UEK295_DOKUMENTATION-LB1

Adrian Jasaroska

2. APRIL 2025

CSBE

Inhalt

1 Abbildungsverzeichnis

2 Tabellenverzeichnis

3 Quellenverzeichnis

Table 1 Quellen verzeichnis

WAS?	QUELLE?
CODE SNIPPETS	Präsentation Blocks
TEMPLATE	Boilerplate von Herrn Gauch
CODE INSPIRATION	Noah Lezama (Kollege von mir) Ich fragte ihm was er so alles für Klassen erstellt hat und hat mir dies gesagt, ich habe dann denn Code von denn Präsentationen genommen und wenn ich Fragen hatte, hat er mir auch geholfen
CODE REVIEW	Herr Daniel Schmitz hat mir Empfehlungen gegeben und gewisse Dinge erklärt die ich simplerweise von denn Präsentationen genommen habe
YOUTUBE VIDEO	https://www.youtube.com/watch?v=9SGDpanrc8U Dieses Youtube Video hat mir am anfang geholfen bei Java Spring Boot einigermassen klar zu kommen

4 Arbeitsjournal

4.1 23.03.2025

4.1.1 Tätigkeits Tabelle

Table 2 Erster Tag Tätigkeits Tabelle

Nr.	Tätigkeit	Beteiligte Personen	Soll (Std.)	Ist (Std.)
1	Initiales Projekt-Setup	AJ	1	1.5
2	Boilerplate als Vorlage eingerichtet	AJ	0.5	0.5
3	Projektstruktur geplant	AJ	1	1
	Totale Soll/IST		**2.5**	**3**

4.1.2 Beschreibung des Tagesablaufes

Heute habe ich mit dem Projekt begonnen. Nach dem Input von Herrn Iannattone über das neue Projekt, habe ich mich zunächst eingearbeitet und die grundlegende Struktur geplant. Als Basis habe ich eine Boilerplate-Vorlage verwendet und in das Repository integriert. Dies half mir, schnell eine Grundstruktur zu haben, auf der ich aufbauen konnte. Die Zeit habe ich hauptsächlich mit der Einrichtung der Entwicklungsumgebung und der Planung der Projektarchitektur verbracht. Ich habe etwas mehr Zeit als geplant benötigt, da ich mich erst mit dem Thema vertraut machen musste.

4.2 25.03.2025

4.2.1 Tätigkeits Tabelle

Table 3 Zweiter Tag Tätigkeits Tabelle

Nr.	Tätigkeit	Beteiligte Personen	Soll (Std.)	Ist (Std.)
1	Docker-Compose eingerichtet	AJ	1.5	2
2	MariaDB-Verbindung hergestellt	AJ	1	1.5
3	Controller erstellt	AJ	2	2.5
	Totale Soll/IST		**4.5**	**6**

4.2.2 Beschreibung des Tagesablaufes

Heute habe ich mich auf die Infrastruktur und die Controller-Schicht konzentriert. Zuerst habe ich Docker-Compose für das Projekt konfiguriert, um eine konsistente Entwicklungsumgebung zu schaffen. Das hat etwas länger gedauert als erwartet, da ich einige Probleme mit den Ports lösen musste. Anschließend habe ich die Verbindung zur MariaDB-Datenbank eingerichtet und getestet.

Der Großteil des Tages wurde dann mit der Erstellung der Controller verbracht. Ich habe die grundlegende Struktur für die REST-Endpoints implementiert und dabei versucht, einer klaren API-Design-Strategie zu folgen. Die Controller-Implementierung war aufwändiger als geplant, da ich zusätzliche Validierungen einbauen wollte. Insgesamt bin ich mit dem Fortschritt zufrieden, obwohl ich mehr Zeit als geplant benötigt habe.

4.3 26.03.2025

4.3.1 Tätigkeits Tabelle

Table 4 Dritter Tag Tätigkeits Tabelle

Nr.	Tätigkeit	Beteiligte Personen	Soll (Std.)	Ist (Std.)
1	Komplettes Projekt hinzugefügt	AJ	3	4
2	Code-Überprüfung	AJ	1	1.5
3	Adminer hinzugefügt	AJ	0.5	0.5
	Totale Soll/IST		**4.5**	**6**

4.3.2 Beschreibung des Tagesablaufes

Heute habe ich das komplette Projekt in das Repository integriert. Ich habe die fehlenden Komponenten wie Services, Repositories und Entitäten implementiert und mit den vorhandenen Controllern verbunden. Diese Arbeit war umfangreicher als erwartet, da ich während der Implementierung einige Designentscheidungen überdenken musste.

Nach der Implementierung habe ich eine Code-Überprüfung durchgeführt, um sicherzustellen, dass alles funktioniert und den Best Practices entspricht. Ich habe einige Verbesserungen vorgenommen, besonders im Bereich der Exception-Handling und Validierung.

Zum Schluss habe ich Adminer als Datenbank-Management-Tool zu Docker hinzugefügt, um die Datenbankadministration zu erleichtern. Ich habe festgestellt, dass es noch einige temporäre Probleme mit der Benutzbarkeit gibt, die ich in den kommenden Tagen lösen muss. Trotz der Herausforderungen bin ich mit dem Fortschritt zufrieden.

4.4 02.04.2025

4.4.1 Tätigkeits Tabelle

Table 5 Vierter Tag Tätigkeits Tabelle

Nr.	Tätigkeit	Beteiligte Personen	Soll (Std.)	Ist (Std.)
1	Datenbank-Tabellenerstellung korrigiert	AJ	1.5	2
2	Authentifizierung angepasst	AJ	2	2.5
3	Checkliste hinzugefügt	AJ	0.5	0.5
4	Code mit Thunder Client getestet	AJ	1	1.5
5	Swagger UI und Endpoint-Schutz angepasst	AJ	1	1
	Totale Soll/IST		**6**	**7.5**

4.4.2 Beschreibung des Tagesablaufes

Heute habe ich mehrere wichtige Aspekte des Projekts verbessert. Zunächst habe ich Probleme mit der Datenbank-Tabellenerstellung behoben, die zuvor Fehler verursacht hatten. Die Korrekturen waren etwas zeitaufwändiger als geplant, da ich einige Beziehungen zwischen den Entitäten neu definieren musste.

Anschließend habe ich die Authentifizierung angepasst und dabei festgestellt, dass es Probleme mit 403 Forbidden-Fehlern bei der Registrierung über Thunder Client gab. Diese konnte ich nach einiger Fehleranalyse beheben.

Um den Projektfortschritt besser zu verfolgen, habe ich eine Checkliste erstellt, die die verbleibenden Aufgaben und den aktuellen Status dokumentiert. Das hilft mir, den Überblick zu behalten und Prioritäten zu setzen.

Nach der Implementierung habe ich umfangreiche Tests mit Thunder Client durchgeführt und dabei ein Problem mit der Platzierung des Bearer-Tokens gefunden und behoben. Schließlich habe ich den Schutz für die Swagger UI, Login- und Registrierungs-Endpoints entfernt, da diese logischerweise ohne Token zugänglich sein müssen - schließlich benötigen Benutzer Zugang zu diesen Funktionen, bevor sie ein Token haben.

Obwohl ich mehr Zeit als geplant benötigt habe, konnte ich mehrere kritische Probleme lösen und das Projekt ist nun in einem deutlich stabileren Zustand.

5 Namenschema der Daten

5.1 Klassen und Entitäten

5.1.1 Cases

Es gibt ein paar Standards zu welche Cases verwendet werden können, da gibt es ***snake_case, PascalCase, camelCase*** etc***.*** Es gibt auch sprachen die ein Best-Practice Case haben, natürlich kann man in jedem Case programmieren, man könnte auch in Java in snake_case programmieren oder in C# in camelCase oder JavaScript in snake_case aber es gibt für jede Sprache eine Best Practice. In Python sollte man in snake_case arbeiten, in JavaScript in camelCase und in C# in PascalCase, das sind Normen/Standards die in der Softwarewelt fest gelegt wurden und es ist auch zu empfehlen diese Best Practices so viel wie möglich beim eigenen Code zu implementieren

Abbildung 1 Standards von Cases in der Programmierwelt

5.1.2 Form der Entitäten

Dies ist eine Persönliche Präferenz aber ich habe meine Entitäten immer im Singular geschrieben. Es gibt viele die dies nicht so toll finden, nun habe ich es so implementiert, weil ich es gerne so beschreibe auch wenn es mehr als ein Datensatz in der Tabelle gibt.

5.1.3 Beispiele

Als Beispiel habe ich folgende Entitäten: User, Product und Category

Diese habe ich auch in PascalCase geschrieben, weil ich es am logischten finde dies so zu machen für Entitäten überhaupt, sprich ich nenne Sie bei jeder Programmiersprache in PascalCase, hiermit meine ich aber nur die Entitäten, natürlich jetzt in JavaScript schreibe ich Variablen und Funktionen in camelCase.

5.2 Field Names

5.2.1 Cases

Ich habe schon erklärt was Cases sind, In bezug auf Field Names habe ich immer camelCase verwendet

5.2.2 Namenskonventionen

Ich habe versucht so simple und beschreibende Namen zu geben für die Methoden und allgemein allen Namen im Code

5.2.3 Beispiele

Ich habe für denn Benutzernamen „username“, für denn Passwort „password“ für die Beschreibung „description“ und für denn Preis „price“.

5.3 Primary Keys

5.3.1 Entitäts Regel

Ich habe mir schon immer eine Regel aufgestellt bei Entitäten und dies ist das ich Konsistent eine „id“ Field deklariere. Dies ist auch Best Practice für Datenmodellen

Diese Id gebe ich immer denn Datentyp „Long“, dies ist dementsprechend auch Best Practice

5.3.2 Automatische Id vergebung

Man könnte jede Id von jeden Datensatz manuell vergeben aber dies ist das dümmste was man machen kann, es braucht unnötig Zeilen, Rechenkraft in der CPU und alles ist unnötig. Die Magie die wirklich sehr empfehlenswert ist bei „Id‘s“ und dies ist auch Best Practice ist das man denn Schlüsselwort „Auto Increment“ verwendet, dies vergibt jeden Datensatz automatisch die nächste verfügbare Id, was man sonst noch machen könnte ist eine uuid zu verwenden, dies verwendet man öfter als normale Id da dies eigentlich unleserlich ist und allerlei keine grosse Bedeutung hat ausser natürlich das es zu einem Datensatz gebunden ist.

5.4 Foreign Keys und Relationships

5.4.1 Foreign Key Cases

Ich verwende meistens für Foreign Keys „snake_case“ weil dies auch Best Practice ist und ich das gerne übersichtlich unterscheiden kann ob jetzt etwas ein Primary Key oder ein Foreign Key

Diesen Pattern verwende ich „entity_name_id“ und dies ist das „snake_case“

5.4.2 Relationship Cases

Da habe ich gerne camelCase verwendet sowie man das in meinen Code sieht haben wir als Beispiel „category“ in der Products Klasse

5.5 Collection Tables

5.5.1 Namen

Für die Tatsächlich Tabellennamen in der Datenbank verwende ich snake_case. Wie gefolgt sieht das so aus in diesem Pattern „product“ Zum beispiel auch für die Benutzer Rollen Collection ist es „user_id“

5.6 Column Constraints

5.6.1 Spezielle Attributen

Ich habe es gesetzt sodass bei denn erforderlichen Columns, diese auch Nullable sind, dies bedeutet dass der Wert „null“ ein mögliche Eingabe ist und dies so in die DB gespeichert wird.

Genauso hab ich bei manchen Columns auch „Uniqueness“ eingefügt, dies beduetet das es keine doppelte Angaben für diese Column gibt. Wenn ein gewisses Wert genommen wird, kann nur dieser Column diesen Wert haben und alle anderen neuen Anfragen mit diesem Wert werden blockiert, die Daten werden nicht gespeichert und ein Fehler tritt hervor.

Dies macht Sinn so zu implementieren bei Usernames, sodass es keine gleiche Usernames vergeben werden können

5.7 Projektstruktur und File Namen

```
.
├── Checklist.md
├── docker-compose.yml
├── mvnw
├── mvnw.cmd
├── pom.xml
├── README.md
├── src
│   ├── main
│   │   ├── java
│   │   │   └── ch
│   │   │       └── csbe
│   │   │           └── productmanagment
│   │   │               ├── controller
│   │   │               │   ├── AuthController.java
│   │   │               │   ├── CategoryController.java
│   │   │               │   ├── ProductController.java
│   │   │               │   └── UserController.java
│   │   │               ├── dto
│   │   │               │   ├── AuthRequest.java
│   │   │               │   ├── AuthResponse.java
│   │   │               │   ├── category
│   │   │               │   ├── product
│   │   │               │   └── user
│   │   │               ├── model
│   │   │               │   ├── Category.java
│   │   │               │   ├── Product.java
│   │   │               │   └── User.java
│   │   │               ├── ProductManagmentApplication.java
│   │   │               ├── repository
│   │   │               │   ├── CategoryRepository.java
│   │   │               │   ├── ProductRepository.java
│   │   │               │   └── UserRepository.java
│   │   │               ├── security
│   │   │               │   ├── AuthService.java
│   │   │               │   ├── JwtAuthFilter.java
│   │   │               │   ├── JwtService.java
│   │   │               │   └── SecurityConfig.java
```

```
|   |   |           └── service
|   |   |               ├── CategoryServiceImpl.java
|   |   |               ├── CategoryService.java
|   |   |               ├── ProductServiceImpl.java
|   |   |               ├── ProductServiceInterface.java
|   |   |               ├── UserServiceImpl.java
|   |   |               └── UserService.java
|   |   └── resources
|   |       ├── application.yml
|   |       └── db
|   |           └── migration
|   |               └── V1.0.0__Base_DDL.sql
|   └── test
|       └── java
|           └── ch
|               └── csbe
|                   └── productmanagment
|                       └── ProductManagmentApplicationTests.java
├── structure.txt
└── target
    ├── classes
    |   ├── application.yml
    |   ├── ch
    |   |   └── csbe
    |   |       └── productmanagment
    |   |           ├── controller
    |   |           |   ├── AuthController.class
    |   |           |   ├── CategoryController.class
    |   |           |   ├── ProductController.class
    |   |           |   └── UserController.class
    |   |           ├── dto
    |   |           |   ├── AuthRequest.class
    |   |           |   └── AuthResponse.class
    |   |           ├── model
    |   |           |   ├── Category.class
    |   |           |   ├── Product.class
    |   |           |   └── User.class
    |   |           ├── ProductManagmentApplication.class
    |   |           ├── repository
    |   |           |   ├── CategoryRepository.class
```

```
|   |       |   ├── ProductRepository.class
|   |       |   └── UserRepository.class
|   |       ├── security
|   |       |   ├── AuthService.class
|   |       |   ├── JwtAuthFilter.class
|   |       |   ├── JwtService.class
|   |       |   └── SecurityConfig.class
|   |       └── service
|   |           ├── CategoryService.class
|   |           ├── CategoryServiceImpl.class
|   |           ├── ProductServiceImpl.class
|   |           ├── ProductServiceInterface.class
|   |           ├── UserService.class
|   |           └── UserServiceImpl.class
|   └── db
|       └── migration
|           └── V1.0.0__Base_DDL.sql
└── generated-sources
    └── annotations

39 directories, 58 files
```

6 Beschreibung der Endpoints

6.1 Authentifizierung (/auth)

6.1.1 Anmelden

Table 6 Anmelde Maske

Pfad	POST: /auth/login
Zweck	Benutzer authentifizieren und JWT-Token generieren
Parameter	Body: AuthResponse (Username, Password)
Antwort	Benutzer-Object
http-Status	200 (Erfolg), 401 (Unauthorized / fehlende oder falsche Login Daten)

6.1.2 Registrieren

Table 7 Registrier Maske

Pfad	POST: /auth/register
Zweck	Neues Benutzerkonto erstellen
Parameter	Body: AuthRequest (Username, Password)
Antwort	Benutzer-Object
http-Status	200 (Erfolg)

6.2 Benutzerverwaltung (/users)

6.2.1 Benutzer erstellen

Table 8 Benutzer Erstellen Maske

Pfad	POST: /users
Zweck	Neuen Benutzer anlegen
Parameter	Body: Benutzer-Object
Antwort	Erstellter Benutzer
http-Status	200 (Erfolg)

6.2.2 Benutzer abrufen

Table 9 Benutzer abrufen Maske

Pfad	GET: /users/{id}
Zweck	Benutzer anhand von ID abrufen
Parameter	Pfad: id (Long)
Antwort	Benutzer-Object
http-Status	200 (Erfolg), 404 (Nicht gefunden)

6.2.3 Alle Benutzer abrufen

Table 10 Alle Benutzer abrufen Maske

Pfad	GET: /users
Zweck	Alle Benutzer auflisten
Parameter	Pfad: id (Long), Body: Benutzer-Object
Antwort	Liste von Benutzern
http-Status	200 (Erfolg)

6.2.4 Benutzer aktualisieren

Table 11 Benutzer aktualisieren Maske

Pfad	PUT: /users/{id}
Zweck	Bestehende Benutzer aktualisieren
Parameter	Pfad: id (Long), Body: Benutzer-Object
Antwort	Aktualisierter Benutzer
http-Status	200 (Erfolg)

6.2.5 Benutzer löschen

Table 12 Benutzer löschen Maske

Pfad	DELETE: /users/{id}
Zweck	Benutzer entfernen
Parameter	Id (Long)
Antwort	Keine
http-Status	204 (Kein Inhalt)

6.2.6 Benutzer befördern

Table 13 Benutzer befördern Maske

Pfad	PUT: /users/promote/{id}
Zweck	Benutzer zum Admin befördern
Parameter	Pfad: id (Long)
Antwort	Keine
http-Status	204 (Kein Inhalt)

6.3 Produktverwaltung (/products)

6.3.1 Produkt erstellen

Table 14 Produkt erstellen Maske

Pfad	POST: /products
Zweck	Neues Produkt erstellen
Parameter	Produkt-Object (Name, Beschreibung, Preis, Kategorie)
Antwort	Erstelltes Produkt
http-Status	200 (Erfolg)

6.3.2 Produkt abrufen

Table 15 Produkt abrufen Maske

Pfad	GET: /products/{id}
Zweck	Produkt anhand von ID abrufen
Parameter	Pfad: id (Long)
Antwort	Produkt Object
http-Status	200 (Erfolg), 404 (Nicht gefunden)

6.3.3 Alle Produkte abrufen

Table 16 Alle Produkte abrufen Maske

Pfad	GET: /products
Zweck	Alle Produkte abzurufen
Parameter	Keine
Antwort	Liste von allen Produkten
http-Status	200 (Erfolg)

6.3.4 Produkt aktualisieren

Table 17 Produkte aktualisieren Maske

Pfad	PUT: /products/{id}
Zweck	Bestehendes Produkt aktualisieren
Parameter	Pfad: Id (Long), Body: Produkt-Object
Antwort	Aktualisiertes Produkt
http-Status	200 (Erfolg)

6.3.5 Produkt löschen

Table 18 Produkte löschen Maske

Pfad	DELETE: /products/{id}
Zweck	Produkt entfernen
Parameter	Pfad: id (Long)
Antwort	Keine
http-Status	204 (Kein Inhalt)

6.4 Kategorienverwaltung (/categories)

6.4.1 Kategorie erstellen

Table 19 Kategorien erstellen Maske

Pfad	POST: /categories
Zweck	Neue Kategorie hinzufügen
Parameter	Body: Kategorie-Objekt
Antwort	Erstellte Kategorie
http-Status	200 (Erfolg)

6.4.2 Kategorie abrufen

Table 20 Kategorie abrufen Maske

Pfad	GET: /categories/{id}
Zweck	Kategorien anhand von ID auflisten
Parameter	Keine
Antwort	Kategorie-Object
http-Status	200 (Erfolg), 404 (Nicht gefunden)

6.4.3 Alle Kategorien abrufen

Table 21 Alle Kategorien abrufen Maske

Pfad	GET /categories
Zweck	Alle Kategorien auflisten
Parameter	Keine
Antwort	Liste von Kategorien
http-Status	200 (Erfolg), 404 (Nicht gefunden)

6.4.4 Kategorie aktualisieren

Table 22 Kategorie aktualisieren Maske

Pfad	PUT: /categories/{id}
Zweck	Bestehende Kategorie aktualisieren
Parameter	Pfad: id (Long), Body: Kategorie-Object
Antwort	Aktualisierte Kategorie
http-Status	200 (Erfolg)

6.4.5 Kategorie löschen

Table 23 Kategorie löschen Maske

Pfad	DELETE: /categories/{id}
Zweck	Kategorie entfernen
Parameter	Pfad: id (Long)
Antwort	Keine
http-Status	204 (Kein Inhalt)

6.5 Testing von Endpunkten

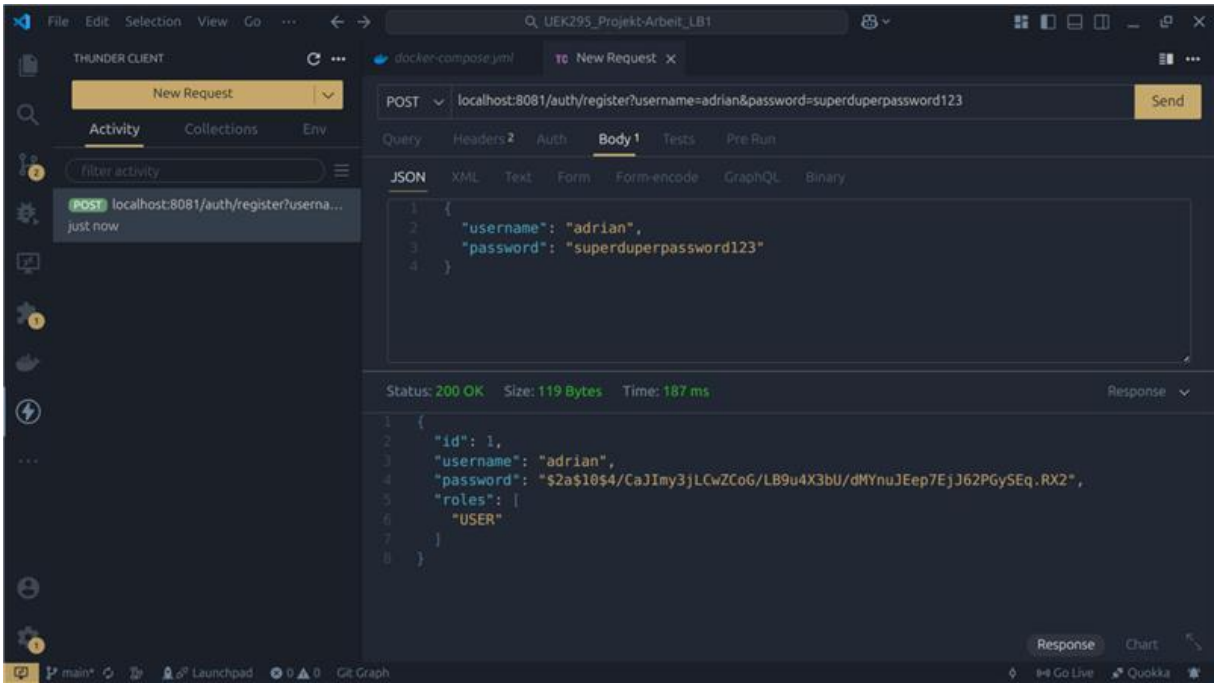

Abbildung 2 Register Test

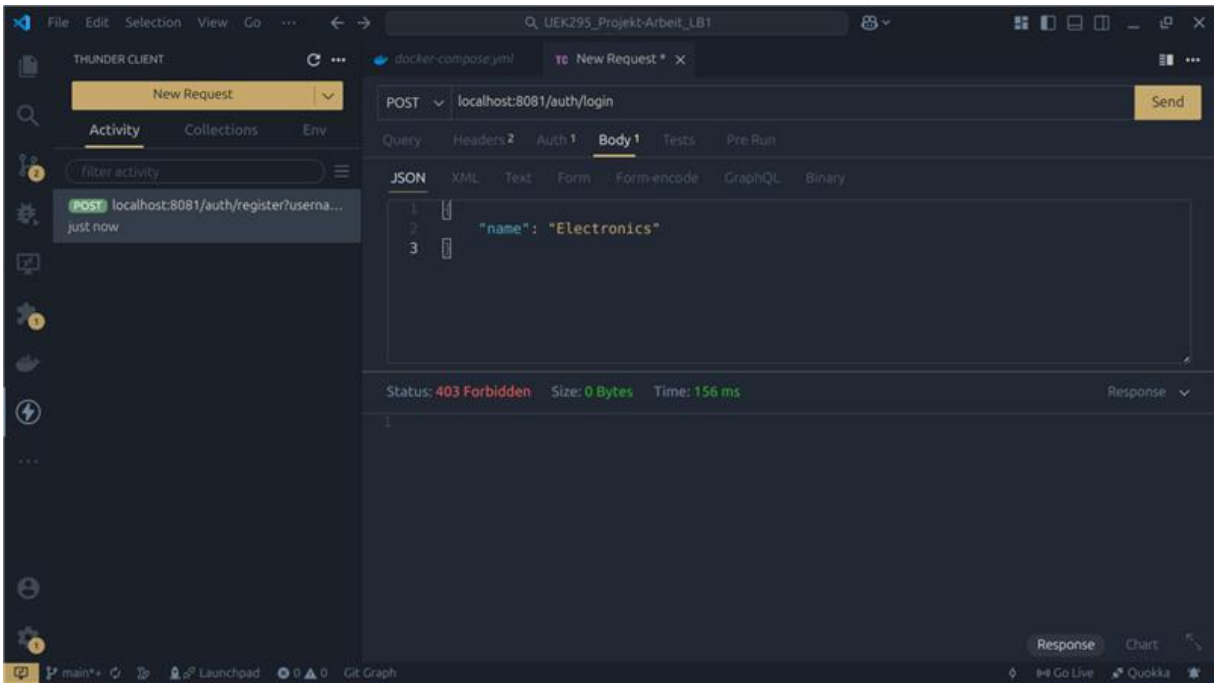

Abbildung 3 Login Test - False Data on Purpose

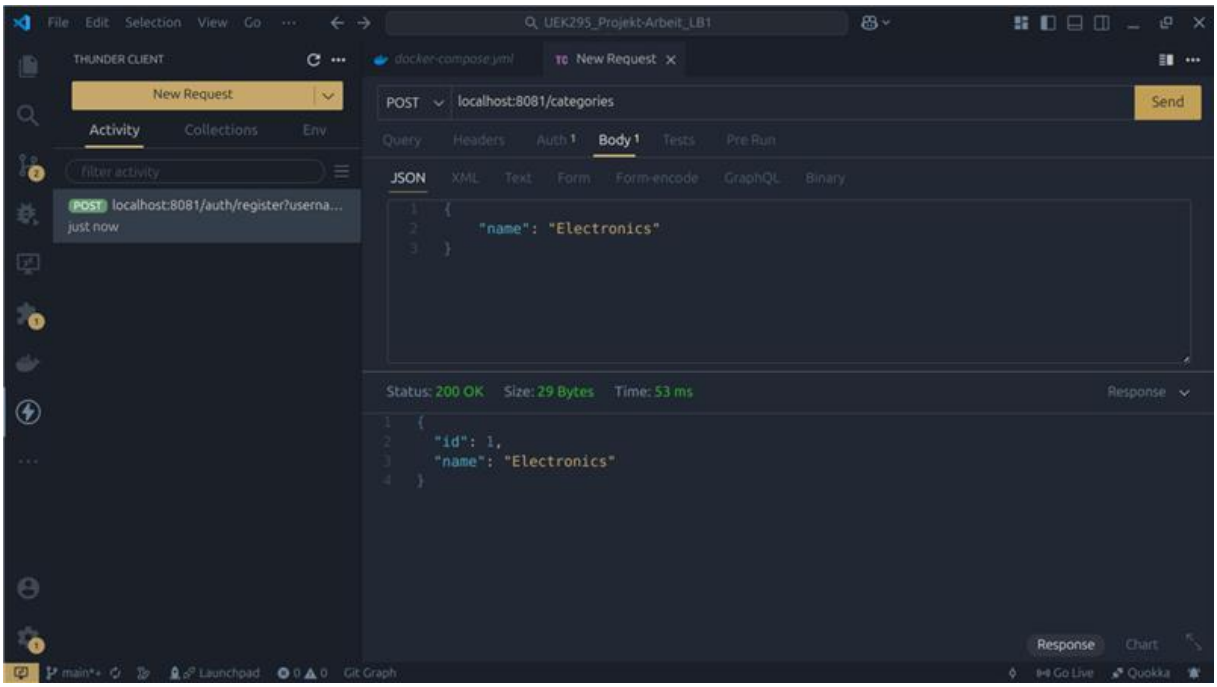

Abbildung 4 Categories Test with Bearer Token (Success)

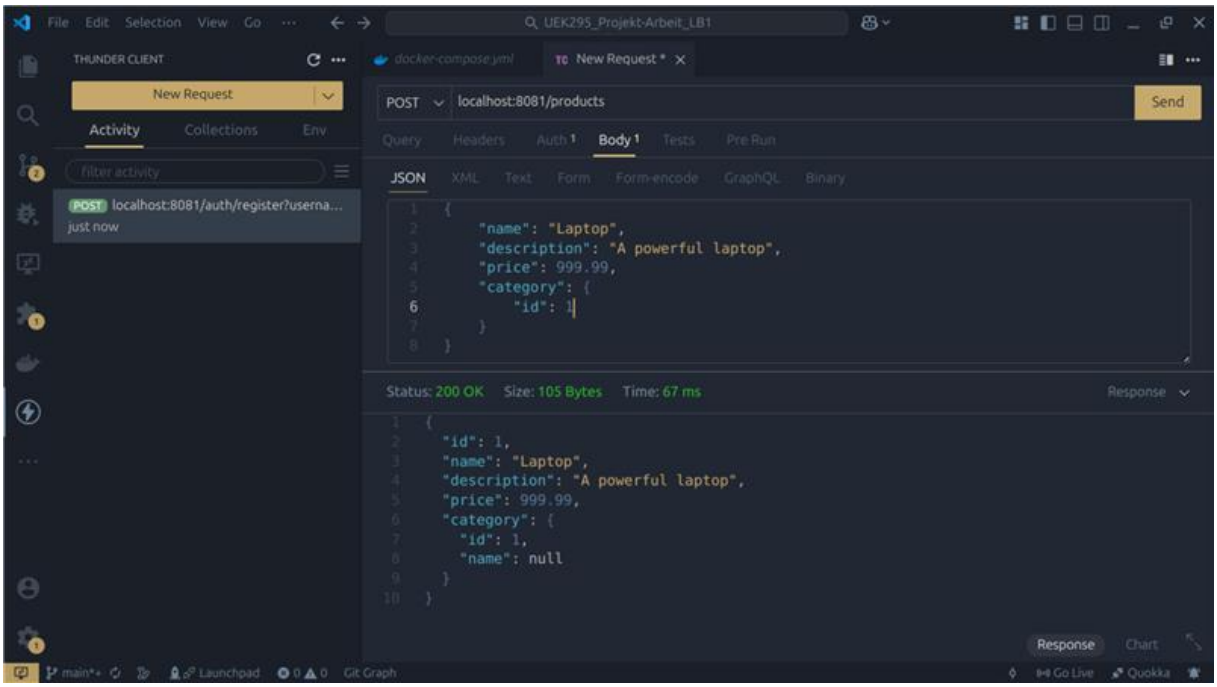

Abbildung 5 Creating a Product with the given Category (chosen by ID)

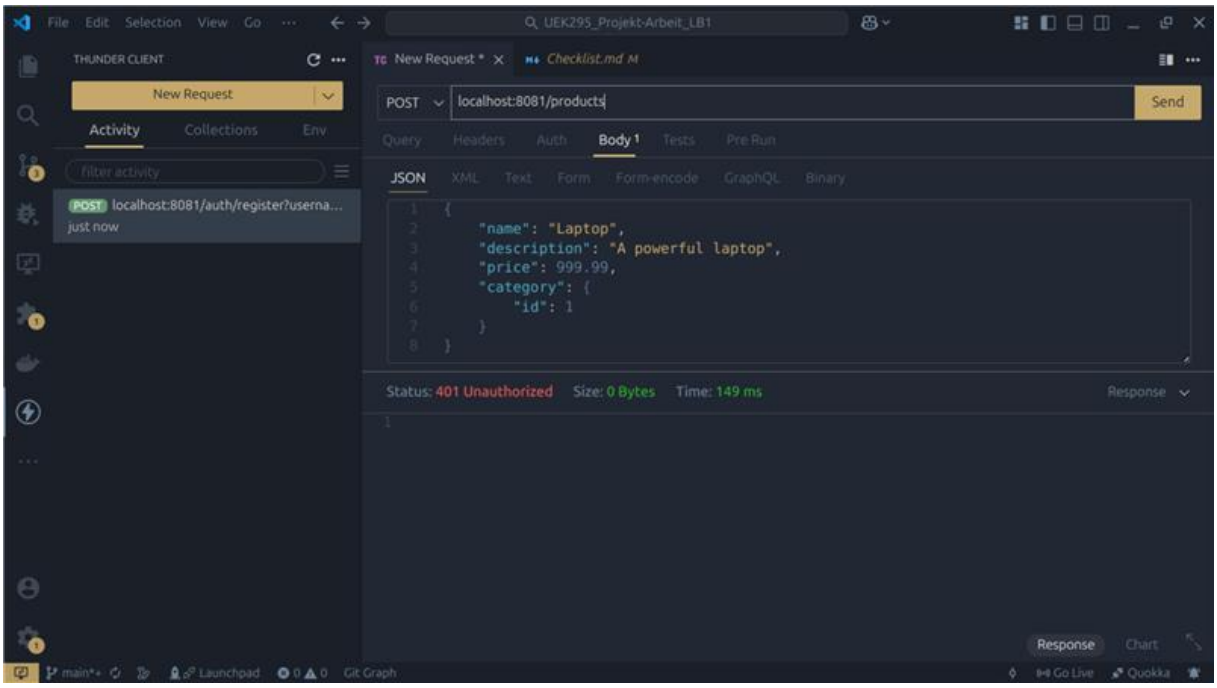

Abbildung 6 Unauthorized 401 weil ich denn Token entfernt habe

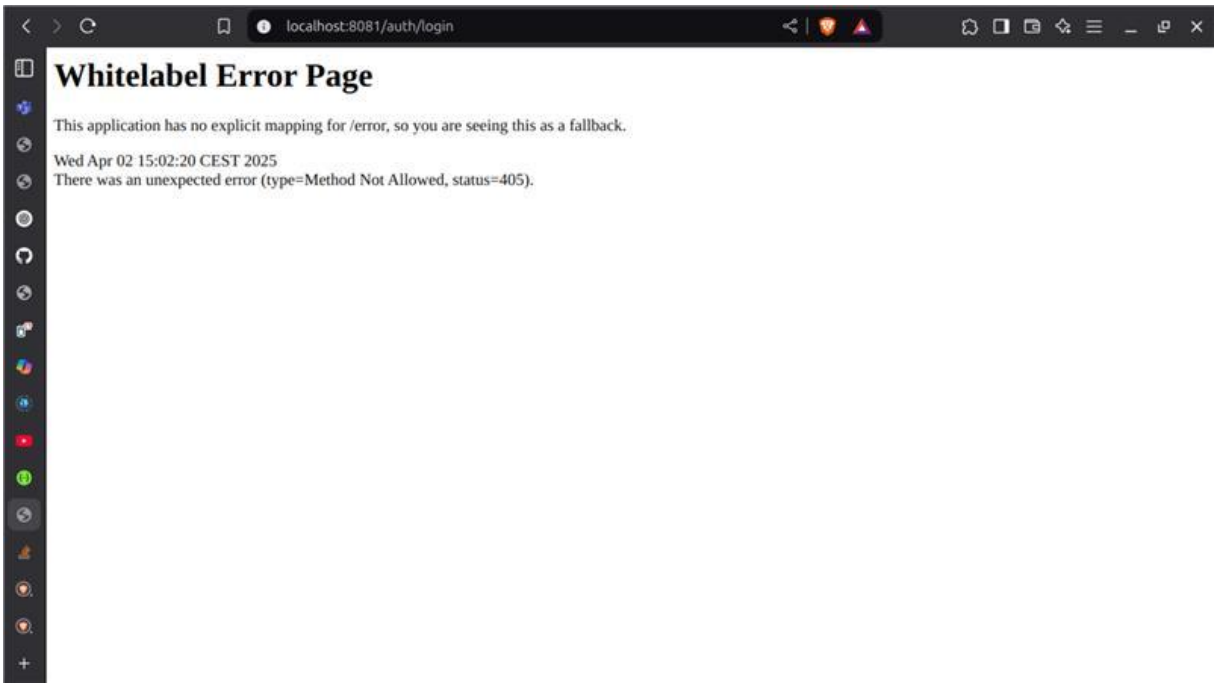

Abbildung 7 Wenn man versucht die Backend Application im Frontend zu accessen, dies wollte ich mal so testen

Dies sind Tests die ich auf die Endpoints getan habe mihilfe von Thunder Client und ich habe explizit die Status Codes getester, dieses 405 ist eine Ausnahme auf der ich zugestossen bin.