TECHNISCHE HOCHSCHULE LÜBECK

BRIDGE BUILDER

Blend Salihu

Web- und Cloud Computing-Projekt
2023

Inhaltsverzeichnis

Spielkonzept

In diesem Spiel muss eine Brücke mittels Mausklicken über einer Klippe gebaut werden. Mit jedem Level erhöht sich der Schwierigkeitsgrad, indem die Zeit hierfür vermindert wird. Um das Spiel zu spielen, muss sich der Spieler anfangs mit einem Usernamen und Passwort registrieren. Sollte der Spieler in einem Level scheitern, so kann dieser sich einloggen und ab dem zuletzt gespieltem Level weiterspielen oder seinen/ihren Account löschen.

Frontend

Für das Frontend wurde HTML5 für die Benutzeroberfläche und CSS3 für die Animationen, sowie als Clientseitige Sprache Dart verwendet.

Für die Anbindung an das Backend wird eine REST-API genutzt.

Es wurde eine asynchrone Programmierung in Dart mit Hilfe von Streams umgesetzt.

Model

Das Model beinhaltet die Spiellogik von Bridge Builder. Diese ist unabhängig von der View und dem Controller. Über das Model können die Daten gespeichert, aktualisiert und verabeitet werden.

Beispielsweise befinden sich in der User.dart Funktionen, die die Kommunikation zwischen dem Controller und der REST-API ermöglicht. Wenn eine Formulareingabe über die View erzeugt wurde, wird dies durch den Controller registriert, welcher die Funktionen von User.dart nutzt, um die Eingaben zu verändern.

View

Die View stellt die Daten aus dem Model dar in einer Benutzerfreundlichen Art als Benuutzeroberfläche dar.

Controller

Der Controller nimmt Interaktionen zwischen dem Model und der View entgegen und leitet diese weiter an die jeweils andere Instanz.

Beispielsweise wird über den Controller eine HTTP-Anfrage an den Server geschickt, sobald eine Formulareingabe bestätigt wird.

Backend

Als Webframework wurde Flask in Python verwendet. Durch die Erweiterung Flask-User wird eine in die Webanwendung integrierte Benutzerverwaltung ermöglicht.

Das RMM-Level 2 wird durch die REST-API erreicht.

Als Orchestrierungsplattform wurde Kubernetes verwendet.

Continuous Deployment

Dadurch, dass jede stabile Änderung in Produktion deployed wird, finden die Qualitätstests in Produktion statt. Manuell wurde die REST-API mit curl getestet.

Exception Handling

Beim Registrieren, Einloggen und Löschen des Accounts werden Meldungen angezeigt. Je nach Fall wird angezeigt, dass die Handlung entweder erfolgreich war oder nicht, dass ein Feld nicht ausgefüllt wurde oder ob der Username bereits vergeben ist.

Die Meldungen sehen wie folgt aus:

- Registration successful
- Failed to register user
- Username and password are required
- Login successful
- Failed to log in
- Account deleted successfully
- Failed to delete account
- Username and password are required for account deletion
- Username already exists

HTTP-Requests

Folgende HTTP-Requests wurden für die Entwicklung von Bridge Builder verwendet:

- POST → Registrierung, Log in

  In Abbildung 1 ist der HTTP-Request POST für das Registrieren eines neuen Users dargestellt.

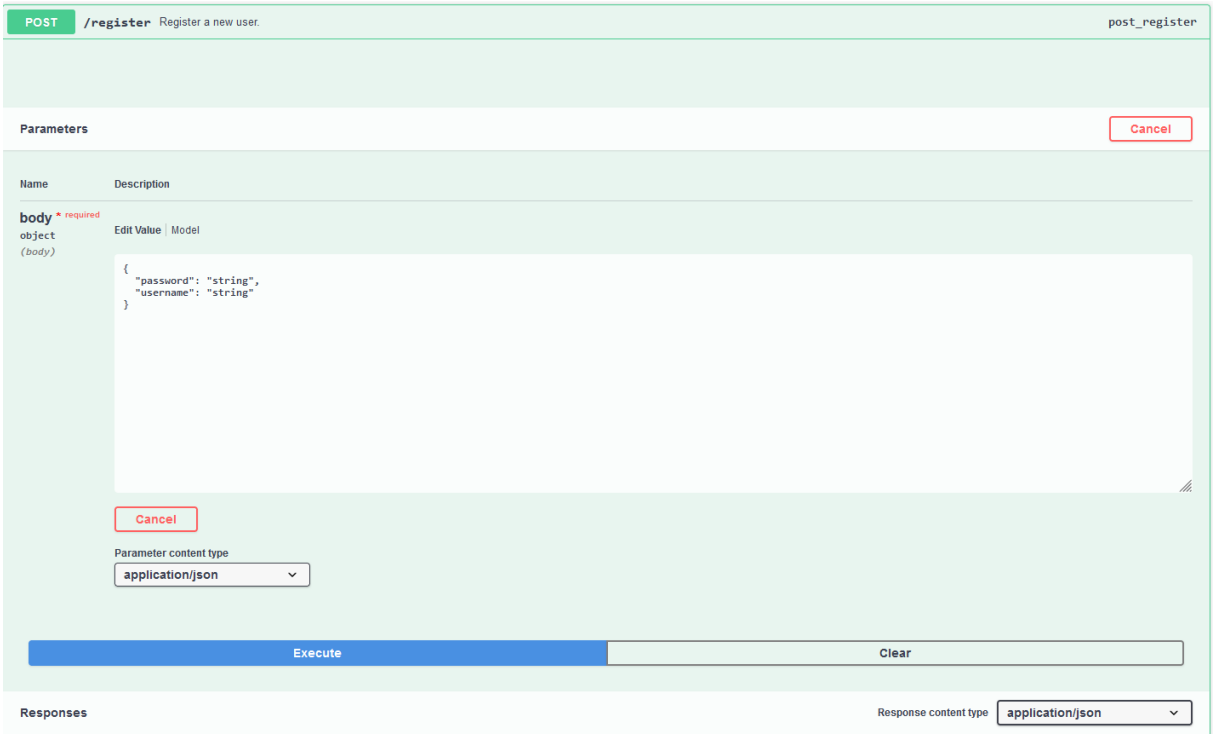

Abbildung 1: Swagger-UI - Eingabe des POST-Requests für das Registrieren eines Users

In Abbildung 2 ist die Ausgabe vom HTTP-Request POST dargestellt. Mit Hilfe der curl-Anfrage werden die Daten mit HTTP übertragen. Der HTTP-Statuscode 201 gibt dann die Meldung ‚User created successfully' aus, sollte die Erstellung erfolgreich gewesen sein. Der HTTP-Statuscode 400 gibt die Meldung ‚Bad request, missing username or password' aus, sollte Pflichtparameter fehlen. Der HTTP-Statuscode 500 gibt die Meldung ‚Internal Server Error' aus, sollte der Server einen internen Fehler haben.

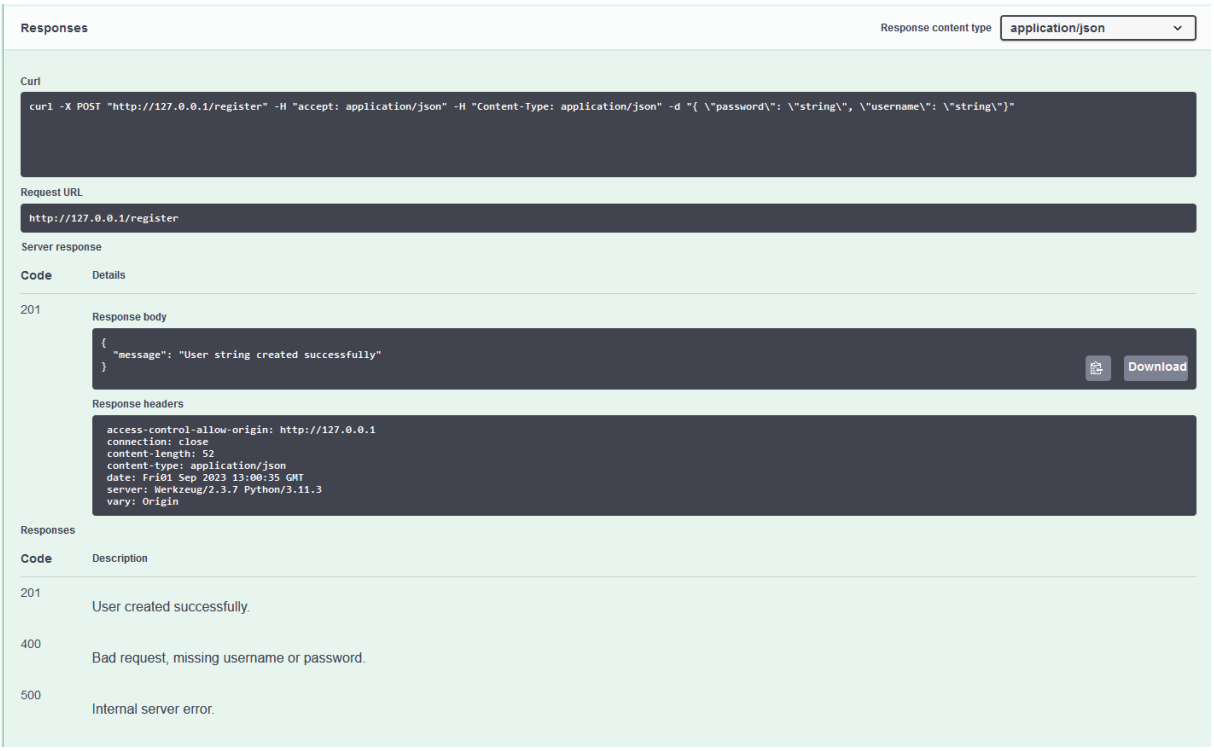

Abbildung 2: Swagger-UI - Ausgabe vom POST-Request für das Registrieren eines Users

In Abbildung 3 ist der HTTP-Request POST für das Einloggen eines Users dargestellt.

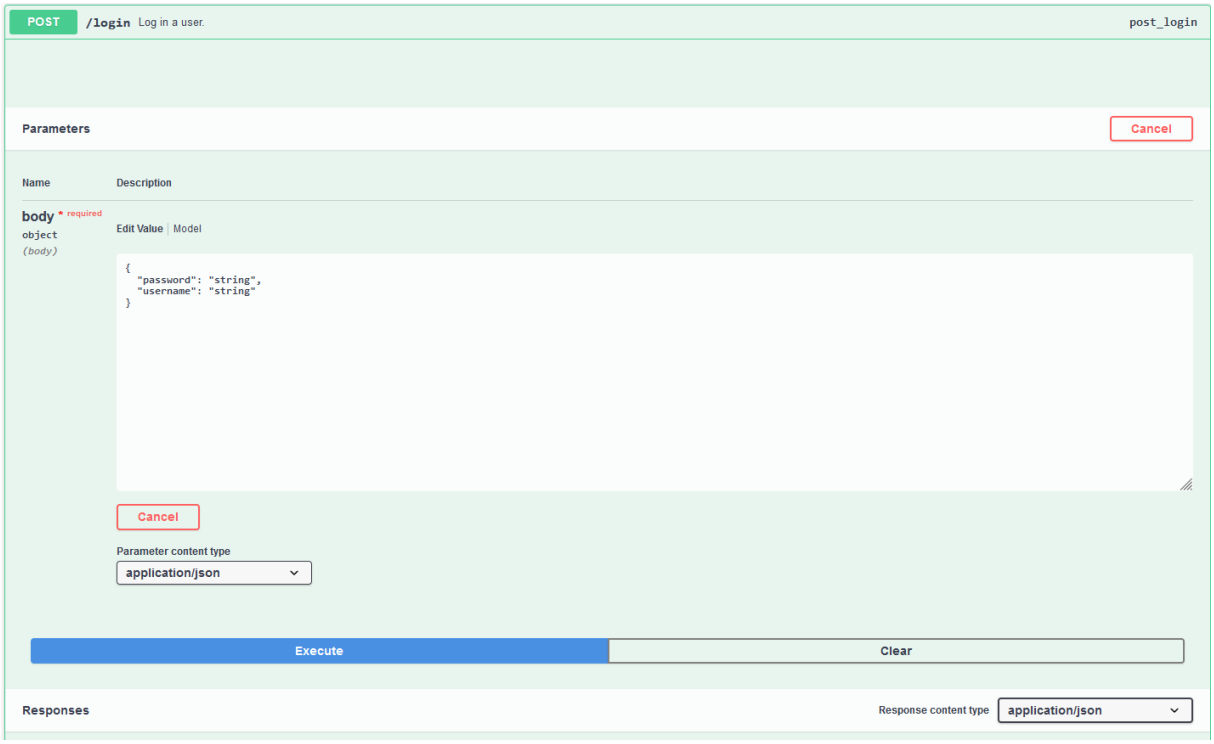

Abbildung 3: Swagger-UI - Eingabe des POST-Requests für das Einloggen eines Users

In Abbildung 4 ist die Ausgabe vom HTTP-Request POST dargestellt. Mit Hilfe der curl-Anfrage werden die Daten mit HTTP übertragen. Der HTTP-Statuscode 200 gibt dann die Meldung ‚User logged in successfully' aus, sollte die Anfrage erfolgreich gewesen sein. Der HTTP-Statuscode 401 gibt die Meldung ‚Unauthorized, invalid username or password' aus, sollte die Person nicht authorisiert sein, die Aktion auszuführen bzw. darauf zuzugreifen. Der HTTP-Statuscode 500 gibt die Meldung ‚Internal Server Error' aus, sollte der Server einen internen Fehler haben.

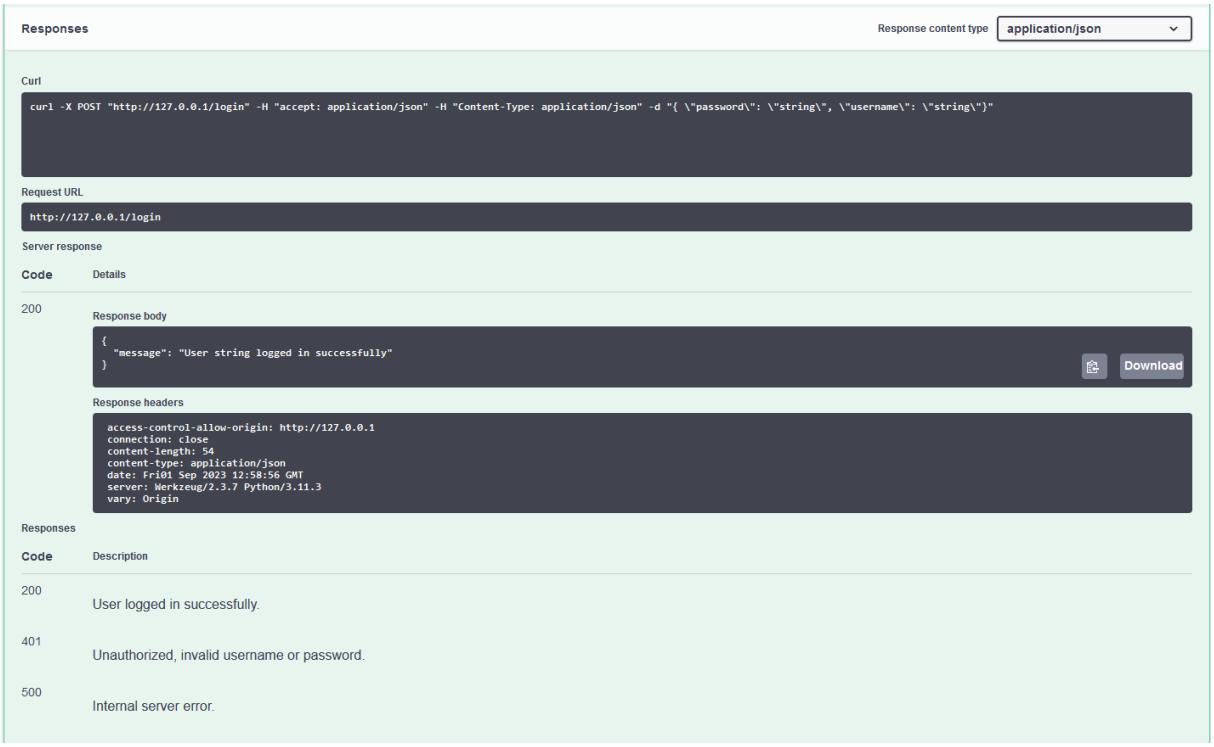

Abbildung 4: Swagger-UI- Ausgabe vom POST-Request für das Einloggen eines Users

- PUT → Update von Passwort und Username

In Abbildung 5 ist der HTTP-Request PUT für das Updaten eines Usernames dargestellt.

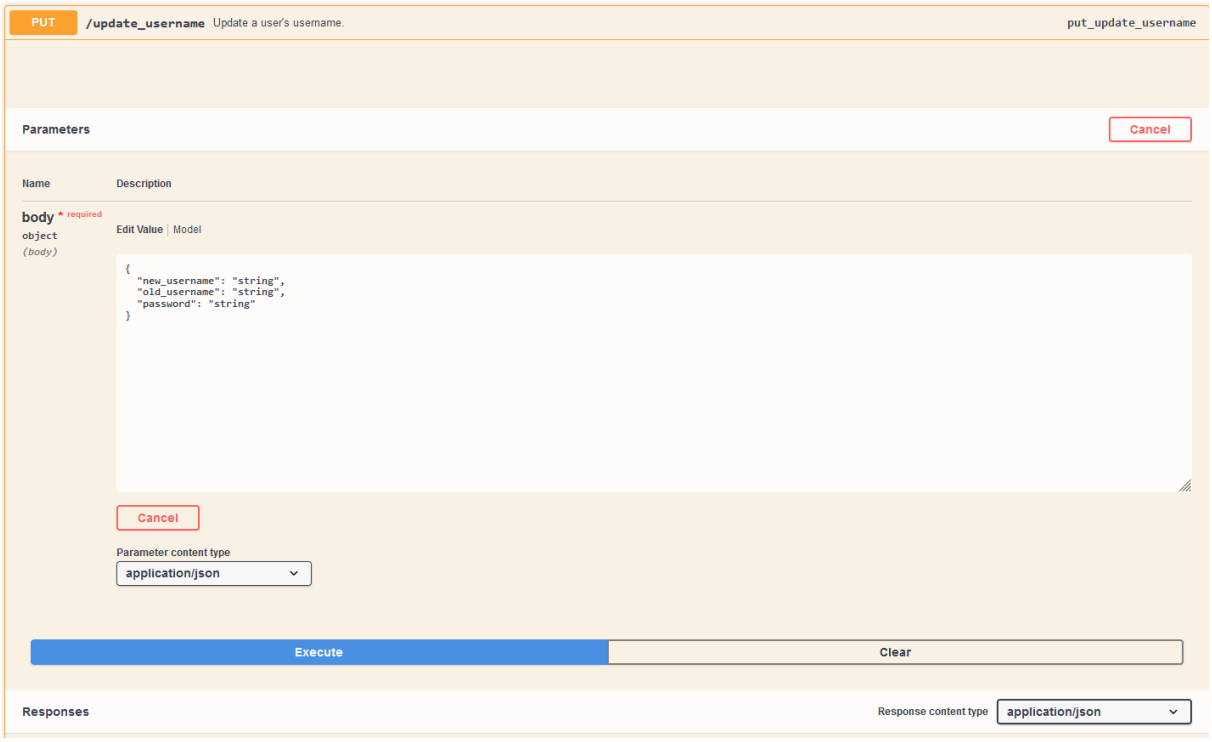

Abbildung 5: Swagger-UI - Eingabe des PUT-Requests für das Updaten eines Usernames

In Abbildung 6 ist die Ausgabe vom HTTP-Request PUT dargestellt. Mit Hilfe der curl-Anfrage werden die Daten mit HTTP übertragen. Der HTTP-Statuscode 200 gibt dann die Meldung ‚Username updated successfully' aus, sollte die Anfrage erfolgreich gewesen sein. Der HTTP-Statuscode 500 gibt die Meldung ‚Internal Server Error' aus, sollte der Server einen internen Fehler haben.

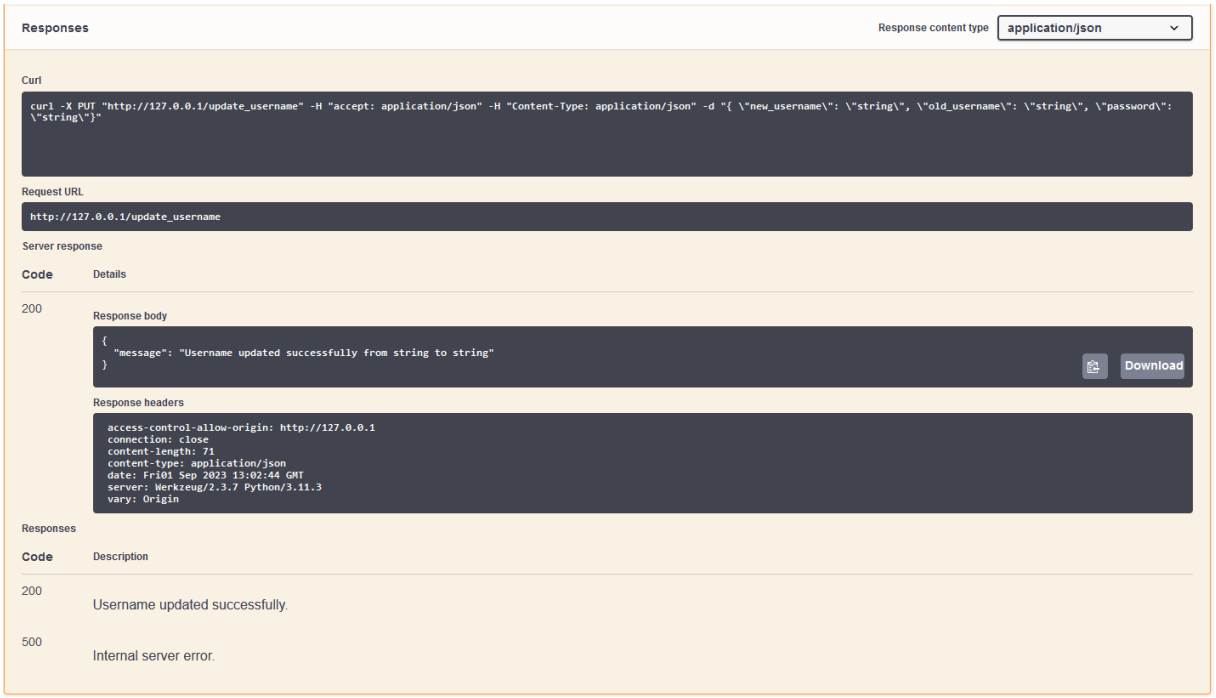

Abbildung 6: Swagger-UI - Ausgabe vom PUT-Request für das Updaten eines Usernames

In Abbildung 7 ist der HTTP-Request PUT für das Updaten eines Userpasswortes dargestellt.

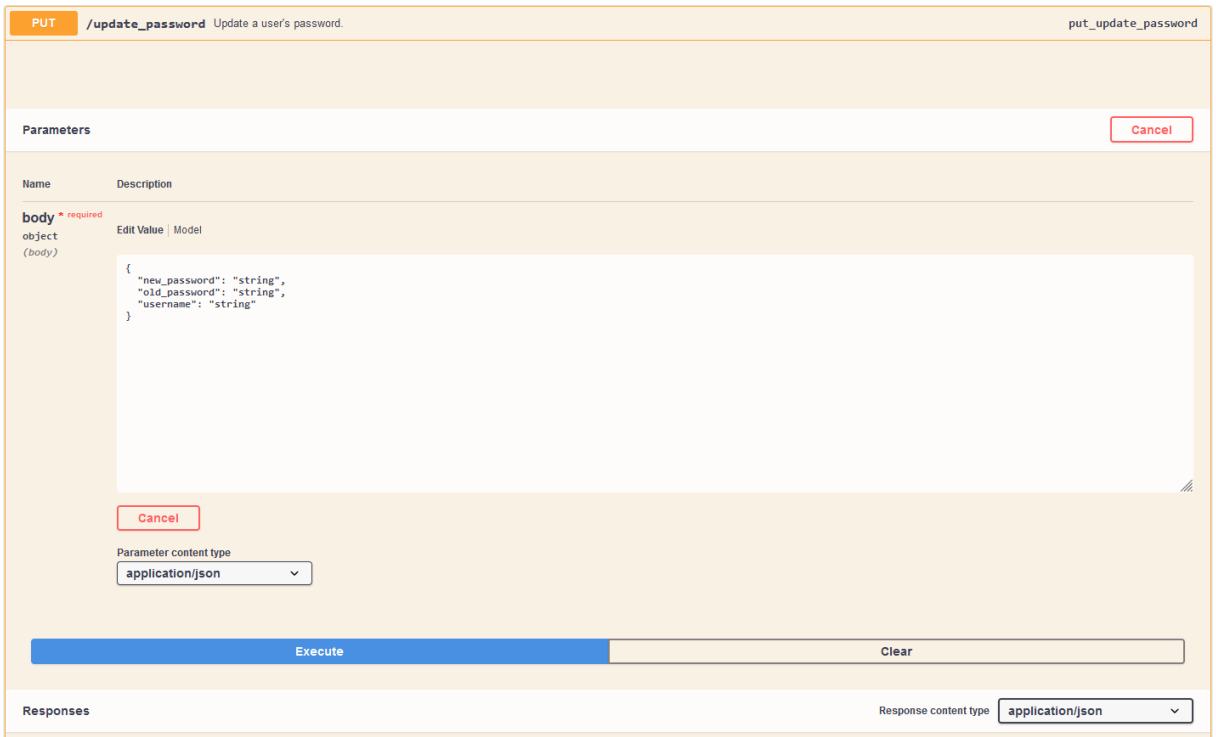

Abbildung 7: Swagger-UI - Eingabe des PUT-Requests für das Updaten eines Userpasswortes

In Abbildung 8 ist die Ausgabe vom HTTP-Request PUT dargestellt. Mit Hilfe der curl-Anfrage werden die Daten mit HTTP übertragen. Der HTTP-Statuscode 200 gibt dann die Meldung ‚Password updated successfully' aus, sollte die Anfrage erfolgreich gewesen sein. Der HTTP-Statuscode 500 gibt die Meldung ‚Internal Server Error' aus, sollte der Server einen internen Fehler haben.

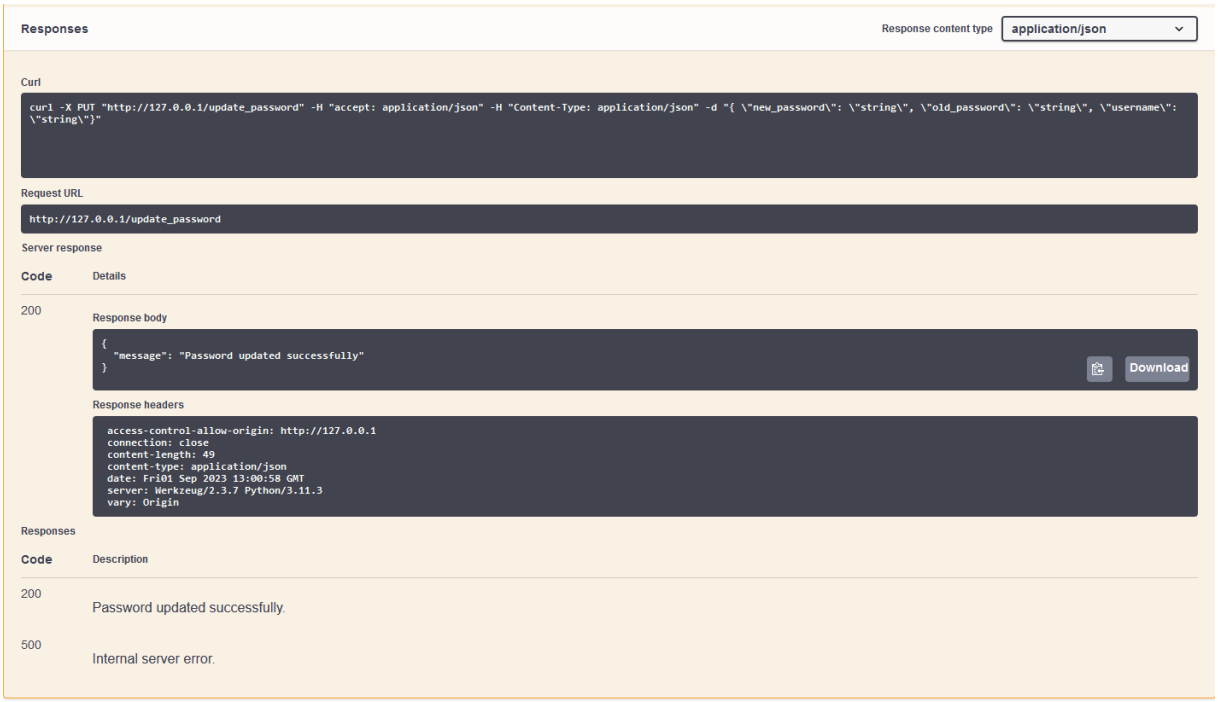

Abbildung 8: Swagger-UI - Ausgabe vom PUT-Request für das Updaten eines Userpasswortes

- DELETE → Löschen eines Accounts

In Abbildung 9 ist der HTTP-Request DELETE für das Löschen eines Users dargestellt.

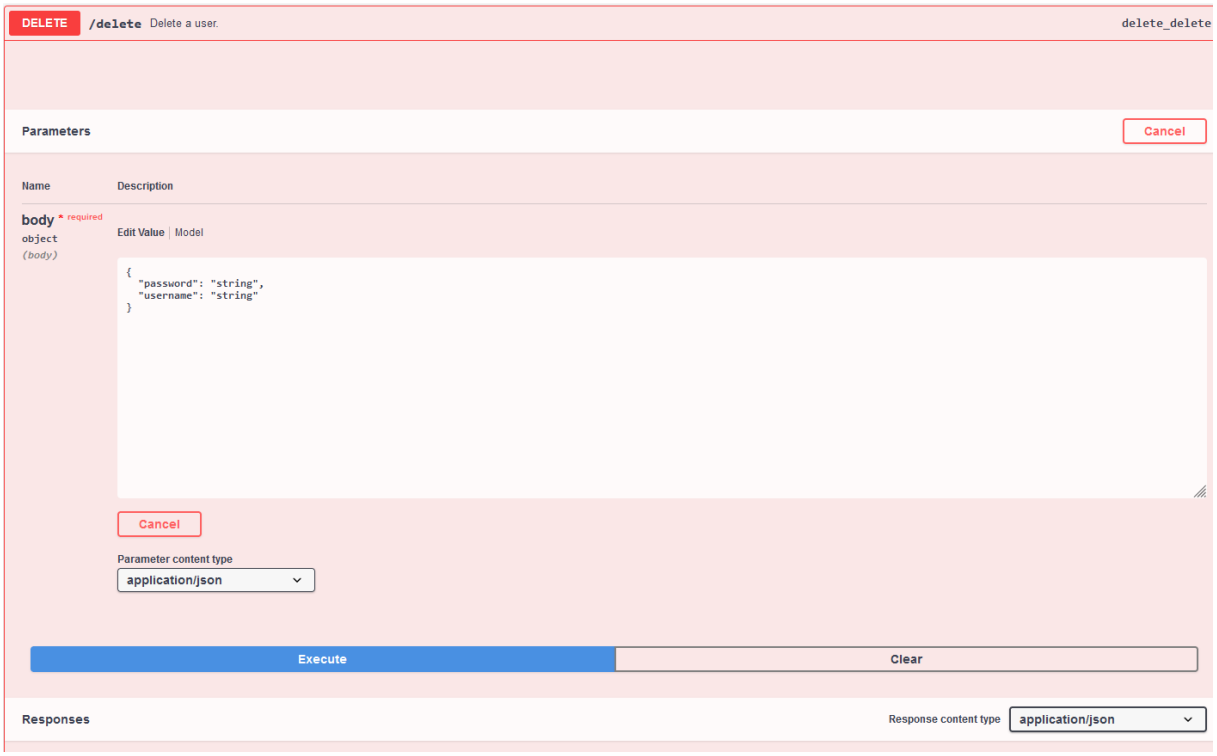

Abbildung 9: Swagger-UI - Eingabe des DELETE-Requests für das Löschen eines Users

In Abbildung 10 ist die Ausgabe vom HTTP-Request DELETE dargestellt. Mit Hilfe der curl-Anfrage werden die Daten mit HTTP übertragen. Der HTTP-Statuscode 200 gibt dann die Meldung ‚User deleted successfully' aus, sollte die Anfrage erfolgreich gewesen sein. Der HTTP-Statuscode 401 gibt die Meldung ‚Unauthorized, invalid username or password' aus, sollte die Person nicht authorisiert sein, die Aktion auszuführen bzw. darauf zuzugreifen. Der HTTP-Statuscode 500 gibt die Meldung ‚Internal Server Error' aus, sollte der Server einen internen Fehler haben.

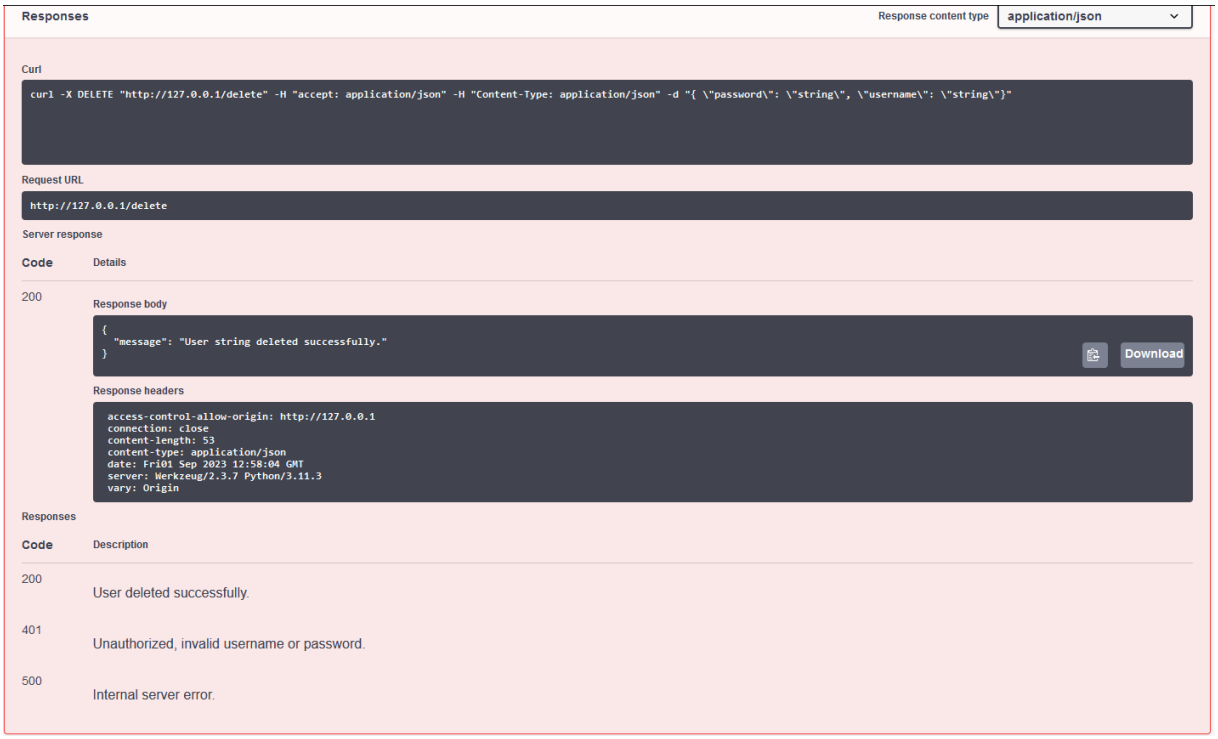

Abbildung 10: Swagger-UI - Ausgabe vom DELETE-Request für das Löschen eines Users

- GET → Für das Rendern der HTML

Pipeline

Für die Pipeline wird die Sqlite.yaml und die Sqlite-volume.yaml deployed. Zudem wurde in der .gitlab-ci.yml Ingress auf true gesetzt.

Kommunikation zwischen Dart und dem Backend

Die Kommunikation zwischen Dart und dem Backend erfolgt über das Cross-Origin Resource Sharing (CORS), der Webclients Cross-Origin-Requests ermöglicht.

Datenbank

Als Datenbank wird SQLite verwendet. Diese ist eine serverlose, leichtgewichtige, relationale Datenbank und somit für das Spielkonzept geeignet.

Schwierigkeiten

In meiner Präsentation habe ich angegeben, dass ich als Clientseitige Programmiersprache JavaScript verwenden werde, jedoch wurde mir schnell klar, dass die Einarbeitung in die mir neue Programmiersprache nicht so einfach als gedacht sein würde, weshalb ich beschloss von JavaScript auf Dart zu wechseln, da diese Programmiersprache zum einen in der Vorlesung als auch im Praktikum mit vielen Beispielen und Erläuterungen vermittelt wurde, und zum anderen sehr nahe an Java angelehnt ist, sodass es kaum eine Umstellung für mich darstellte.

Anfangs habe ich Flask-User verwendet, jedoch habe ich diese Funktion mit der Zeit vergessen und habe eigene Funktionen zur Benutzerverwaltung verwendet.

Konzeptionelles Modell als UML-Klassendiagramm

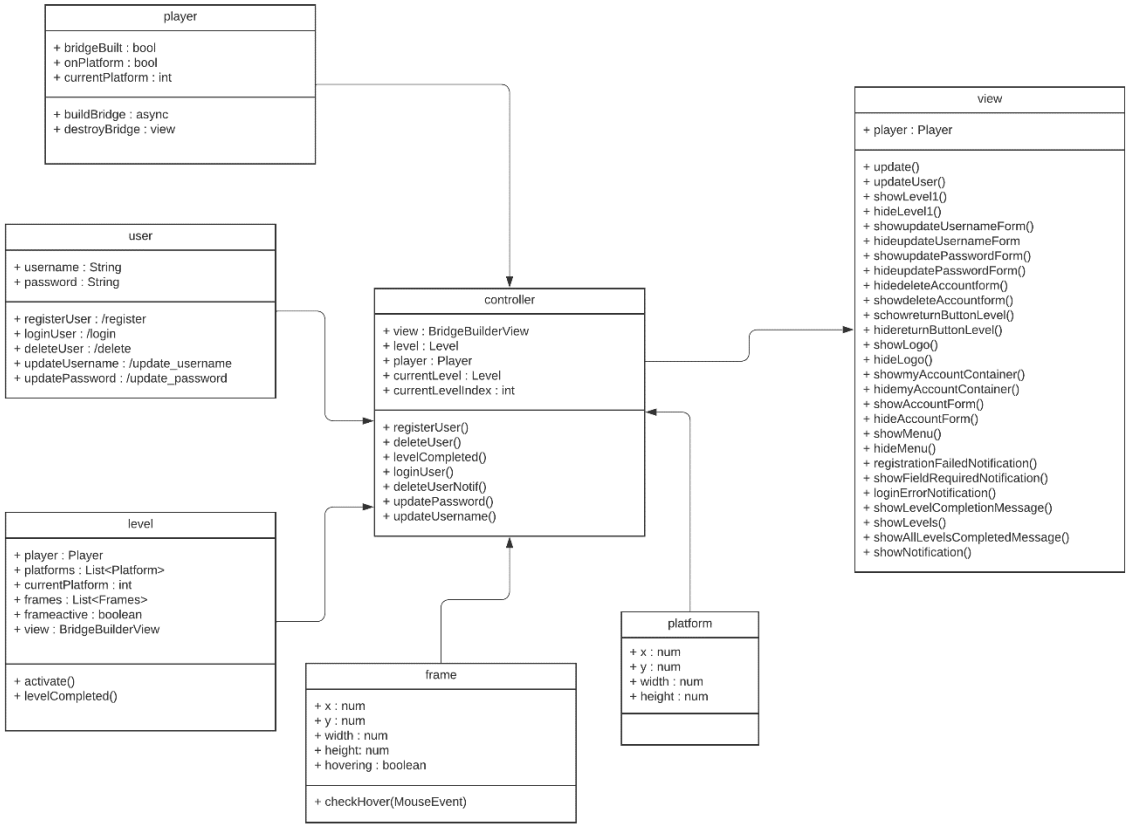

Abbildung 11: Konzeptionelles Modell als UML-Klassendiagramm

Der Controller hatte die Verantwortung für Benutzerinteraktionen und das Manipulieren des DOM-Baums mithilfe der View. Außerdem oblag ihm die Bestätigung von Benutzeraktionen, wie beispielsweise das Auslösen eines Klicks auf Schaltflächen. Dies ermöglichte beispielsweise nach der Registrierung das Senden von Anfragen an die REST-API über eine bereitgestellte Funktion in der user.dart. Die Funktion ermöglichte

das Übermitteln von HTTP-Anfragen an die entsprechende App-Routenadresse. In der player.dart war der Controller für die Spiellogik verantwortlich, einschließlich der Handhabung von Kollisionen und ähnlichen Aspekten. Bei den Leveln waren die platform.dart-Klasse und die frame.dart-Klasse für die Größenanpassungen und die Implementierung von Plattformen zuständig, die vom Spieler überwunden werden konnten.

Anforderungen – The „Backend“ Edition

#	Anforderung	Ausführung
AF-1	Game als Single-Page-App	Bridge Builder ist ein Ein-Spieler-Game und als HTML-Single Page App konzipiert. Es wird als statische Website von beliebigen Webservern (HTTP) und der mir zur Verfügung gestellten Deployment Pipeline automatisiert bereitgestellt.
AF-2	Balance zwischen technischer Komplexität und Spielkonzept	Bridge Builder hat ein interessantes Spielkonzept und eine vergleichbare Komplexität wie die Spiele aus der Hall-of-Fame. Das Spiel schnell und intuitiv erfassbar und angenehm auf dem Target-Device zu spielen.
AF-3	DOM-Tree- und MVC-basiert	Bridge Builder folgt dem MVC-Prinzip und nutzt den DOM-Tree als View.
AF-4	Target Device: Desktop Browser	Bridge Builder ist mit HTML5 Desktop-Browsern spielbar. Dies wurde vor allem auf den Browsern Firefox und Chrome geprüft.
AF-5	Intuitive Spielfreude	Bridge Builder ist schnell und intuitiv erfassbar und erzeugt durch die Schwierigkeitserhöhung in den Leveln Spielfreude.
AF-6	Das Spiel muss ein Stateful Backend vorsehen (z.B. für High Scores, Add-On Level, etc.)	Das Backend von Bridge Builder ist über eine HTTP-basierte REST-Schnittstelle vor einer Standard-Datenbank (SQLite) realisiert wurden. Das REST-Backend wurde auf Basis von Python realisiert. Als Webframework wurde Flask verwendet.

		Die REST-Prinzipien (CRUD-Schema auf POST, GET, PUT, DELETE zu mappen) wurden eingehalten und ein RMM-Level von 2 wurde angestrebt und erreicht. Das Stateful Backend ist mit einem geeigneten Konzept (Ingress) in containerisierter Form im Kubernetes Cluster automatisiert mittels der Deployment Pipeline deployed wurden und passt sich sinnvoll in dem Spielkonzept ein. Dem/Der Spieler/in wird die Möglichkeit eingeräumt, sich geeignet zu authentifizieren und seinen/ihren Account zu löschen.
AF-7	Client-seitige Sprache für das Frontend	Als Client-seitige Sprache wurde Dart verwendet und das Frontend mit einem geeigneten Konzept (Ingress), welches den Zugriff auf Dienste innerhalb eines Kubernetes-Clusters über eine externe URL (https://webapp-10260.edu.k8s.th-luebeck.dev/) ermöglicht, in containerisierter Form im Kubernetes Cluster automatisiert mittels der Deployment Pipeline deployed worden.
AF-8	Keine Spielereien	In Bridge Builder wurde auf Sound- und Videospielereien verzichtet und die Animationen wurden mittels CSS realisiert.
AF-9	Wiederverwenden von Grafiken oder anderen gestaltenden Elementen	Bestehende Grafiken und andere gestaltende Elemente in Bridge Builder wurden wiederverwendet, da diese lizenzfrei genutzt werden können bzw. einer Lizenz unterliegen, die mindestens die nicht kommerzielle Nutzung erlaubt.
AF-10	Dokumentation	Aus der Dokumentation geht die Gesamtarchitektur hervor. Die Funktionalitäten und Verantwortlichkeiten der Architektur Tiers wurden beschrieben und deren Zusammenspiel erläutert. Aus der Frontend-Dokumentation geht das konzeptionelle Modell des Spiels hervor.