FACHHOCHSCHULE
ERFURT UNIVERSITY
OF APPLIED SCIENCES
Angewandte
Informatik

Projektdokumentation CleverCash

Semesterprojekt Modul Grundlagen Java2

Antong Götz, Jakob Roch, Lilou Steffen, Richard Prax

Abgabedatum: 5. August 2024

Betreuer: Robert Zeranksi, Christian Chartron,
Xander Van der Weken

Inhaltsverzeichnis

Listings

1 Projektidee

CleverCash ist eine Finanzverwaltungsplattform, die Teams mit mehreren Beteiligten einen einfachen Zugang und Überblick über ihre Finanzen bietet. Die Plattform ermöglicht eine transparente Darstellung der gesamten finanziellen Situation eines Teams, sowohl insgesamt als auch für einzelne Beteiligte. Einnahmen und Ausgaben werden übersichtlich und visuell dargestellt, um eine einfache Nachverfolgung zu ermöglichen.

Mit CleverCash können Beteiligte flexibel Geld zur Verfügung stellen und haben die Möglichkeit, mehr Mitbestimmung über Finanzentscheidungen zu erlangen. Darüber hinaus erleichtert die Plattform alltägliche Aufgaben wie das Bestellen von Pizza für die Kollegschaft oder das Buchen von Mietwagen. Ein weiteres Highlight ist die Möglichkeit, gemeinsam auf größere Anschaffungen, wie einen hochwertigen Kaffee-Vollautomaten, zu sparen. CleverCash macht die Finanzverwaltung kollaborativer und flexibler, um den Bedürfnissen von Teams und Gemeinschaften gerecht zu werden.

2 Technologien

Technologien, welche innerhalb des Projektes verwendet wurden:

- Java 21
- SpringBoot
- Gitlab
- JUnit5
- Docker
- pgAdmin
- PostgreSQL
- Postman
- Maven
- JaCoCo
- Jackson
- Lombok
- SWAGGER

Spezifische Versionen der einzelnen Technologien und Abhängigkeiten, welche innerhalb des Projektes genutzt wurden sind den POM.xml Dateien zu entnehmen!

3 Architektur

Das Projekt CleverCash ist als Maven-Multi-Modul-Projekt strukturiert und besteht aus zwei Hauptmodulen: dem *Api Modul* und dem *Storage Modul*. Diese Module arbeiten zusammen, um die Funktionalitäten der Anwendung bereitzustellen. Die Anwendung nutzt das Spring Boot Framework zur Erleichterung der Konfiguration und Verwaltung der Komponenten.

3.1 Modulübersicht

3.1.1 Api Modul

Das Api Modul enthält die Hauptkomponenten für die Verwaltung der REST-API. Dazu gehören:

- **Controller:** Verantwortlich für die Verarbeitung der HTTP-Anfragen und -Antworten.
- **Services:** Enthalten die Geschäftslogik der Anwendung.
- **Config:** Beinhalten verschiedene Konfigurationsklassen für die Anwendung.
- **DTOs (Data Transfer Objects):** Klassen, die für den Datenaustausch zwischen den verschiedenen Schichten der Anwendung verwendet werden.
- **Mapper:** Eine Mapping Klasse mit der Funktionalität Database Objects (DBOs) und Data Transfer Obejct (DTOs) ineinerander zu konvertieren.

3.1.2 Storage Modul

Das Storage Modul umfasst die Datenpersistenzschicht der Anwendung und beinhaltet:

- **Models:** JPA-Entitäten, die die Datenbanktabellen darstellen.
- **Repositories:** Schnittstellen für den Datenzugriff, die von Spring Data JPA bereitgestellt werden.

Datenbank: Die Anwendung verwendet eine PostgreSQL-Datenbank. Die Repositories kommunizieren mit der PostgreSQL-Datenbank, um Daten zu speichern und abzurufen . Dies ermöglicht eine robuste und skalierbare Lösung für die Datenverwaltung.

3.2 Architekturdiagramme

Die nachfolgenden Diagramme sollen vereinfacht die einzelnen Module noch einmal darstellen, wobei jeweils die einzelnen Module exrtahiert voneinander als auch im Zusammenspiel betrachtet werden sollen.

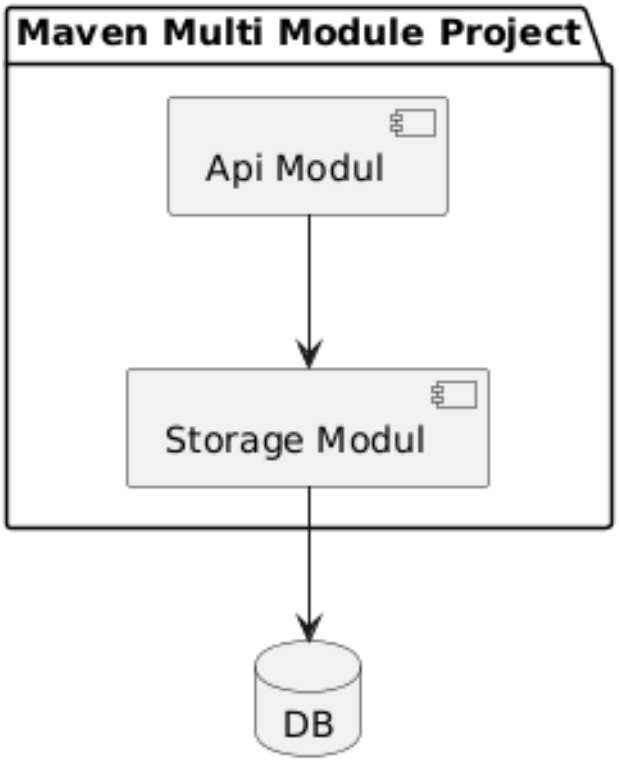

Abbildung 3.1: Übersicht der Module

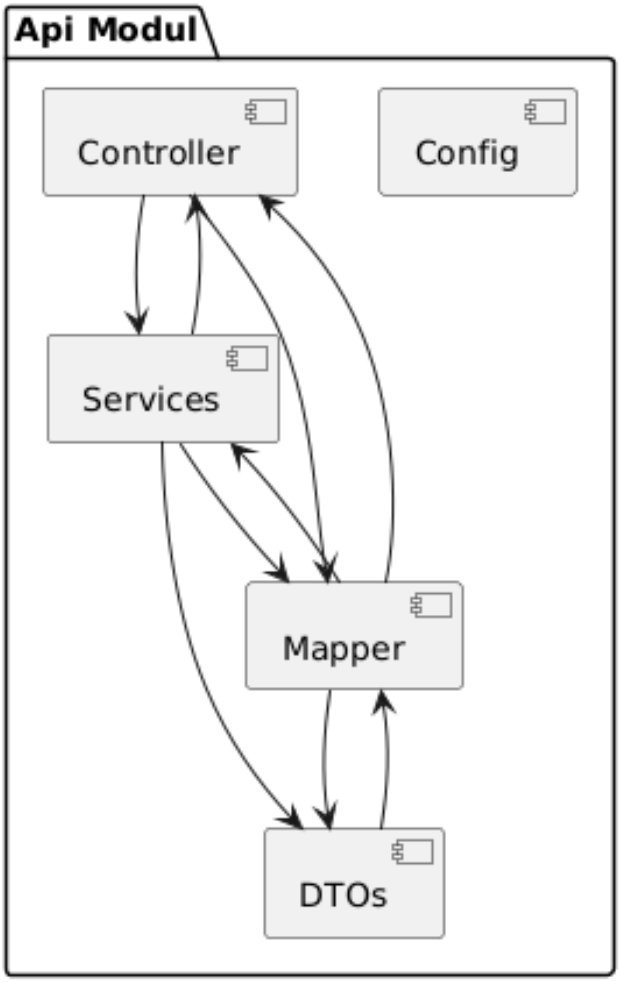

Abbildung 3.2: Komponenten des Api Moduls

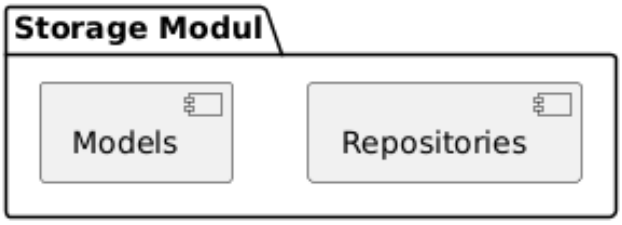

Abbildung 3.3: Komponenten des Storage Moduls

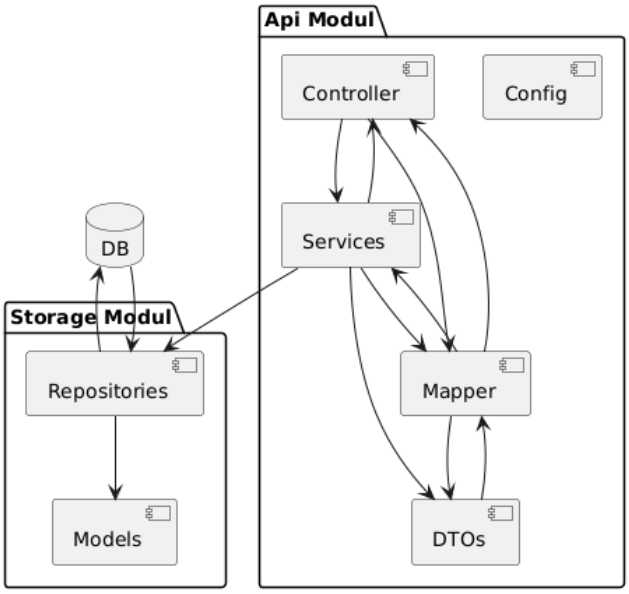

Abbildung 3.4: Zusammenarbeit zwischen den Modulen

4 How-to-run

4.1 Schritte zum Starten der Anwendung

1. Kopiere das Git-Repository in ein lokales Verzeichnis:

```
git clone https://git.ai.fh-erfurt.de/prgj2-24/clevercash.git
```

2. Starte die Docker-Engine.
3. Öffne die Bash im Projektordner
4. Stelle sicher, dass die Ports 8080, 5432, 5050 auf der lokalen Maschine frei verfügbar sind.
5. Starte die Anwendung mit folgendem Befehl:

```
docker compose up --build
```

4.2 Links und Zugangsdaten

- PGAdmin:
  - E-Mail: `Admin@Admin.com`
  - Passwort: `admin`
- BackEnd Port: 8080
- DB Port: 5432

4.3 Anleitung für PGAdmin

Falls bei der ersten Anmeldung keine Datenbank vorhanden ist, sind die folgenden Schritte auszuführen:

1. Rechtsklick auf *Servers*
2. *Register -> Server*
3. Wähle einen Namen deiner Wahl
4. Klicke auf *Connection*

5. Host name: *postgres_cc*
6. Password: *admin*
7. Klicke auf *Save*

Die Datenbanktabellen sind unter folgendem Pfad zu sehen:

- *[Name]->Databases->CleverCashDB->Schemas->Tables*

5 REST-Schnittstellen

Das nachfolgende Kapitel thematisiert die implementierten REST-Schnittstellen und wie man sie testen kann. Die vollständige Dokumenation der Endpunkte ist nach Start der Applikation in der Swagger-Doku zu finden (mehr dazu im Abschnitt zu SWAGGER). Beim Teststen der Endpunkte ist es notwenig, dass man eine bestimmte Reihenfolge einhält. Da in diesem Projekt die Authentifizierung über einen JWT implementiert wurde, ist es notwendig zuerst einen User über die Registrierung zu erstellen, darüber erhält man einen gültigen Access-Token mit welchem man daraufhin weitere Endpunkte testen kann. Weitere Abhängigkeiten in der Reihenfolge der Abfragen werden im Verlauf des Kapitels thematisiert.

5.1 Swagger

Die vollständige Dokumenation der REST-Endpunkte ist nach start der Applikation erreichbar unter:

- http://localhost:8080/swagger-ui/index.html

In diesem Kapitel werden anhand von Testdaten mögliche Szenarien der Applikation durchgespielt, sie sind in dieser Reihenfolge, und mit den hier zur Verfügung gestellten Testdaten, auszuführen um Fehler und Unstimmigkeiten zu vermeiden:

Registrierung
Wie bereits erwähnt ist es notwendig, dass zu beginn des Tests ein User registriert wird über den Authentication Controller, dies kann mit Hilfe dieser Testdaten:

Listing 5.1: Registrierung

```
{
    "firstName": "Max",
    "lastName": "Mustermann",
    "email": "MaxMustermann@gmail.com",
    "birthDate": "2000-01-01",
    "password": "Test1234!"
}
```

Das ausführen liefert den HTTP-Code 201 zurück, was uns sagt, dass der User erfolgreich im System angelegt wurde, außerdem erhalten wir die zugehörige UserID, welche man notieren sollte, und den Access-Token, welcher essentiell ist für weitere Abfragen. Der Token sollte im nachfolgende Kopiert werden und in Swagger im “Authorize” Button eingefügt

werden, nun können alle anderen Endpunkte getestet werden. Bei der Registrierung sollten spezifisch diese Werte genommen werden, da sowohl eine Falsche Datumseingabe, als auch die Eingabe eines zu schwachen Passwortes den Registrierungsprozess unterbrechen.

User Controller

Nach der Registrierung ist es nun möglich den User-Controller aktiv zu testen. Es muss stehts darauf geachtet werden, dass die zugehörige UserID eingetragen wird, welche bei der Registrierung als return Value zurückgegeben wurde. Beim Testen der Endpunkte kann man verschiedene Situationen durchspielen, beispielsweise eine Valide Abfrage über die eigenen User Informationen, welche den Status Code 200 zurückgeben sollte, testet man die Abfrage eines Users mit einer anderen ID, schlägt die Authentifizierung fehl und man bekommt eine 403 Forbidden Exception, was bedeutet das der eingeloggte User mit dem hinterlegten JWT nicht auf die Daten anderer User zugreifen darf.

Das Updaten des Users kann mit den folgenden Daten erfolgen:

Listing 5.2: Update User

```
{
    "firstName": "Max",
    "lastName": "Mustermann",
    "email": "MaxMustermann@gmail.com",
    "birthDate": "1990-01-01",
    "password": "Test1234!"
}
```

Der Delete Endpunkt sollte vorerst nicht ausgeführt werden, da dann in der Folge der User nicht mehr im System existiert und die anderen Endpunkte nicht getestet werden können.

Adress Controller

Der Adress Controller ermöglicht es, dem User eine Adresse zu seinem persönlichen konto hinzuzufügen und auch wieder zu entfernen. Eine Adresse kann erfolgreich mit den folgenden Werten angelegt werden:

Listing 5.3: Adresse hinzufügen

```
{
    "postalCode": "99999",
    "country": "TestLand",
    "city": "TestStadt",
    "street": "TestStrasse",
    "streetNumber": "1",
    "state": "TestStaat"
}
```

Bei diesem Test gilt genau das gleiche wie bei der Registrierung, es sollten genau diese Werte genommen werden, damit keine Probleme währen des Prozessen auftreten.

Bankaccount Controller

Der Bankaccount Controller stellt das Herzstück der Applikation, er ist verantwortlich für das Erstellen und Löschen von Bankaccounts für einen User. Es ist wieder wichtig die richtige UserID zu verwenden wenn Anfragen getestet werden. Das Anlegen eines Bankaccount ist zwingend notwendig, bevor weitere Controller wie Installment, Saving oder auch Transaction getestet werden können. Einen Bankaccount kann erstellt werden für den registrierten User mit folgenden Testdaten.

Listing 5.4: Bankaccount hinzufügen

```
{
    "name": "TestBankAccount",
    "balance": "10000",
    "dailyLimit": "1000"
}
```

Mit diesen Daten wird erfolgreich ein Bankaccount zum User hinzugefügt, die ID, welche zurückgegeben wird beim Erstellen sollte notiert werden, damit weitere Endpunkte getestet werden können.

Ein Bankaccount kann mit folgenden Daten auch geupdated werden.

Listing 5.5: Bankaccount aktualisieren

```
{
    "name": "AndererName",
    "balance": "20000",
    "dailyLimit": "1000"
}
```

Der Bankaccount sollte nicht gelöscht werden bevor nicht alle tests durchgeführt wurden.

Weitere Controller

Wurden die vorherigen Schritte alle erfolgreich ausgeführt sollte es jetzt möglich sein ohne Probleme die restlichen Controller zu testen. In der folgende sind noch einmal 3 Beispiele dargstellelt wie einzelne Sachen hinzugefügt werden können.
Das hinzufügen einer Transaction kann mit Hilfe Folgender Daten erfolgen.

Listing 5.6: Transaction anlegen

```
{
    "amount": 800,
    "description": "Laptop"
}
```

Das hinzufügen eines Savings kann mit Hilfe Folgender Daten erfolgen.

Listing 5.7: Saving anlegen

```
{
    "name": "TestSaving",
    "amount": "500",
    "durationInMonths": 3,
    "startDate": "2025-06-01",
    "description": "SmartPhone"
}
```

Das hinzufügen eines Installments kann mit Hilfe Folgender Daten erfolgen.

Listing 5.8: Installment anlegen

```
{
    "name": "Test",
    "amount": "500",
    "amountPerRate": "100",
    "durationInMonths": "5",
    "description": "Smartphone",
    "startDate": "2025-06-01"
}
```

Es soll darauf hingewiesen werden, das hier lediglich ein kleiner Ausschnitt der Funktionalitäten wiedergespiegelt wurde, jedoch können auf dieser Grundlage die Endpunkte gut getestet werden. Bei einigen Endpunkten gibts es Filteroptionen, daher ist es dort sinnvoll mehrere Obejekte anzulegen, wenn die Filter getestet werden sollen. Desweiteren handelt es sich hier auch nur um postive Fälle, jeglicher Input sollte in diesem Beispiel keinerlei Fehler aufweisen, das System ist jedoch mit Validierungen gegenübern Fehlern teilweise Robust, kleine Ausnahmen sind im Kapitel der Known-Bugs aufgeführt.

5.2 Postman

Die REST-Endpunkte können neben SWAGGER auch über Postman getestet werden, es gilt die gleiche Reihenfolge wie im vorherigen Kapitel beschrieben. Die Postman Collection muss noch über folgenden Weg importiert werden:

- Postman öffnen und über *File* → *Import* die Postman Collection importieren.

Die Collection ist neben der Dokumentation im Docs Ordner des Projektes hinterlegt. Es ist darauf zu achten, dass bei allen Pathvariablen mit “Fixed Values” gearbeitet wurde, daher ist es notwendig abzugleichen, dass bei der Registrierung auch die richtige UserID und bei der Erstellung des Bankaccounts auch dort die richtige ID ausgegeben wurde, wenn nicht ist dies anzupassen bevor die Endpunkte getestet werden können. Der JWT ist ebenfalls bei jeder Abfrage neu zu hinterlegen.

6 Known-Bugs

In diesem Kapitel sollen nicht nur bekannte Fehler thematisiert werden, sondern auch Funktionalitäten welche im Rahmen des Projektes nicht umgesetzt wurden, welche uns jedoch durchaus bewusst sind das diese noch implementiert werden müssten bevor so ein System einsatzfähig ist.

Thema ID
In dem vorliegenden Projekt wurden im Bereich der ID stets mit reinen Integern hantiert, uns ist bewusst, dass dies nicht sicher ist, da man auf der Grundlage einer ID einfach versuchen kann andere IDs zu eraten. Sollte das System online gehen ist es notwendig, dass man die ganze Thematik auf generierte UUID umbaut.

Rollen/Rechte Konzept
Mit dem JWT wurde in unserem System die Grundlage der Authentifizierung gelegt, außerdem erfolgt schon bei den einzelnen Abfragen die Autorisierung über die Identität des Users welcher die Abfrage stellt. Ein Rollen/Rechte Konzept ist im System noch gar nicht implementiert, könnte jedoch auf der geschaffenen Grundlage zusätzlich implementiert werden.

Address Situation
Der jetzige Stand erlaubt es lediglich eine Adresse anzulegen in der Datenbank, wenn sie einem User zugewiesen wird. Sollte das System erweitert werden und es werden noch an anderen Stellen Adressen benötigt und mit diesen hantiert, muss noch eine Funktion hinzugefügt werden, sodass man auch Adressen unanhängig von Usern anlegen kann. Außerdem fehlt die Funktionalität, dass eine Adresse aus der Datenbank entfernt wird, sofern kein User mehr damit in Verbindung steht.

Validierung
Die Validierung des User Input ist an einigen Stellen etwas schwammig gestaltet und es wird ein wenig an den Verstand des Nutzers appeliert. Ein simples Beispiel ist die Überprüfung des Start- und Enddatums beim Filtern, es erfolgt keine Validierung, dass die Startzeit vor der Endzeit liegt, der Output beschränkt sich derzeit auf eine leere Ausgabe. Desweiteren wurde die Validierung nicht an allen Stellen des Projektes gleichermaßen durchgeführt und gehandhabt mittelt Exception-Handling.

Javadoc
Die Generierung des JavaDoc ist fehlerhaft, beziehungsweise funktioniert gar nicht, da be-

stimmte Abhängigkeiten nicht gefunden werden können.

Designentscheidungen

Im Verlauf der Planung haben sich folgende Designentscheidungen ergeben:

- Balance und SavingAmount werden der einfachheit halber im BankAccount redundant gepspeichert => theoretisch berechenbar
- Auf eine eigene History Klasse verzichten wir in unserem Modell, jeder BankAccount hält seine Transactionen
- alreadyPaidAmount bei der Transaction wird gespeichert, damit bestimmt werden kann was die Ratenzahlung abgeschlossen ist
- currentSaving wurde im Bankaccount entfernt, stattdessen hat das jedes Saving ein isActive Flag bekommen => in der Programmlogik wird nun geprüft das nur eins gleichzeitig aktiv ist

7 Zeitplanung

Name	Datum	Aufgabe/Modul
Richard	30.04.2024	Übernahme Java1 Projekt, Setup Projektstruktur für Java2
Richard	30.04.2024	Dockerfile erstellt, CRUD User angefangen zu implementieren
Richard	02.05.2024	CRUD User beendet, notwendige Klassen miterstellt und eingefügt
Richard	03.05.2024	Models und Repos erstellt
Richard	07.05.2024	Überarbeitung ER Modell aus Java 1
Richard	08.05.2024	Setup Maven-Multi-Module Projekt, shareAccount und switchAccount implementiert
Jakob	10.05.2024	Weiter Anpassungen ER-Modell
Richard	12.05.2024	Refactoring bankAccount controller und andere Klassen
Richard	13.05.2024	Refactoring beendet vorerst, Zugriffsmodifikatoren überarbeitet, Logikfehler behoben, logic nesting einheitlich gemacht
Richard	14.05.2024	Implementierung SavingScheduler
Richard	17.05.2024	Implementierung InstallmentScheduler, Refactoring SavingScheduler
Richard	18.05.2024	Klassendiagramm überarbeitet nach Konsultation, Designentscheidungen in Doku notiert
Gruppe	25.05.2024	Besprechung und Abänderungen im Klassendiagramm, weitere Projektplanung
Richard	25.05.2024	Datenmodell angepasst, Logik angepasst, Teststruktur aufgesetzt, erste Tests geschrieben als Orientierung, Annotationen überarbeitet
Richard	26.05.2024	Dockerfile setup + docker compose für DB und alles drum und dran
Richard	27.05.2024	Logik angepasst an neue Models, Routing angepasst, Routing getestet, angefangen Unit tests zu bearbeiten
Jakob	27.05.2024	Zyklische Beziehungen in DB entfernt
Richard	28.05.2024	Tests behoben, routing angepasst
Richard	29.05.2024	Exception handling angefangen, bestehende Logik angepasst, Tests angepasst

Richard/ Jakob	01.06.2024	Merge Request review (Umbau MariaDB zu Postgresql), Annotationen überarbeitet, Code angepasst
Richard	06.06.2024	Modells überarbeitet, Logik Anpassungen, Refactoring
Richard	07.06.2024	Controller refactoring und Aufteilung
Richard	08.06.2024	Services überarbeitet
Richard	09.06.2024	Neues Projekt zusammengemerged mit vorhandenem Code, neues Routing etc.
Jakob	12.06.2024	Aufteilung Services
Jakob/ Richard	14.06.2024	Refactor Filter zu FilterService
Richard	17.06.2024	Tests hinzugefügt AuthController, AddressController
Richard	17.06.2024	Komplett JavaDoc für Services
Richard	18.06.2024	Komplett JavaDoc für Controller
Richard	19.06.2024	Address Service tests, User Service tests
Jakob	20.06.2024	Tests BankAcocuntService und SavingService
Jakob	23.06.2024	Tests BankAcocuntController und SavingServiceController
Richard	25.06.2024	InstallmentService tests
Richard	27.06.2024	InstallmentController tests
Jakob	30.06.2024	UserController Tests
Richard	30.06.2024	Arbeit an der Abschlusspräsentation
Richard	02.07.2024	Vollständige Implementierung DTO Pattern
Gruppe	05.07.2024	Abschlusspräsentation
Richard	05.07.2024	Swagger setup, angefangen Tests zu erneuern
Jakob	05.07.2024	Tests ausgebessert
Richard	08.07.2024	Error/Success DTO eingebaut, Controller angepasst, Return Codes angepasst, Exceptions angepasst, Dokumentation der Controller (JavaDoc + Swagger) neu gemacht
Richard	09.07.2024	Neue Validierung des User Input eingebaut mit Exception Handling
Richard	10.07.2024	Angefangen Tests anzupassen und Logik Fehler zu beheben, Implementierung JWT
Richard	12.07.2024	Refactoring Controller, Response Codes angepasst, Formatierung etc.
Richard	15.07.2024	Refactoring Controller und Services Beendet
Jakob	17.07.2024	Tests auf Controller auf JWT anpassen
Richard	20.07.2024	Dokumentation überarbeitet
Anton	27.07.2024	JavaDoc für die Services überarbeitet
Richard	28.07.2024	Kleine Anpassungen im JavaDoc, Tests beendet
Richard	30.07.2024	Dokumenation weiter geschrieben
Richard	31.07.2024	Dokumenation vollständig fertig gemacht mit allen Diagrammen, Testdaten, Beschreibungen etc
Richard	05.08.2024	Postman Collection Testdaten fertig gemacht und durchgetestet