mueller and Johnston	Wolff-Ziemann	voluptatem	-137.45	
1993-09-21	Heathcote-Marks	Hartmann, Larson and Turcotte	repellat	-136.08	
1994-03-20	Borer and Sons	Luetgten, Gerlach and Toy	soluta	-92.49	
1997-04-19	Zulauf-Powlowski	Doyle-West	eligendi	157.19	
1998-07-04	Barrows, Bernier and Harris	Zulauf-Powlowski	repellat	-150.4	
2000-05-09	Borer and Sons	Schmitt, Okuneva and Mayert	soluta	115.34	
2004-05-09	Hagenes, Bednar and Bernhard	Ward, Walker and Lebsack	maxime	69.58	
2004-08-30	Ryan-O'Kon	Doyle-West	quasi	51.36	
2006-08-21	Carter, Keeling and Rempel	Klocko, Terry and Jenkins	minus	-131.1	
2013-09-04	Pfannerstill LLC	Zboncak Ltd	itaque	30.54	
2016-02-29	Bechtelar, Kuhn and Waters	Mills-Kovacek	repellat	12.13	

Abbildung 1: Front-End Transaktionsübersicht

## 2.4. Zielerreichung

Die als Muss definierten Anforderungen gemäss Aufgabenstellung wurden erfüllt. Es können mittels REST-API-Abfragen zu Accounts, Jobs und deren Labels erstellt und abgeändert werden. Weiterhin können mit Transaktion zwischen den Accounts Geldbeträge verschoben und Jobs zugewiesen werden.

Weiterhin konnte eine umfangreiche Test-Suite mit über 190 Tests erstellt werden.

Zudem steht eine minimale Webanwendung zur Verfügung mit welchem Nutzer ihre Daten bearbeiten können.

Die Authentifizierung und Autorisierung von Nutzer sowie Multiuser konnte aus zeitlichen Gründen nicht umgesetzt werden.

Folglich konnten Gemeinschaftskonten aus dem selbigen Grund ebenfalls nicht umgesetzt werden.

## 2.5. Fazit

Durch den Einsatz des Teams in der Entwicklung der Softwareteilen konnten Erfahrungen in der Erstellung und Handhabung von RESTful-API mit Hilfe des Laravel Frameworks gesammelt werden. Ebenfalls konnten Erfahrungen in der Entwicklung in der Verwendung von Containern gesammelt werden.

Obwohl die Implementation von Nutzerkonten und Gemeinschaftskonten nicht umgesetzt werden konnten, sind wir mit den erreichten Leistungen und dem in den insgesamt 400 Lernstunden erstellten Produkt sehr zufrieden.

## 2.6. Ausblick

Die Entwicklung von FISCUS ist mit dem Abschluss der VDA noch lange nicht beendet.

Es war von Beginn an der Wunsch beider Teammitgliedern ein Produkt zu erstellen, welches nach der VDA mit den gewünschten Features ergänzt und anschliessend veröffentlicht wird.

So wird das Projekt FISCUS unter der bereits bestehender GitLab Repository Gruppe FISCUS als Open-Source Projekt weiterhin zur Verfügung stehen.

## 3. Einleitung

### 3.1. Motivation

Vor einem Jahr suchte ich Cyrill Näf nach einer Budgetierung Lösung um meine Privaten Finanzen zu Verwalten. Dabei ist mir Aufgefallen, dass es unzählige Budgetierung-Applikationen auf dem Markt gibt. Bei etwas genauerer Betrachtung wurde mir klar, dass nur eine von ihnen ein klares Konzept mit "Erfolgsrezept" bietet. Leider hat die bereits bestehende Lösung ein paar Haken. Unter anderem ist sie kostenpflichtig, unterstützt keine Gemeinschaft-Konten (Partner / Familien Konten) und wird von einer amerikanischen Firma betrieben und entwickelt.

### 3.2. Aufgabenstellung

Vordiplomarbeit 2020/21

#### Personal Budgeting

##### Beschreibung

Mit der Budgetierungslösung Fiscus (Vorläufiger Projektname) soll es in erster Linie möglich sein die persönliche finanzielle Lage abbilden und zu können.

Die Besonderheit gegenüber den diversen anderen Budgetierungstools bildet die Herangehensweise «jeder Franken hat eine Aufgabe».

In anderen Worten soll das gegebene Budget aufgeteilt werden in Sparziele diverser Arten, kontinuierlichen Kosten, sowie Aufgaben (beispielsweise ein neues Auto, Hochzeit usw.) welche in vorgegebener Zeit erreicht werden sollen.

Das Budget wird durch jegliche realen Geldquellen (zum Beispiel Sparkonten, Lohnkonten oder Portemonnaie) gebildet und wird bei Erhalt den verschiedenen Aufgaben zugeteilt.

Durch das Definieren von Sparzielen kann errechnet werden wie viel der Nutzer pro Monat für die jeweilige Aufgabe auf die Seite legen muss, um das Sparziel im gesteckten Zeitrahmen erreichen zu können.

Durch die zeitlichen Gegebenheiten der VDA steht das Backend im Vordergrund.

Die optionale Entwicklung des Frontend dient lediglich zur Veranschaulichung und ist als MVP «minimal viable product» zu sehen.

## Aufgabenstellung

Folgende Möglichkeiten müssen verfügbar sein als REST HTTP API:

1. Erstellen, bearbeiten, anzeigen und löschen eines Budgets
2. Erstellen, bearbeiten, anzeigen und löschen eines Budgets Accounts
3. Erstellen, bearbeiten, anzeigen und löschen von Aufgaben
4. Erstellen, bearbeiten, anzeigen und löschen von Aufgaben Labels
5. Erstellen, bearbeiten, anzeigen und löschen von Transaktionen
  - Negativ-Belastende Transaktionen
  - Positiv-Belastende Transaktionen
  - Interne Transaktionen (z.B. Umbuchungen)

Zudem sind folgende Möglichkeiten wünschenswert:

6. Erstellen, bearbeiten, anzeigen und löschen von Sparzielen pro Aufgabe
  - Regelmässiges Sparziel
  - Bis Datum Sparziel
  - Sparziel
7. Backend Unit Tests
8. Backend End to End Tests
9. Web Client
  - Erstellen, bearbeiten, anzeigen und löschen eines Budgets
  - Erstellen, bearbeiten, anzeigen und löschen eines Budgets Accounts
  - Erstellen, bearbeiten, anzeigen und löschen von Aufgaben
  - Erstellen, bearbeiten, anzeigen und löschen von Aufgaben Labels
  - Erstellen, bearbeiten, anzeigen und löschen von Transaktionen
    - Negativ-Belastende Transaktionen
    - Positiv-Belastende Transaktionen
    - Interne Transaktionen (z.B. Umbuchungen)
10. Authentifizierung & Autorisierung
11. Gemeinschaft Budgets
  - Mehrere User Authentifizierung & Autorisierung

Die Ausführungen in Ihrem Vorschlag sind integraler Bestandteil dieser Aufgabenstellung.

Gliedern Sie das Projekt in sinnvolle Teilaufgaben und erstellen Sie eine vollständige Dokumentation gemäss Bestimmungen zur VDA.

<b>Versand der Aufgabenstellung</b>	<b>06. Oktober 2021</b>
<b>Abgabe der Dokumentation</b>	<b>07. Februar 2022</b>

Freundliche Grüsse



Uwe Singer

### 3.3. Informationen zum Dokument

Tabelle 1: Persönliche Angaben des Projektteams

Klasse:	20I	20I
Name:	Näf	Stutz
Vorname:	Cyrill	Nicolas
Adresse Privat:	Brunnenstrasse 1 8610 Uster	Kugelgasse 3 8492 Wila
NATEL Privat:	076 824 83 84	076 570 33 26
E-Mail Privat:	* <a href="mailto:cyrill@naef.family">cyrill@naef.family</a>	* <a href="mailto:nicolasstutz@hotmail.com">nicolasstutz@hotmail.com</a>
Arbeitgeber:	SIX Group Services AG	Bruker Switzerland AG
Rufnummer Geschäft:	079 901 59 20	044 825 96 82
E-Mail Geschäft:	<a href="mailto:cyrill.naef@six-group.com">cyrill.naef@six-group.com</a>	<a href="mailto:nicolas.stutz@bruker.com">nicolas.stutz@bruker.com</a>

## 4. Pflichtenheft

### 4.1. Sinn und Zweck

Mit der Budgetierungslösung FISCUS soll es möglich sein die persönliche finanzielle Lage abbilden zu können.

Die Besonderheit gegenüber den diversen anderen Budgetierungstools bildet die Herangehensweise «jeder Franken hat eine Aufgabe». Dies bedeutet, dass optimalerweise das gesamte Budget des Nutzers bis auf den letzten Franken aufgeteilt wird in Aufgaben und Sparziele diverser Arten.

### 4.2. Abgrenzung

Die Dokumentation des Produktes (Quellcode, Codekommentare, Installationsanleitung, Fehlerbehebung usw.) wird nicht in deutscher Sprache verfasst, sondern wird ausschliesslich auf Englisch verfügbar sein.

Unterstützung für Rechts nach Links Sprachen und Inhalt wird ausgeschlossen.

Barrierefreiheit im Internet (a11y) wird ausgeschlossen. (Wikipedia, 2022)

### 4.3. Voraussetzung

Für die Umsetzung der Anwendung sind Grundkenntnisse in der Erstellung von Softwareprodukten allgemein vorausgesetzt. Weiterhin müssen Kenntnisse in mindestens einer Objektorientierten Programmiersprache vorhanden sein. Da es sich bei der Projektarbeit um eine Webanwendung handelt, müssen ebenfalls Grundkenntnisse über Server-Client Verbindungen sowie dem einhergehenden Aufbau von http-Verbindungen zum Zweck der Daten Be- und Verarbeitung vorhanden sein.



## 4.4. Erweiterungsmöglichkeiten

Durch die zeitliche Begrenzung der VDA liegt der Fokus auf der Entwicklung des Backends der Webanwendung.

Die Entwicklung soll so durchgeführt werden, dass die Daten über definierte API's durch ein Frontend abgerufen werden können. Dabei soll nach gängigem MVC Patern vorgegangen werden, um die Wahl des Frontends grundsätzlich offen zu lassen.

## 4.5. Technische Anforderungen

Der Evaluation der Entwicklungssoftware ist Teil des Projekts.

Anhand der folgenden Kriterien soll ein geeigneter Softwarestack evaluiert werden:

- Die verwendete Software soll die Kenntnisse des Projektteams erweitern oder vertiefen.
- Kostenfaktoren
  - o Anschaffung der Entwicklungssoftware
  - o Hosting entwickelten Daten auf Hosting Partnern
- Hosting Möglichkeiten
  - o Muss auf den meisten Webhosting Plattformen lauffähig sein.
  - o Webhosts in der Schweiz verfügbar. (Beispiel Hoststar, Hostpoint)
- Um die Erweiterbarkeit durch Dritte zu gewährleisten, soll die Entwicklungssoftware einer möglichst freien Lizenzart unterliegen. (Beispielsweise GNU GPL<sup>1</sup>)
- Langlebigkeit
  - o Ein langlebiges Produkt, welches nicht in absehbarer Zeit ersetzt wird.
  - o Die Entwickler Community, welche das Produkt nutzt, soll möglichst aktiv sein.
- Komplexität
  - o Die Entwicklungssoftware soll von hause aus Strukturen bieten, welche eine einfache Wartung und Erweiterung begünstigen.
  - o Geringe Anlernzeit
  - o Die Entwicklungsumgebung soll einfach aufsetzbar sein.

Da es sich um ein Open Source Projekt handelt, soll der Quellcode des Backends auf der Plattform GitLab öffentlich zugänglich gemacht werden. Neben dem Quellcode des Backends sollen ebenfalls die Dokumentation zur API sowie das Frontend öffentlich verfügbar gemacht werden. Diese Repositorien sollen unter einer Gruppe zusammengeführt und veröffentlicht werden.

Die Dokumentation bezüglich der VDA soll ebenfalls in dieser Gruppe vorhanden sein, jedoch nicht für die Öffentlichkeit frei zugänglich sein.

Die Schnittstellen sollen mit Unittest überprüft werden. Die Testprotokolle bilden das Proof of Concept und sollen zum Projektabschluss übergeben werden.

Zur einfachen Veranschaulichung der Rückgabewerten der im Backend definierten Schnittstellen kann optional ein Frontend erstellt werden. Dieses ist als MVP zu sehen und ist nicht als Endnutzer Produkt zu betrachten.

---

<sup>1</sup> <https://www.gnu.org/licenses/gpl-3.0.en.html>

Durch die API übergebenen Daten sollten von folgenden Webbrowsern fehlerfrei empfangen und durch das Frontend dargestellt werden.

- Microsoft Edge
- Firefox
- Google Chrome
- Opera

### 4.6. Allgemeine Übersicht der Funktionen

Mit der Budgetierungslösung soll der Nutzer zu jedem Zeitpunkt über das Wissen verfügen, wieviel Geld er für Sparziele (Bsp. Geplante Hochzeit, Steuern) und Aufgaben (Bsp. Miete, Stromrechnung) für den jeweiligen Monat noch zur Verfügung hat.

Der Nutzer definiert diverse reale Geldquellen, wie beispielsweise Sparkonten, Lohnkonten oder Portemonnaie. Diese werden einem Budget zugeteilt.

Die Summe, welche sich im jeweiligen Budget befindet, soll durch den Nutzer auf diverse Aufgaben und Sparziele verteilt werden. Es soll durch den Nutzer angestrebt werden, dass der verteilte Betrag der Summe des Budgets entspricht. Somit erhält jeder Franken des Budgets einen Auftrag.

Durch das Definieren von Sparzielen und Aufgaben wird errechnet wie viel der Nutzer pro Monat für die jeweilige Sparzielen auf die Seite legen muss, um das Sparziel im gesteckten Zeitrahmen erreichen zu können.

### 4.7. Detaillierte Funktionsbeschreibungen

Folgende Möglichkeiten müssen als REST HTTP API verfügbar sein:

1. Erstellen, bearbeiten, anzeigen und löschen eines Budgets
2. Erstellen, bearbeiten, anzeigen und löschen eines Budgets Accounts
3. Erstellen, bearbeiten, anzeigen und löschen von Aufgaben
4. Erstellen, bearbeiten, anzeigen und löschen von Aufgaben Labels
5. Erstellen, bearbeiten, anzeigen und löschen von Transaktionen
  - Negativ-Belastende Transaktionen
  - Positiv-Belastende Transaktionen
  - Interne Transaktionen (z.B. Umbuchungen)

Zudem sind folgende Möglichkeiten wünschenswert:

6. Web Client (Als MVP)
  - Erstellen, bearbeiten, anzeigen und löschen eines Budgets
  - Erstellen, bearbeiten, anzeigen und löschen eines Budgets Accounts
  - Erstellen, bearbeiten, anzeigen und löschen von Aufgaben
  - Erstellen, bearbeiten, anzeigen und löschen von Aufgaben Labels
  - Erstellen, bearbeiten, anzeigen und löschen von Transaktionen
    - Negativ-Belastende Transaktionen
    - Positiv-Belastende Transaktionen
    - Interne Transaktionen (z.B. Umbuchungen)

7. Erstellen, bearbeiten, anzeigen und löschen von Sparzielen pro Aufgabe
  - Regelmässiges Sparziel
  - Bis Datum Sparziel
  - Sparziel
8. Backend Unit Tests
9. Nutzer Authentifizierung & Autorisierung
10. Gemeinschaft Budgets
  - Mehrere Nutzer Authentifizierung & Autorisierung

## 4.8. Schnittstellen

### 4.8.1. Informatikschnittstellen

Das Backend stellt über die definierte API die Daten zur Verfügung. Die entsprechenden Endpunkte sollen im Verlauf des Projekts definiert werden.

Ein unabhängiges Frontend muss die Daten empfangen und darstellen können. Dabei sollen die Schnittstellen die Vorgaben nach REST API erfüllen, also RESTful sein.

## 4.9. Leistungsanforderungen

Folgende Ziele sollen im Verlauf der VDA umgesetzt werden:

- Zum Abgabetermin am 07.02.2022 soll eine vollständige Dokumentation über das Projekt übergeben werden.
- Nach der Erstellung der Projektplanung und vor Beginn der Entwicklungsarbeiten soll eine Entwicklungssoftware evaluiert werden, welche den definierten Anforderungen entspricht.
- Nach der Evaluation der Entwicklungssoftware und vor Beginn der Implementation soll die Softwarestruktur erstellt werden, welche den Anforderungen an die Kenntnisse, Kostenfaktoren, Hosting Möglichkeiten, Erweiterbarkeit, Langlebigkeit und Komplexität entspricht.
- Mit der Abgabe der Dokumentation am 07.02.2022 soll ein Test-Suite übergeben werden, welcher die API-Spezifikation fehlerfrei überprüft.

## 4.10. Qualitätsmanagement

Mit einem Test-Suite soll die Backend internen Funktionen und Abläufe mit Unit-Tests auf ihre funktionelle Korrektheit überprüft werden.

Weiterhin soll anhand der API-Spezifikation die Rückgabewerte überprüft werden. Dies umfasst neben der Korrektheit der gelieferten Daten ebenfalls die Überprüfung bei falschen oder ungültigen Angaben.

Mit den API spezifischen Tests sollen ebenfalls http-Status Überprüfungen durchgeführt werden.

## 4.11. Adaptionen

Kann das Kann-Ziel «GUI Entwicklung» nicht erreicht werden, soll nach erfolgtem Projektabschluss dieses noch erstellt werden. Eine grundsätzliche Anpassung des erstellten Backends soll dabei nicht nötig sein.

Die Entwicklung der API soll so durchgeführt werden, dass die Software effizient wartbar und erweiterbar ist um auch zukünftig die Software um Features erweitern zu können.

## 4.12. Kosten

### 4.12.1. Personalaufwand

*Tabelle 2: Übersicht Personalaufwand*

Name	Funktion	Aufwand
Cyrill Näf	Projektteilnehmer	200 Lernstunden (45min)
Nicolas Stutz	Projektteilnehmer	200 Lernstunden (45min)
Uwe Singer	Schulinterne Betreuung	VDA Gespräche 2 x 30min

### 4.12.2. Material- und Maschinenkosten

Die zu evaluierende Entwicklungssoftware soll, wenn möglich kostenfrei in der Anschaffung sowie der kommerziellen Nutzung sein. Für Software, welche allenfalls eine Effizienzsteigerung in der Entwicklung des Projekts von über 50% erbringt, kann ein einmaliger Betrag von bis zu 500CHF aufgewendet werden.

## 5. Projektplanung

### 5.1. Kanban

Kanban ist eine agile Arbeitsmethode. Kanban hat das Ziel, die Durchlaufzeit innerhalb eines Prozesses zu optimieren, indem Engpässe im Prozess vermieden werden. Das Ziel ist es einen möglichst konstanten Fluss von Aufgaben vom Backlog ins Done zu verschieben. Dabei wird ein Board, auch Kanban-Board, als Hilfsmittel verwendet.

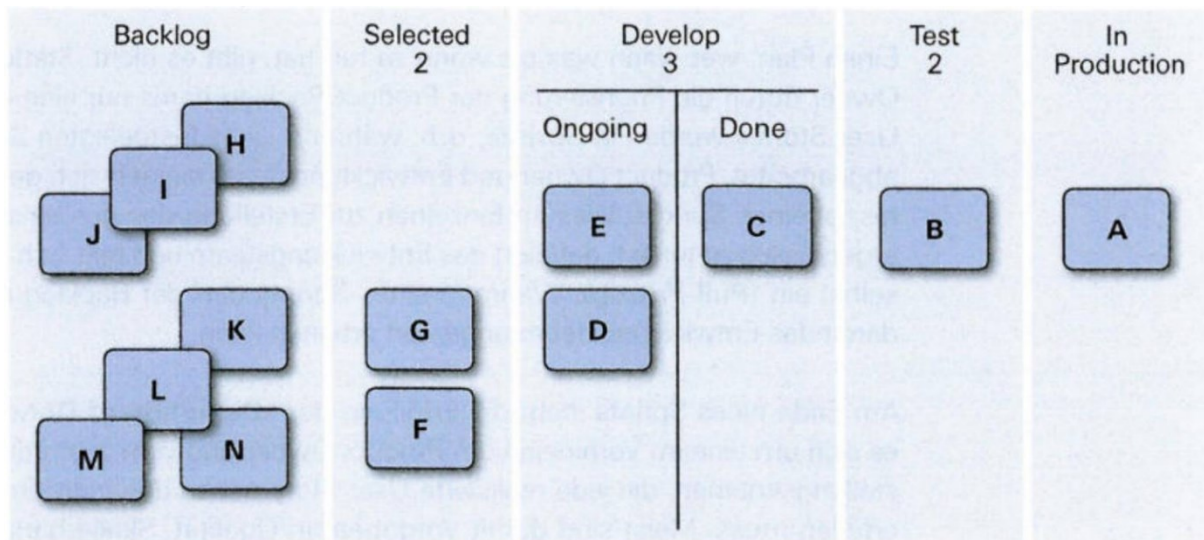


Abbildung 2: Beispiel Kanban Board

Die Zahlen im Board geben das "Work in Progress (WIP)" Limit an. Dieses Limit soll dabei helfen den Flow der Aufgaben aufrecht zu erhalten.

Kanban kann flexibel an das eigene Projekt und Arbeitsanforderungen angepasst werden.

In diesem Projekt haben wir Kanban in Kombination mit regelmässigen Austauschpunkten verwendet. Die Austauschpunkte dienten zur Diskussion der aktuellen Herausforderungen und dem Fortschritt. Das in Gitlab integrierte Kanban Board wurde zur Visualisierung verwendet. Als Work in Progress Limit wurden 3 Tasks pro Spalte verwendet.

## 5.2. Termine, Meilensteine

### 5.2.1. Termine

Tabelle 3: Projekt Termine

Nr.	Topic	Teilnehmer	Datum
1	VDA Gespräche 1	Uwe Singer, Cyrill Näf, Nicolas Stutz	26.11.2021
2	VDA Gespräche 2	Uwe Singer, Cyrill Näf, Nicolas Stutz	07.01.2021
3	VDA elektronische Abgabe der Dokumentation	Cyrill Näf, Nicolas Stutz	07.02.2022
4	VDA Präsentation	Uwe Singer, Cyrill Näf, Nicolas Stutz	12.03.2022

### 5.2.2. Meilensteine

Tabelle 4: Projekt Termine

Nr.	Titel	Lieferobjekt	Datum
1	Abschluss Projektplanungsphase	Pflichtenheft, Zeitplan	01.12.2021
2	Abschluss Konzeptphase	Evaluierte Entwicklungssoftware, definierte Softwarearchitektur, API-Spezifikation	27.12.2021
3	Abschluss Implementationsphase	Test-Suite	25.01.2022
4	Dokumentationsphase	Projektdokumentation	07.02.2022

5.3. Zeitplanung

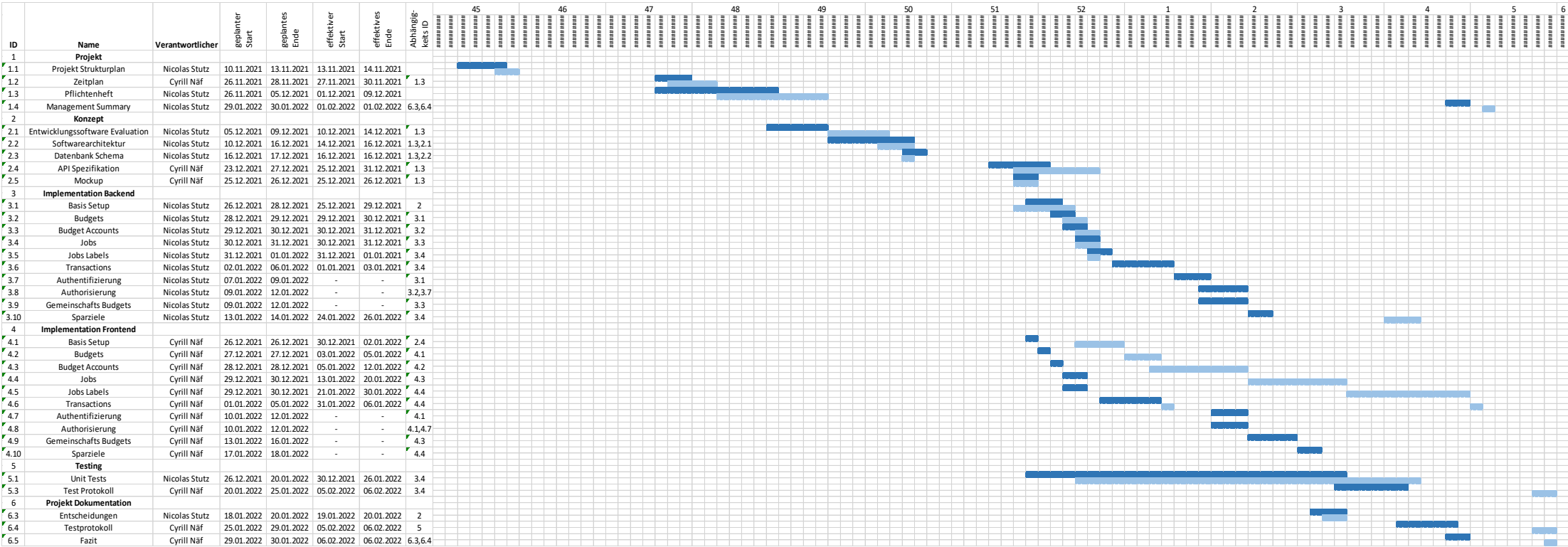


Abbildung 3: Projekt Zeitplan  
Der Zeitplan ist in besserer Auflösung im Anhang unter dem Namen 04\_Zeitplan.pdf abgelegt.

5.4. Kritischer Pfad

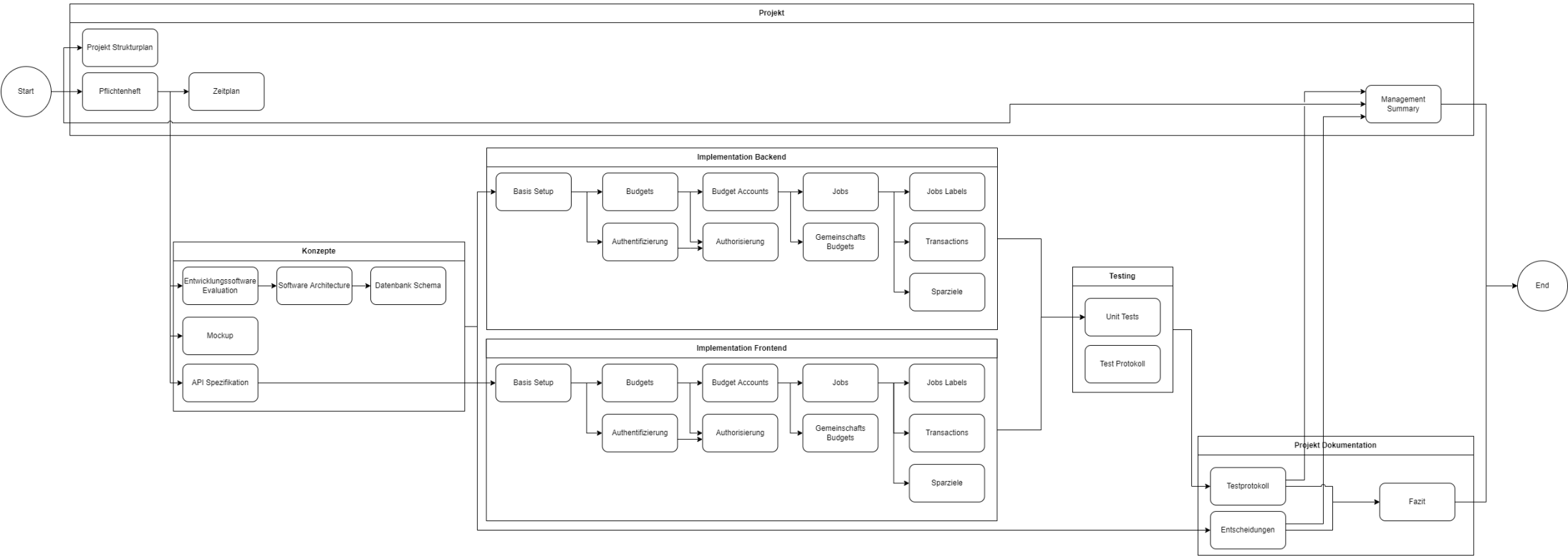


Abbildung 4: Kritischer Pfad  
Der kritische Pfad ist in besserer Auflösung im Anhang unter dem Namen 06\_Kritischer\_Pfad.png abgelegt.

## 6. Systementwurf

## 6.1. Evaluation Backend Entwicklungssoftware

Um für das Projekt die geeignetste Entwicklungssoftware einzusetzen, soll aus dem enormen Umfang vorhandener Backend Frameworks das passendste Produkt gewählt werden.

### 6.1.1. Vorgehen

Die grosse Anzahl von Backend Lösungen soll in einem ersten Schritt reduziert und auf 5 Produkte beschränkt werden.

Im zweiten Schritt werden die 5 Produkte anhand diverser Kriterien untereinander verglichen und gegenübergestellt.

Nach erfolgter Gegenüberstellung wird das zu verwendende Produkt anhand der Vergleiche und gesetzten Kriterien gewählt.

### 6.1.2. Erste Einschränkung

Die folgende Übersicht zeigt den Trend der Popularität von Server-Side Frameworks seit Januar 2010.

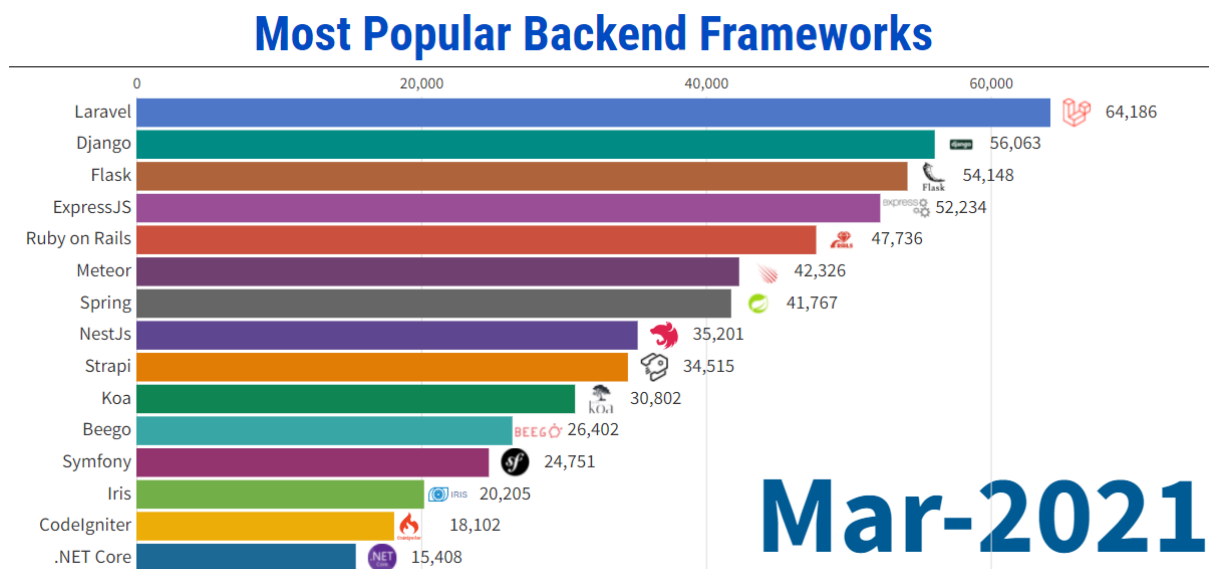


Abbildung 5: Populärste Backend Frameworks (statisticsanddata, 2021)

## FISCUS Personal Budgeting

Aus der Übersicht lässt sich bereits die Auswahl der vorhandenen und populären Frameworks für Backend-Lösungen ableiten.

Folgende Kriterien tragen massgebend zur Entscheidung welche Frameworks für die engere Auswahl in Betracht gezogen werden bei:

- Die zu verwendende Software soll die Kenntnisse des Projektteams erweitern oder vertiefen.
- Kostenfaktoren
  - o Anschaffung der Entwicklungssoftware
  - o Hosting entwickelten Daten auf Hosting Partnern
- Hosting Möglichkeiten
  - o Muss auf den meisten Webhosting Plattformen lauffähig sein.
  - o Webhosts in der Schweiz verfügbar. (Beispiel Hoststar, Hostpoint)
- Um die Erweiterbarkeit durch Dritte zu gewährleisten, soll die Entwicklungssoftware einer möglichst freien Lizenzart unterliegen. (Beispielsweise GNU GPL)
- Langlebigkeit
  - o Ein langlebiges Produkt, welches nicht in absehbarer Zeit ersetzt wird.
  - o Die Entwickler Community, welche das Produkt nutzt, soll möglichst aktiv sein.
- Komplexität
  - o Die Entwicklungssoftware soll von hause aus Strukturen bieten, welche eine einfache Wartung und Erweiterung begünstigen.
  - o Geringe Anlernzeit
  - o Die Entwicklungsumgebung soll einfach aufsetzbar sein.

Anhand dieser Kriterien sollen folgende Frameworks im nächsten Schritt untereinander verglichen werden:

- Laravel
- Django
- Flask
- Ruby on Rails
- Spring



## 6.1.3. Vergleich der engeren Wahl

Für die genannten Kriterien besonders positiven Aspekte der Frameworks sind folgend grün hinterlegt.

Tabelle 5: Vergleich Softwareevaluation

Framework	Laravel	Django	Flask	Ruby on Rails	Spring Boot
Programmiersprache	PHP	Python	Python	Ruby	Java
Popularität	Platz 1	Platz 2	Platz 3	Platz 5	Platz 7
Im Einsatz seit	2011	2005	2010	2004	2002
Kenntnisse des Teams	Nicolas: Keine Kenntnisse Cyrill: PHP gute Kenntnisse, Laravel keine Kenntnisse	Nicolas: Python Grundkenntnisse, Django keine Kenntnisse Cyrill: Python gute Kenntnisse, Django Grundkenntnisse	Nicolas: Python Grundkenntnisse, Flask keine Kenntnisse Cyrill: Python gute Kenntnisse, Flask Grundkenntnisse	Nicolas: Keine Kenntnisse Cyrill: Keine Kenntnisse	Nicolas: Java 3 Module an HFU, Spring keine Kenntnisse Cyrill: Java 3 Module an HFU, Spring keine Kenntnisse
Kosten Anschaffung	Programmiersprache und Framework kostenfrei	Programmiersprache und Framework kostenfrei	Programmiersprache und Framework kostenfrei	Programmiersprache und Framework kostenfrei	Programmiersprache und Framework kostenfrei
Kosten Projekt-Hosting	Hostpoint.ch: ab 12.90 CHF (Hostpoint, 2022)  Hoststar.ch: ab 6.90 CHF (Hoststar, 2022)	Hostpoint.ch: - (Hostpoint, 2022)  Hoststar.ch: ab 6.90 CHF (Hoststar, 2022)	Hostpoint.ch: - (Hostpoint, 2022)  Hoststar.ch: ab 6.90 CHF (Hoststar, 2022)	Hostpoint.ch: - (Hostpoint, 2022)  Hoststar.ch: - (Hoststar, 2022)	Hostpoint.ch: - (Hostpoint, 2022)  Hoststar.ch: - (Hoststar, 2022)
Umfang, Komplexität	Vollumfänglich, featurereich, geringe Komplexität	Vollumfänglich, featurereich, mittlere Komplexität	Wenig Features, geringe Komplexität	Vollumfänglich, featurereich, mittlere Komplexität	Vollumfänglich, featurereich, mittlere Komplexität

### 6.1.4. Entscheidung und Begründung

Aus der engeren Wahl sticht das Framework Laravel aus den folgenden Gründen hervor:

- Seit Ende 2016 befindet sich Laravel in den obersten Drei der populärsten Backend Frameworks. Seit Februar 2018 befindet es sich mit einigem Abstand an der Spitze.
  - Laravel wurde 2011 veröffentlicht. Somit ist es kein Newcomer auf dem Markt und konnte sich bereits bewähren. Es kann davon ausgegangen werden, dass es auch in den nächsten Jahren weiterhin aktiv unterstützt wird.
  - PHP sowie Laravel ist für private Zwecke, aber auch für kommerzielle Projekte kostenfrei.
  - PHP unterliegt der PHP-Lizenz, Laravel der MIT-Lizenz.
  - PHP-Dateien können bei den meisten Hosting-Anbietern am kostengünstigsten veröffentlicht werden.
  - Das Laravel-Framework ist vollumfänglich und bietet von Grund auf diverse Funktionen:
    - o Eloquent ORM
    - o Blade Template- Engine
    - o Datenbank-Seeding
    - o Unit-Testing
    - o Automatisches Laden von PHP-Klassen
    - o Reverse Routing
    - o Migrationsschemata für eine Versionskontrolle von Datenbankänderungen
- (Wikipedia Laravel, 2022)

Weitere Anmerkungen bezüglich der Wahl des Frameworks:

- Beide Projektmitgliedern haben keine Kenntnisse über Laravel. Durch die Erfahrung in anderen Objektorientierten Programmiersprachen sowie der geringen Komplexität des Frameworks ist mit einer geringen Einarbeitungszeit zu rechnen.
- Laravel bietet mit «Homestead» die Möglichkeit eine vom lokalen System abgekoppelten Entwicklungsumgebung zu nutzen. Hierzu wird Laravel sowie zusätzliche Software (Ubuntu, Git, MySQL, Node u.s.w. ) mittels Vagrant in eine VM geladen. Dies ermöglicht die systemunabhängige Entwicklung sowie ein einfacherer Austausch zwischen den Projektmitgliedern.

## 6.2. Evaluation Frontend Entwicklungssoftware

Die Evaluation der Frontend Technologie wurde etwas einfacher gestaltet. Dafür gibt es zwei Gründe. Das Frontend ist ein Kann-Ziel und es ist ein “Minimal Viable Product” (MVP) das nur zur Veranschaulichung des Backends dienen soll. Sie wurde an das JavaScript Modul der Höheren Fachschule Uster angelehnt welches während der Projektzeit durchgeführt wurde.

Als Entscheidungsoptionen gab es zwei Möglichkeiten, reines JavaScript an sich oder mit einem Framework kombinieren.

Der Hauptfokus des JavaScript Unterrichts lag auf reinem JavaScript ohne Bibliotheken oder Frameworks.

### 6.2.1. JavaScript

#### 6.2.1.1 Vorteile

Die Komplexität von reinem JavaScript ist gering. Es wird genau das verwendet und entwickelt, was benötigt wird. Durch die Durchführung des JavaScript Unterrichts sind viele Basis Elemente und Strukturen bereits bekannt.

Reines JavaScript bietet in modernen Browsern auch die Möglichkeit zur Erstellung von Komponenten. Für ältere Browser müsste dann allerdings ein Polyfill verwendet werden.

#### 6.2.1.2 Nachteile

Reines JavaScript muss auf den Browser Speicher zurückgreifen. Es gibt verschiedene Möglichkeiten wie den Lokal Speicher oder Session Speicher. Die Problematik hierbei ist das Ablegen in strukturierter Form, aber auch der geregelte Zugriff. Die strukturierte Form könnte mittels der Browser Datenbank gelöst werden, diese bringt allerdings eine sehr hohe Komplexität mit sich. Für die Zugriffe, wer wann was lesen oder ändern darf, müsste ein Protokoll- und Verhaltensregelwerk erstellt werden.

Ein weiterer Nachteil ist die Skalierung des JavaScripts. Die Skalierung bezieht sich auf die Anzahl der Komponenten und unterschiedlichen Ansichten. Grundsätzlich fehlt es an Rendering oder Templates Möglichkeiten, um verschiedene Ansichten vorzubereiten und bei Benutzer Interaktion auszutauschen.

### 6.2.2. Framework

#### 6.2.2.1 Vorteile

Die aktuellen JavaScript Frameworks wie z.B. Angular, React oder Vue bringen alle ein eigenes Template und Rendering Lösung für ihre Komponenten mit sich. Dabei wird in jedem Fall eine hierarchische Struktur verwendet.

Ein Zentraler Speicher wie bei Vue der Vuex wird ebenfalls mitgeliefert. Dabei ist das Protokoll und die Verhaltensregeln bereits definiert. Im Fall von Vuex ist dies Event basiert. Bei der Entwicklung steht ebenfalls ein Rollback und Inspektion Feature zur Verfügung. Dies erlaubt eine genaue Analyse welche Komponenten zu welchem Zeitpunkt den Speicher verändert haben. Selbst die Veränderung wird protokolliert.

#### 6.2.2.2 Nachteile

Da die Frameworks ihr eigenes Rendering und Templates Lösungen bringen werden auch eigene JavaScript Komponenten verwendet. Diese Komponenten sind leider nicht mit den standard JavaScript Komponenten verwendbar.

Die Frameworks bringen durch die eigenen Komponenten sowie das Rendering und Templates eine neue Komplexität in die Applikation.

### 6.2.3. Entscheidung

Anhand der oben aufgeführten Vor- und Nachteile haben wir uns für den Framework Ansatz entschieden. Massgebend zur Entscheidung beigetragen hat auch der Fakt das es sich um ein Kann-Ziel handelt. Hierbei haben wir den eigenen Lernfaktor in den Vordergrund gestellt.

Als JavaScript Framework haben wir Vue gewählt da es eine gute Mischung zwischen React und Angular bietet.

## 6.3. Software-Architektur

Die Software-Architektur wird nach den Grundlagen des C4<sup>2</sup> Models definiert. (c4model.com, 2022) Die bildliche Darstellung wird mittels der Software draw.io<sup>3</sup> vorgenommen. Dieses kann neben den spezifischen Formen des C4 Models auch diverse Standards (Bsp. UML Klassen Schemata) darstellen.

Mit dem C4 Model wird das Design auf vier Level verteilt, wobei jedes Level ein weiterer Detailgrad abstrahiert.

Context -> Übersicht über grundlegende Teile des Softwaresystem

Container -> Zeigt den Aufbau eines Systemteils.

Component -> Zeigt aus welchen Komponenten ein Container besteht.

Class -> Zeigt Implementationscode typischerweise in Form von Klassendiagrammen und Datenbankschemata.

In den folgenden Darstellungen sind ausschliesslich die Bestandteile der Backend API Applikation dargestellt, da diese als Muss-Ziel definiert ist und aus zeitlichen Gründen vor allem auf diese konzentriert wird.

---

<sup>2</sup> <https://c4model.com/>

<sup>3</sup> <https://www.diagrams.net/>

## 6.3.1. System Context

Der FISCUS Kontext umfasst zwei relevante Teile.

Ein Nutzer von FISCUS, mit eigenem Nutzerprofil, erstellt Budgets, Accounts und Aufgaben. Er führt Transaktionen durch und ruft Informationen über den momentanen Stand ab.

Die Webapplikation ermöglicht dem Benutzer seine Daten anzuzeigen, neue Budgets, Accounts und Aufgaben zu erstellen und Änderungen am eigenen Nutzerprofil zu tätigen. Die Webapplikation ruft Informationen über das Nutzerprofil und angeforderte Daten über Budgets, Accounts und Aufgaben ab.

### Legende

Software System

Externe Person

### System Context Diagram FISCUS Budgetierungslösung

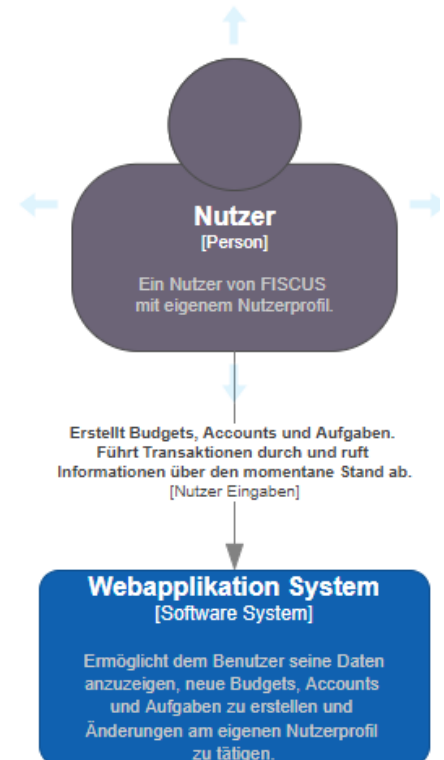


Abbildung 6: FISCUS System Context

## 6.3.2. Container Diagram

### 6.3.2.1 Webapplikation Container

Der Nutzer greift über seinen Webbrowser mittels dem Domainnamen (Bsp. fisco.ch) auf die Webapplikation zu. Er erhält die Webseite, über welche er Änderungen vornehmen und Informationen abrufen kann.

Die Webapplikation führt Abrufe gegenüber der API-Applikation durch um die gewünschten Daten zu laden oder anzupassen. Dies geschieht mittels gängigen REST-Schnittstellen.

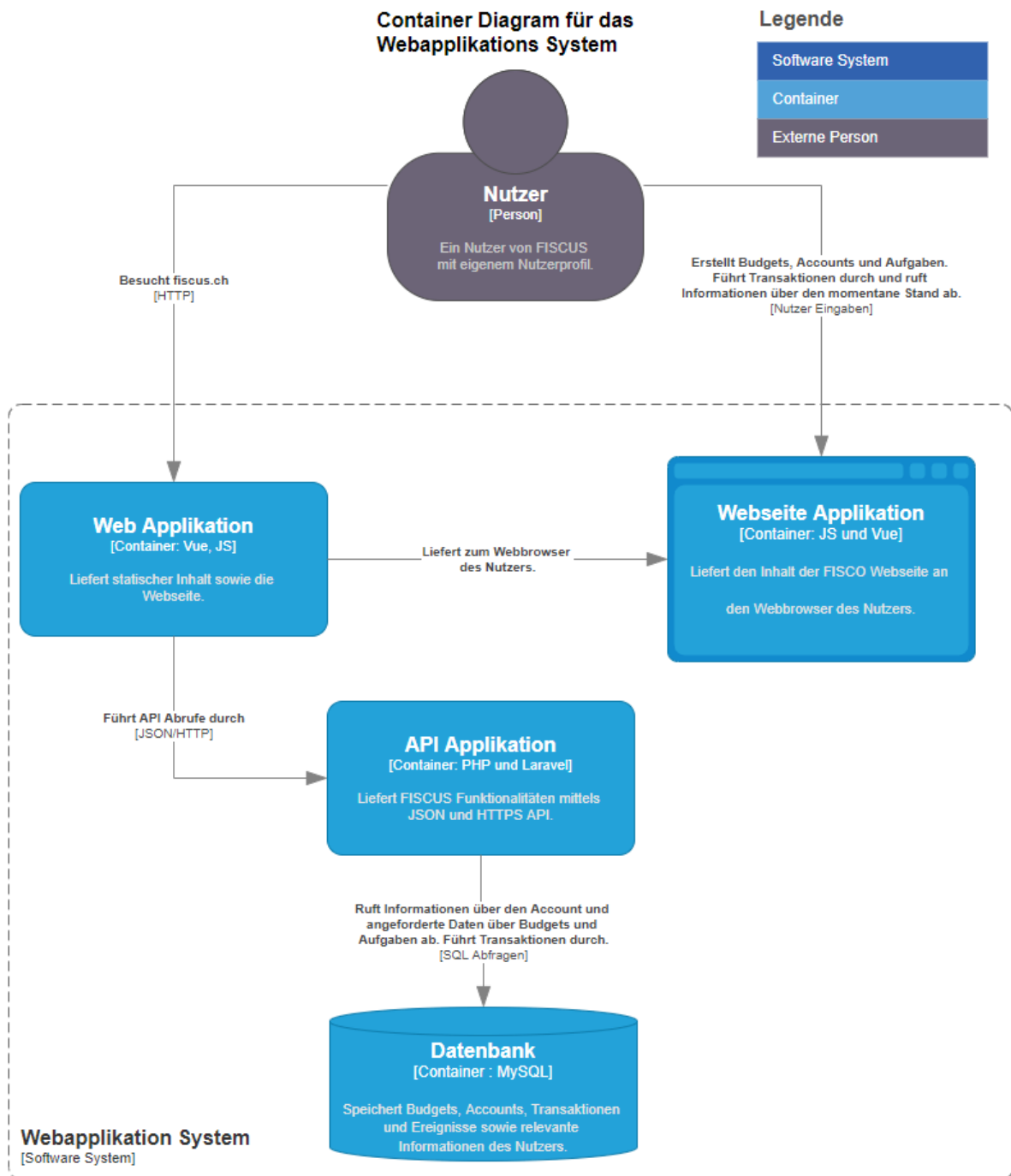


Abbildung 7: FISCUS Container Diagram

## 6.3.3. Component Diagram

### 6.3.3.1 API Applikation Container

Die Anfragen der Web Applikation werden den entsprechenden Controllern zugewiesen.

Die Accounts, Jobs und Transaktionen Controller sind dem Budget Controller zugeteilt.

Der Nutzer Logins und Token Erstellungen werden über den Nutzer Controller durchgeführt. Dieser verwendet Sicherheitsbestandteile, welche die Autorisierung und Authentifizierung sicherstellen, hier als Sicherheits Component dargestellt.

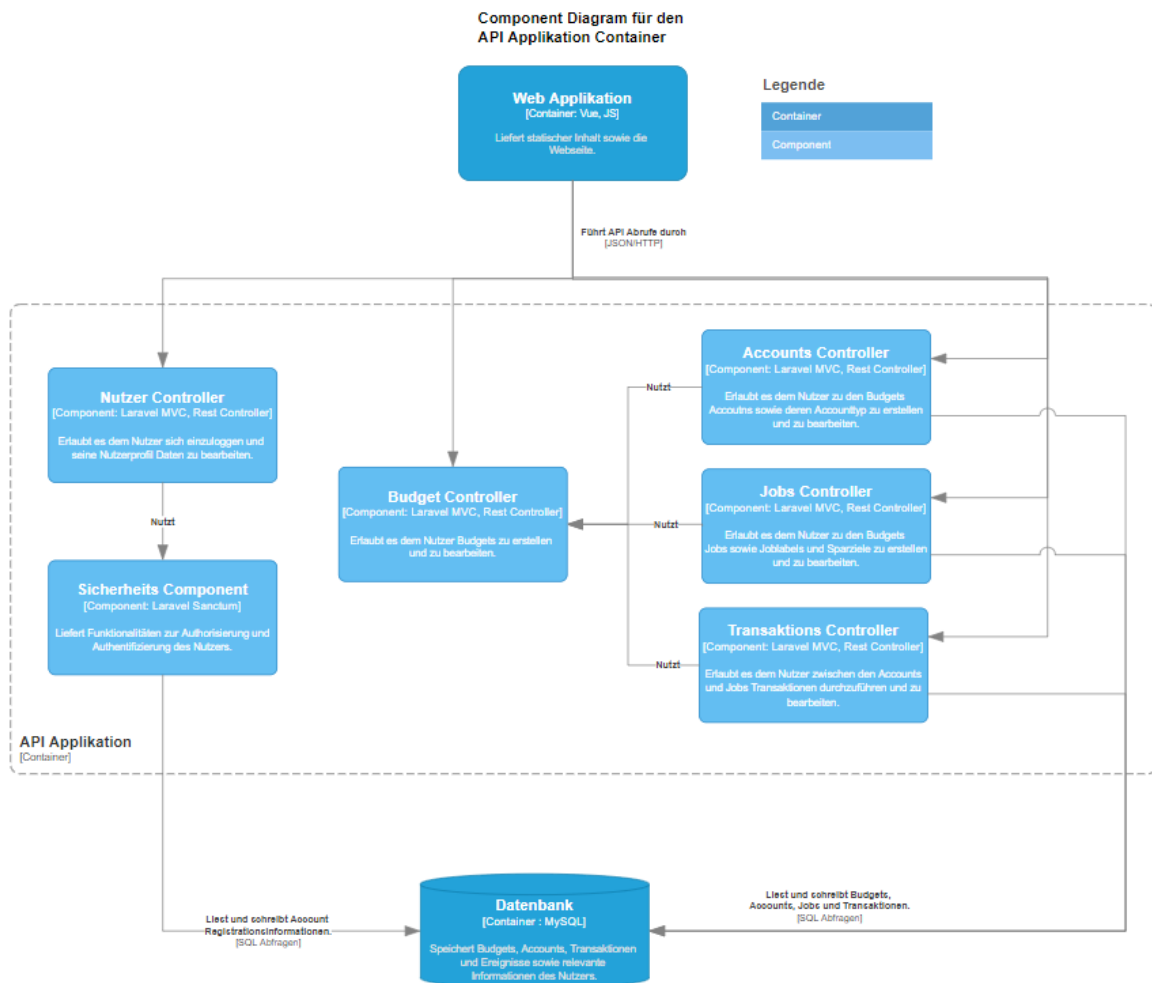


Abbildung 8: FISCUS API Applikation Container Diagram

### 6.3.3.2 Datenbank Container

Die Nutzeraccount Datenbank ist aus Sicherheitsgründen systematisch von der Nutzerdaten Datenbank getrennt. Mittels Accesstokens können Nutzer sich autorisieren und ihrer Autorisierungsstufen entsprechende Änderungen an den Nutzerdaten durchführen.

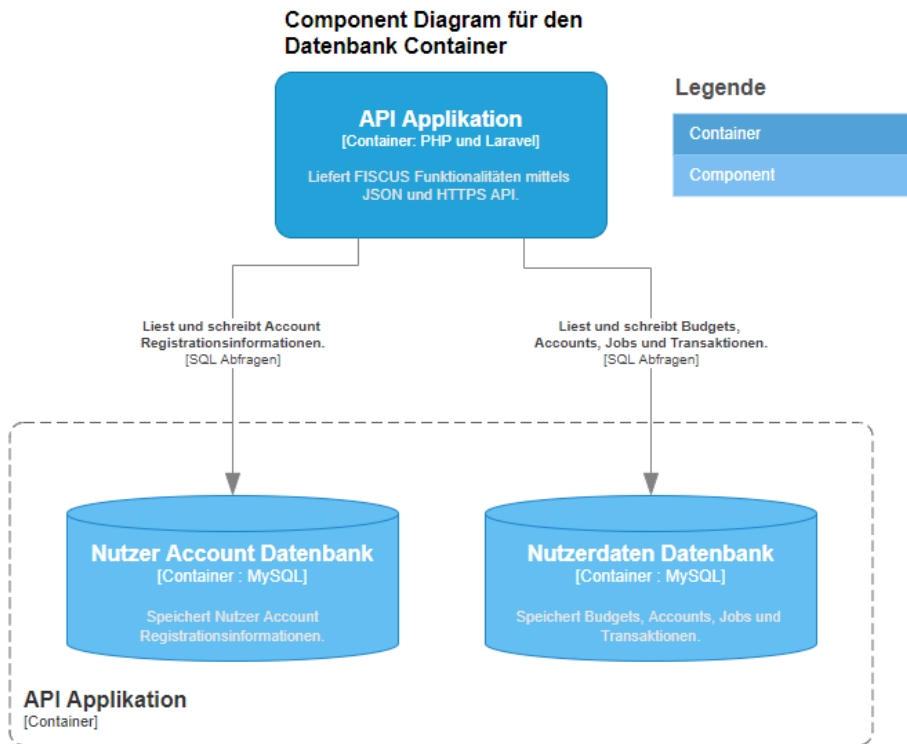


Abbildung 9: FISCUS Datenbank Container Diagram

### 6.3.4. Class Diagram

Durch Herrn Simon Brown, der Ersteller des C4 Models, wird empfohlen die vierte Ebene des C4 Models in der Planungsphase nicht zu erstellen, da diese sich während der Realisierung durchgehend ändern werden. (c4model.com, 2022) Weiterhin kann sich die Arbeit erspart werden, da durch die IDE entsprechende Klassendiagramme automatisiert erstellt werden können. Für eine konzeptionelle Grundlage sind dennoch folgend das Klassendiagramm mit den bereits absehbaren Modellen und Controllern sowie ein Datenbankschema aufgeführt.



### 6.3.4.1 API Applikations Container Klassendiagram

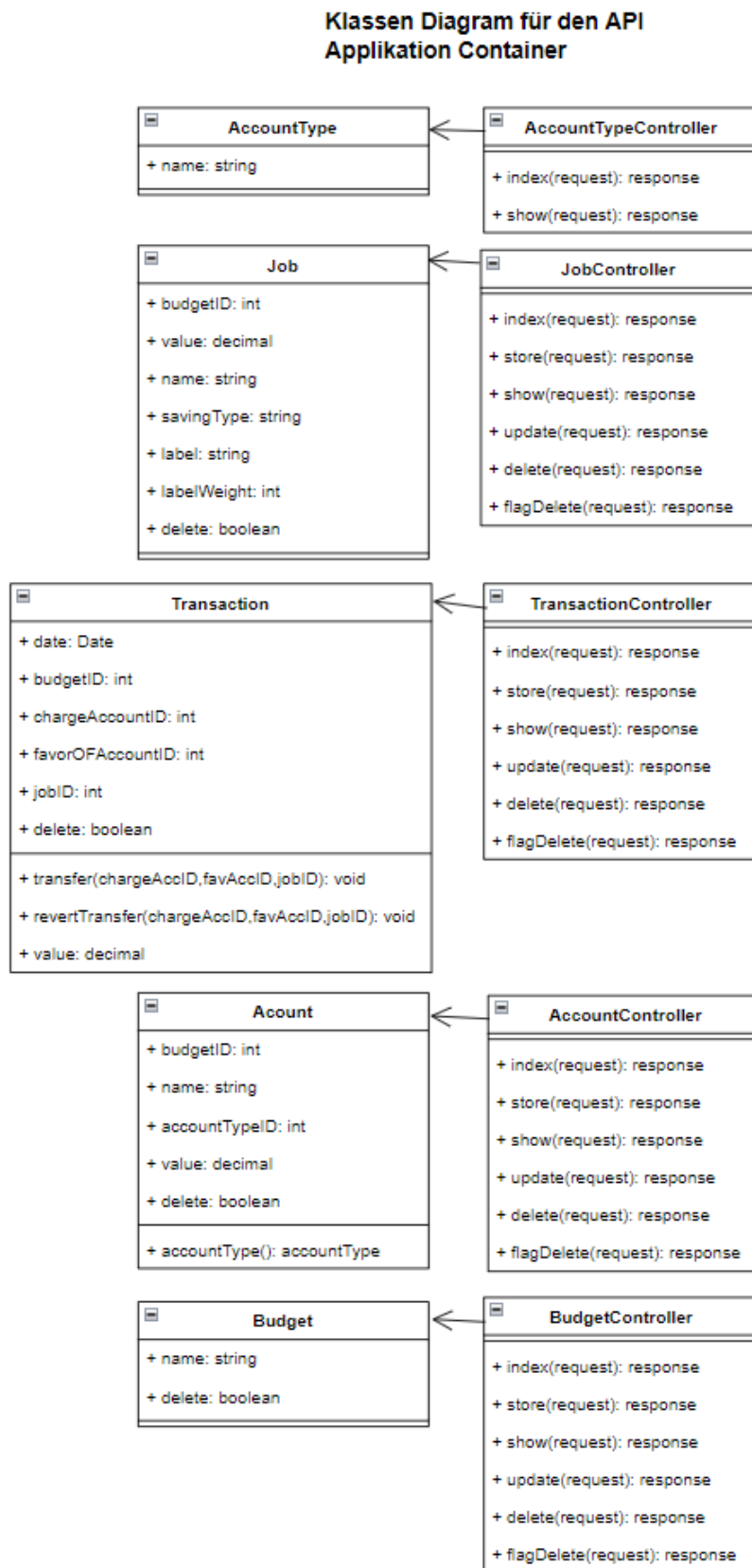


Abbildung 10: FISCUS Klassen Diagram

### 6.3.4.2 Nutzerdaten Datenbank Schema Diagram

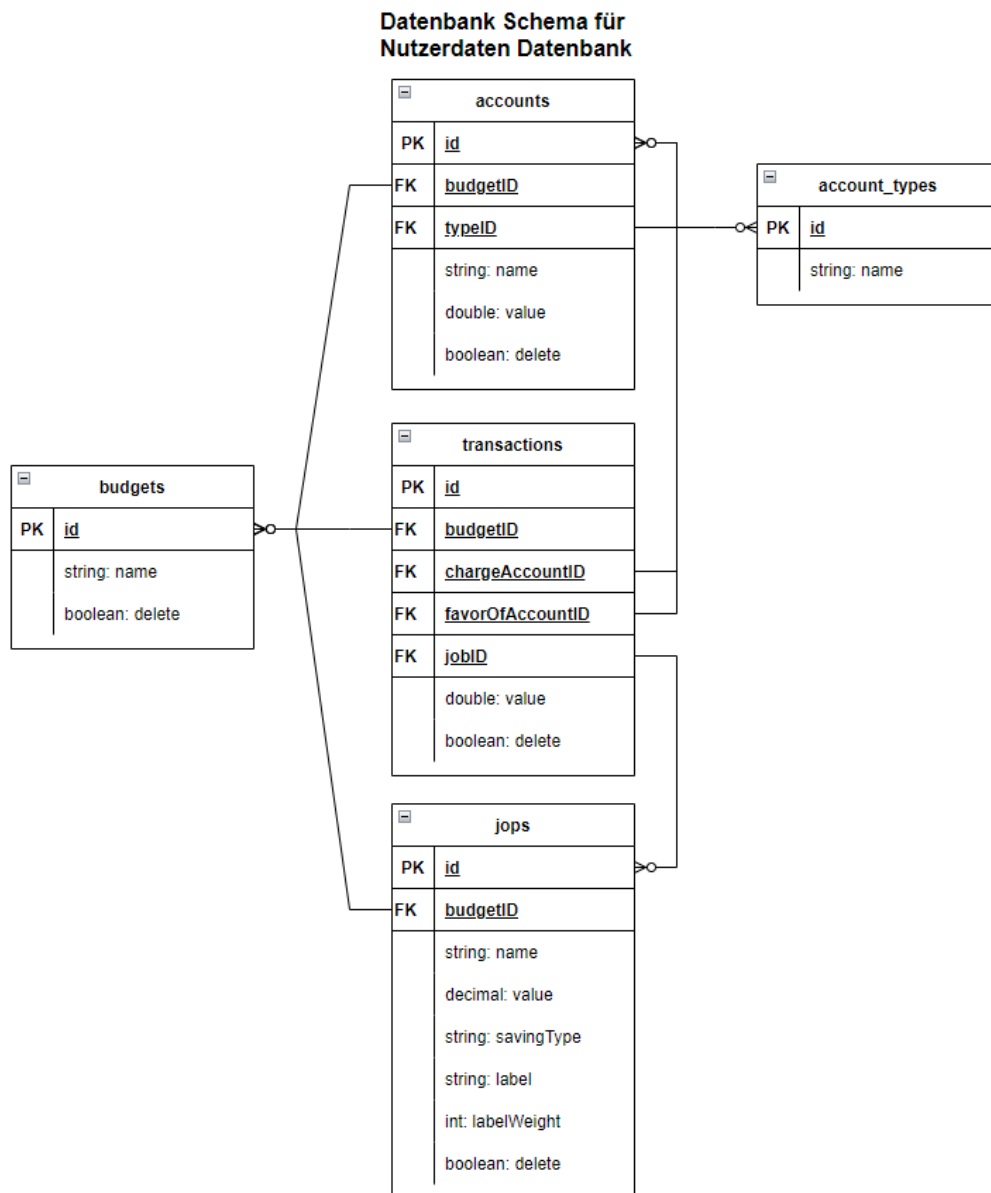


Abbildung 11: FISCUS Datenbank Schema

## 7. Realisierung

### 7.1. REST API-Schnittstellendefinition

Um unabhängig voneinander Arbeiten zu können sowie das System offen zu gestalten haben wir uns für eine Client-Server Architektur entschieden. Um den Datenaustausch zu vereinheitlichen und veröffentlichen, dokumentieren wir die REST API-Schnittstelle.

Um REST APIs zu beschreiben und dokumentieren ist der OpenAPI 3 Standard der heutige Industrie Standard. Die API-Deklaration kann unter Anhang (03\_API-Dokumentation.yaml) eingesehen werden, sie besteht aus beinahe 100 Zeilen. Die UI Version als PDF ebenfalls im Anhang unter dem Dateinamen 02\_API-Dokumentation.pdf eingesehen werden.

## 7.1.1. Aufbau

Der OpenAPI 3 Standard wird mittels eines YAML Dokumentes beschrieben. In der folgenden Abbildung ist die Struktur des Dokumentes ersichtlich.

```
openapi: 3.0.2
info:
  title: Fiscus API
  version: '1.0'
servers:
  - url: https://api.server.test/api/
components:
  securitySchemes:
    BasicAuth:
      type: http
      scheme: basic
  schemas: {}
  responses: {}
  parameters: {}
security:
  - BasicAuth: []
paths: {}
```

Abbildung 12: OpenAPI 3 Aufbau & Struktur

Zur Basisstruktur gehört die Version von OpenAPI die verwendet wird, der Titel als auch die Version der dokumentierten API-Schnittstelle.

Der “Components” Bereich fasst die wiederverwendbaren Definitionen zusammen. Diese dienen dazu Duplikationen zu vermeiden und das Dokument übersichtlicher zu gestalten.

Der “Security” Abschnitt definiert die Sicherheit Umsetzung, welche für die ganze API gültig ist. In unserem Fall wird eine HTTP(S) Basic Authentication über alle Endpunkte definiert.

## 7.1.2. Schemas

Schemas definieren die Objekte, welche von der API behandelt werden. Die Objekte dienen zur Spezifikation des erwarteten Formates welches primär zur Deduplizierung der verwendeten Elemente im Dokument dient.

```
schemas:
  id:
    description: A unique identifying id (uuid) - helper schema to make ID's look all the same (foreign keys, query parameters & url parameters)
    type: integer
  ID:
    description: A unique identifying id (uuid) - helper schema to make ID's look all the same (object ID)
    properties:
      id:
        $ref: '#/components/schemas/id'
  Base:
    type: object
    description: A base schema used for generic properties every other schema contains
    properties:
      created_at:
        type: string
        format: date
      updated_at:
        type: string
        format: date
  BudgetBase:
    type: object
    description: A budget holds all accounts of an entity (Person or Group of Persons)
    properties:
      name:
        type: string
    required:
      - "name"
  Budget:
    type: object
    description: A budget holds all accounts of an entity (Person or Group of Persons)
    allOf:
      - $ref: '#/components/schemas/ID'
      - $ref: '#/components/schemas/BudgetBase'
      - $ref: '#/components/schemas/Base'
```

Abbildung 13: Beispiel Schemadeklaration

## 7.1.3. Responses

Responses definieren die Antworten, welche von der API zurückgegeben werden. Die globale Definition der Antworten dienen primär zur Deduplizierung der verwendeten Antworten im Dokument.

```
responses:
  Budget:
    description: Budget
    content:
      application/json:
        schema:
          $ref: '#/components/schemas/Budget'
```

Abbildung 14: Beispiel Responsedeklaration

In der Deklaration wird das Format wie JSON oder XML angegeben. In unserem Fall ist dies immer JSON.

Im “schema” Part ist die Wiederverwendung des Budget Objekts ersichtlich. Dies bedeutet, dass die Budget Antwort der API ein Objekt des Typen Budget zurückgibt.

## 7.1.4. Paths

Der Abschnitt “Paths” in der API-Dokumentation ist der wichtigste Abschnitt. In ihm werden alle Endpunkte deklariert. Wir versuchen für jedes Objekt die API so generisch und gleich zu gestalten wie nur möglich. Dies ist sehr zeitaufwändig, bewährt sich aber in der Verwendung der Schnittstelle sehr.

Folgende API-Endpunkte gelten für die meisten Objekte, “<Objekt(e)>” dient dabei als Platzhalter:

```
GET      /<Objekte>
GET      /<Objekt>
POST     /<Objekt>/ {<Objekt>ID}
PUT      /<Objekt>/ {<Objekt>ID}
DELETE   /<Objekt>/ {<Objekt>ID}
```

Der Inhalt in den geschwungenen Klammern “{}” definiert einen Platzhalter / Parameter, bei dem ein variabler Wert angegeben werden kann. In unserem Fall ist dies die Identifikation (ID) des Objektes.

```
/budgets:
  get:
    description: Gets all budgets visible to the user
    parameters:
      - $ref: '#/components/parameters/offsetParam'
      - $ref: '#/components/parameters/limitParam'
    responses:
      '200':
        description: OK
        content:
          application/json:
            schema:
              type: array
              items:
                $ref: '#/components/schemas/Budget'
      '401':
        $ref: '#/components/responses/Unauthorized'
    tags:
      - budget
```

Abbildung 15: Beispiel GET /<Objekte>

### Beispiel GET /<Objekte>

Unter “Responses” werden die Antworten des Endpunktes definiert. In diesem Fall ist das der HTTP Status Code 200 und 401. Unter diesen ist das Antwortformat definiert. Die 200 Status-Code Antwort ist speziell, da hier eine Liste von Budget Objekten zurückgegeben wird.

Die Tags dienen zur Gruppierung der “Paths” in der Visualisierung.

```

/budget:
  post:
    description: Creates a new budget
    requestBody:
      required: true
      content:
        application/json:
          schema:
            $ref: '#/components/schemas/BudgetBase'
    responses:
      '201':
        $ref: '#/components/responses/Budget'
      '406':
        $ref: '#/components/responses/ValidationError'
      '401':
        $ref: '#/components/responses/Unauthorized'
    tags:
      - budget
/budget/{budgetID}:
  parameters:
    - name: budgetID
      in: path
      required: true
      schema:
        $ref: '#/components/schemas/id'
  get:
    description: Gets a budget
    responses:
      '200':
        $ref: '#/components/responses/Budget'
      '401':
        $ref: '#/components/responses/Unauthorized'
      '404':
        $ref: '#/components/responses/NotFound'
    tags:
      - budget
  put:
    description: Updates the budget
    requestBody:
      required: true
      content:
        application/json:
          schema:
            $ref: '#/components/schemas/BudgetBase'
    responses:
      '200':
        $ref: '#/components/responses/Budget'
      '401':
        $ref: '#/components/responses/Unauthorized'
      '404':
        $ref: '#/components/responses/NotFound'
      '406':
        $ref: '#/components/responses/ValidationError'
    tags:
      - budget
  delete:
    description: Deletes a budget
    responses:
      '200':
        description: OK
      '401':
        $ref: '#/components/responses/Unauthorized'
      '404':
        $ref: '#/components/responses/NotFound'
    tags:
      - budget

```

Abbildung 16: Beispiel Deklaration der restlichen Endpunkte

Die Parameter beschreiben die möglichen Parameter für jeden Endpunkt. Diese sind auch global definiert. Dabei gibt es zwei unterschiedliche Arten von Parametern. Die Query und die URL Parameter.

Beispiel mit Query-Parameter:

```

GET /budgets
GET /budgets?limit=100

```

Beispiel mit URL-Parameter:

```

GET /budget/{budgetID}
GET /budget/12332

```

Hauptunterschied der /<objekt> und /<objekt>/<ObjektID> Endpunkte ist der requestBody. Dieser definiert ob und in welchem Format ein Objekt erwartet wird. Beispielsweise die POST Methode des Objektes Budget erwartet ein Objekt vom Schema Typ BudgetBase in der Anfrage. Das erwartete Objekt soll im JSON-Format geliefert werden.

Als Ausnahme zu dieser Struktur gelten die Membership Endpunkte. Für die Membership Endpunkte existieren keine PUT Endpunkte. Grund hierfür ist, dass die Membership Endpunkte eine n:n Beziehung zwischen zwei unabhängigen Objekten beschreiben. Um den Aktualisierungsfall nicht komplizierter als nötig zu gestalten, muss für die Aktualisierung einer Beziehung die Beziehung gelöscht und neu erstellt werden.

## 7.2. Realisierung Backend API

### 7.2.1. Basis Setup

Für die Implementation wird, wie unter Kapitel Evaluation der Entwicklungssoftware definiert das Framework Laravel eingesetzt, welches als Programmiersprache PHP verwendet. Mit Laravel Homestead kann die Entwicklung unabhängig vom verwendeten Hostsystem betrieben werden. Beim Aufbau des Basis Setups ist die aktuelle Version von Homestead die Version v12.8.0.

Bei Homestead handelt es sich um eine Entwicklungsumgebung welche sich in einem vom Hostsystem abgekoppelten Container als VM läuft. Für den Betrieb der VM verwenden wir Virtual Box Version 6.1.26. Die Erstellung der VM wird mit Vagrant automatisiert. Anhand der im Dokument Homestead.yaml enthaltenen Konfigurationen werden diverse Funktionen und grundlegende Einstellung geladen.

Hinweis:

Die zum Zeitpunkt des Basissetup aktuelle Version von Virtual Box 6.1.28 ist nicht mit Homestead v12.8.0 sowie Vagrant 2.2.19 kompatibel.

Homestead bietet inkludierte sowie optionale Software, welche mit der Installation der VM mitaufgesetzt wird.

Folgende Software wird aus dem durch Homestead gebotenen Softwarepaket verwendet:

*Tabelle 6: Homestead intern verwendete Software*

Name	Version	Zweck
Ubuntu	20.04.3 LTS	VM OS
PHP	V8.0.11	Programmiersprache
MySql	Ver 8.0.26	Datenbank
Composer	2.2.4	Dependency und Library management Werkzeug
Artisan	8.77.1	Laravel Framework CLI Werkzeug zur einfachen Erstellung von Softwareteilen

Um auf das Laravel CLI zugreifen zu können wird mittels der Software PuTTY Version 0.74 eine Verbindung mit der VM erstellt. Mittels der im Dokument Homestead.yaml definierten IP-Adresse wird über Port 22 (SSH) die Verbindung aufgebaut.

#### 7.2.1.1 Laravel CLI Befehle

Laravel bietet mit Artisan ein mächtiges Werkzeug zur kompletten Erstellung der Grundstrukturen der Softwarekomponente.

Im Folgenden eine Übersicht über die im Verlauf der VDA meist verwendeten Laravel CLI Befehle.

Tabelle 7: Laravel CLI Befehle

Befehl	Zweck
php artisan make:model ExampleModel -a --api	Erstellt Model mit Namen ExampleModel. Erstellt dazugehörige Fabrik, Migration, Seeder, Controller (nur API spezifische Methoden) und Request Validations Dokumente.
php artisan migrate:fresh --seed	Löscht alle vorhandenen Datenbank Tabellen, migriert die Tabellen anhand der definierten Migrations-Dokumente neu und lädt die Tabellen mit per Faker eingeschränkten zufälligen Werten.
php artisan test	Führt alle Tests durch. Wahlweise mit Filter Option --filter createMethod
composer dump-autoload	Lädt Dokument Pfade neu um Dokument welche nicht über Laravel CLI erstellt wurden zu verzeichnen.

## 7.2.2. MVC Pattern

Das Laravel Framework wird mittels dem MVC Pattern strukturiert. Durch das Laravel CLI erstellte Dateien werden konsequent den entsprechenden Strukturen zugeordnet.

Mit dieser Backend Lösung soll jedoch lediglich eine API geschaffen werden, welche anschliessen die Daten an das Frontend Web-UI liefert. Folgend werden daher die Schritte zur Implementation des View ausgeschlossen.

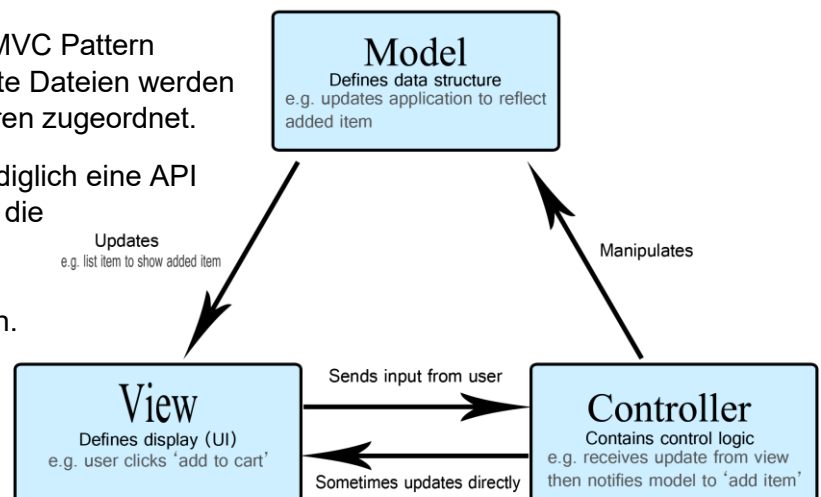


Abbildung 17: MVC Pattern (developer.mozilla.org, 2022)

## 7.2.3. Budgets

Durch die Abhängigkeiten der weiteren Features wird als erstes das Budget implementiert.

Mit der Laravel CLI Befehl «php artisan make:model Budget -a --api» werden die Grundstrukturen erstellt.

### 7.2.3.1 Datenbank Migration

In der Migrations Klasse wird die entsprechende Budget Tabelle sowie deren Felder definiert, welche bei der Datenmigration der Datenbank übermittelt wird.

Tabelle 8: Budget Migration Felder

Feld	Definition
id	Eindeutige inkrementierende Identifikationsnummer
name	Name des Budgets
delete	Markiert den gelöscht-Status des Budgets

Budgets haben keine Abhängigkeiten gegenüber weiteren Tabellen.

```
Schema::create('budgets', function (Blueprint $table) {
    $table->increments('id');
    $table->string('name');
    $table->boolean('delete');
    $table->timestamps();
});
```

Abbildung 18: Quellcode Budget Migrations Klasse

Um Anhand effektiv in der Datenbank vorhandener Budgets testen zu können wurden mit dem «artisan make:» Befehl eine Seeder sowie eine Fabrik Klasse erstellt.

In der Fabrik Klasse werden den Feldern des Budgets zufällige Werte mitgegeben, welche bei jedem neu erstellten Budget aus einem eingeschränkt zufälligen Bereich ausgewählt werden. Hierfür wird die PHP Faker Bibliothek verwendet, über welche beispielsweise zufällige Namen von Firmen beim Seeding der Datenbank den Budgets übergeben werden können.

```
class BudgetFactory extends Factory
{
    /**
     * Define the model's default state.
     *
     * @return array
     */
    public function definition()
    {
        return [
            'name' => $this->faker->company(),
            'delete' => $this->faker->boolean($chanceOfGettingTrue = 25)
        ];
    }
}
```

Abbildung 19: Quellcode Budget Fabrik Klasse

Mit der Seeder Klasse wird nun definiert, wie viele Budgets bei der Migration der budgets Tabelle erstellt werden sollen.

```
class BudgetSeeder extends Seeder
{
    /**
     * Run the database seeds.
     *
     * @return void
     */
    public function run()
    {
        Budget::factory()->times(10)->create();
    }
}
```

Abbildung 20: Quellcode Budget Seeder Klasse

Sind diese Daten definiert kann mit dem entsprechenden Laravel CLI Befehl die Daten migriert werden. Falls gewünscht auch mit Seeding.



## 7.2.3.2 Model

Durch das Laravel Framework werden im Hintergrund diverse Verbindungen geschlossen. So ist es nicht nötig die Budget Felder im Model erneut zu definieren. Nur Eigenschaften und Funktionen, welche nicht den Standardparametern entsprechen müssen, explizit aufgeführt werden.

Mit der \$fillable Variable, welche bereits durch Laravel definiert wurde, können Beispielsweise die Felder des Budgets definiert werden, welche durch http-POST-Methoden verändert werden dürfen.

```
class Budget extends Model
{
    use HasFactory;
    protected $table = 'budgets';

    protected $primaryKey = 'id';
    protected $fillable = ['name', 'delete'];

    /**
     * The attributes that should be cast to native types.
     *
     * @var array
     */
    protected $casts = [
        'delete' => 'boolean',
    ];
}
```

Abbildung 21: Quellcode Budget Model Klasse

## 7.2.3.3 Controller

Gemäss dem MVC Pattern werden im Controller die Daten abgeholt und zusammengestellt welche anschliessend dem View, hier über die API and das Web-UI zur Verfügung gestellt werden sollen.

Die Controller der API verfügen grundsätzlich über fünf Methoden.

Tabelle 9: Controller Standard Methoden

Methode	Rückgabewert
index	Liefert eine Liste der Objekte zurück.
store	Speichert ein Objekt in die Datenbank.
show	Liefert ein spezifisches Objekt zurück, normalerweise anhand der Objekt ID
update	Überschreibt ein Objekt auf Datenbank mit aktuellen Werten.
destroy	Löscht ein Objekt von der Datenbank.

Da der Nutzer die Möglichkeit haben sollte, Budgets, Accounts, Jobs und Transactions zu löschen, aber bei Bedarf auch wiederherzustellen ist bei diesen Modellen eine weitere Methode mit dabei welche anstatt der destroy Methode verwendet wird.

Die Modelle Budgets, Accounts, Jobs und Transactions halten daher das Feld «delete» des Typs Boolean, über welches bei der Abfrage über die index Methode die als «true» markierten Objekte per Default aussortiert werden.

```
public function index(Request $request)
{
    $helper = new ControllerHelper();
    $delete = $helper->setQueryParam($request->query("delete"), DeleteEnum::FALSE);

    if ($delete == DeleteEnum::ALL) {
        $indexBudgets = Budget::
            orderBy('id', 'ASC')->get();
    } else{
        $indexBudgets = Budget::
            where('delete', '=', $helper->reverseCastDelete($delete))->
            orderBy('id', 'ASC')->get();
    }

    return $indexBudgets;
}
```

Abbildung 22: Quellcode Budget Controller Klasse mit delete Selektor

### 7.2.3.4 Request

Mit den store und update Methoden werden per http-Request entsprechende Objekte zur Speicherung, respektive Auffrischung geliefert. Die Objekte werden mit spezifischen Request Klassen auf ihre Validität überprüft. Das Laravel Framework behandelt die auftretenden Fehler und liefert eine Fehlermeldung mit entsprechendem http-Status.

```
public function rules()
{
    return [
        'name' => 'required|string|min:1|max:30',
        'valueGoal' => 'required|numeric|between:-999999999.99,999999999.99|regex:/^[-]?[0-9]*.[0-9]?[0-9]?$/ ',
        'dateGoal' => [new DateGoalRequired($this->savingType), 'date', 'date_format:Y-m-d'],
        'budgetID' => 'required|integer|exists:budgets,id',
        'savingType' => 'required|integer|min:0|max:2',
    ];
}
```

Abbildung 23: Quellcode Job Request Validierung Klasse

Kann unter den gegebenen Validierungsmöglichkeiten keine entsprechende gefunden werden können selbst Regeln mit passender Fehlermeldung definiert werden.

### 7.2.3.5 API-Definition

Erreichbar sind die Budgets über die im File /app/routes/api.php definierten Schnittstellen.

Schnittstelle	http-Methode	Rückgabewert
Route::get('/budgets', [BudgetController::class, 'index']);	GET	Liefert eine Liste aller Budgets.
Route::post('/budget', [BudgetController::class, 'store']);	POST	Speichert ein neues Budget in der Datenbank.
Route::get('/budget/{budgetID}', [BudgetController::class, 'show']);	GET	Liefert eine Budget Entität anhand der in der URL beschriebenen Budget ID.
Route::put('/budget/{budgetID}', [BudgetController::class, 'update']);	PUT	Überschreibt ein Budget anhand der in der URL beschriebenen Budget ID mit den gelieferten Werten.
Route::delete('/budget/{budgetID}', [BudgetController::class, 'flagDelete']);	DELETE	Markiert ein Budget als gelöscht. Man beachte die aus der BudgetController verwendete Methode «flagDelete».

```

/**
 * Budgets
 */
Route::get('/budgets', [BudgetController::class, 'index']);
Route::post('/budget', [BudgetController::class, 'store']);
Route::get('/budget/{budgetID}', [BudgetController::class, 'show']);
Route::put('/budget/{budgetID}', [BudgetController::class, 'update']);
Route::delete('/budget/{budgetID}', [BudgetController::class, 'flagDelete']);

```

Abbildung 24: Quellcode Budget API

## 7.2.4. Accounts

Nach der Implementation der Budgets folgen die Accounts.

Ein Account entspricht einer realen Geldquelle, also beispielsweise ein Sparkonto, Debit Konto oder einfach das Portemonnaie. Sie enthalten dementsprechend einen Wertanteil der folgend auf die verschiedenen Jobs verteilt werden sollen.

### 7.2.4.1 Datenbank Migration

Auch die Accounts erhalten einen eigenen Namen sowie die Zuordnung zu einem Budget mittels Budget ID. Weiterhin wird den Accounts einen Account Typ zugeteilt. Mit diesem kann durch den Nutzer definiert werden um was für einen Account Typ es sich handelt. (Sparkonto, Debit Konto usw.)

Tabelle 10: Account Migration Felder

Feld	Definition
id	Eindeutige inkrementierende Identifikationsnummer
budgetID	Referenzierte Budget ID
typeID	Referenzierte Account Typ ID
name	Name des Accounts
value	Enthaltener Wertanteil
delete	Markiert den gelöscht-Status

## FISCUS Personal Budgeting

Bei den Feldern budgetID und typeId handelt es sich um die Identifikatoren von 1:n Beziehungen. In der Migrationklasse wird ein entsprechender Fremdschlüssel definiert. Diesem wird ebenfalls ein Vorgehen im Fall einer Löschung der Entität mitgegeben. Mit «set null» bleibt die Entität bestehen, mit «cascade» wird sie entfernt.

```
Schema::create('accounts', function (Blueprint $table) {
    $table->increments('id');
    $table->unsignedInteger('budgetID')->nullable("false");
    $table->unsignedInteger('typeID')->nullable("false");
    $table->string('name');
    $table->double('value');
    $table->boolean('delete');
    $table->timestamps();
    $table->foreign('typeID')
        ->references('id')
        ->on('account_types')
        ->onDelete('set null');
    $table->foreign('budgetID')
        ->references('id')
        ->on('budgets')
        ->onDelete('set null');
});
```

Abbildung 25: Quellcode Account Migration Klasse

### 7.2.4.2 Controller

Die Hilfsklasse «ControllerHelper» enthält Methoden, welche in den meisten Controller Klassen Anwendung finden. Ein Beispiel dafür ist die setQueryParameter Methode. Diese müssen zwei Parameter übergeben werden. Der erste Parameter entspricht einem Request Query-Parameter. Wird dieser nicht gesetzt wird der zweite Parameter als Default von der Methode zurückgegeben.

```
public function index(Request $request)
{
    $helper = new ControllerHelper;
    $offset = $helper->setQueryParameter($request->query("offset"), OffsetLimitConstant::OFFSET);
    $offset = $helper->checkIntVal($offset, OffsetLimitConstant::OFFSET);
    $limit = $helper->setQueryParameter($request->query("limit"), OffsetLimitConstant::LIMIT);
    $limit = $helper->checkIntVal($limit, OffsetLimitConstant::LIMIT);
    $delete = $helper->setQueryParameter($request->query("delete"), DeleteEnum::FALSE);
    $budgetID = $request->query("budgetID");

    if ($delete == DeleteEnum::ALL) {
        $indexAccounts = Account::
            where('budgetID', $budgetID)->
            orderBy('id', 'ASC')->get();
    } else{
        $indexAccounts = Account::
            where('budgetID', $budgetID)->
            where('delete', '=', $helper->reverseCastDelete($delete))->
            orderBy('id', 'ASC')->get();
    }

    $indexAccounts = $indexAccounts->slice($offset, $limit);
    $displayAccounts = [];

    foreach($indexAccounts as $accounts){
        array_push($displayAccounts, $accounts);
    }

    return $displayAccounts;
}
```

Abbildung 26: Quellcode Account Controller Index Methode

Die API bietet die Möglichkeit mit dem Request Query-Parameter «delete» nur die als gelöscht markierten, die nicht gelöscht markierten oder alle Accounts anzuzeigen. Mit dem DeleteEnum werden die entsprechenden Werte gegengeprüft. Hier sei angemerkt, dass PHP erst ab Version 8.1 Enum Klassen implementiert. Die PHP-Version 8.1 ist erst seit Homestead v13.0.0 verfügbar. Daher wurde unter /app/Enums/ eine eigene abstrakte Klasse mit Enum entsprechender Funktionalität erstellt, von welcher unter anderem DeleteEnum erbt.

Da es dem Nutzer offengestellt ist hunderte Accounts pro Budget zu erstellen, wird aus Performance Gründen die aus der Datenbank abgerufene Liste in Abschnitte unterteilt. Der URL können die Query-Parameter «offset» und «limit» mitgegeben werden. Die Standardwerte sind 0 für offset und 50 für limit. Werden also die Werte 15 dem offset und 25 dem limit mitgegeben, werden ausschliesslich der Accounts der Position 15 bis 40 zurückgegeben.

### 7.2.5. Tests

Nach der Erstellung der Budgets, Accounts und AccountType API bietet es sich an Tests für die existierenden Schnittstellen zu erstellen. Andernfalls muss bei jeder Änderung jede Schnittstelle inklusive möglicher fehlerhafter Angaben einzeln geprüft werden, was selbsterklärend mit einem hohen Aufwand verbunden ist.

Die genau Testdefinition ist unter dem Kapitel Testing aufgeführt und kann anhand eines Beispiels nachempfunden werden.

### 7.2.6. Jobs

Mit der Implementation der Aufgaben, folgend Jobs genannt, wird auch das USP von FISCUS in die Tat umgesetzt.

Es soll dem Nutzer die Möglichkeit geboten werden bestimmten Sparziele Geld zuzuweisen.

Sinnbildlich hält jeder Job ein Topf, in welchen Geld aus den Accounts zugeteilt wird. Idealerweise wird die gesamte Summe aller dem Budget zugewiesenen Accounts den Jobs zugewiesen. Wird das Geld real ausgegeben wie beispielsweise beim Kauf eines neuen Autos werden den Jobs der Betrag wieder abgezogen. Der Geldtopf ist also die Summe aus Zuteilungen (assignedValue) und Ausgaben (expenseValue).

Abstrahiert betrachtet können die Sparziele in drei Kategorien unterteilt werden.

*Tabelle 11: Sparziele Kategorien*

Feld	Definition	Beispiel
Fortlaufend	Monatlich wiederkehrender Betrag.	Miete, Telefonrechnung
Bestimmter Geldbetrag	Erreichung eines bestimmten Geldbetrags ohne Enddatum.	Neues Auto, Notfallbatzen
Bestimmtes Datum	Erreichung eines Geldbetrags bis zu einem gesetzten Datum.	Hochzeit, Steuern

Anhand dieser Kategorien soll der Nutzer die Jobs erfassen können. Dabei muss die Job Klasse die geeignete Validierung erfüllen und so festlegen, dass ein beim Typ «bestimmtes Datum» ein Enddatum erfasst werden muss, jedoch bei «fortlaufend» dies nicht der Fall ist.

### 7.2.6.1 Datenbank Migration

Das Migrations Dokument für die Jobs weist wiederum die Felder «name» und «delete» sowie eine inkrementierende ID und die zugewiesene Budget ID auf.

Mit dem Feld «savingType», zu welchem ebenfalls eine Enum besteht, wird die entsprechende Funktionalität und anzeigewerte im Model bestimmt.

Weiterhin sind die Felder «valueGoal» und «dateGoal» vorhanden, wobei Letzteres durch die Gegebenheiten der Sparziel Kategorien null entsprechen darf.

*Tabelle 12: Job Migration Felder*

Feld	Definition
id	Eindeutige inkrementierende Identifikationsnummer
budgetID	Referenzierte Budget ID
savingTypeID	Referenzierte integer des SavingTypeEnum
name	Name des Jobs
valueGoal	Zu erreichende Geldsumme.
dateGoal	Enddatum des Jobs.
delete	Markiert den gelöscht-Status

### 7.2.6.2 MonthlySaving

Dem Nutzer wird im Web-UI eine Monatsansicht zur Verfügung gestellt, in welcher er einerseits Informationen über die monatlich zu sparender Geldsumme sowie dem bereits ausgegebenen Betrag informiert wird. Weiterhin kann er über die jeweiligen Monaten Geldbeträge den Jobs zuweisen. Hingegen soll es dem Nutzer nicht möglich sein, Ausgaben auf die jeweiligen Monate zu buchen. Dies soll mittels der Erfassung von Transaktionen funktionieren.

Die monatlichen Sparbeiträge und Ausgaben werden dynamisch erstellt. Der Nutzer kann über das UI alle Monate und Jahre individuell angezeigt bekommen aber erst beim Zuweisen eines Geldbetrags wird eine «MonthlySaving» Entität effektiv in der Datenbank mit der entsprechenden Job ID Referenz erstellt.

### 7.2.6.3 Model

Neben den Standard Variablen wie \$cast, \$fillable usw. weist das Job Model ebenfalls die durch Laravel gegebene Variabel \$appends sowie drei zusätzliche Methoden auf.

Wie man sich denken mag, hängt \$appends durch Entwickler erstellte Methoden and die zu liefernden Entitäten an.

Die getMonthlyGoalAttribute Methode liefert anhand der gewählten Sparziel Kategorie den jeweiligen Betrag, welcher monatlich zu sparen ist.

Tabelle 13: Monatlicher Sparbetrag nach Sparziel Kategorie

Feld	Definition
Fortlaufend	Wird durch die zu erreichende Gesamtsumme des Jobs gebildet.
Bestimmter Geldbetrag	Liefert 0. Hier steht kein monatliches Ziel an.
Bestimmtes Datum	Errechnet sich aus der folgenden Kalkulation. $\text{monatlicher Betrag} = \frac{\text{Sparzielsumme} - \text{Zugeteilte Summe} + \text{Erfasste Ausgaben}}{\text{Ziel Datum} - \text{aktuell angezeigter Monat}}$

Mit der `getStoredValueAttribute` Methode wird den gelieferten Jobs der momentane Stand des Spartopfs mitgegeben. Dieser errechnet sich aus der Summe der erfassten Zuteilungen aller Monate sowie aller Ausgaben, welche dem Job zugeteilt sind.

Die dritte Methode `getMonthAttribute` liefert dem Job die `MonthlySaving` Entität des aktuellen Monats. Über den Request Query-Parameter «month» kann ebenfalls ein bestimmter Monat selektiert und ausgegeben werden.

### 7.2.7. Labels

Um dem Nutzer die Möglichkeit zu bieten die erstellten Jobs zu kategorisieren, können den Jobs Labels zugeteilt werden. Der Nutzer kann die Namen der Labels frei wählen.

Ebenfalls kann Nutzer den Labels eine Gewichtung in Form eines Integers zuordnen. Anhand der Gewichtung können die Labels nach ihrer Wichtigkeit sortiert werden. Die Höhe der Gewichtung ist dem Nutzer überlassen.

#### 7.2.7.1 Kreuztabelle

Jobs können mehrere Labels zugewiesen sein. Ein Label kann ebenfalls mehreren Jobs zugeteilt sein. Dies entspricht einer n:n Beziehung. Um dies zu bewerkstelligen wird die Kreuztabelle «memberships\_\_job\_labels» erstellt.

Bei der Zuweisung eines Labels an einen Job durch den Nutzer wird der Kreuztabelle ein entsprechender Eintrag hinzugefügt.

### 7.2.8. Transactions

Nach den Jobs können nun die Transaktionen implementiert werden. Die Transaktionen behandeln die Buchungen zwischen den Accounts sowie die Buchung von Ausgaben auf die Jobs.

#### 7.2.8.1 Datenbank Migration

Das Datenbank Schema der Transaktionen enthält neben dem Datum, an dem die Transaktion effektiv stattfand oder stattfinden wird, ebenfalls den zu buchenden Betrag. Weiterhin enthält das Schema die Accounts und den Job welche von der Betragsverschiebung betroffen sind.

Tabelle 14: Transaktion Migration Felder

Feld	Definition
id	Eindeutige inkrementierende Identifikationsnummer
budgetID	Referenzierte Budget ID
date	Nutzer definiertes Datum der Transaktion
chargeAccountID	ID des Accounts welcher der Betrag abgezogen wird
favorOfAccountID	ID des Accounts auf den der Betrag gebucht wird
chargeJobID	ID des Jobs dem der Betrag zugeordnet werden soll.
value	Zu buchender Betrag
delete	Markiert den gelöscht-Status

### 7.2.8.2 Transfer / Invertierter Transfer

Das Transaktion Model enthält zwei Methoden, welche für die effektive Übertragung des Betrags verwendet werden.

Die «transfer» Methode hält drei Parameter, \$chargeAccountID, \$favorOfAccountID und \$value. Erwartungsgemäss wird vom Account mit der \$chargeAccountID der Betrag \$value abgezogen und zu \$favorOfAccountID addiert.

Die reverseTransaction Methode kommt zur Anwendung, wenn eine Transaktion abgeändert oder gelöscht wird. Diese Methode hält als Parameter eine Transaktion ID. Anhand dieser werden die Account ID's und der gebuchte Betrag ermittelt und die Buchung rückgängig gemacht.

```
public function transfer ($chargeAccountID, $favorOfAccountID, int $value){
    $chargeAccount = Account::find($chargeAccountID);
    $favorOfAccount = Account::find($favorOfAccountID);

    $this->negativeCharge($chargeAccount, $value);
    $this->positiveCharge($favorOfAccount, $value);
}

public function reverseTransfer ($transactionID){
    $transaction = Transaction::find($transactionID);
    $this->positiveCharge(Account::find($transaction->chargeAccountID), $transaction->value);
    $this->negativeCharge(Account::find($transaction->favorOfAccountID), $transaction->value);
}
```

Abbildung 27: Transaktion Transfer und reverse Transfer Methoden

Hinweis:

Man beachte, dass in beiden Methoden die Job ID der Transaktion nicht behandelt wird. Mehr dazu unter Kapitel Fazit.

## 7.3. Realisierung Frontend

### 7.3.1. Basis Setup

Für die Implementation wird, wie unter Kapitel Evaluation der Entwicklungssoftware definiert das Framework Vue eingesetzt, welches als Programmiersprache JavaScript verwendet. Beim Aufbau des Basis Setups ist die aktuelle Version von Vue 3, allerdings verwenden wir Version 2. Dies wird damit begründet das die UI Bibliothek Vuetify noch nicht mit der neuen Vue 3 Version kompatibel ist.

Folgende JavaScript Bibliotheken werden für das Frontend verwendet:



Name	Version	Zweck
axios	^0.24.0	HTTP Request Client Bibliothek
core-js	^3.6.5	Polyfill
vue	^2.6.11	Vue Framework
vue-router	^3.2.0	Router für das Vue Framework
vuetify	^2.4.0	UI & Komponenten Bibliothek für das Vue Framework
vuex	^3.4.0	Zentraler Speicher für das Vue Framework

Tabelle 15: Vue Framework & Bibliothek Versionen

### 7.3.1.1 Frontend CLI Befehle

Im Folgenden eine Übersicht über die im Verlauf der VDA meist verwendeten Frontend CLI Befehle.

Befehl	Zweck
npm run serve	Startet das Vue Framework mit dem Entwicklungsserver
npm run build	Erstellt eine Produktions Webseite
npm run lint	Führt Qualitäts Prüfungen bezüglich der Formatierung durch und korrigiert Fehler automatisch sofern möglich

Tabelle 16: Frontend CLI Befehle

### 7.3.2. UI Pattern & Architektur

#### 7.3.2.1 Pattern

Das Frontend wird mittels des Komponenten Pattern implementiert. Das Pattern wird durch das Framework Vue vorgegeben.

Die Idee des Komponenten Patterns ist, einzelne Funktionen & Design Elemente in möglichst kleine eigenständige Komponenten zu packen. Dies wird für uns in diesem Fall bereits von der Vuetify UI Bibliothek übernommen.

Diese einzelnen Komponenten werden anschliessend zu Ansichten zusammengefasst, welche vom Router geladen werden.

## 7.3.2.2 Architektur

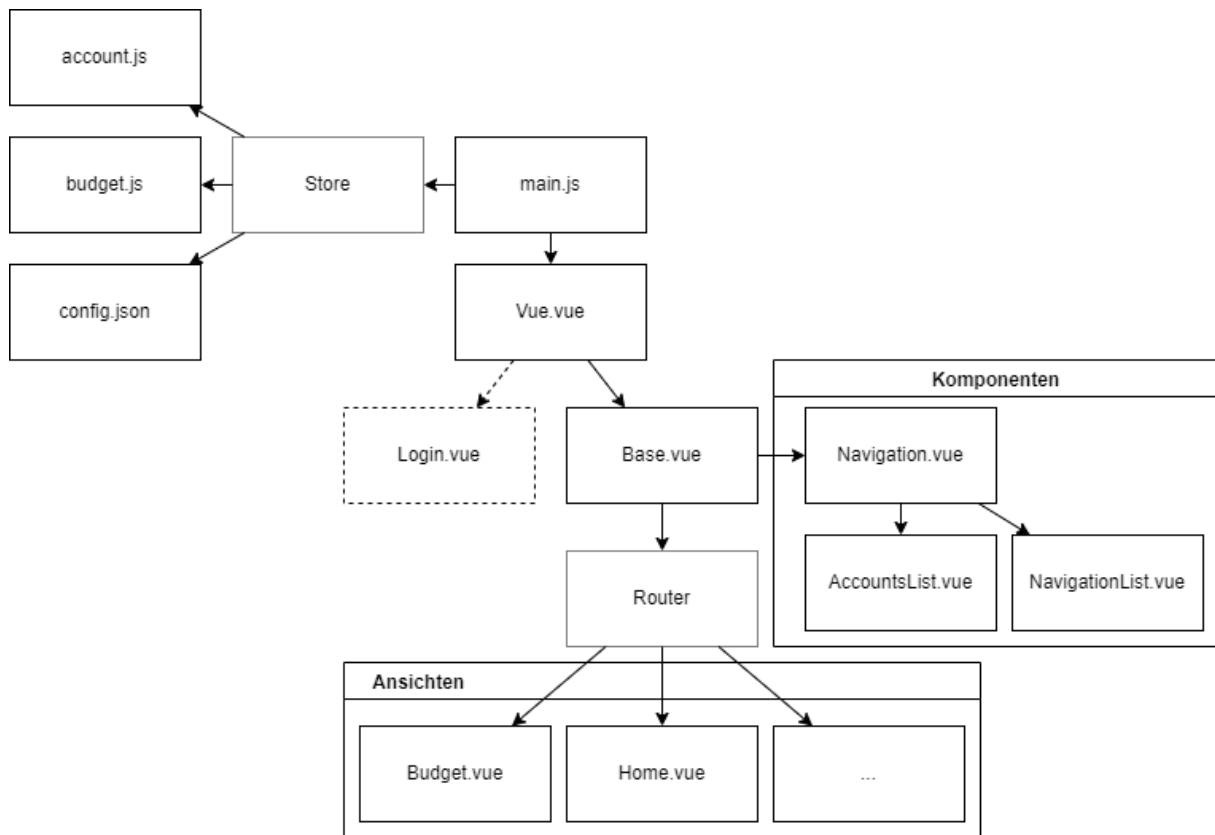


Abbildung 28: UI Pattern & Architektur

Das Komponenten Pattern funktioniert mittels Vererbung. Initial wird die Datei `main.js` gestartet. Diese instanziiert den Vuex Store, lädt die Konfiguration in den Store und instanziiert die Vue Applikation.

Normalerweise würde man den Router direkt in die Vue Applikation integrieren. Hier wurde der Router in die Base Ansicht integriert. Dies soll eine einfache Erweiterung mit dem Login Systems ermöglichen (gestrichelte `Login.vue` Box). Hierzu müsste lediglich eine Kondition in die Vue Applikation eingebaut werden.

Die `Base.vue` instanziiert die Navigation sowie den Router als eigenständige Komponenten.

Dadurch wird eine flexible Navigation zwischen den einzelnen Ansichten wie `Budget.vue` oder `Home.vue` ermöglicht, ohne dass die Navigation mehrfach instanziiert werden muss.

### 7.3.3. Beispiel Komponente/Ansicht

Technisch gesehen ist eine Ansicht exakt das gleiche wie eine Komponente. Sie wird lediglich in einem anderen Verzeichnis gespeichert.

Eine Komponente setzt sich aus einem Pseudo HTML Teil beginnend mit "<template>", einem JavaScript Teil beginnend mit "<script>" und einem CSS Styling Teil, beginnend mit "<style>", zusammen.

#### 7.3.3.1 Pseudo HTML

Die HTML Struktur der Komponente. Dabei wird HTML mit Pseudo HTML vermischt. Unter Pseudo HTML versteht man hier HTML Komponenten (Standard) und Vue Framework Komponenten.

#### 7.3.3.2 JavaScript

Die JavaScript Anweisungen und Funktionen die von der Komponente zur korrekten Funktion benötigt werden. Bei der Verwendung von Komponenten müssen diese hier importiert und unter "components" deklariert werden.

#### 7.3.3.3 CSS Styling

Das CSS Styling wird verwendet, um CSS Styles an nur dieser Komponente anzubringen. Die CSS-Klassen und ID's sind nur in diesem Kontext der Komponente gültig. Dies wird für kleine Modifikationen an Design Elementen verwendet.

```
<template>
  <v-app>
    <navigation></navigation>
    <v-main>
      <v-container>
        <router-view />
      </v-container>
    </v-main>
  </v-app>
</template>

<script>
import Navigation from "../components/Navigation.vue";

export default {
  name: "Base",
  components: {
    Navigation,
  },
  data: () => ({
    direction: "top",
    hover: true,
    right: true,
    bottom: true,
    transition: "slide-y-reverse-transition",
  }),
  computed: {
    budget() {
      return this.$store.getters.budgetID;
    },
  },
};
</script>
<style>
#create .v-speed-dial {
  position: absolute;
}

#create .v-btn--floating {
  position: relative;
}
</style>
```

Abbildung 29: Beispiel Komponente / Ansicht

### 7.3.4. Beispiel Store

```
const Store = {
  state: () => ({
    backendAPI: "",
  }),
  mutations: {
    INSERT_BACKENDAPI(state, payload) {
      state.backendAPI = payload;
    },
  },
  actions: {
    insertConfig({ commit }, payload) {
      commit("INSERT_BACKENDAPI", payload.backendAPI);
    },
  },
  getters: {
    backendAPI(state) {
      return state.backendAPI;
    },
  },
};

export default Store;

export const mutations = Store.mutations;
export const getters = Store.getters;
export const actions = Store.actions;
```

Abbildung 30: Beispiel Vuex Store

Ein Funktionierender Vuex Store besteht aus einem State, Mutations, Actions und Getters

#### 7.3.4.1 State

Der State definiert die Struktur des Speichers.

#### 7.3.4.2 Mutations

Mutations beschreiben, wie der Speicher verändert wird. Der Speicher darf nur über Mutations verändert werden.

#### 7.3.4.3 Actions

Actions werden von den Komponenten und Bibliotheken aufgerufen. Sie sind der «Weg» wie Komponenten standardisiert und koordiniert den State verändern können.

#### 7.3.4.4 Getters

Getters dienen der Abfrage des Speichers.

## 8. Testing

### 8.1. Backend Testing

#### 8.1.1. Testdefinition

Getestet werden die definierten API-Routen auf ihre Rückgabewerte sowie deren Response-Status. Weiterhin werden Funktionen der definierten Modelle auf deren korrekte Funktionsweise überprüft.

Der Überprüfung untersteht neben der korrekten Funktionsweise ebenfalls das Fehlerhandling. Getestet wird mit fehlenden und fehlerhaften Angaben, um Abstürze und Datenkorruption zu provozieren und erwartungsgemäss zu behandeln.

Die Tests werden in einer Mischung zwischen Blackbox und Whitebox verfahren getestet. Die Unit Tests dienen dem Whitebox Testverfahren, für das Blackbox Verfahren wird Postman verwendet. Das Blackbox-Verfahren wird nur für die komplexen Abfragen wie den Transaktionen eingesetzt und aus Platzgründen nicht in diesem Dokument dokumentiert.

## 8.1.2. Typische Testszenarien

Anhand der Create Methode werden folgend die typischen Testszenarien einer Methode dargestellt. Diese werden bei allen Controllern sowie bei allen darin definierten Methoden (Bsp. index, create, show, update, delete, flagDelete) durchgeführt.

### 8.1.2.1 CreateTest

Testklasse welche das Testing einer Create Methode abbildet.

### 8.1.2.2 Test create

Testet die Create Methode auf ihre korrekte funktionsweise. Es wird überprüft, ob ein entsprechendes Objekt in der Datenbank abgelegt wurde. Weiterhin wird der http-Status 201 überprüft und ob die aufgebaute Session keine Fehler enthält.

### 8.1.2.3 Test create with invalid path

Überprüft bei einer fehlerhaften Pfadangabe der Rückgabewert auf den http-Status 404.

### 8.1.2.4 Test create with invalid method

Überprüft bei einer fehlerhaften Methode der Rückgabewert auf den http-Status 405.

### 8.1.2.5 Test create with empty parameter

Überprüft, ob bei fehlenden oder leeren Query-Parametern entsprechende Fehler geworfen werden sowie den http-Status 302 rückgegeben wird.

### 8.1.2.6 Test create with invalid parameter type

Überprüft, ob bei der Angabe falscher Typen (Bsp. String anstatt Integer) in den Query-Parametern entsprechende Fehler geworfen werden sowie den http-Status 302 rückgegeben wird.

### 8.1.2.7 Test create with invalid value length

Überprüft, ob bei der Angabe der Query-Parameter unerlaubte Werte der Typen Integer und Decimal angegeben und entsprechende Fehler geworfen werden sowie den http-Status 302 rückgegeben wird.

```
/**
 * Test Account create.
 */
public function testAccountCreate()
{
    $response = $this->post('/api/account',[
        "name" => "testAccountCreate",
        "budgetID" => "2",
        "value" => "546",
        "typeID" => "3"
    ]);
    $testAccount = Account::latest('id')->first();

    $response->assertStatus(201);
    $response->assertSessionHasNoErrors();

    $this->assertEquals($response['id'], $testAccount['id']);

    $this->assertDatabaseHas('accounts',[
        "name" => "testAccountCreate",
        "budgetID" => "2",
        "value" => "546",
        "typeID" => "3"
    ]);
}
```

Abbildung 31: Test Beispiel Create Account Test

### 8.1.3. Vorbereitung

Um den Test die nötigen Daten zur Durchführung der Unittest zu liefern, müssen erst entsprechende Datensätze erstellt werden. Die Erstellung der Datensätze geschieht über Laravel's Migration, Seeder und Factory Klassen sowie den Laravel artisan Funktionalitäten.

Der entsprechende Befehl um die Datenbank zu initialisieren ist:

```
php artisan migrate:fresh --seed
```

Um die Tests zu starten, sollte folgender Befehl ausgeführt werden:

```
php artisan test
```

Weiterhin können spezifische Test mit folgendem Filter abgefragt werden:

```
php artisan test --filter createTest
```

Dem Filter können ebenfalls Teilnamen übergeben werden. Dadurch wird ermöglicht über die Benamslung Testgruppen zu überprüfen.

### 8.1.4. Relevante Software

*Tabelle 17: Backend Test Relevante Software*

Software	Version
Vagrant	2.2.19
VirtualBox	6.1.28
PuTTY	0.74
Backend Quellcode	Gemäss /Anhang/01_FISCUS/backend/
Postman	9.11.0
VS Code	1.63.2
vagrant-winnfsd Plugin	1.4.0
Windows 10 Pro	20H2

## 8.2. Frontend Testing

### 8.2.1. Testdefinition

Das Frontend wird mittels des Blackbox-Verfahrens getestet. Alle Tests werden entsprechend von Hand durchgeführt.

### 8.2.2. Testszenarien

Ein Beispiel Szenario Anhand des "Accounts" wird in diesem Dokument dokumentiert.

#### 8.2.2.1 Accounts Anzeigen

Testen, ob bei der Auswahl eines Budgets die passenden Accounts angezeigt werden.

#### 8.2.2.2 Account Erstellen

Testen, ob ein Account erstellt werden kann. Bedingung hierbei ist, dass das UI die Account Anzeige selbstständig aktualisiert.

#### 8.2.2.3 Account Bearbeiten

Testen, ob ein Account aktualisiert werden kann. Bedingung hierbei ist, dass das UI die Account Anzeige selbstständig aktualisiert.

#### 8.2.2.4 Account Löschen

Testen, ob ein Account gelöscht werden kann. Bedingung hierbei ist, dass das UI die Account Anzeige selbstständig aktualisiert.

### 8.2.3. Vorbereitung

Um die Frontend Tests durchführen zu können muss erst das Environment installiert und gestartet werden.

Voraussetzung für die Tests ist die installierte und laufende Backend API.

Um dem Frontend das richtige Backend zu konfigurieren muss die Datei "frontend/public/config.json" angepasst werden. Diese sollte bereits korrekt konfiguriert sein in der abgegebenen Version.

Danach kann das Frontend gestartet werden wie in der Installation Anleitung beschrieben.

### 8.2.4. Relevante Software

Als Testumgebung dient der lokale Computer mit NodeJS und der installierten API. Folgende Versionen sind relevant für die Durchführung der Tests:

Tabelle 18: Frontend Test Relevante Software

Software	Version
NodeJS	6.14.13
package.json	Korrekte Version wird im Frontend Ordner mitgeliefert.
Chrome Browser	98.0.4758.81
Edge Browser	98.0.1108.43
Brave Browser	V1.35.100
Backend Quellcode	Gemäss /Anhang/01_FISCUS/backend/
Frontend Quellcode	Gemäss /Anhang/01_FISCUS/frontend/

## 8.3. Testprotokoll

Tabelle 19: Übersicht Testfälle

Nummer	Testfall	Status
1	Unit Tests	OK
2	Account Anzeigen	OK
3	Account Erstellen	OK
4	Account Aktualisieren	NOK
5	Account Löschen	OK

### 8.3.1. Unit Testing

Tabelle 20: Testfall 1

Testfall	1
Testgegenstand	Unit Tests
Beschreibung	WhiteBox Testing der Backend API
Bezug	Backend
Details	Befehl: <code>php artisan test</code>
Soll-Ergebnis	Alle 196 Unit Tests werden erfolgreich bestanden.
Ist-Ergebnis	PASS Tests\Feature\Http\Controllers\AccountTypes\AccountTypesIndex ✓ account types index ✓ account types index invalid path ✓ account types index invalid method  Alle 196 Tests wurden erfolgreich bestanden (Output wurde gekürzt)
Status	OK
Kommentar	Die ausführlichen Ist-Ergebnisse können im Anhang /Anhang/05_Unit_Test_Result.
Datum	01.02.2022
Tester	Nicolas Stutz



8.3.2. Account Anzeigen

Tabelle 21: Testfall 2

Testfall	2
Testgegenstand	Account Anzeigen
Beschreibung	Testen ob bei der Auswahl eines Budgets die passenden Accounts angezeigt werden.
Bezug	Frontend
Details	-
Soll-Ergebnis	Alle Accounts des selektierten Budgets werden angezeigt.
Ist-Ergebnis	<div>Die Accounts des gewählten Budgets werden angezeigt: <div><div>Fiscus Budget</div><div><div><div><div><div></div></div><div>Dashboard</div></div><div><div><div></div></div><div>Transactions</div></div><div><div><div></div></div><div>Budget</div></div></div></div><div>Budget Accounts</div><div><div>Hagenes, Bednar and Bernha... Value: 647156.49</div><div>Stehr-Herzog Value: 714579.58</div><div>Cruickshank Ltd Value: 666220.12</div><div>Barrows, Bernier and Harris Value: 507460.49</div><div>Konopelski, Satterfield and ... Value: 366901.02</div><div>Zboncak Ltd Value: -999.34</div><div>Stanton-Crona Value: 921673.64</div><div>ADD ACCOUNT</div></div><div>Abbildung 32: Testfall 2 Accountübersicht</div></div></div>
Status	OK
Kommentar	
Datum	01.02.2022
Tester	Cyrill Näf
Bemerkung	

8.3.3. Account Erstellen

Tabelle 22: Testfall 3

Testfall	3																		
Testgegenstand	Account Erstellen																		
Beschreibung	Testen, ob ein Account erstellt werden kann. Bedingung hierbei ist, dass das UI die Account Anzeige selbstständig aktualisiert.																		
Bezug	Frontend																		
Details	-																		
Soll-Ergebnis	Alle Accounts des selektierten Budgets werden angezeigt inklusive des neu erstellten Accounts.																		
Ist-Ergebnis	Die Accounts des gewählten Budgets werden angezeigt inklusive des neu erstellten Accounts: <div><div><div><div><div>Fiscus Budget</div><div><div>Dashboard</div><div>Transactions</div><div>Budget</div><div>Budget Accounts</div><div>Hagenes, Bednar and Bernha... Value: 647156.49</div><div>Stehr-Herzog Value: 714579.58</div><div>Cruickshank Ltd Value: 666220.12</div><div>Barrows, Bernier and Harris Value: 507460.49</div><div>Konopelski, Satterfield and ... Value: 366901.02</div><div>Zboncak Ltd Value: -999.34</div><div>Stanton-Crona Value: 921673.64</div><div>ADD ACCOUNT</div></div></div><div><div>Account</div><div><div>Name</div><div>Nicolas-Wallet</div></div><div><div>Standard</div><div>Wallet</div></div><div>CREATE</div></div></div></div><div><div><div>Fiscus Budget</div><div><div>Dashboard</div><div>Transactions</div><div>Budget</div><div>Budget Accounts</div><div>Hagenes, Bednar and Bernha... Value: 647156.49</div><div>Stehr-Herzog Value: 714579.58</div><div>Cruickshank Ltd Value: 666220.12</div><div>Barrows, Bernier and Harris Value: 507460.49</div><div>Konopelski, Satterfield and ... Value: 366901.02</div><div>Zboncak Ltd Value: -999.34</div><div>Stanton-Crona Value: 921673.64</div><div>Nicolas-Wallet Value: 0</div><div>ADD ACCOUNT</div></div></div><div><div>Account</div><div><div>Name</div><div>Nicolas-Wallet</div></div><div><div>Standard</div><div>Wallet</div></div><div>UPDATEDELETE</div></div><div><div>Transactions</div><table><tr><th>Dessert (100g serving)</th><th>Calories</th><th>Fat (g)</th><th>Carbs (g)</th><th>Protein (g)</th><th>Iron (%)</th></tr><tr><td>Frozen Yogurt</td><td>159</td><td>6</td><td>24</td><td>4</td><td>1%</td></tr><tr><td>Ice cream sandwich</td><td>237</td><td>9</td><td>37</td><td>4.3</td><td>1%</td></tr></table><div>Rows per page: 51-2 of 2</div></div></div></div> <div>Abbildung 33: Testfall 3 neuer Account erstellt</div> <div>Abbildung 34: Testfall 3 neuer Account in Account Übersicht</div>	Dessert (100g serving)	Calories	Fat (g)	Carbs (g)	Protein (g)	Iron (%)	Frozen Yogurt	159	6	24	4	1%	Ice cream sandwich	237	9	37	4.3	1%
Dessert (100g serving)	Calories	Fat (g)	Carbs (g)	Protein (g)	Iron (%)														
Frozen Yogurt	159	6	24	4	1%														
Ice cream sandwich	237	9	37	4.3	1%														

Status	OK
Kommentar	
Datum	01.02.2022
Tester	Cyrill Näf
Bemerkung	

8.3.4. Account Bearbeiten

Tabelle 23: Testfall 4

Testfall	4																		
Testgegenstand	Account Bearbeiten																		
Beschreibung	Testen, ob ein Account aktualisiert werden kann. Bedingung hierbei ist, dass das UI die Account Anzeige selbstständig aktualisiert.																		
Bezug	Frontend																		
Details	-																		
Soll-Ergebnis	Alle Accounts des selektierten Budgets werden angezeigt, der angepasste Account wird aktualisiert.																		
Ist-Ergebnis	<div><div>Die Accounts des gewählten Budgets werden angezeigt. Der zu Editierende Account wird nicht aktualisiert.</div><div><div><div><div><div>Fiscus Budget</div><div><div><div>Dashboard</div><div>Transactions</div><div>Budget</div></div><div><div>Budget Accounts</div><div>Hagenes, Bednar and Bernha... Value: 647156.49</div><div>Stehr-Herzog Value: 714579.58</div><div>Cruickshank Ltd Value: 666220.12</div><div>Barrows, Bernier and Harris Value: 507460.49</div><div>Konopelski, Satterfield and ... Value: 366901.02</div><div>Zboncak Ltd Value: -999.34</div><div>Stanton-Crona Value: 921673.64</div><div>Nicolas-Wallet Value: 0</div><div>ADD ACCOUNT</div></div></div><div><div>Account</div><div><div>Name</div><div>Nicolas-Wallet-Edited</div></div><div><div>Standard</div><div>Wallet</div></div><div><div>UPDATE</div><div>DELETE</div></div></div><div><div>Transactions</div><table><tr><th>Dessert (100g serving)</th><th>Calories</th><th>Fat (g)</th><th>Carbs (g)</th><th>Protein (g)</th><th>Iron (%)</th></tr><tr><td>Frozen Yogurt</td><td>159</td><td>6</td><td>24</td><td>4</td><td>1%</td></tr><tr><td>Ice cream sandwich</td><td>237</td><td>9</td><td>37</td><td>4.3</td><td>1%</td></tr></table><div>Rows per page: 51-2 of 2</div></div></div></div><div>Abbildung 35: Testfall 4 abgeänderter Account</div></div></div></div>	Dessert (100g serving)	Calories	Fat (g)	Carbs (g)	Protein (g)	Iron (%)	Frozen Yogurt	159	6	24	4	1%	Ice cream sandwich	237	9	37	4.3	1%
Dessert (100g serving)	Calories	Fat (g)	Carbs (g)	Protein (g)	Iron (%)														
Frozen Yogurt	159	6	24	4	1%														
Ice cream sandwich	237	9	37	4.3	1%														
Status	NOK																		
Kommentar	Mängel muss nach der Projektphase behoben werden.																		
Datum	01.02.2022																		
Tester	Cyrill Näf																		
Bemerkung																			

8.3.5. Account Löschen

Tabelle 24: Testfall 5

Testfall	5
Testgegenstand	Account Löschen
Beschreibung	Testen, ob ein Account gelöscht werden kann. Bedingung hierbei ist, dass das UI die Account Anzeige selbstständig aktualisiert.
Bezug	Frontend
Details	-
Soll-Ergebnis	Alle Accounts des selektierten Budgets werden angezeigt, der gelöschte Account verschwindet.
Ist-Ergebnis	<div>Alle Accounts des gewählten Budgets werden angezeigt, der gelöschte Account verschwindet:</div> <div><div><div><div>Fiscus Budget</div><div><div>Dashboard</div><div>Transactions</div><div>Budget</div></div><div><div>Budget Accounts</div><div>Hagenes, Bednar and Bernha... Value: 647156.49</div><div>Stehr-Herzog Value: 714579.58</div><div>Cruickshank Ltd Value: 666220.12</div><div>Barrows, Bernier and Harris Value: 507460.49</div><div>Konopelski, Satterfield and ... Value: 366901.02</div><div>Zboncak Ltd Value: -999.34</div><div>Stanton-Crona Value: 921673.64</div><div>ADD ACCOUNT</div></div></div></div><div>Home</div></div> <div>Abbildung 36: Testfall 5 Löschen eines Accounts</div>
Status	OK
Kommentar	
Datum	01.02.2022
Tester	Cyrill Näf
Bemerkung	

## 9. Benutzeranleitung

### 9.1. Installation Backend

#### 9.1.1. Softwarevoraussetzung

Folgende Software ist für die Entwicklung mit dem Laravel Homestead Framework vorausgesetzt.

*Tabelle 25: Übersicht Softwarevoraussetzung*

Name	Version	Zweck
Virtual Box	6.1.28	VM Software
Vagrant	2.2.19	Automatisierung der Erstellung der VM Entwicklungsumgebung
PuTTY	0.74	Verbindungswerkzeug für den Zugriff die VM
FISCUS ZIP	Nach Abgabe	FISCUS Projekt in ZIP Format. Es hält die weiter benötigte Software

Virtual Box, Vagrant und PuTTY können anhand ihrer Installationsanweisung installiert werden.

Weiterhin empfiehlt das Softwareteam folgende Software für die Arbeit an FISCUS.

*Tabelle 26: Empfohlene Software*

Name	Version	Zweck
Postman	V 9.11.0	API-Abfragewerkzeug
VS Code	1.63.2	IDE

#### 9.1.2. Ordnerstruktur

Das ZIP File «VDA-2022-HFI-Näf\_Cyrill-HFI-Stutz\_Nicolas.zip» kann an einen geeigneten Ort entpackt werden.

Für die folgenden Angaben ist der Ordner /Anhang/backend der Ausgangspunkt.

Im Dokument Homestead.yaml sind für Vagrant notwendige Daten hinterlegt. Hier können bei Bedarf diverse Features aktiviert und weitere Konfigurationen definiert werden.

Im Dokument /app/.env befindet sich unter anderem die Applikations-URL, sowie die Zugangsdaten und Schnittstelle zur Datenbank.

Durch das umfangreiche Angebot von Features des Laravel Frameworks, kann die generierte Ordnerstruktur schnell unübersichtlich wirken.

Folgend eine Übersicht der Pfade, unter welchen das Projektteam Dokumente erstellt und bearbeitet hat.

Tabelle 27: Übersicht Dokumentpfade

Pfad	Enthält
/app/Enums/	Enums
/app/GlobalConstants/	Globale Variablen
/app/Helpers/	Global verwendete Methoden
/app/Controllers/	Controllers
/app/Requests/	Request Validierungsdokumente
/app/Models/	Models
/app/Rules/	Individuelle Validierungsregeln
/database/factories/	Fabriken zur Erstellung zufälliger Entitäten
/database/migrations/	Migrations Dokumente
/database/seeders/	Seeders
/tests/Feature/http/Controllers/	Controller Tests
/tests/Feature/Models	Model Methoden Tests

### 9.1.3. Starten der Entwicklungsumgebung

Um die VM mit Laravel Homestead zu starten, muss mittels Vagrant und Virtual Box die VM gestartet werden.

Tabelle 28: Vorgehen zum Starten der Homestead VM

Nr.	Beschreibung	
1	Öffne die Eingabeaufforderung als <u>Administrator</u>	
2	Navigiere zum FISCUS Order, dieser enthält das Homestead.yaml Dokument	
3	Führe folgende Befehl aus: vagrant up	
4	Warte, bis die Vagrant die VM eingerichtet hat. Dies kann beim erstmaligen Starten bis zu 20 Minuten dauern.	

Die Entwicklungsumgebung sollte somit bereit sein. Eine Virtual Box VM läuft im Hintergrund. Mit dem Starten von Virtual Box kann der Status der VM eingesehen werden.

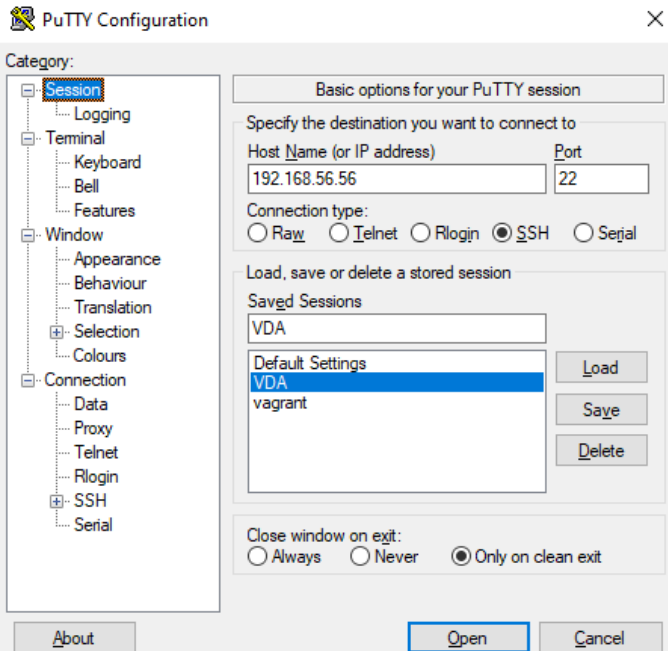
Hinweis:

Wird die VM über Vagrant geschlossen, kann es vorkommen, dass Virtual Box nicht alle Instanzen schliesst. Dadurch kann Vagrant die VM nicht erneut starten und ein Fehler wird geworfen. Um dieses Problem zu beheben, muss über den Taskmanager die vorhandenen Virtual Box Prozesse beendet werden. Anschliessend kann mit dem Befehl «vagrant up» wie gewohnt die VM gestartet werden können.

## 9.1.4. VM Zugriff

Auf die VM kann per PuTTY zugegriffen werden.

Tabelle 29: Vorgehen zum Zugriff auf die VM via PuTTY

Nr.	Beschreibung	
1	Starte PuTTY	
2	Fülle folgende Felder unter Session aus: Host Name: 192.168.56.56 Connection type: SSH	
3	Starte die Verbindung mit Open. Das Laravel CLI öffnet.	
4	Login mit folgenden Daten: login as: vagrant password: vagrant	
5	Wechsle mit «cd code/» zum Quellcodeverzeichnis	

Hier können die verschiedenen Laravel CLI Befehle ausgeführt werden. Weiterhin ist es möglich die Datenbank einzusehen. Die Zugangsdaten für die Datenbank sind im `/app/.env` Dokument hinterlegt.

Mit dem folgenden Befehl kann die Datenbank mit Sample Daten geladen werden:

```
php artisan migrate:fresh --seed
```

Hinweis:

Das Quellcodeverzeichnis der VM ist mit dem auf dem Hostsystem gekoppelt. Änderungen des Quellcodes auf dem Hostsystem werden demnach in die VM übernommen.

## 9.2. Installation Frontend

### 9.2.1. Softwarevoraussetzung

Folgende Software ist für die Entwicklung des Vue Framework vorausgesetzt.

Software	Version
NodeJS	6.14.13
package.json	Korrekte Version wird im Frontend Ordner mitgeliefert.
Chrome Browser	98.0.4758.81
Edge Browser	98.0.1108.43
Brave Browser	V1.35.100

*Tabelle 30: Übersicht Softwarevoraussetzung*

Weiterhin empfiehlt das Softwareteam folgende Software für die Arbeit an FISCUS.

Name	Version	Zweck
Vue.js devtools (Chromium)	5.3.4	Vue Devtools für Chromium basierte Browser
VS Code	1.63.2	IDE

*Tabelle 31: Empfohlene Software*

### 9.2.2. Ordnerstruktur

Das ZIP File «VDA-2022-HFI-Näf\_Cyrill-HFI-Stutz\_Nicolas.zip» kann an einen geeigneten Ort entpackt werden.

Für die folgenden Angaben ist der Ordner /Anhang/frontend der Ausgangspunkt.

Im Dokument public/config.json sind die für die Vue Applikation notwendige Datenkonfigurationen hinterlegt. Diese können bei Bedarf angepasst werden.

Folgend eine Übersicht der Pfade, unter welchen das Projektteam Dokumente erstellt und bearbeitet hat.

Pfad	Enthält
/public	Statische Dateien welche im Produktion Bundle enthalten sein sollten
/src	Enthält den Vue Applikation Code
/src/components	Enthält die selbst Entwickelten Komponenten
/src/plugins	Enthält die Vuetify UI Bibliothek
/src/router	Vue router und Navigation Definition
/src/service	Services sind kleine Hilfs Klassen, die das Entwickeln erleichtern
/src/store	Vuex Store Definitionen
/src/views	Komponente welche als Ansichten verwendet werden

*Tabelle 32: Übersicht Dokumentpfade*



### 9.2.3. Frontend Starten

Um die Website zu starten, muss mittels NodeJS der Development Server inklusive Build Prozess gestartet werden.

Nr.	Beschreibung	Befehl
1	Navigiere in den /frontend Ordner im FISCUS Ordner.	
2	Installiere die Benötigten Abhängigkeiten	npm install
3	Starten des Entwicklung Server	npm run serve
4	Warte, bis der Entwicklungsserver gestartet ist. Dies kann einige Minuten dauern.	

*Tabelle 33: Vorgehen zum Starten der Frontend Applikation*

Die Frontend Applikation sollte somit bereit sein.

### 9.2.4. Zugriff

Um auf das nun gestartete Frontend zuzugreifen wird lediglich ein Browser benötigt in dem die URL <http://localhost:8080> geöffnet wird.

### 9.3. Gebrauchsanweisung des Fiscus Web-UI



Abbildung 37: Frontend Home Page (Erster Start)

Beim ersten Öffnen der Webapplikation muss oben rechts ein Budget ausgewählt werden. Über diese Selektion kann das Budget bei Bedarf auch gewechselt werden.

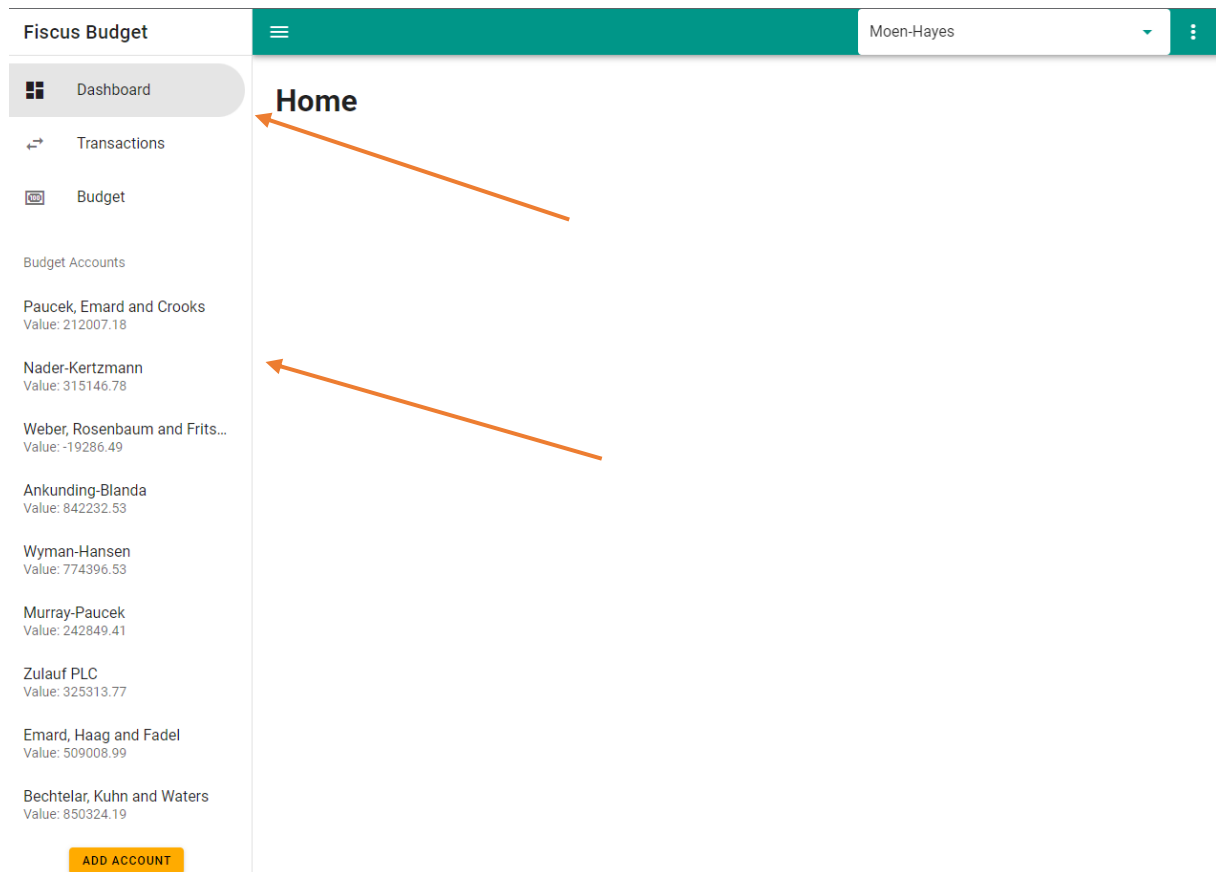


Abbildung 38: Frontend Home Page (Nach Budgete selektion)

Beim Selektieren des Budgets werden automatisch die Account Daten geladen und man wird auf die Home/Dashboard Seite geleitet.

# FISCUS Personal Budgeting

Fiscus Budget

Dashboard

Transactions

Budget

Budget Accounts

Paucek, Emard and Crooks  
 Value: 212007.18

Nader-Kertzmann  
 Value: 315146.78

Weber, Rosenbaum and Frits...  
 Value: -19286.49

Ankunding-Blanda  
 Value: 842232.53

Wyman-Hansen  
 Value: 774396.53

Murray-Paucek  
 Value: 242849.41

Zulauf PLC  
 Value: 325313.77

Emard, Haag and Fadel

Moen-Hayes

Transactions

Date	Charge Account	Favour Account	Charge Job	Value	Action
1971-07-06	Zulauf-Powlowski	Rohan, Nienow and Wisoky	omnis	-51.34	
1982-01-11	Hagenes, Bednar and Bernhard	Rohan, Nienow and Wisoky	omnis	92.64	
1988-09-19	Heathcote LLC	Torp-Rohan	consequatur	-174.91	
1989-04-26	Bartell, Johnson and Sanford	Heathcote-Marks	voluptas	-149.45	
1991-12-04	Dietrich, Orn and Rosenbaum	Kemmer-Kris	et	-53.66	
1998-03-02	Becker Inc	Weber, Rosenbaum and Fritsch	tenetur	-185.73	
2006-11-06	Dickinson, Becker and Conroy	Hartmann, Larson and Turcotte	repellat	-74.2	
2007-09-19	Douglas-Jones	Schinner Ltd	repellat	-174.75	
2012-09-08	Ward-Keeling	Dare, Jerde and Schmitt	et	118.64	
2012-12-24	Stehr-Herzog	Kreiger-Quitzon	velit	-196.52	
2014-12-12	Feil, Cremin and Gorczany	Dietrich, Orn and Rosenbaum	error	-11.85	
2019-04-29	Carter, Keeling and Rempel	Kemmer-Kris	vitae	-49.77	

Abbildung 39: Frontend Transaction Page

Beim Auswählen der Transactions werden alle Transaktionen des aktuellen Budgets angezeigt. Die Anzahl der geladenen Transaktionen beschränkt sich auf die ersten 100. Beim Scrollen zum Ende der Liste werden die nächsten 100 Transaktionen on the fly geladen. Dies funktioniert so lange, bis alle Transaktionen geladen wurden.

## FISCUS Personal Budgeting

**Fiscus Budget**

Dashboard

Transactions

**Budget**

Budget Accounts

Paucek, Emard and Crooks  
Value: 212007.18

Nader-Kertzmann  
Value: 315146.78

Weber, Rosenbaum and Frits...  
Value: -19286.49

Ankunding-Blanda  
Value: 842232.53

Wyman-Hansen  
Value: 774396.53

Murray-Paucek  
Value: 242849.41

Zulauf PLC  
Value: 325313.77

Emard, Haag and Fadel  
Value: 509008.99

Bechtelar, Kuhn and Waters  
Value: 850324.19

**ADD ACCOUNT**

**Tomato**

Name	Value	Action
voluptatem		
consequatur		
omnis		
soluta		
minus		
voluptas		
repellat		
eligendi		
eligendi		
vitae		
quo		
voluptas		

**MediumAquaMarine**

Name	Value	Action
------	-------	--------

Abbildung 40: Frontend Budget Page

Beim Auswählen der Budget Ansicht Werden alle Labels und deren Jobs geladen. Ein Job kann zu mehreren Labels gehören.

**Fiscus Budget**

Dashboard

Transactions

**Budget**

Budget Accounts

Paucek, Emard and Crooks  
Value: 212007.18

Nader-Kertzmann  
Value: 315146.78

**Weber, Rosenbaum and Frits...**  
Value: -19286.49

Ankunding-Blanda  
Value: 842232.53

Wyman-Hansen  
Value: 774396.53

Murray-Paucek

**Account**

Name  
Weber, Rosenbaum and Fritsch

Standard  
Saving Account

**UPDATE DELETE**

**Transactions**

Dessert (100g serving)	Calories	Fat (g)	Carbs (g)	Protein (g)	Iron (%)
Frozen Yogurt	159	6	24	4	1%
Ice cream sandwich	237	9	37	4.3	1%

Rows per page: 5 1-2 of 2

Abbildung 41: Frontend Account Page

Beim Anklicken eines Accounts wird der aktuelle Account geladen.

## 10. Fazit

### 10.1. Offene Punkte

Da die effektive Realisierungszeit der VDA auf weniger als 100 Arbeitsstunden begrenzt ist, wurde gewisse Teile der Software so entwickelt, dass ein funktionierendes aber oftmals suboptimales Produkt entstand. Folgend eine Zusammenfassung dieser Konzepte sowie ein kurzer Lösungsansatz, welcher nach der Abgabe der VDA am bestehenden Produkt implementiert werden könnte.

#### 10.1.1. Transaktion Transfer

Die Funktionalitäten der transfer und reverseTransfer Methoden des Transaktion Model sind nicht einheitlich und ineffektiv.

Auf der einen Seite halten die Accounts einen Wert, auf den mit der Transaktion Subtraktionen und Additionen ausgeübt werden. Auf der anderen Seite wird lediglich die jobID der Transaktion zugewiesen. Eine Verschiebung eines Geldbetrags zur JobID findet nicht statt. Mit dem Laden eines jeden Jobs müssen also alle Transaktionen durchgegangen werden und anhand der Job ID über jeden vorhandenen Monat die Beträge gefilterter werden. Dieses Vorgehen ist nicht skalierbar und daher für einen Produktivsystem nicht ausreichend.

Die jeweiligen monthlySaving Entitäten eines Jobs sollten daher ebenfalls ein Feld «expenseValue» enthalten, worauf der zu transferierenden Betrag gebucht wird.

#### 10.1.2. Frontend MVP

Im Frontend MVP sind nicht alle “Kann” Funktionalitäten implementiert. Für eine Übersicht kann die Tabelle im Unterkapitel «Vergleich Zielerfüllung Pflichtenheft» des Kapitels «Fazit» konsultiert werden.

Rückwirkend betrachtet sollte in der Frontend Architektur vermehrt auf Komponenten gesetzt werden. Aktuell sind die Ansichten zu gross und enthalten zu viel Logik. Dies erhöht die Komplexität um ein Vielfaches. Der wiederverwendungs Wert sinkt dadurch auch stark. Mit einer höheren Anzahl kleinen Komponenten steigt die Modularität.

Eine weitere Unschönheit ist, dass die Komponenten und Ansichten die API-Aufrufe enthalten. Hier wäre es schöner, wenn diese konsequent von einem Service zur Verfügung gestellt würden. An einigen Stellen wurde solch ein Ansatz ausprobiert. Eine saubere Implementierung davon ist allerdings noch ausstehend.

## 10.2. Vergleich Zielerfüllung Pflichtenheft

Die als Muss definierten Anforderungen gemäss Aufgabenstellung wurden erfüllt. Es können mittels REST-API-Abfragen zu Accounts, Jobs und deren Labels erstellt und abgeändert werden. Weiterhin können mit Transaktion zwischen den Accounts Geldbeträge verschoben und Jobs zugewiesen werden.

Weiterhin konnte eine umfangreiche Test-Suite mit über 190 Tests erstellt werden.

## FISCUS Personal Budgeting

Zudem steht eine minimale Webanwendung zur Verfügung mit welcher Nutzer ihre Daten einsehen können.

Die Authentifizierung und Autorisierung von Nutzer sowie Multiuser konnte aus zeitlichen Gründen nicht umgesetzt werden.

Folglich konnten Gemeinschaftskonten aus dem selbigen Grund ebenfalls nicht umgesetzt werden.

Tabelle 34: Zusammenfassung Zielerreichung

Nr.	Ziel	Ziel Art	Ziel erfüllt	Kommentar
1	Erstellen, bearbeiten, anzeigen und löschen eines Budgets	Muss	✓	
2	Erstellen, bearbeiten, anzeigen und löschen eines Budgets Accounts	Muss	✓	
3	Erstellen, bearbeiten, anzeigen und löschen von Aufgaben	Muss	✓	
4	Erstellen, bearbeiten, anzeigen und löschen von Aufgaben Labels	Muss	✓	
5	Erstellen, bearbeiten, anzeigen und löschen von Transaktionen	Muss	✓	
6	Web Client (Als MVP)	Kann		
6.1	Erstellen, bearbeiten, anzeigen und löschen eines Budgets	Kann	(✓)	Aus zeittechnischen Gründen teilweise erfüllt.
6.2	Erstellen, bearbeiten, anzeigen und löschen eines Budgets Accounts	Kann	(✓)	Aus zeittechnischen Gründen teilweise erfüllt.
6.3	Erstellen, bearbeiten, anzeigen und löschen von Aufgaben	Kann	(✓)	Aus zeittechnischen Gründen teilweise erfüllt.
6.4	Erstellen, bearbeiten, anzeigen und löschen von Aufgaben Labels	Kann	(✓)	Aus zeittechnischen Gründen teilweise erfüllt.
6.5	Erstellen, bearbeiten, anzeigen und löschen von Transaktionen	Kann	(✓)	Aus zeittechnischen Gründen teilweise erfüllt.
7	Erstellen, bearbeiten, anzeigen und löschen von Sparzielen pro Aufgabe	Kann	✗	Wurde direkt in die Jobs integriert.
8	Backend Unit Tests	Kann	✓	
9	Nutzer Authentifizierung & Autorisierung	Kann	✗	Aus zeittechnischen Gründen nicht erfüllt.
10	Gemeinschaft Budgets	Kann	✗	Aus zeittechnischen Gründen nicht erfüllt.

## 10.3. Lernaspekt

### 10.3.1. Nicolas Stutz

Das Konzept der Backend Entwicklung sowie die Erstellung von REST-Schnittstellen kannte ich nur ansatzweise. Durch die Erstellung des Backends lernte ich diese Konzepte besser

kennen und implementieren. Weiterhin konnte ich ein besseres Verständnis für das MVC Pattern erlangen.

Da ich als Quereinsteiger in die Softwareentwicklung kam, war es ebenfalls eine neue und sehr positive Erfahrung in einem Team über eine längere Zeit an einem Projekt zu arbeiten.

### 10.3.2. Cyrill Näf

Als Systemtechniker ist die Software Entwicklung Neuland für mich. Eine neue Erfahrung, die Zusammenarbeit im Team war sehr positiv. Dabei erkannte ich wie wichtig es ist sich auf Konzepte, Schnittstellen und Interfaces zu einigen. In unserem Konkreten Fall war das die API-Dokumentation welche auch als Spezifikation diente. Sie ermöglichte die reibungslose Zusammenarbeit zwischen den Projektmitgliedern.

Die Entwicklung des Frontends mit JavaScript und einem Framework zeige mir wie anspruchsvoll Frontend Entwicklung ist. Die grösste Herausforderung für mich war die suboptimale IntelliSense Unterstützung von JavaScript. Diese lässt sich darauf zurückzuführen, dass JavaScript nicht typisiert ist.

## 10.4. Ausblick

Die Entwicklung von FISCUS ist mit dem Abschluss der VDA noch lange nicht beendet.

Es war von Beginn an der Wunsch beider Teammitgliedern ein Produkt zu erstellen, welches nach der VDA mit den gewünschten Features ergänzt und anschliessend veröffentlicht wird.

So wird das Projekt FISCUS unter der bereits bestehender GitLab Repository Gruppe FISCUS als Open-Source Projekt weiterhin zur Verfügung stehen und weiterentwickelt.

## 11. Anhang

Dateiname	Beschreibung
01_FISCUS	Quellcode des Backend und Frontend
02_API-Dokumentation.pdf	API-Dokumentation als Swagger UI
03_API-Dokumentation.yaml	API-Dokumentation als Swagger Quellcode
04_Zeitplan.pdf	Projektzeitplan
05_Unit_Test_Result.pdf	Liste aller Unit Tests Resultate
06_Kritischer_Pfad.png	Kritischer Pfad

Abbildung 42: Übersicht Anhang

## 12. Glossar und Verzeichnisse

### 12.1. Glossar

API .....	<i>Application Programming Interface</i>
CLI .....	<i>Command Line Interface</i>
GUI .....	<i>Graphical user interface</i>
HTML .....	<i>Hypertext Markup Language</i>
HTTP .....	<i>Hypertext transfer protocol</i>
ID .....	<i>Identifikationsnummer</i>
IDE .....	<i>Integrated development environment</i>
JSON .....	<i>JavaScript Object Notation</i>
MIT .....	<i>Massachusetts Institute of Technology</i>
MVC .....	<i>Model View Controller Software Pattern</i>
MVP .....	<i>Minimal Viable Product</i>
PHP .....	<i>Programmiersprache PHP: Hypertext Preprocessor</i>
REST .....	<i>Representational state transfer</i>
UML .....	<i>Unified Modeling Language</i>
USP .....	<i>Unique selling proposition</i>
VDA .....	<i>Vordiplomarbeit</i>
VM .....	<i>Virtuelle Maschine</i>
XML .....	<i>Extensible Markup Language</i>
yaml .....	<i>Yet another markup language</i>



## 12.2. Literaturverzeichnis

*c4model.com*. (15. 1 2022). Von <https://c4model.com/> abgerufen

*developer.mozilla.org*. (15. 1 2022). Von <https://developer.mozilla.org/en-US/docs/Glossary/MVC> abgerufen

*Hostpoint*. (5. 1 2022). Von <https://www.hostpoint.ch/managed-flex-server/managed-flex-server.html> abgerufen

*Hoststar*. (3. 1 2022). Von <https://www.hoststar.ch/de> abgerufen

*Hoststar*. (5. 1 2022). Von <https://www.hoststar.ch/de/hosting> abgerufen

*statisticsanddata*. (2021, 12 26). Retrieved from <https://statisticsanddata.org/data/most-popular-backend-frameworks-2012-2021/>

*Wikipedia*. (3. 1 2022). Von [https://de.wikipedia.org/wiki/Barrierefreies\\_Internet](https://de.wikipedia.org/wiki/Barrierefreies_Internet) abgerufen

*Wikipedia Laravel*. (6. 1 2022). Von <https://en.wikipedia.org/wiki/Laravel> abgerufen

## 12.3. Abbildungsverzeichnis

Abbildung 1: Front-End Transaktionsübersicht .....	5
Abbildung 2: Beispiel Kanban Board .....	13
Abbildung 3: Projekt Zeitplan .....	14
Abbildung 4: Kritischer Pfad .....	14
Abbildung 5: Populärste Backend Frameworks (statisticsanddata, 2021) .....	15
Abbildung 6: FISCUS System Context .....	21
Abbildung 7: FISCUS Container Diagram .....	22
Abbildung 8: FISCUS API Applikation Container Diagram .....	23
Abbildung 9: FISCUS Datenbank Container Diagram .....	24
Abbildung 10: FISCUS Klassen Diagram .....	25
Abbildung 11: FISCUS Datenbank Schema .....	26
Abbildung 12: OpenAPI 3 Aufbau & Struktur .....	27
Abbildung 13: Beispiel Schemadeklaration .....	27
Abbildung 14: Beispiel Responsedeklaration .....	28
Abbildung 15: Beispiel GET /<Objekte> .....	28
Abbildung 16: Beispiel Deklaration der restlichen Endpunkte .....	29
Abbildung 17: MVC Pattern (developer.mozilla.org, 2022) .....	31
Abbildung 18: Quellcode Budget Migrations Klasse .....	32
Abbildung 19: Quellcode Budget Fabrik Klasse .....	32
Abbildung 20: Quellcode Budget Seeder Klasse .....	32
Abbildung 21: Quellcode Budget Model Klasse .....	33
Abbildung 22: Quellcode Budget Controller Klasse mit delete Selektor .....	34
Abbildung 23: Quellcode Job Request Validierung Klasse .....	34
Abbildung 24: Quellcode Budget API .....	35
Abbildung 25: Quellcode Account Migration Klasse .....	36
Abbildung 26: Quellcode Account Controller Index Methode .....	36
Abbildung 27: Transaktion Transfer und reverse Transfer Methoden .....	40
Abbildung 28: UI Pattern & Architektur .....	42
Abbildung 29: Beispiel Komponente / Ansicht .....	43
Abbildung 30: Beispiel Vuex Store .....	44
Abbildung 31: Test Beispiel Create Account Test .....	45
Abbildung 32: Testfall 2 Accountübersicht .....	49
Abbildung 33: Testfall 3 neuer Account erstellt .....	50
Abbildung 34: Testfall 3 neuer Account in Account Übersicht .....	50
Abbildung 35: Testfall 4 abgeänderter Account .....	51
Abbildung 36: Testfall 5 Löschen eines Accounts .....	52
Abbildung 37: Frontend Home Page (Erster Start) .....	58
Abbildung 38: Frontend Home Page (Nach Budgete selektion) .....	58
Abbildung 39: Frontend Transaction Page .....	59
Abbildung 40: Frontend Budget Page .....	60
Abbildung 41: Frontend Account Page .....	60
Abbildung 42: Übersicht Anhang .....	63

## 12.4. Tabellenverzeichnis

Tabelle 1: Persönliche Angaben des Projektteams.....	8
Tabelle 2: Übersicht Personalaufwand .....	12
Tabelle 3: Projekt Termine .....	13
Tabelle 4: Projekt Termine .....	13
Tabelle 5: Vergleich Softwareevaluation.....	17
Tabelle 6: Homestead intern verwendete Software .....	30
Tabelle 7: Laravel CLI Befehle .....	31
Tabelle 8: Budget Migration Felder.....	31
Tabelle 9: Controller Standard Methoden .....	33
Tabelle 10: Account Migration Felder .....	35
Tabelle 11: Sparziele Kategorien.....	37
Tabelle 12: Job Migration Felder .....	38
Tabelle 13: Monatlicher Sparbetrag nach Sparziel Kategorie .....	39
Tabelle 14: Transaktion Migration Felder.....	40
Tabelle 15: Vue Framework & Bibliothek Versionen .....	41
Tabelle 16: Frontend CLI Befehle .....	41
Tabelle 17: Backend Test Relevante Software .....	46
Tabelle 18: Frontend Test Relevante Software.....	48
Tabelle 19: Übersicht Testfälle .....	48
Tabelle 20: Testfall 1 .....	48
Tabelle 21: Testfall 2 .....	49
Tabelle 22: Testfall 3 .....	50
Tabelle 23: Testfall 4 .....	51
Tabelle 24: Testfall 5 .....	52
Tabelle 25: Übersicht Softwarevoraussetzung.....	53
Tabelle 26: Empfohlene Software.....	53
Tabelle 27: Übersicht Dokumentpfade.....	54
Tabelle 28: Vorgehen zum Starten der Homestead VM.....	54
Tabelle 29: Vorgehen zum Zugriff auf die VM via PuTTY .....	55
Tabelle 30: Übersicht Softwarevoraussetzung.....	56
Tabelle 31: Empfohlene Software.....	56
Tabelle 32: Übersicht Dokumentpfade.....	56
Tabelle 33: Vorgehen zum Starten der Frontend Applikation.....	57
Tabelle 34: Zusammenfassung Zielerreichung .....	62