Das Grosse SQL-Spiel

The SQL-Alchemist

Software-Entwicklungspraktikum (SEP)
Sommersemester 2015

Technischer Entwurf

Auftraggeber:
Technische Universität Braunschweig
Institut für Informationssysteme
Prof. Dr. Wolf-Tilo Balke
Mühlenpfordtstraße 23, 2.OG
D-38106 Braunschweig

Betreuer: Jan-Christoph Kalo

Auftragnehmer:

Name	E-Mail-Adresse
Gabriel Ahlers	g.ahlers@tu-braunschweig.de
Majid Dashtiepielehroud	m.dashtiepielehroud@tu-braunschweig.de
Ronja Friebe	r.friebe@tu-braunschweig.de
Stefan Hanisch	stefan.hanisch@tu-braunschweig.de
Fabio Luigi Mazzone	f.mazzone@tu-braunschweig.de
Nicole Naczk	n.naczk@tu-braunschweig.de
Denis Nagel	denis.nagel@tu-braunschweig.de
Luca Porcello	l.porcello@tu-braunschweig.de
Christian Reineke	c.reineke@tu-braunschweig.de
Christian Sander	christian.sander@tu-braunschweig.de
Carl Schiller	c.schiller@tu-braunschweig.de
Levent Muzaffer Üner	l.uener@tu-braunschweig.de
Sören van der Wall	s.van-der-wall@tu-braunschweig.de
Daniel Wolfram	d.wolfram@tu-braunschweig.de

Braunschweig, 1. Juli 2015

Inhaltsverzeichnis

1 Einleitung **7**
1.1 Projektdetails 8

2 Analyse der Produktfunktionen **9**
2.1 Analyse von Funktionalität <F10>: <Nutzer registrieren> 10
2.2 Analyse von Funktionalität <F20>: <Nutzer anmelden> 11
2.3 Analyse von Funktionalität <F30>: <Nutzer abmelden> 12
2.4 Analyse von Funktionalität <F40>: <Profil einsehen> 13
2.5 Analyse von Funktionalität <F60>: <Passwort ändern> 14
2.6 Analyse von Funktionalität <F70>: <Avatar ändern> 15
2.7 Analyse von Funktionalität <F80>: <Benutzer löschen> 16
2.8 Analyse von Funktionalität <F90>: <Audioeinstellungen bearbeiten> 17
2.9 Analyse von Funktionalität <F100>: <Spielstand zurücksetzen> 18
2.10 Analyse von Funktionalität <F110>: <Tutorial spielen> 19
2.11 Analyse von Funktionalität <F120>: <Story spielen> 20
2.12 Analyse von Funktionalität <F130>: <SQL-Trainer spielen> 21
2.13 Analyse von Funktionalität <F140>: <Minispiel spielen> 23
2.14 Analyse von Funktionalität <F150>: <Hausaufgaben bearbeiten> 24
2.15 Analyse von Funktionalität <F160>: <Ranglisten einsehen> 24
2.16 Analyse von Funktionalität <F170>: <Spieler suchen> 25
2.17 Analyse von Funktionalität <F180>: <Hausaufgabenergebnisse einsehen> 26
2.18 Analyse von Funktionalität <F190>: <Benutzer befördern> 27
2.19 Analyse von Funktionalität <F210>: <Eine Trivia-Aufgabe erstellen> 28
2.20 Analyse von Funktionalität <F220>: <Benutzeraufgaben bewerten> 29
2.21 Analyse von Funktionalität <F230>: <Hausaufgaben erstellen> 30

3 Resultierende Softwarearchitektur **31**
3.1 Komponentenspezifikation 31
3.2 Schnittstellenspezifikation 33
3.3 Protokolle für die Benutzung der Komponenten 38

4 Verteilungsentwurf **39**

5 Implementierungsentwurf **40**
5.1 Implementierung von Komponente <C10>: Das Front-End: . . . 40
5.1.1 me.ScreenObject . . . 40
5.1.2 game.AlertScreen . . . 44
5.1.3 game.HighscoreScreen . . . 44
5.1.4 game.DefaultLoadingScreen . . . 45
5.1.5 game.LoginScreen . . . 45
5.1.6 game.SignUpScreen . . . 45
5.1.7 game.SettingsScreen . . . 46
5.1.8 game.ProfileScreen . . . 46
5.1.9 game.TaskScreen . . . 47
5.1.10 game.TitleScreen . . . 47
5.1.11 me.GUI_Object . . . 48
5.1.12 me.Entity . . . 51
5.1.13 game.PlayerEntity . . . 52
5.1.14 game.LevelEntity . . . 53
5.1.15 game.CoinEntity . . . 53
5.1.16 game.SpikeEntity . . . 54
5.1.17 game.ScrollEntity . . . 54
5.1.18 game.EnemyEntity . . . 54
5.1.19 me.Renderable . . . 57
5.2 Implementierung von Komponente <C20>: Das Back-End . . . 59
5.2.1 Model Classes – Profile . . . 60
5.2.2 Paket-/Klassendiagramm . . . 60
5.2.3 Erläuterung . . . 61
5.2.4 Model Classes – Part 1 . . . 64
5.2.5 Paket-/Klassendiagramm . . . 64
5.2.6 Erläuterung . . . 65
5.2.7 Model Classes – Part 2 . . . 71
5.2.8 Paket-/Klassendiagramm . . . 71
5.2.9 Erläuterung . . . 72
5.2.10 Model Classes – Part 3 . . . 78
5.2.11 Paket-/Klassendiagramm . . . 78
5.2.12 Erläuterung . . . 79
5.2.13 Helper and Secured Classes . . . 86
5.2.14 Paket-/Klassendiagramm . . . 86
5.2.15 Erläuterung . . . 87
5.2.16 Controller Classes . . . 99
5.2.17 Paket-/Klassendiagramm . . . 99

5.2.18 Erläuterung 100

6 Datenmodell **109**
6.1 Erläuterung 109

7 Konfiguration **123**

8 Änderungen gegenüber Fachentwurf **124**

9 Erfüllung der Kriterien **125**
9.1 Musskriterien 125
9.2 Sollkriterien 125
9.3 Kannkriterien 126

Abbildungsverzeichnis

1.1 Zustandsdiagramm zum oberflächlichen Spielablauf 7

2.1 Sequenzdiagramm zur Registrierung 10
2.2 Sequenzdiagramm zum Login 11
2.3 Sequenzdiagramm zum Login 12
2.4 Sequenzdiagramm zur Darstellung der Profilübersicht 13
2.5 Sequenzdiagramm zum Ändern des Passworts 14
2.6 Sequenzdiagramm zum Ändern des Avatars 15
2.7 Sequenzdiagramm zum Löschen des Benutzers 16
2.8 Sequenzdiagramm zum Ändern der Audioeinstellungen 17
2.9 Sequenzdiagramm zum Zurücksetzen des Spielstands 18
2.10 Sequenzdiagramm für das Tutorial 19
2.11 Aktivitätsdiagramm für den Story Mode 20
2.12 Sequenzdiagramm für den SQL-Trainer 21
2.13 Sequenzdiagramm für das Minigame 23
2.14 Sequenzdiagramm für die Ranglisten 24
2.15 Sequenzdiagramm zum Suchen eines anderen Users 25
2.16 Sequenzdiagramm zum einsehen der eigenen Hausaufgabenergebnisse 26
2.17 Sequenzdiagramm zum Befördern eines Users 27
2.18 Sequenzdiagramm für das Erstellen von Aufgaben 28
2.19 Sequenzdiagramm zum bewerten User-erstellter Aufgaben. 29

3.1 Komponentendiagramm. 31
3.2 Back End - Statechart 38

4.1 Verteilungsdiagramm für SQL-Alchemst 39

5.1 Klassendiagramm für die verschiedenen Screens 43
5.2 Klassendiagramm für die verschiedenen Screens 50
5.3 Klassendiagramm für die verschiedenen Front-End-Entitäten 56
5.4 Klassendiagramm für die Renderables 58
5.5 Klassendiagramm für die Model-Klasse Profile $\langle C20 \rangle$ 60
5.6 Klassendiagramm für einige der Model-Klassen $\langle C20 \rangle$ 64
5.7 Klassendiagramm für einige der Model-Klassen $\langle C20 \rangle$ 71

5.8 Klassendiagramm für einige der Model-Klassen $\langle C20 \rangle$ 78
5.9 Klassendiagramm für die Helper- und Secured-Klassen $\langle C20 \rangle$ 86
5.10 Klassendiagramm für die Controller-Klassen $\langle C20 \rangle$ 99

6.1 Klassendiagramm zum SQL-Alchemist 122

1 Einleitung

Dieses Dokument dient dazu, einen tiefer gehenden Überblick über den internen Ablauf der zu entwickelnden Software zu bieten. Es werden zuerst die wichtigsten Funktionen einzeln beschrieben, wobei Sequenzdiagramme zu Hilfe genommen werden. Danach wird der Aufbau der Komponenten beschrieben, sowie deren Verteilung gezeigt und am Ende die Datenverwaltung näher beleuchtet.
Ziel ist es, dass ein Entwickler nach dem Lesen dieses Dokuments die Software nachentwickeln könnte.

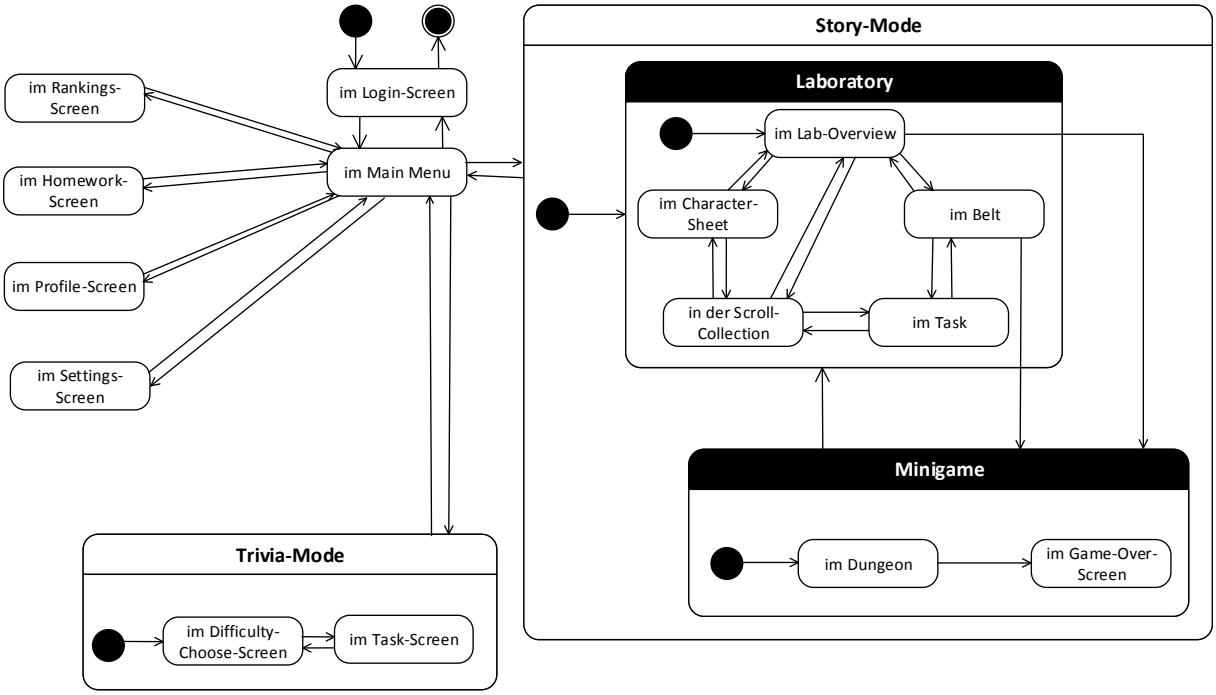

Abbildung 1.1: Zustandsdiagramm zum oberflächlichen Spielablauf

Um eine Verständnisgrundlage zu schaffen ist in Abbildung 1.1 die allgemeine Menüführung in einem Zustandsdiadramm dargestellt.

Nach dem Einloggen gelangt man in das Main Menu. Von dort aus gelangt man in die drei Spielmodi (Story, Trivia, Homework), die Einstellungen, das Profil und in die Ranglisten.

Der Story Mode hat wiederum eine eigene Übersicht, Laboratory genannt. Von hier aus kann der User sich seine aktuellen Playerstatistiken im Charakter Sheet ansehen, kann sich in der

Scrollcollection um die Herstellung von Potions und Enchantments kümmern, diese danach in seinen Belt einfügen um sich im Anschluss dem Minispiel zu widmen. Im Minispiel werden dem User verschiedene Hindernisse vorgesetzt, die es zu überwinden gilt. Scheitert der User, so gelangt er in den Game Over Screen, in dem ihm angezeigt wird wie viele Lofi-Coins (die Spiel-Währung) und welche Scrolls er in dieser Runde eingesammelt hat. Nach Bestätigung befindet sich der User zurück im Laboratory.

Der Trivia-Mode stellt einen SQL-Trainer dar. Entscheidet der User sich dafür diesen zu verwenden, gelangt er in einen Screen, in dem er sich für einen Schwierigkeitsgrad entscheiden muss. Hat der User dies getan, erhält er eine, dem ausgewählten Schwierigkeitsgrad entsprechende, Aufgabe und kann diese lösen.

Der Homework-Mode funktioniert auf die gleiche Weise wie der Trivia-Mode, lediglich die Auswahl des Schwierigkeitsgrads entfällt in diesem Fall. Die Funktion soll den Hausaufgaben-Ablauf der Vorlesung RDB1 unterstützen.

In den Rankings werden die Spieler nach verschiedenen Kriterien in Ranglisten sortiert. Darüber ist es auch möglich, durch die Eingabe des Benutzernamens, nach anderen Spielern zu suchen und sich deren Profil anzeigen zu lassen. Das eigene Profil ist über das Main Menu zu erreichen und einsehbar. Selbiges gilt für die Spieleinstellungen.

1.1 Projektdetails

Die Anzahl an Lofi-Coins, sowie die Anzahl an Scrolls, die der User pro Tag einsammeln kann, werden beschränkt. Dies soll verhindern, dass der User hauptsächlich oder ausschließlich das Minispiel spielt, was dem Zweck der Anwendung, nämlich dem Üben von SQL entgegenwirkt. Auf der anderen Seite soll so der Spieler zum Wiederkommen animiert werden.

Des Weiteren wurde im Pflichtenheft erwähnt, dass Spieler sich durch Aufgabenerstellung ins Spiel mit einbringen können. Um dabei jedoch eine gewisse Qualität gewährleisten zu können, wird es nur beförderten Nutzern möglich sein, eigene Aufgaben zu erstellen. Wie ein User befördert wird, wird über die Spielzeit oder über Leistungen im Spiel geregelt.

2 Analyse der Produktfunktionen

Im Folgenden werden die im Pflichtenheft benannten Funktionen näher beschrieben und erklärt. Dabei wird davon ausgegangen, dass der User sich in dem Interface befindet, in dem er die erklärte Funktion initiieren kann.

Des Weiteren ist zu erkennen, dass, jedes mal wenn das Back-End nach der Registration oder dem Login aktiv wird, es zuerst einen „checkSession()“-Methodenaufruf startet. Dies ist die Überprüfung des Back-Ends, ob es selbst den User kennt, ob er die gebrauchten Rechte für die Aktion hat und ob die Aktion auch auf die zum User gehörenden Daten ausgeführt wird.

2.1 Analyse von Funktionalität <F10>: <Nutzer registrieren>

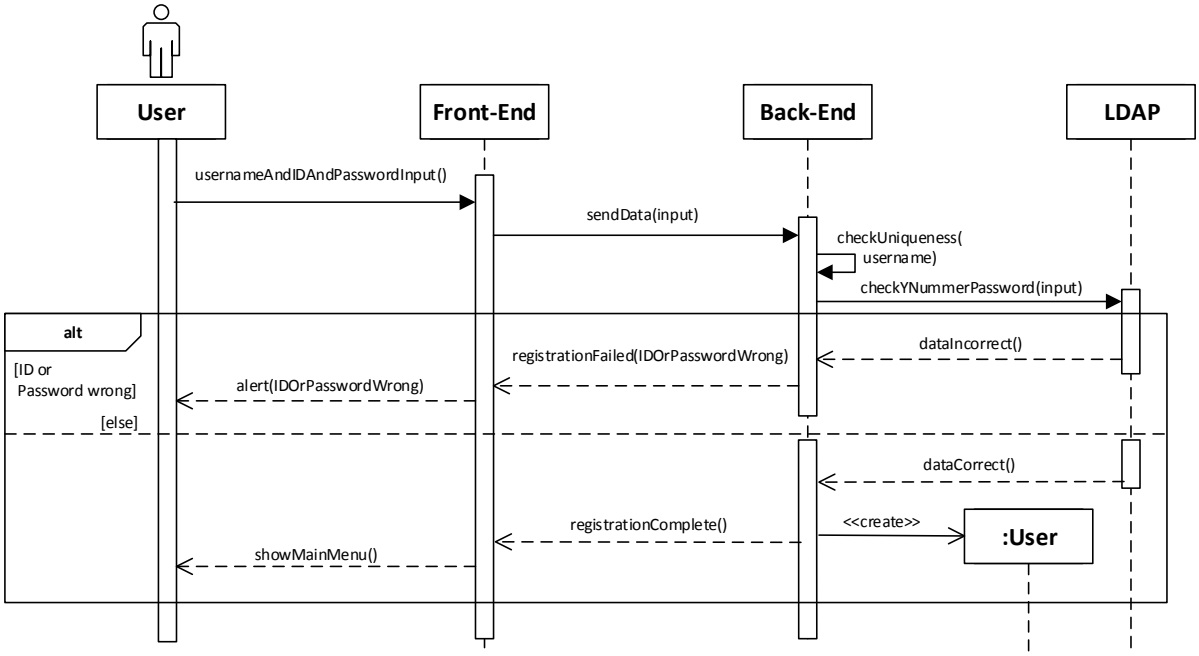

Abbildung 2.1: Sequenzdiagramm zur Registrierung

In Diagramm 2.1 wird der Vorgang der Registrierung beschrieben.

Dafür muss der User zuerst einen Username, eine E-Mail-Adresse oder seine y-Nummer (insgesamt ID genannt) und ein Passwort (im Falle einer y-Nummer das zur y-Nummer gehörende Passwort) angeben. Diese Eingaben werden dann an das Back-End übertragen. Hier wird zuerst geprüft ob der Username schon vergeben ist. Sollte das der Fall sein, wird dies dem User mitgeteilt und er muss einen neuen Username angeben. Wenn der Username noch nicht vergeben war, wird, im Falle dass es sich bei der ID um eine E-Mail-Adresse handelt, geprüft, ob diese schon vorhanden ist. Wenn die ID eine y-Nummer ist, wird diese, inklusive des eingegebenen Passworts, an das „LDAP“ geschickt und dort überprüft. Sollte der jeweils zutreffende Schritt fehlschlagen, muss der User seine Eingaben ändern, beziehungsweise korrigieren und der Prozess beginnt von vorn.

Sollte der Ablauf erfolgreich abgeschlossen worden sein, erstellt das Back-End ein neues User-Objekt, speichert dies in seiner Datenbank und gibt die Rückmeldung, dass die Registrierung erfolgreich gewesen ist. Der User wird dann in das Hauptmenü weitergeleitet.

2.2 Analyse von Funktionalität <F20>: <Nutzer anmelden>

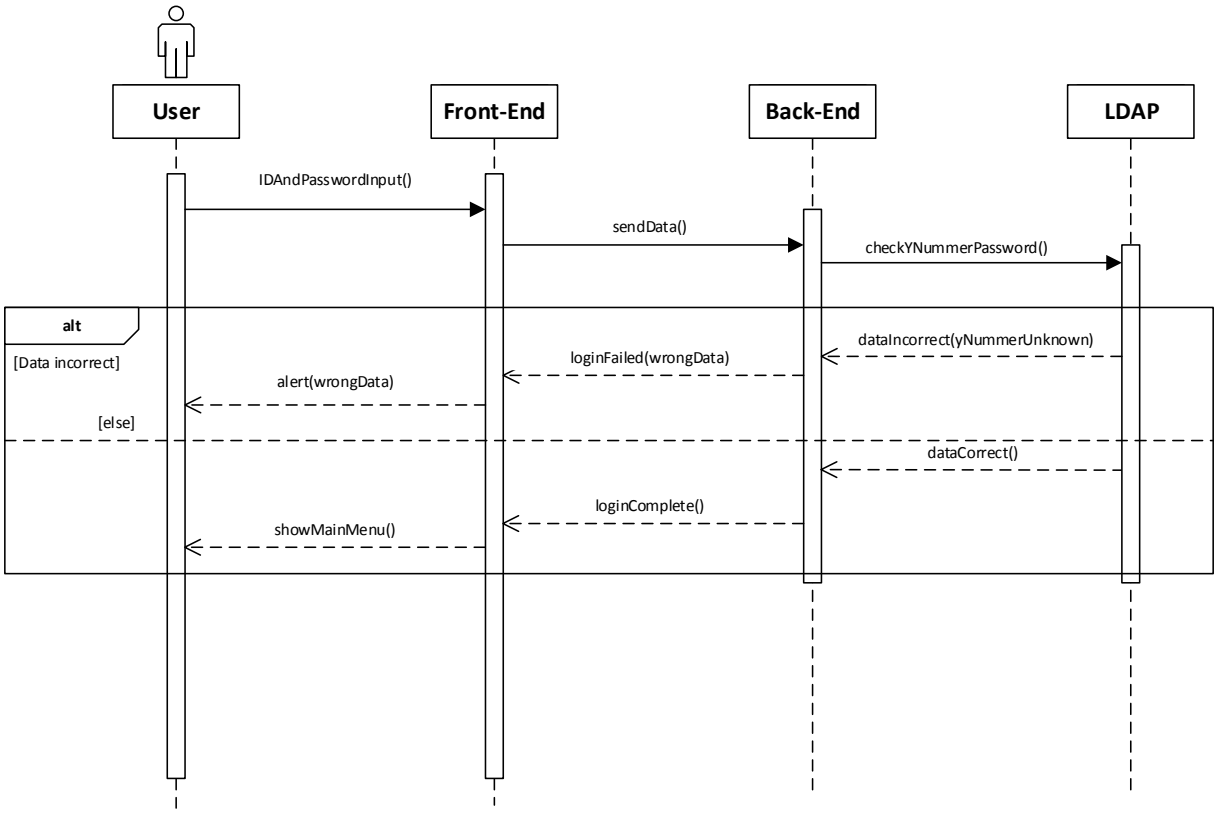

Abbildung 2.2: Sequenzdiagramm zum Login

Das Diagramm 2.2 beschreibt den Login-Vorgang.

Hierbei muss der User die von ihm registrierte ID (siehe <F10>) und sein Passwort angeben. Diese werden, je nach Typ der ID, dann entweder mit der eigenen Datenbank verglichen, oder, sollte die ID eine y-Nummer sein, an das „LDAP“ geschickt und dort überprüft.
Sollten die eingegeben Daten nicht korrekt sein, wird dies dem User mitgeteilt und er bekommt die Möglichkeit seine Eingaben zu korrigieren. Sind die Daten korrekt wird der User in das Hauptmenü weitergeleitet.

2.3 Analyse von Funktionalität <F30>: <Nutzer abmelden>

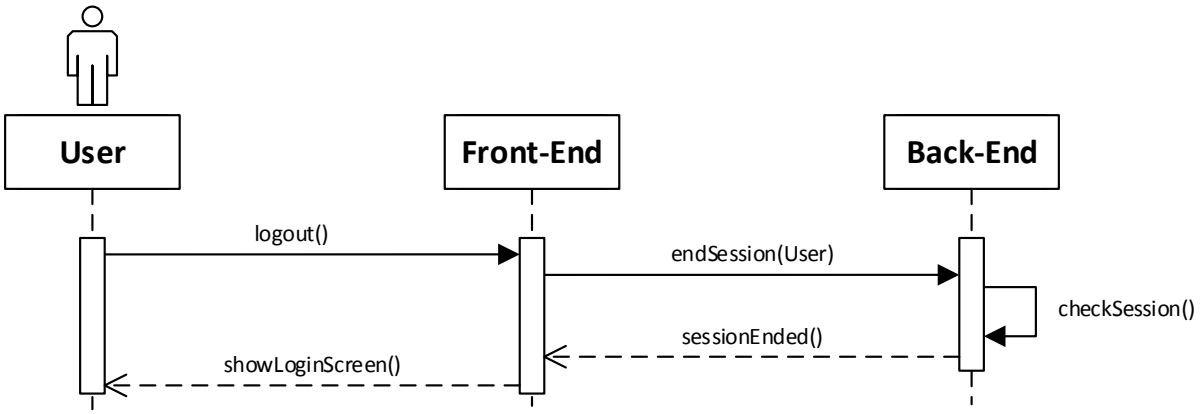

Abbildung 2.3: Sequenzdiagramm zum Login

Das Diagramm 2.3 beschreibt den Ablauf des Ausloggens.

Entscheidet sich der User dazu sich auszuloggen, wird diese Information an das Back-End gesendet. Dieses beendet die Session des Users. Ist das geschehen wird eine Benachrichtigung an das Front-End gesendet und der User wird zurück zum Login-Screen geleitet.

2.4 Analyse von Funktionalität <F40>: <Profil einsehen>

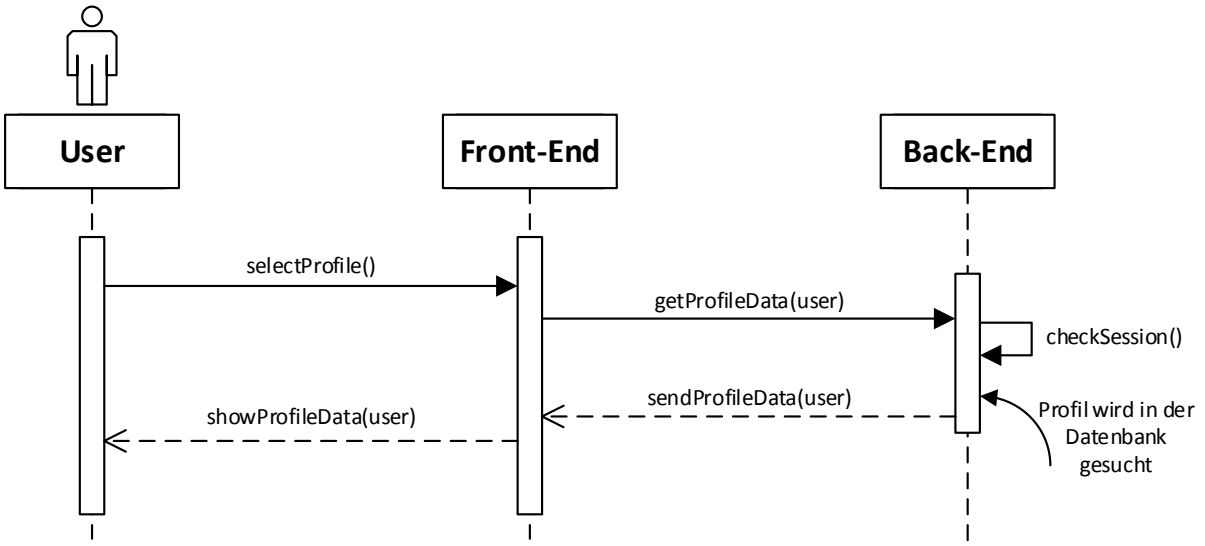

Abbildung 2.4: Sequenzdiagramm zur Darstellung der Profilübersicht

Das Diagramm 2.4 zeigt was passiert, wenn man sich das eigene Userprofil anzeigen lassen möchte.

Um das Profil des Users anzuzeigen, werden zuerst die Userdaten vom Back-End angefordert. Das Back-End lädt dann die Profildaten aus den Userdaten und schickt diese zurück an das Front-End. Dort werden sie, für den User einsehbar, im Profil-Interface angezeigt.

2.5 Analyse von Funktionalität <F60>: <Passwort ändern>

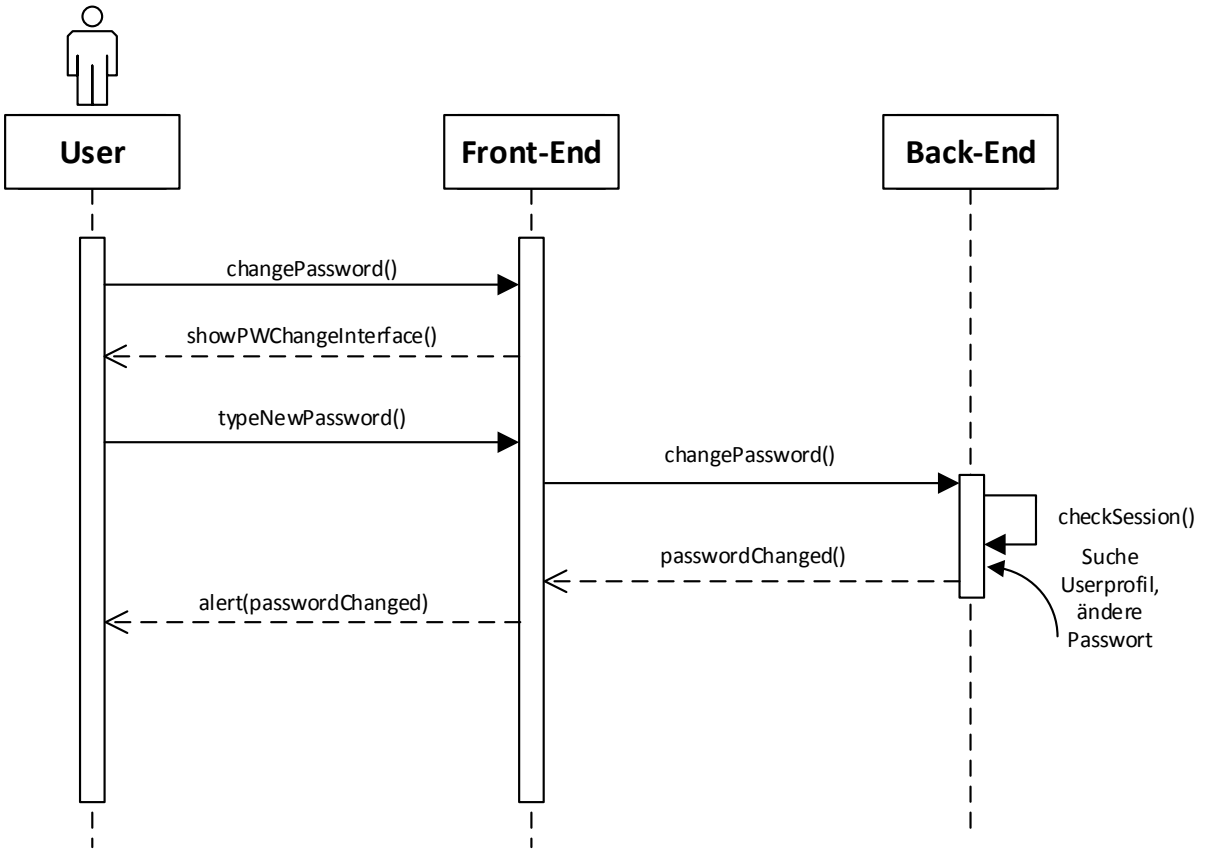

Abbildung 2.5: Sequenzdiagramm zum Ändern des Passworts

Das Diagramm 2.5 beschreibt das Ändern des Passwortes.

Das Ändern des Passwortes steht nur nicht-studentischen Nutzern zur Verfügung. Entscheidet sich der Nutzer, sein Passwort zu ändern, kann er dies mit einem Klick auf den „Change Password “-Button tun. Zuerst muss er sein altes Passwort eingeben und im Anschluss kann er ein neues Password erstellen. Dies muss danach noch einmal bestätigt werden und wenn alles erfolgreich war, wird das geänderte Passwort an das Back-End geschickt und dort in den Nutzerdaten vermerkt. Sollte während des Vorgangs ein Fehler auftreten, bleibt das alte Passwort bestehen und der User wird darüber in Kenntnis gesetzt.

2.6 Analyse von Funktionalität <F70>: <Avatar ändern>

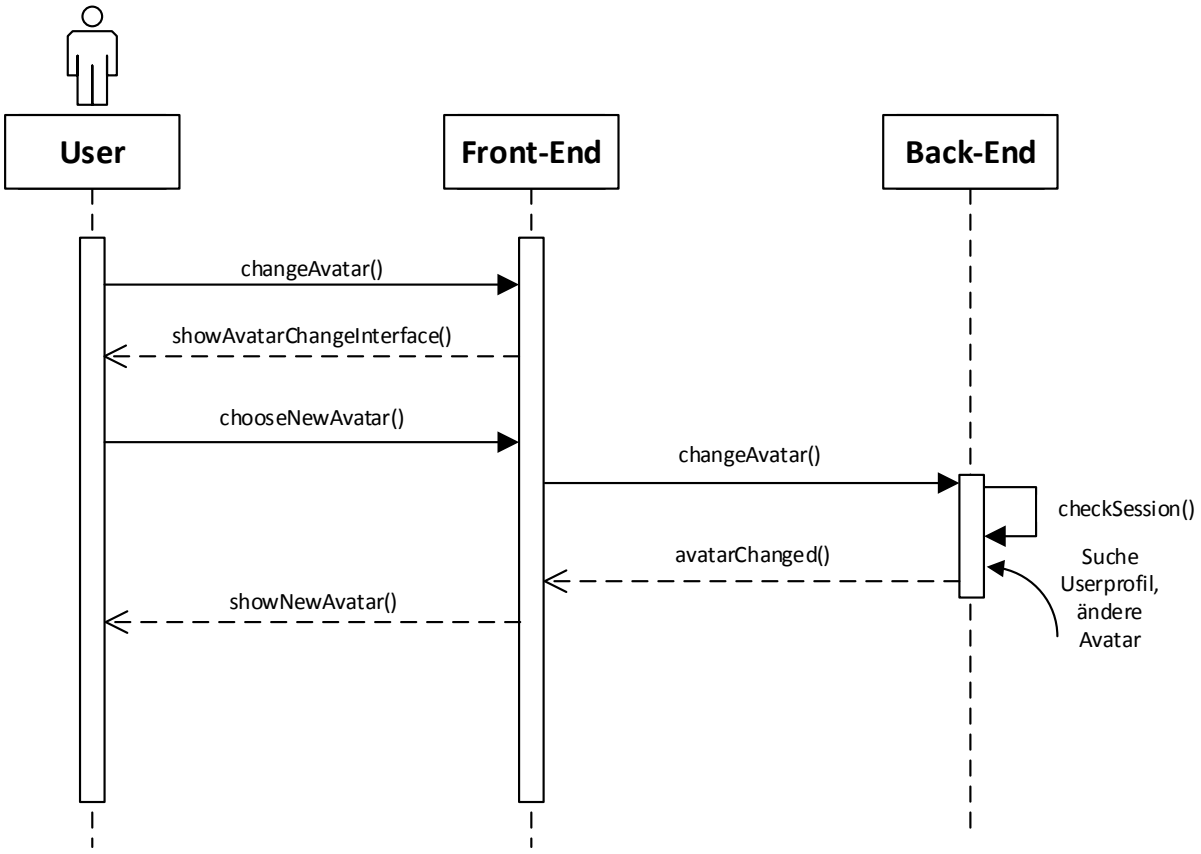

Abbildung 2.6: Sequenzdiagramm zum Ändern des Avatars

Das Diagramm 2.6 beschreibt das Ändern des Avatars.

Klickt der User auf den „Change Avatar “-Button, werden ihm alle seine verfügbaren Avatare angezeigt und er kann sich für einen entscheiden. Hat er dies getan wird die Änderung der Einstellung vermerkt, ans Back-End geschickt, dort gespeichert und dann, wieder zurück im Front-End, aktualisiert angezeigt.

2.7 Analyse von Funktionalität <F80>: <Benutzer löschen>

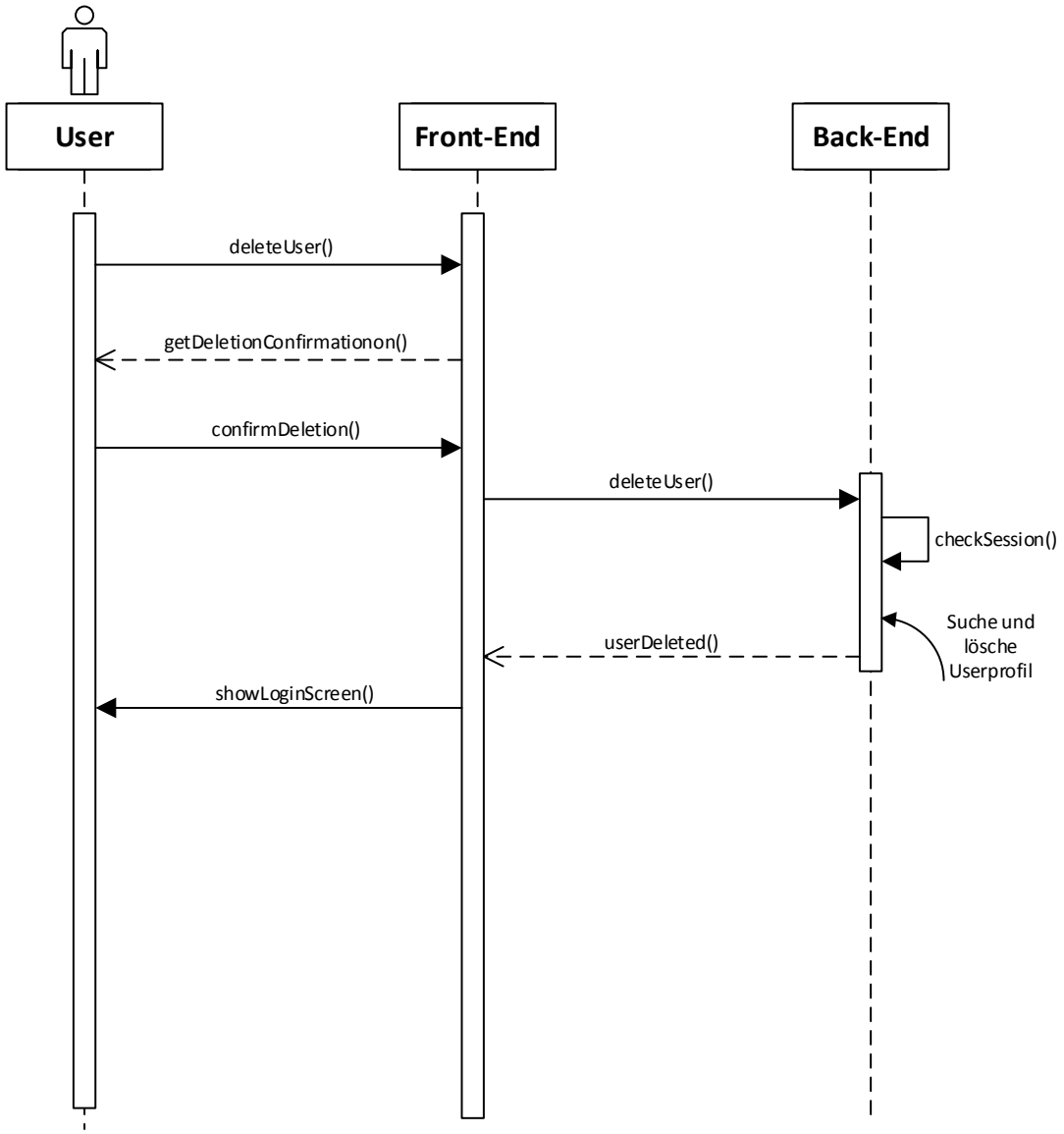

Abbildung 2.7: Sequenzdiagramm zum Löschen des Benutzers

Diagramm 2.7 zeigt, was passiert, wenn der User seinen Account löschen möchte. Möchte der User sein Profil löschen, kann er dies im Settings-Interface tun, welches dementsprechend erst geladen werden muss. Wenn der User nun den „Delete User“-Button drückt, wird er erst noch einmal gefragt, ob er sein Profil wirklich löschen möchte. Beantwortet der User dies positiv, geht eine Mitteilung an das Back-End, wo das Profil gelöscht wird. Der User wird dann auf den Login-Screen geleitet, wo es ihm frei steht, sich wieder zu registrieren.

2.8 Analyse von Funktionalität <F90>: <Audioeinstellungen bearbeiten>

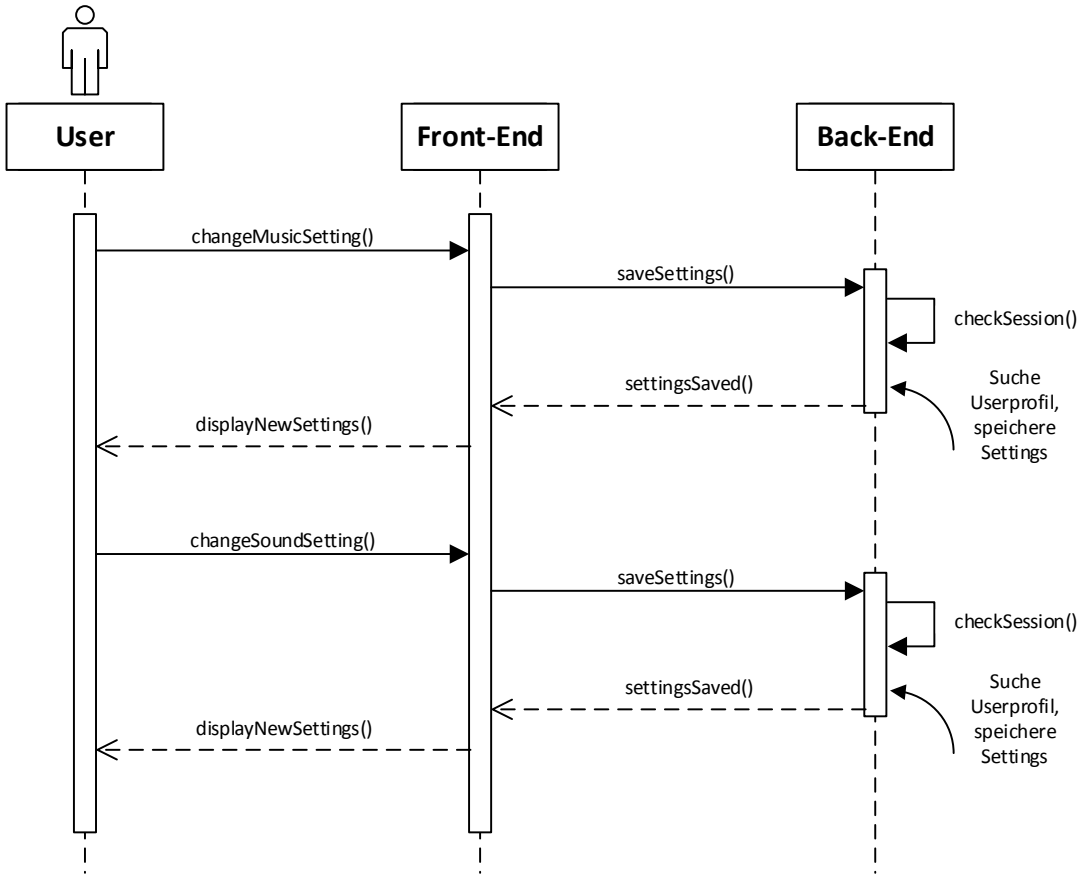

Abbildung 2.8: Sequenzdiagramm zum Ändern der Audioeinstellungen

Diagramm 2.8 zeigt, was passiert, wenn der User seine Audio-Optionen ändert.

Zum Ändern der Audiofunktionen stehen Knöpfe bereit, die sowohl die Hintergrundmusik („Music“) als auch die Soundeffekte, wie Sprungsounds und Klicksounds, aus- beziehungsweise einstellen. Jegliche Änderung wird sofort ans Back-End gesendet, dort gespeichert und dann im Profil aktualisiert.

2.9 Analyse von Funktionalität <F100>: <Spielstand zurücksetzen>

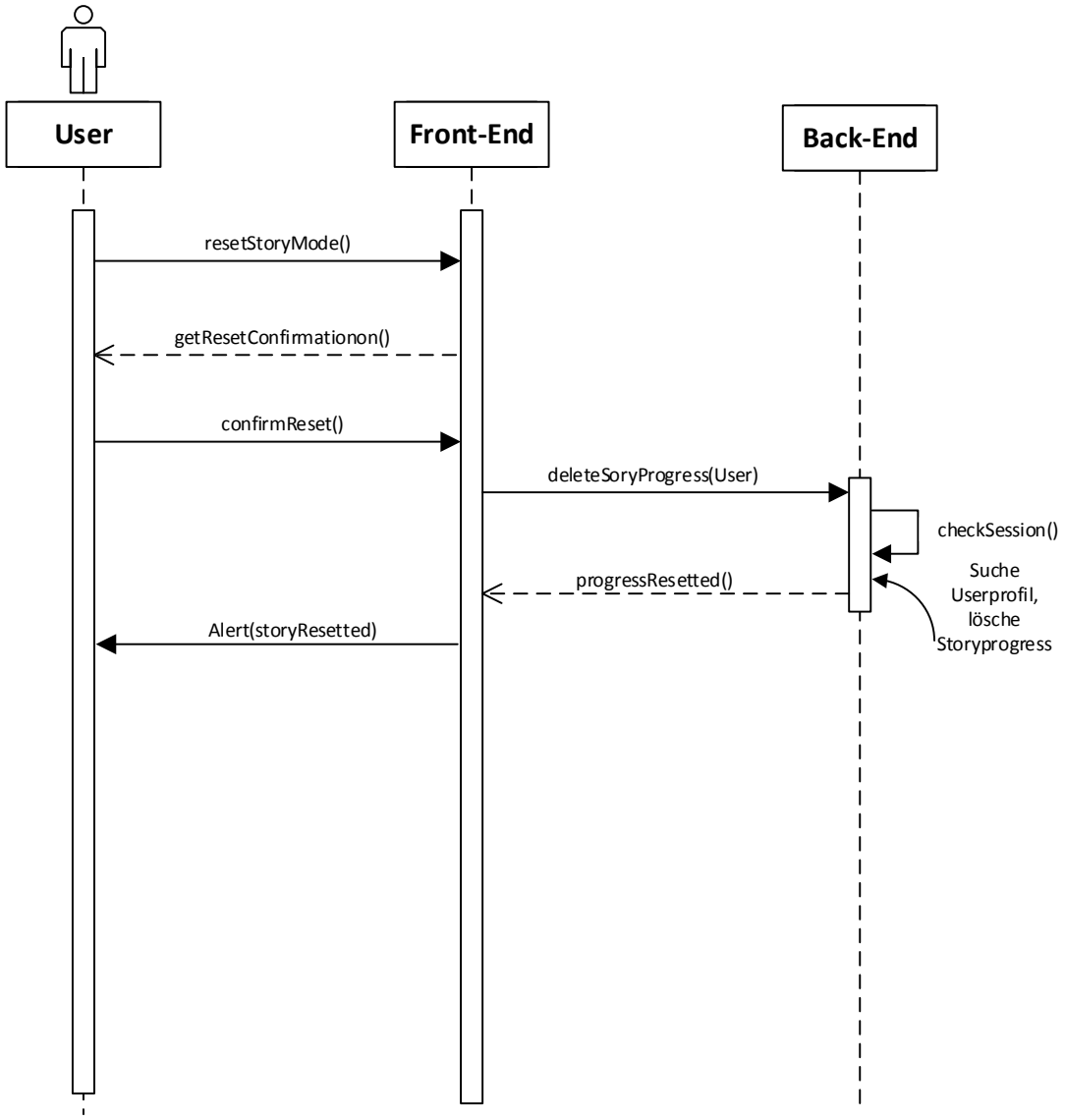

Abbildung 2.9: Sequenzdiagramm zum Zurücksetzen des Spielstands

Diagramm 2.9 beschreibt das Zurücksetzen des Storyfortschritts.

Der User kann per Knopfdruck seinen Story-Fortschritt zurücksetzen. Tut er dies, muss er vorher noch einmal seine Zustimmung zum Löschvorgang geben. Ist auch dies geschehen, wird die Anweisung zum Löschen des Storyfortschritts des Users an das Back-End geschickt und dort ausgeführt.

2.10 Analyse von Funktionalität <F110>: <Tutorial spielen>

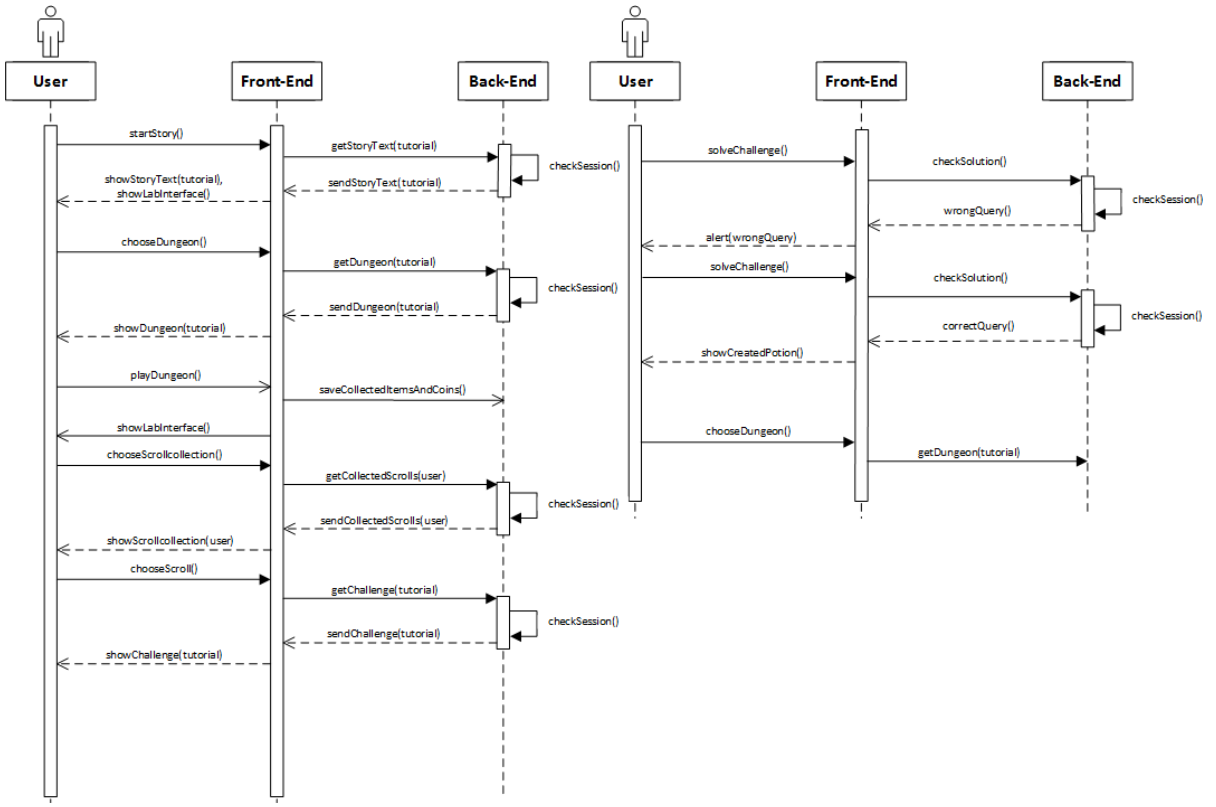

Abbildung 2.10: Sequenzdiagramm für das Tutorial

Diagramm 2.10 zeigt den Ablauf im Tutorial.

Wird das Tutorial gestartet, werden zuerst alle Tutorial-Texte vom Back-End angefordert. Diese werden dann, wenn benötigt, dem User angezeigt. Zuerst wird der User in den Dungeon geleitet. Dazu werden die Leveldaten vom Back-End geladen, sodass der User den Dungeon betreten und spielen kann. Hierbei kann er Schriftrollen („Scrolls“) und Münzen („Lofi-Coins“) einsammeln, was jeweils sofort ans Back-End gesendet und dort im Story-Progress bzw. in den Profildaten gespeichert wird. Scheitert der User an einer Hürde im Dungeon, bekommt er zuerst einen Game-Over-Screen angezeigt, auf dem zu sehen ist, was er eingesammelt hat und wird dann zurück in den Labor-Screen geleitet. Dort wird er per Tutorial-Text zur Scrollcollection geleitet, um sich dort für ein Trank-Rezept zu entscheiden. Ist dies getan, wird aus dem Back-End eine für das Rezept passende Aufgabe angefordert und dem User angezeigt. Für diese Aufgabe hat der User keine Versuchsbegrenzung. Die Eingaben des Users werden dabei immer an das Back-End gesendet und dort kontrolliert. Hat der User die Aufgabe richtig gelöst, erhält er eine Potion und kann diese danach im Dungeon verwenden.

2.11 Analyse von Funktionalität <F120>: <Story spielen>

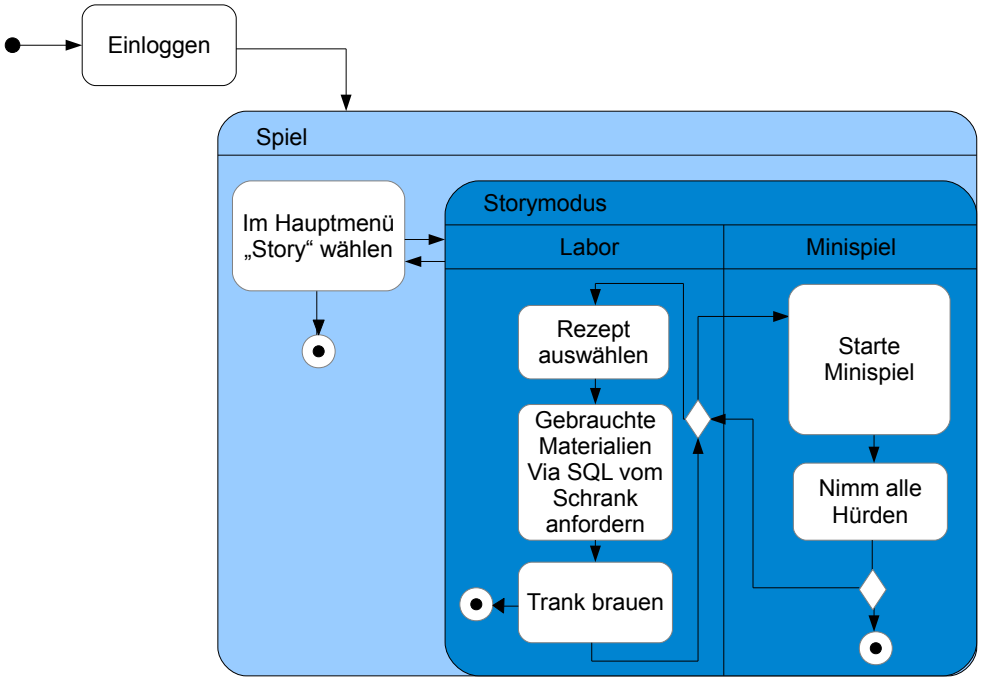

Abbildung 2.11: Aktivitätsdiagramm für den Story Mode

Der Story-Mode besteht zu sich abwechselnden Teilen aus SQL-Trainer und Minispiel. Um dies besser zu zeigen, ist in Abbildung 2.11 das Aktivitätsdiagramm für den Story Mode aus dem Pflichtenheft abgebildet.
Darin ist zu sehen, dass man sich im Labor einen oder mehrere Tränke brauen kann (SQL-Trainer-Anteil), um diese dann im Dungeon zu verwenden (Minispiel-Anteil). Um die beiden Teile für sich besser beschreiben zu können, sind diese in den folgenden zwei Funktionen separat aufgeführt.

2.12 Analyse von Funktionalität <F130>: <SQL-Trainer spielen>

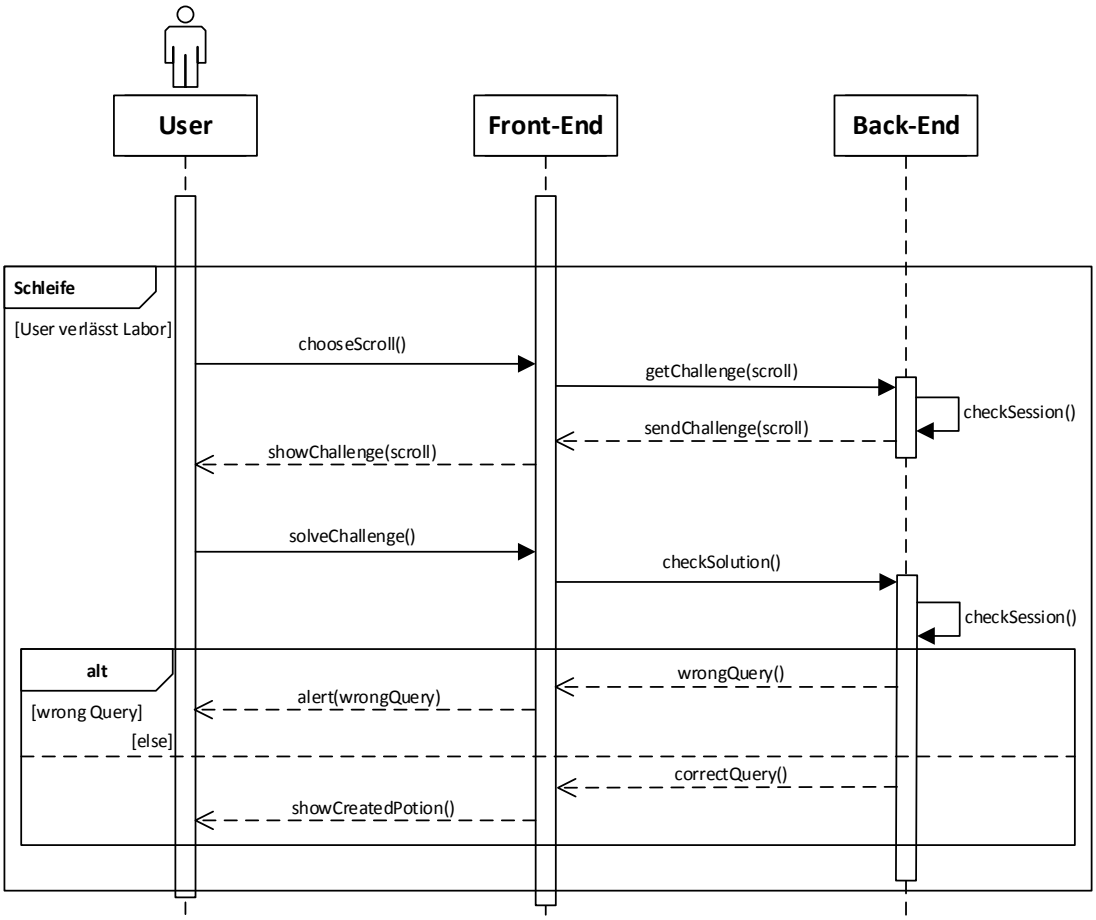

Abbildung 2.12: Sequenzdiagramm für den SQL-Trainer

Der SQL-Trainer ist Teil der drei Spielmodi (Story, Trivia, Homework) und wird einmal (wie er im Trivia-Mode verwendet wird) erklärt. Die leichten Abweichungen der anderen Spielmodi werden im Anschluss erwähnt. Diese benötigen wenig bis gar keine weitere Erklärung, da sie nur einen oberflächlichen Unterschied machen. Zuerst werden dem User fünf Schwierigkeitsgrade angezeigt, aus denen er wählen kann. Hat er sich für einen Schwierigkeitsgrad entschieden, wird dies dem Back-End mitgeteilt, welches daraufhin eine, dem gewählten Schwierigkeitsgrad entsprechende, Aufgabe bereitstellt. Diese kann dann vom User gelöst werden. Die Eingaben werden dabei immer an das Back-End geschickt und dort auf Richtigkeit geprüft. Ist die Prüfung positiv verlaufen, wird der User wieder zur Schwierigkeitsgrad-Auswahl weitergeleitet.

Abweichungen zum Story-Mode:
Es werden keine Schwierigkeitsgrade angezeigt, sondern schon eingesammelte Rezepte für Tränke. Diese beinhalten in ihrer Definition schon die Schwierigkeitsgrade für die daraus hervorgehenden Aufgaben.

Abweichungen zum Homework-Mode:
Beim Homework-Mode wird die Auswahl des Schwierigkeitsgrades in jeglicher Form übersprungen und der Aufgabentext wird direkt angezeigt.

2.13 Analyse von Funktionalität <F140>: <Minispiel spielen>

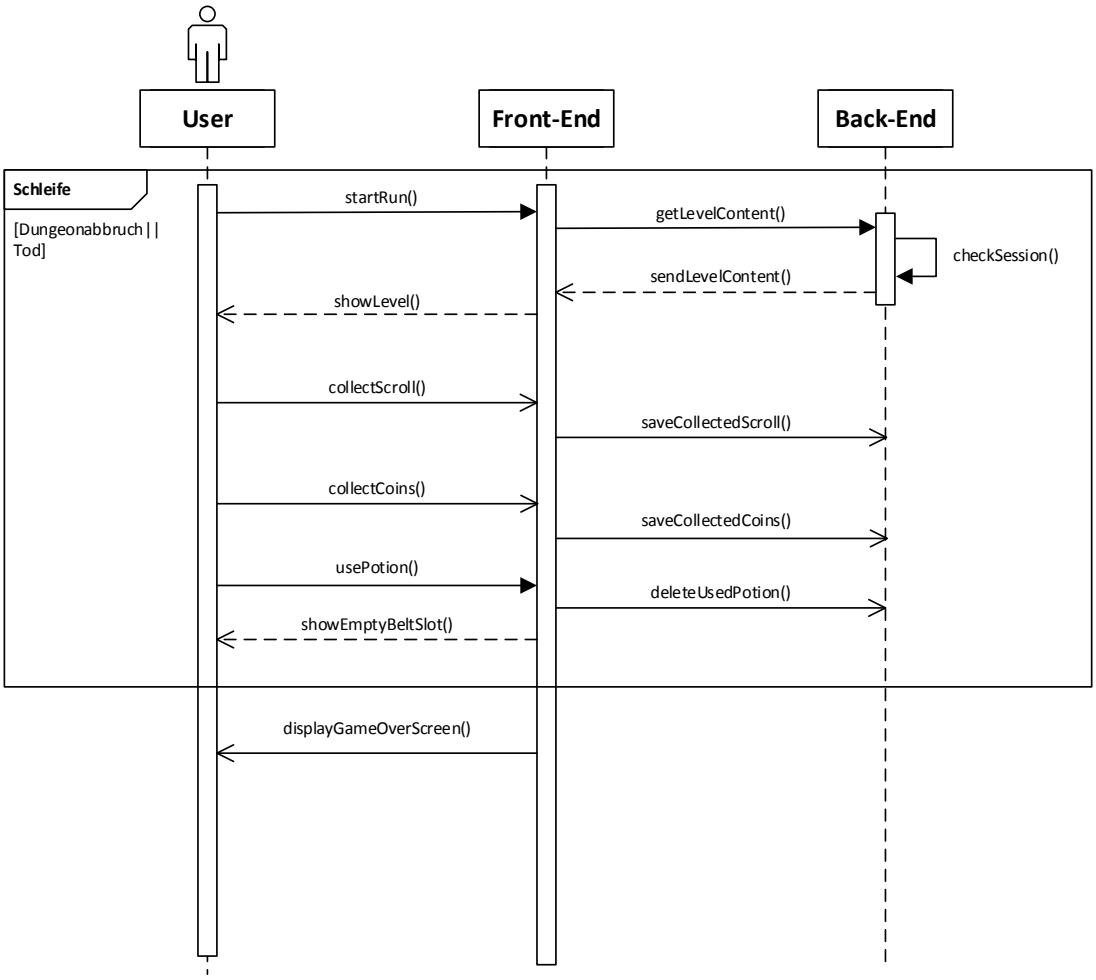

Abbildung 2.13: Sequenzdiagramm für das Minigame

Die Abbildung 2.13 zeigt die Abläufe im Dungeon.

Im Minispiel bewegt sich die Figur stetig nach rechts. Der User kann springen, wodurch er verschiedene Hindernisse überwinden kann. Des Weiteren kann der User Lofi-Coins und Scrolls einsammeln. Dies wird sofort im Back-End vermerkt und im Spielerprofil und im Storyprogress gespeichert. Dem User ist zudem möglich, vorher erstellte Potions zu verwenden, um deren Effekte zur Überwindung von Hindernissen zu nutzen.

2.14 Analyse von Funktionalität <F150>: <Hausaufgaben bearbeiten>

Die Bearbeitung von Hausaufgaben funktioniert wie das Nutzen des SQL-Trainers. Genauere Erklärungen sind unter (F130) zu finden. Von der dortigen Beschreibung gibt es allerdings folgende Abweichungen:

Bei den Hausaufgaben handelt es sich um Aufgabenpakete, welche durch die Lehrenden des Moduls RDB1 erstellt werden und den Studenten, die das Modul belegen, zugewiesen werden. Alle anderen Nutzer der Anwendung haben keinen Zugriff auf diese Aufgaben. Die gestellten Aufgabenpakete werden dann als Teil der Studienleistung betrachtet, welche erfüllt werden muss, um RDB1 zu bestehen. Daher sind die Aufgaben nur in den vorgesehenen Bearbeitungszeiträumen erreichbar und zur Bearbeitung freigegeben.

2.15 Analyse von Funktionalität <F160>: <Ranglisten einsehen>

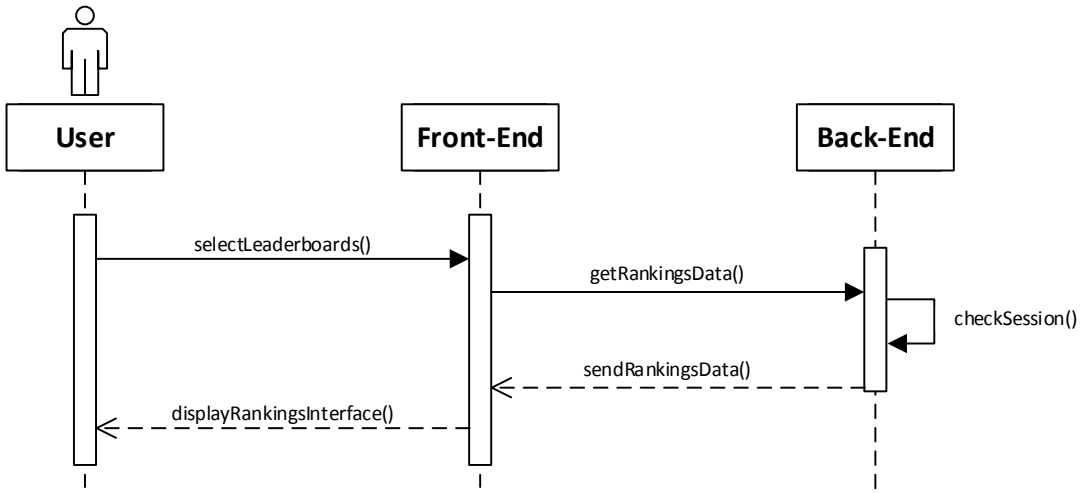

Abbildung 2.14: Sequenzdiagramm für die Ranglisten

In Abbildung 2.14 wir das Anzeigen der Ranglisten dargestellt.

Möchte der User die Ranglisten einsehen, wird eine Anfrage an das Back-End gesendet. Das Back-End stellt dann die Daten für die Ranglisten zusammen und sendet diese zurück an das Front-End, wo diese dann für den User einsehbar präsentiert werden.

2.16 Analyse von Funktionalität <F170>: <Spieler suchen>

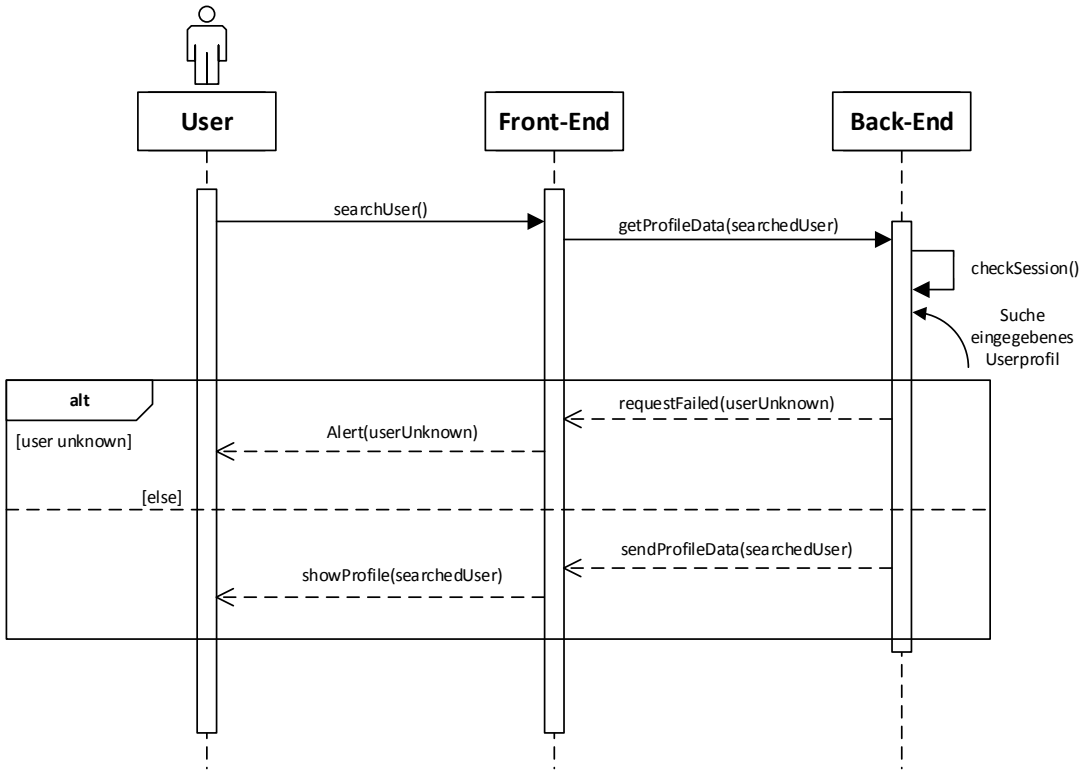

Abbildung 2.15: Sequenzdiagramm zum Suchen eines anderen Users

Im Diagramm 2.15 ist zu sehen, wie man nach einem anderen User suchen kann.

Um einen anderen User zu suchen, wird ein Eingabefeld bereitgestellt. Der dort eingegebene Name wird an das Back-End weitergeleitet, dort in der Userdatenbank gesucht und wenn er existiert, wird dessen Profil aus der Datenbank geladen und dem suchenden User angezeigt. Ist der Name in der Datenbank nicht zu finden, wird dies dem suchenden User mitgeteilt.

Der Zweck der Suche ist es, dass die User die Profile anderer Spieler einsehen können, um sich mit diesen vergleichen zu können. Dies kann aus Gründen des Wettbewerbs untereinander oder einfach aus dem Interesse am Fortschritt von bekannten Usern geschehen. Somit bietet die Funktion den Usern eine simple Möglichkeit sich mit anderen Spielern zu vergleichen.

2.17 Analyse von Funktionalität <F180>: <Hausaufgabenergebnisse einsehen>

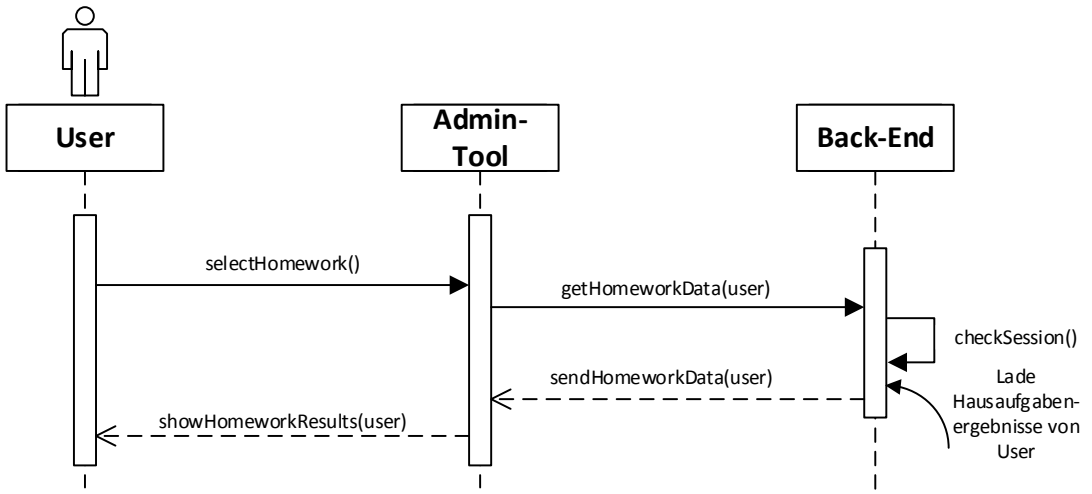

Abbildung 2.16: Sequenzdiagramm zum einsehen der eigenen Hausaufgabenergebnisse

Abbildung 2.16 beschreibt das Anzeigen der Hausaufgabenergebnisse.

Nach dem Einloggen in das Admintool werden die Hausaufgabenergebnisse eines nicht-beförderten Users direkt aus der Datenbank geladen und angezeigt. Ist der User schon „befördert“, muss er erst noch auf den „Show Homework Results “-Button drücken.

2.18 Analyse von Funktionalität <F190>: <Benutzer befördern>

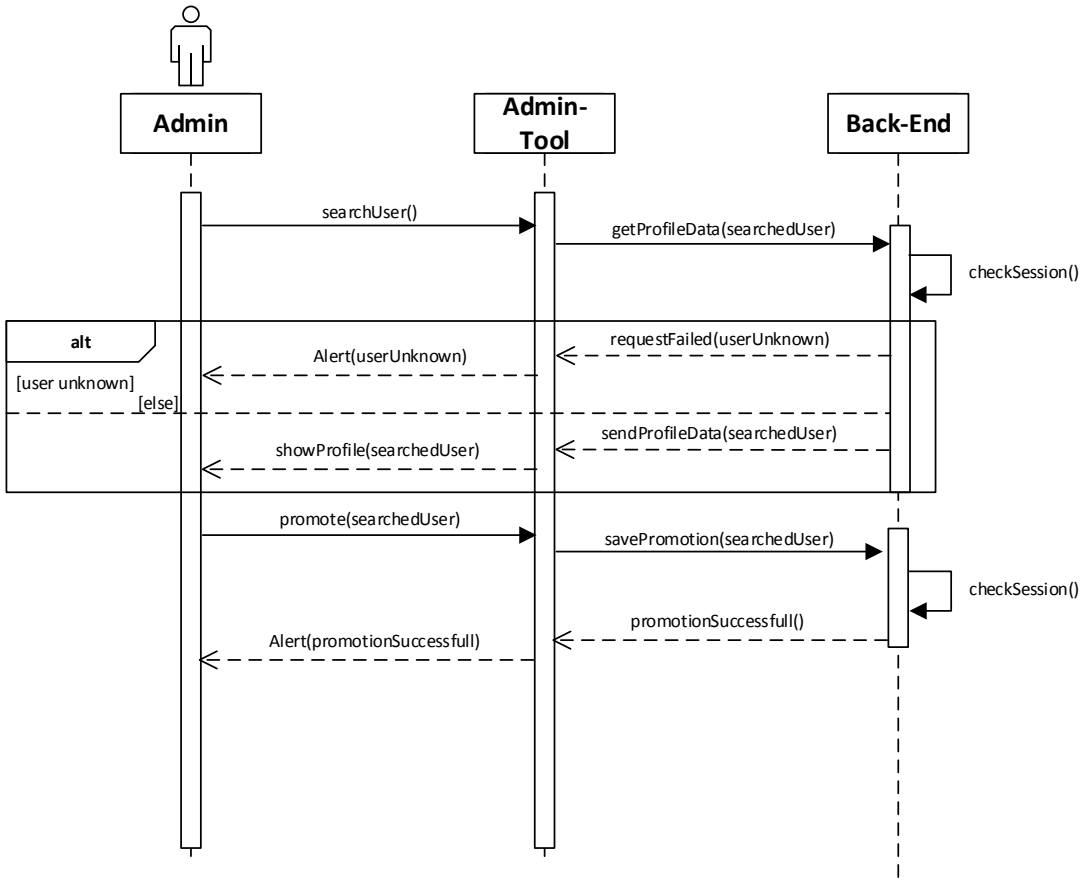

Abbildung 2.17: Sequenzdiagramm zum Befördern eines Users

Abbildung 2.17 beschreibt das Befördern eines Benutzers.

Zuerst wird für den Admin die Userdatenbank angezeigt. Hier kann er entweder manuell oder per Suchfunktion nach einem User suchen und diesen per Knopfdruck befördern, was dann im Back-End in den jeweiligen Userdaten registriert und gespeichert wird.

Der Ablauf, einem User Adminrechte (<F200>) zu geben, gleicht dem des User Beförderns.

2.19 Analyse von Funktionalität <F210>: <Eine Trivia-Aufgabe erstellen>

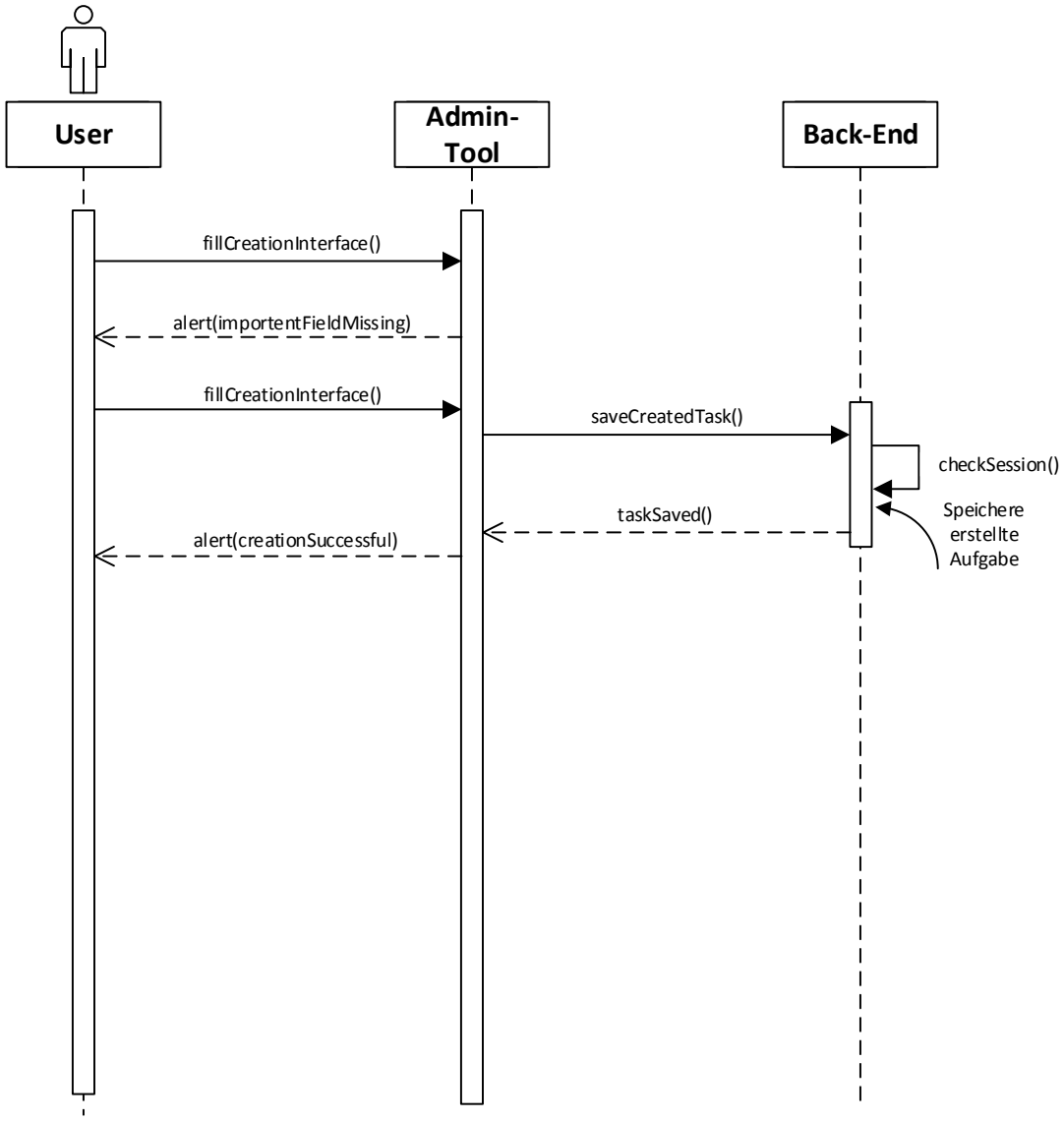

Abbildung 2.18: Sequenzdiagramm für das Erstellen von Aufgaben

Abbildung 2.18 beschreibt das Erstellen von eigenen Aufgaben.

Beförderte Nutzer können selber Aufgaben erstellen, welche später im Trivia Mode verwendet werden können. Dafür steht im Admin-Tool ein Interface zur Verfügung, in das ganz einfach alle zur Aufgabe gehörigen Daten eingetragen und dann an das Back-End gesendet werden, wo die Aufgabe gespeichert wird. Sind nicht alle wichtigen Felder ausgefüllt, wird der User alarmiert und muss dies berichtigen, damit die Aufgabe abgespeichert werden kann.

2.20 Analyse von Funktionalität <F220>: <Benutzeraufgaben bewerten>

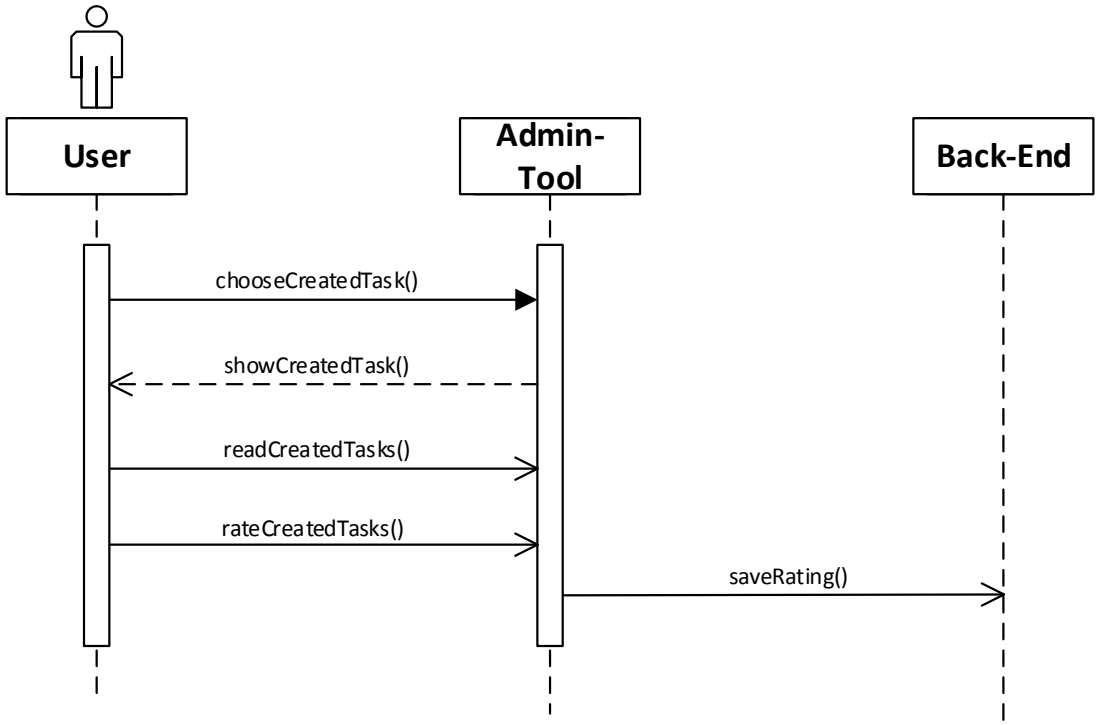

Abbildung 2.19: Sequenzdiagramm zum bewerten User-erstellter Aufgaben.

Abbildung 2.19 zeigt den Ablauf, wie man eine von Usern erstellte Aufgabe bewertet.

In der Liste aller von Usern erstellten Aufgaben kann sich der User Aufgaben aussuchen, diese ansehen, sie bewerten und kommentieren. Die Bewertung wird dann im Back-End für die Aufgabe registriert und gespeichert.

2.21 Analyse von Funktionalität <F230>: <Hausaufgaben erstellen>

Die Erstellung von Hausaufgaben gleicht der userseitigen Erstellung von Aufgaben, ist jedoch nur Administratoren zugänglich.

Der Unterschied zur normalen Aufgabenerstellung besteht darin, dass ein Aufgabenpaket erstellt wird. Dabei kann man entweder Aufgaben für das Paket neu erstellen, oder auf schon einmal erstellte Aufgaben, die in der Datenbank gespeichert sind, zugreifen. Diese Aufgabenpakete können dann noch mit zusätzlichen Parametern versehen werden, wie zum Beispiel einem Bearbeitungszeitraum oder einer Anzahl an Versuchen.

3 Resultierende Softwarearchitektur

Dieser Abschnitt hat die Aufgabe, einen Überblick über die zu entwickelnden Komponenten und Subsysteme zu liefern.

3.1 Komponentenspezifikation

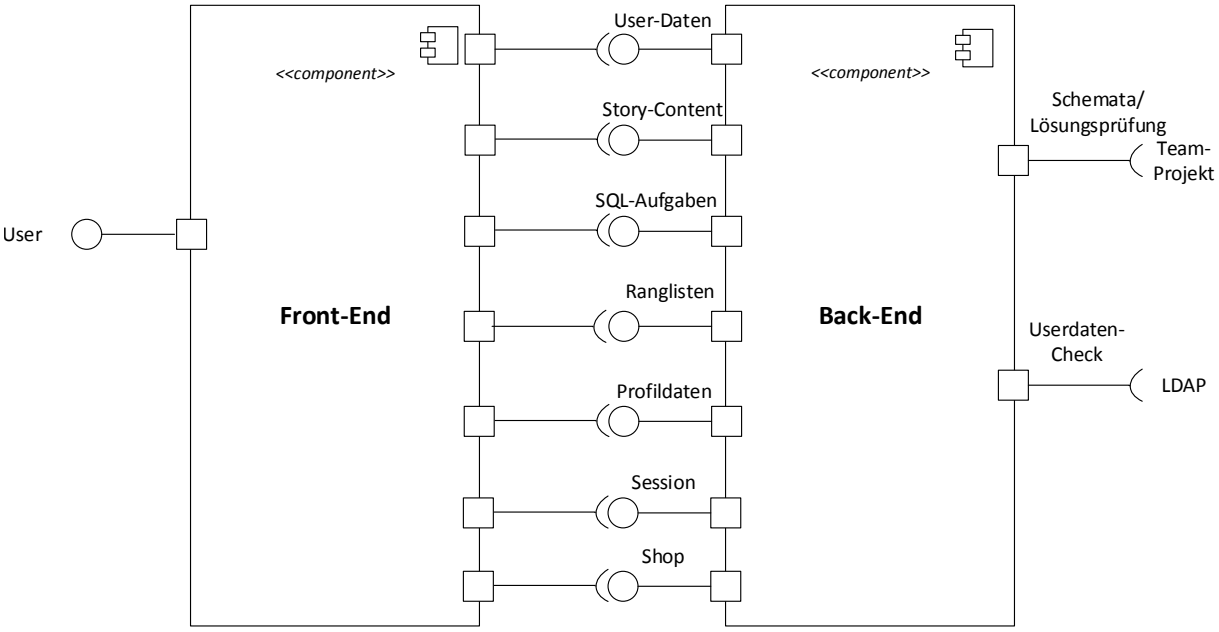

Abbildung 3.1: Komponentendiagramm.

In Abbildung 3.1 ist das Komponentendiagramm zu sehen, das sich aus der Funktionionsanalyse in Abschnitt 2 ergibt. Darin ist zu erkennen, dass die beiden Komponenten, Front-End und Back-End, über verschiedene Interfaces miteinander kommunizieren. Des Weiteren ist zu sehen, dass das Back-End eine Anbindung an das LDAP benötigt, um einen Teil der Nutzerdaten überprüfen zu können.

Komponente $\langle C10 \rangle$**: <Front-End>**
Das Front-End ist ein Interface, das sämtliche vom Back-End angeforderten Daten verarbeitet. Dazu gehört die entsprechende Visualisierung dieser Daten. Das Front-End ist also die Oberfläche der Software, die der User sieht.

Komponente $\langle C20 \rangle$**: <Back-End>**
Das Back-End ist eine Datenbank, die dem Front-End verschiedene Schnittstellen zur Verfügung stellt, um an die angeforderten Daten zu kommen.

3.2 Schnittstellenspezifikation

Im Folgenden werden die einzelnen Schnittstellen der Komponenten aus der Komponentenspezifikation näher erläutert, d.h. die von ihnen zur Verfügung gestellten Operationen werden dokumentiert. Die Tabelle ist dabei um so viele Zeilen zu erweitern, wie es Schnittstellen im Komponentendiagramm gibt. In der innen liegenden Aufteilung ist für jede Operation einer Schnittstelle eine Zeile einzufügen. Reine Set- und Get-Aufrufe brauchen nicht aufgeführt zu werden (sollten auch möglichst nicht komponentenübergreifend auftauchen).

Schnittstelle $\langle I10 \rangle$**: <Session>**

Action and URL	Method	Send	Response Success	Response Error
POST /login	.create	JSON.login	redirect(/profile)	401(JSON.sign.error)
GET /logout	.delete	-	ok	-

Möchte sich ein User einloggen, sendet das Front-End ein JSON-Objekt mit den Login-Daten an das Back-End. Es wird versucht mittels der „create “-Methode eine Session zu erstellen. Bei Erfolg wird die Session angelegt und der User in den Home-Screen weitergeleitet, wobei die Profildaten bereits geladen werden. Bei Misserfolg wird ein HTTP-Error (hier für unautorisierter User) inklusive eines JSON-Objektes zurückgegeben, welches die Ursache des Fehlers beinhaltet. Möchte der User sich wieder ausloggen, wird dies per HTTP-GET-Anfrage übermittelt. Daraufhin wird die aktuelle Session durch die „delete“-Methode gelöscht.

Schnittstelle $\langle I20 \rangle$**: <Userdaten>**

Action and URL	Method	Send	Response Success	Response Error
POST /signup	.create	JSON.signup	redirect(/login)	400(JSON.sign.error)
POST /users	.edit	JSON.user	ok	400(JSON.sign.error)
DELETE /users	.delete	-	ok	400(JSON.sign.error)

Möchte sich ein User registrieren, wird ein JSON-Objekt mit den entsprechenden Daten an das Back-End gesendet, welches daraufhin mittels der „create “-Methode versucht einen neuen User zu erstellen. Bei Erfolg wird der User angelegt und dieser kann sich im Login-Screen einloggen. Bei Misserfolg wird ein HTTP-Error (Bad request) inklusive eines JSON-Objektes zurückgegeben, welches die Ursache für die Fehlermeldung beinhaltet. Außerdem kann der User sein Passwort ändern. Das vom User eingegebene alte und neue Passwort wird per JSON-Objekt an das Back-End ünbermittelt. Dort wird das alte Passwort überprüft. Ist es korrekt, wird das neue Passwort gespeichert, ist es nicht korrekt, wird ein HTTP-Error (Bad request) inklusive eines JSON-Objektes zurückgegeben, welches die Ursache des Fehlers beinhaltet. Möchte der User seinen Account löschen, muss er sein Passwort eingeben. Dieses wird dann per JSON-Objekt an das Back-End gesendet. Bei korrektem Passwort wird der User gelöscht, bei inkorrektem Passwort wird ein Fehler (Bad request) inklusive eines JSON-Objektes zurückgegeben, welches die Ursache des Fehlers beinhaltet.

Schnittstelle ⟨*I*40⟩**: <SQL-Aufgaben>**
Challenge

Action and URL	Method	Send	Response Success	Response Error
GET /challenge	.index	-	JSON.challenge[ ]	-
GET /challenge/:id	.view	-	JSON.challenge	-
POST /challenge	.create	JSON.challenge	ok	-
GET /challenge/story	.story	-	JSON.story	-
GET /challenge/trivia	.trivia	-	JSON.task	-
GET /challenge/homework	.homework	-	JSON.task	-

Im Admin-Tool können die Administratoren die vorhandenen Challenges betrachten oder neue erstellen. Dabei kann der Admin entweder alle Challenges, eine Challenge mit einer bestimmten ID oder alle Challenges eines bestimmten Typs anzeigen lassen. Eine neue Challenge wird im JSON Format an das Back-End gesendet und dort gespeichert.

Task

Action and URL	Method	Send	Response Success	Response Error
GET /tastk/story/:id	.story	-	JSON.task	-
POST /task/story/:id	.storySolve	JSON.task.solve	JSON.task.result	400(JSON.task.result)
GET /task/trivia	.triva	-	JSON.task	-
POST /task/trivia/:id	.triviaSolve	JSON.task.solve	JSON.task.result	400(JSON.task.result)
GET /task/homework	.homework	-	JSON.task	-
POST /task/homework/:id	.homeworkSolve	JSON.task.solve	JSON.task.result	400(JSON.task.result)
GET /task/	.index	-	JSON.task[ ]	-
GET /task/:id	.view	-	JSON.task	-
POST /task/:id	.edit	JSON.task	ok	-
POST /task/:id/rating	.rate	JSON.rating	ok	-
GET /task/:id/comment	.comments	-	JSON.comment[ ]	-
POST /task/:id/comment	.comment	JSON.comment	-	-

Es gibt mehrere Aufgabentypen, welche übermittelt werden müssen. Story sind Aufgaben für Scrolls mit einer gewissen ID. Die Aufrufe Trivia und Homework starten die Bearbeitung eines Aufgabenpaketes. Die /'solve/' Anfragen sind Antworten des Users auf die Statements mit einer bestimmten ID, die im JSON-Format übergeben werden. Bei richtiger Antwort wird ein JSON-Objekt übergeben, welches diese Information enthält. Bei falscher Antwort wird ein HTTP-Error inklusive eines JSON-Objektes zurückgegeben, welches die Ursache des Fehlers beinhaltet (hier die falsche Antwort).

Schnittstelle $\langle I50 \rangle$: <Ranglisten>

Action and URL	Method	Send	Response Success	Response Error
GET /highscore/points	.byPoints	-	JSON.highscore[ ]	-
GET /highscore/time	.byTime	-	<JSON.highscore[ ]	-
GET /highscore/runs	.byRuns	-	<JSON.highscore[ ]	-
GET /highscore/sql	.bySQL	-	<JSON.highscore[ ]	-
GET /highscore/rate	.byRate	-	<JSON.highscore[ ]	-

Hiermit können die Ranglisten in bestimmter Sortierung zurückgegeben werden. Die Rückgabe erfolgt im JSON-Format.

Schnittstelle $\langle I60 \rangle$: <Profildaten>

Action and URL	Method	Send	Response Success	Response Error
GET /profile	.index	-	JSON.playerstate	-
GET /profile/:id	.view	-	JSON.profile	-
GET /profile/character	.character	-	JSON.characterState	-
GET /profile/inventory	.inventory	-	JSON.inventory	-

Die Profildaten werden zu Beginn der Session, direkt nach dem login, vom Back-End angefordert. Das JSON-Objekt enthält den Benutzernamen, aktuelle Einstellungen, ob der User ein Student ist oder nicht, die aktuellen coins und wie nahe der User an den täglichen Limits für coins/scrolls ist. Es können auch die Profile anderer Benutzer betrachtet werden, dies erfolgt über die Profil-ID. Der /'characterState/' ist für das Minispiel von Relevanz, es wird ein JSON-Objekt übergeben, welches die Attribute, gekaufte Avatare und den aktuellen Avatar enthält.

Schnittstelle $\langle I60 \rangle$: <Shop>

Action and URL	Method	Send	Response Success	Response Error
GET /shop	.index	-	JSON.shopitem[ ]	-
POST /shop/:id	.buy	-	ok	400

Wenn der Benutzer auf den Shop klickt, holt sich das Front-End die gesamte Liste der Shopitems in Form eines JSON-Arrays, um sie für den Benutzer darzustellen.

3.3 Protokolle für die Benutzung der Komponenten

Abbildung 1.1 stellt ein Statechart für die Front-End-Komponente (C10) dar. Darauf folgt eine Beschreibung zur Verwendung der Komponente, sodass beides hier nicht mehr erfolgen muss.

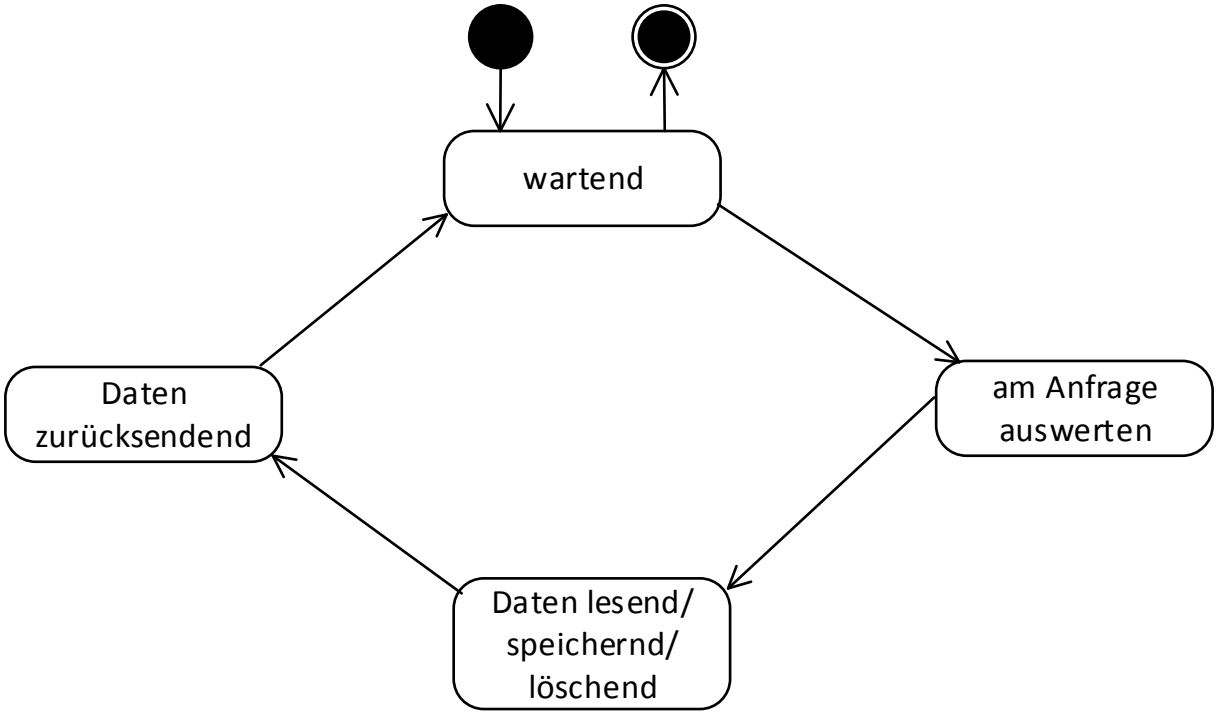

Abbildung 3.2: Back End - Statechart

In Abbildung 3.2 sind die Zustände des Back-Ends (C20) dargestellt. Das Back-End ist eine Datenbank, die auf Anfragen wartet. Erfolgt eine Anfrage an das Back-End, wird diese zuerst ausgewertet, um herauszufinden was getan werden soll. Ist dies geschehen, wird die geforderte Aktion umgesetzt und je nach Art der Aktion und Erfolg oder Misserfolg wird entweder der angeforderte Datensatz oder eine Mitteilung zurückgesendet.

Grundlegend sollte es möglich sein, sowohl das Front-End (C10), als auch das Back-End (C20) wiederzuverwenden. Dabei ist anzumerken, dass die im Front-End angezeigten Daten alle im Back-End lagern, wodurch eine gemeinsame Wiederverwendung am sinnvollsten erscheint. Das Austauschen des SQL-Trainers gegen einen anderen zu erlernenden Inhalt sollte auch möglich sein, wenn sich die Aufgaben in das hier verwendete Schwierigkeitsgrad-Schema (5 Schwierigkeitsgrade) einpassen lassen.

4 Verteilungsentwurf

Das Front-End der Software sowie das mit gelieferte Admin-Tool wird über einen Webbrowser auf dem PC des Benutzers angezeigt. Die Daten dafür sind auf einem Server gespeichert. Um eingegebene Nutzerdaten zu überprüfen wird desweiteren eine Verbindung zum LDAP der TU Braunschweig aufgebaut.

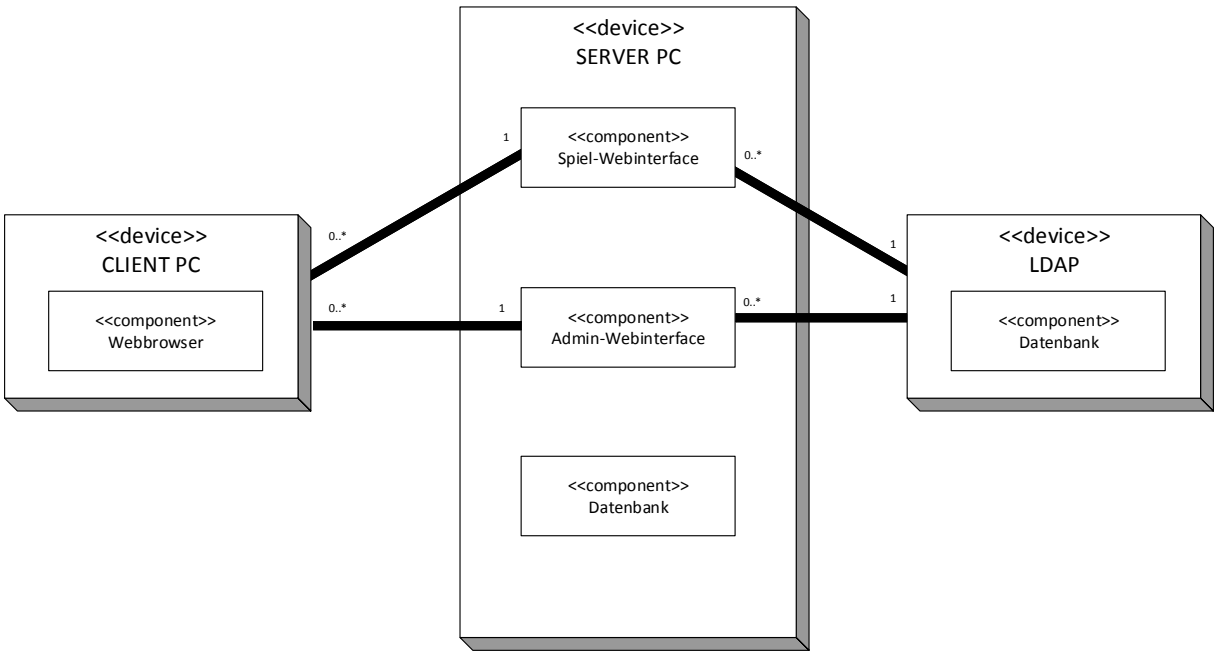

Abbildung 4.1: Verteilungsdiagramm für SQL-Alchemst

5 Implementierungsentwurf

5.1 Implementierung von Komponente <C10>: Das Front-End:

Im folgenden soll detailierter auf die Implementierung der Front-End-Komponente eingegenagen werden.

Das Front-End des SQL-Alchemisten beschreibt die Oberfläche der Applikation. Dabei wird darauf eingegangen, wie und an welcher Stelle das Front-End mit dem Back-End kommuniziert. Außerdem wird detailliert das Layout der Software-Komponente beschrieben.

Bei der Entwicklung der Kompotente wird auf das MelonJS-Framework zurückgegriffen. Dieses ist ein Game-Framework.

Das gesamte Front-End ist aus folgenden Objekten aufgebaut:

5.1.1 me.ScreenObject

In der Applikation befinden sich mehrere ScreenObjects. Diese stellen die einzelnen Bildschirme im Front-End dar. Jeder Screen ist an einen STATE gekoppelt. Wird ein STATE gewechselt, so wird der entsprechende Screen geladen. Jeder Screen enthält weitere Objekte die folgend erläutert werden. Dazu gehören Container, HUDs, Buttons und Sprites. Außerdem besitzt jeder Screen ein eigenes Hintergrundbild welches direkt am Anfang des STATE-Changes geladen wird. Des Weiteren übernehemen alle Screen Objekte durch eine EXTENDS-Beziehung folgende Funktionen:

- init() : Diese Funktion entspricht dem Konstruktor von Java-Klassen.
- onResetEvent() : Diese Funktion wird bei jedem Laden des Screens aufgerufen. Etwaige Veränderungen in den Attributen werden hierbei berücksichtigt und aktualisiert.
- onDestroyEvent() : Das DestroyEvent beendet alle Prozesse die von den Attributen gestartet worden sind.
- update() : Die Update-Funtktion fängt Veränderungen während der Laufzeit des States ab und agiert dementsprechend.

- draw() : Fügt zusätzliche Elemente zu die Screens hinzu die nicht zu den Attributen gehören. Das sind zum einen zum Beispiel Bilder oder Texte.

Folgende ScreenObjects existieren in der Applikation und sind vollständig durch die Extends-Beziehung von me.ScreenObject definiert. Diese sind in dem Digramm 5.1 unter game.ScreenObjekt zusammengefasst.

- game.StartScreen: Dieser Screen ist hat keine besonderen Funktionen und stellt in der Applikation nur einen Einführungs-Screen dar. Man gelangt von diesem Screen mit dem STATE_START-Zustand unmittelbar in den game.LoginScreen.
- game.ReadyScreen: Dieser Screen beschreibt das Labor im Story-Mode des Spiels und wird beim Wechseln in den READY-State aufgerufen.
- game.PlayScreen: Der PlayScreen repräsentiert das Minispiel mit den Runs im Dungeon. Der dazugehörige State ist der PLAY- State.
- game.BeltScreen: Dieser Screen soll den Tränke-Gürtel des Spielers repräsentieren. Hier werden dem Spieler all seine Potions angezeigt und es wird ihm die Möglichkeit gegeben diese in den Belt zu tun um sie im nächsten Run im Dungeon zu verwenden. Erreicht wird dieser Screen durch den Wechsel in den SSTATE_BELTSZustand.
- game.CollectorScreen: In diesem Screen werden dem Spieler, sofern der Zustandswechsel in den STATE_COLLECTOR-Zustand erfolgreich war, Informationen zu den eingesammelten Scrolls und Enchantmets, sowie dem für den Tag festgesetzten Scroll-Limit aufgezeigt.
- game.GameOverScreen: Dieser Screen ist dazu da, um dem Spieler eine Zusammenfassung seines getätigten Runs zu geben. Ihm werden die Anzahl der eingesammelten Scrolls und Enchantments, deren Namen, die Score und die Tiefe des Dungeons angezeigt. Der dazugehörige State ist in diesem Fall der GAMEOVER-State.
- game.TriviaScreen: Wählt der Spieler diesen Screen (mit dem STATE_TRIVIA-Zustand), so kann er SQL-Statements in verschiedenen Schwierigkeitsstufen lösen. Hat der Spieler einen Schwierigkeitsgrad ausgewählt, so wird er durch einen Zustandswechsel in den STATE_TASK-Zustand in den **??** Task-Screen umgeleitet.
- game.ResultScreen: Dieser Screen ist ähnlich wie der GameOverScreen. Hier bekommt der Spieler jedoch eine Zusammenfassung der Ergebnisse seines beantworteten SQL-Statements im **??** TaskScreen. Dieser Zustand lautet dementsprechend STATE_RESULT.
- game.HomeworkScreen: Dieser Screen (Zustand: STATE_HOMEWORK) steht dem Spieler zur Hausaufgabenbearbeitung zur Verfügung.
- game.SheetScreen: In diesem Screen (Zustand: STATE_SHEET) hat der Spieler die Möglichkeit die Spielfigur zu ändern.

- game.ShopScreen: Dieser Screen repräsentiert einen Shop in dem der Spieler zusätzliche Spielfiguren oder einen Beltslot dazukaufen kann.

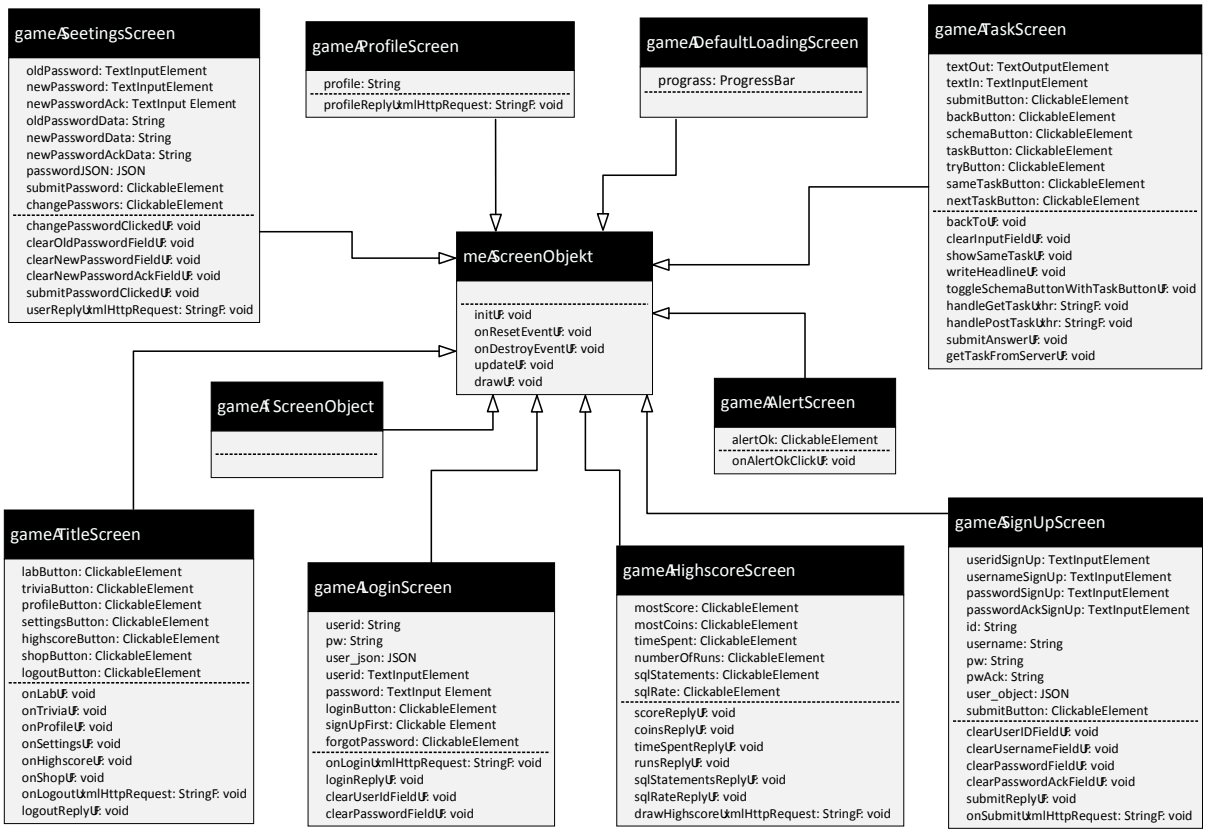

Abbildung 5.1: Klassendiagramm für die verschiedenen Screens

Des Weiteren gibt es folgende Screens mit zusätzlichen Funktionen.

5.1.2 game.AlertScreen

Der AlertScreen (STATE_ALERT) wird aufgerufen, sobald der User mit einem Fehler konfrontiert werden muss. Gibt der User zum Beispiel falsche Eingaben beim Login oder SignUp an, so wird er mittels dieses Screens darauf hingewiesen. Dazu besitzt er die Funktion OnAlertOkClick(). Wird der Alert zur Kenntnis genommen, schickt diese Funktion den Spieler in den letzten ausgewählten State und der automatische Reset dieses Screens wird ausgeführt.

5.1.3 game.HighscoreScreen

Als motivationsfördernden Faktor bietet die Applikation die Möglichkeit Ranglisten einzusehen. Diese befinden sich im HisghscoreScreen (Zustand: HIGHSCORE). Dieser Screen besitzt folgende Funktionen, die die jeweilige Highscore-Ajax-Anfrage an den Server abfängt:

- scoreReply();
- coinsReply();
- timeSpentReply();
- runsReply();
- sqlStatementsReply();
- sqlRateReply();

HINWEIS: Es handelt sich hierbei immer um GET-Anfragen, das heißt, bei diesen Anfragen kommen immer Informationen zurück ohne das neben der Anfrage selbst vorher Informationen an den Server mitgesendet werden müssen. Außerdem gibt besitzt der game.HighscoreScreen noch eine drawHighscore(xmlHttpRequest)-Funktion die als Parameter das zuvor abgefangene Ergebnis der Ajax-Anfrage bekommt und dieses via der draw()-Funktion visualisiert.

5.1.4 game.DefaultLoadingScreen

Der DefaultLoadingScreen ist ein visuell an das SQL-Alchemist-Thema angepasster Lade-Bildschirm. Dieser Screen besitzt das Element Progressbar und enthlt die Funktion progress(). progress() ist eine Funktion, ähnlich der update()-Funktion, mit dem prozentualen Fortschritt des Ladevorganges als Updatevariable, welche in game.js load() gestartet wird.

5.1.5 game.LoginScreen

Dieser Screen (STATE_LOGIN) bietet die Funktionalität des Einloggens in die Applikation. Außerdem wird dem Spieler hier die Möglichkeit gegeben durch einen entsprechenden Zustands-Wechsel in den STATE_SIGNUP in den game.SignUpScreen zu gelangen. Die zusätzlichen Funktionen sind:

- loginReply(): Diese Funktion führt den Login aus. Sie ist dafür zuständig die für den Login notwendigen Daten aus den dafür vorhandenen TextInputElemente (einfache Texteingabefelder) zu filtern, diese in ein JSON-Objekt zu parsen und per Ajax-POST-Anfgrage an den Server zu senden.
- onLogin(xmlHttpRequest): Die Funktion onLogin(xmlHttpRequest) bekommt als Parameter die Antwort der Ajax-Anfrage. Wenn dieser erfolgreich war ändert die Funktion den STATE_MENU und dergame.TitleScreen wird geladen.
- clearUserIdField() und clearPasswordField(): Diese Funktionen dienen rein optischen Zwecken. Sie sorgen dafür, dass der sogenannte Placeholder in den TextInputElements verschwindet und der User ungestört seine Login-Daten eingeben kann.

5.1.6 game.SignUpScreen

Dieser Screen (STATE_SIGNUP) bietet Funktionalität des Registrierens für die Applikation. Nach Erfolg gelangt man in den TItleScreen. Dafür notwendige Funktionen sind:

- submitReply() und onSubmit() sind für den SignUp äqiuvalente Funktionen zum game.LoginScreen .
- clearUserIdField(), clearUserNameField(), clearPasswordField() und clearPasswordAckField() sind ebenfalls äquivalente Funktionen zum Löschen der Placeholder.

5.1.7 game.SettingsScreen

Der SeetingsScreen bietet dem User die Möglichkeit seine Spieleinstellungen zu ändern. Der User kann Sound- und Musikeinstellungen ändern und außerdem auch noch sein Passwort sowohö zu ändern, als auch es zurückzusetzen. Um diese Funktionalität zu gewährleisten fibt es folgende Funktionen:

- changePasswordClicked(): Diese Funktion öffnet drei Textfelder: oldPassword, newPassword und newPasswordAck. Diese sind für eine erfolgreiche Passwortänderung nötig.
- submitPasswordClicked(): Diese Funktion kontrolliert das neue Passwört und dessen Bestätigung. Sollten die übereinstimmen wird userReply() aufgerufen.
- userReply(xmlHttpRequest): Diese Funktion sendet das neue Passwort per Ajax-Anfrage zum Server.

5.1.8 game.ProfileScreen

Im ProfileScreen werden dem User sämtliche Profilinformationen angezeigt. Dafür existiert die Funktion profileReply(), die alle entsprechenden Daten von dem Server bekommt. Dazu führt diese Funktion eine Ajax-.Anfrage aus. Bei Erfolg werden die Profilinformationen mittels draw()-Funktion visualisiert.

5.1.9 game.TaskScreen

Der TaskScreen repräsentiert in der Applikation den SQL-Trainer. für die Funktionalität existieren folgende Funktionen:

- getTaskFromServer(): Diese Funktion sendet die Ajax-Anfrage an den Server und führt im Anschluss writeHeadline() aus. Das Ergebnis der Anfrage wird an handleGetTask() als xhr Parameter übergeben.
- handleGetTask(xhr): Diese Funktion fängt das Ergebnis der Anfrage ab. Dieses enthält die zu lösende SQL-Aufgabe.
- writeHeadline(): Die Funktion schreibt die Überschrift basierend auf die Aufgabe in den Screen.
- toggleSchemaButtonWithTaskButton(): Mit Hilfe dieser Funktion kann der User zwischen der Anzeige des Aufgabentextes und des relationellen Schema wechseln.
- submitAnswer(): Diese Funktion sendet die Lösung des Users an den Server. Dies geschieht auch hier wieder über eine Ajax-Anfrage.
- clearInputField(): Diese Funktion löscht wie im game.LoginScreen beziehungsweise dem game.SignUpScreen bereits erwähnt den Placeholder des Texteingabefeldes.

5.1.10 game.TitleScreen

Der TitleScreen beschreibt die Menüführung der Applikation. Alle Buttons die einen durch einen Zustandswechsel in andere STATES und somit in andere Screens l eitet, sind durch verschiedene Bilder dargestellt. Wichtige Funktionen die hier verwendet werden sind:

- logoutReply(): Diese Funktion sendet die Ajax-POST-Anfrage zum Logout an den Server. Bei Erfolg wird die beim Login erstellte User Session gelöscht.
- onLogout(xmlHttpRequest): Diese Funktion fängt das Ergebnis der Anfrage ab und der User wird den LoginScreen geleitet.

5.1.11 me.GUI_Object

In den oben beschriebenen ScreenObjects befinden sich verschiedene Buttons. Die verwendeten Buttons stehen, wie in Abbildung 5.2 zu sehen is, mit einer EXTENDS-Beziehung zu me.GUI_Object in Verbindung. Alle Buttons erben von me.GUI_Object die Funktion onClick() die ausgeführt wird, wenn der Button geklickt wird. Alle Atrribute die an alle Buttons vererbt werden sind:

- settings.image: Diese Attribut enthält den String mit dem Name des Bildes welches in den Hintergrund des Buttons geladen wird.
- settings.spriteheight : Dieses Attribut definiert die Höhe des Buttons.
- settings.spritewidth: Dieses Attribut definiert dementsprechend die Breite des Buttons.
- z: Der z-Index beschreibt die Höhe des Layers im Canvas. Niedrigere z-Indizes werden hierbei immer von größeren überschrieben.

Folgende Buttons existieren in der Applikation und sind vollständig durch die Extends-Beziehung definiert:

- task: Dieser Button übergibt die Taskvariable difficulty und startet den STATE_TASK.
- skinLeft: Dieser Button befindet sich im SheetScreen und geht die Liste an vorhandenen Avataren nach links ab.
- skinRight: Dieser Button ist der entsprechenden Button der die Liste der Avatare nach rechts durchgeht.
- potionArrow: Dieser Button wird durch einen kleinen Pfeil dargestellt, besitzt das Attribut currentPotion und ist in der Lage den ausgewählten Potion in den ersten freien BeltSlot zu verschieben.
- beltSlotBelt: Dieser Button befindet sich im BeltScreen, besitzt ebenfalls das Attribut des zugehörigen Beltslot und hat besitzt die gegenteilige Funktion zum potion Arrow. Der Button verschiebt den Inhalt des Beltslots in die PotionCollection.
- showPotion: showPotion befindet sich im BeltScreen. Die onClick()-Funktion auf den entsprechenden Potion übergibt die Taskvariblen (potionId und difficulty) an den TaskScreen und startet diesen.
- next: next befindet sich im ResultScreen definiert eine einfache Weiterleitung in den TaskScreen.
- backToMenu: Dieser Button befindet sich in verschiedenen Screen. Möchte der User aus sämtlichen Screens zurück in das Menü gelangen, so ist diese Funktionalität durch diesen Button gegeben.

- backToLab: Dieser Button befindet sich ebenfalls auf verschiedenen Screens. durch einen Klick auf diesen, gelangt der User zurück in das Laboratory.

Außerdem existieren Buttons in der Applikation die zusätzliche update()-Funktion besitzen. Diese Funktion fängt ein Event ab und agiert dementsprechend. Die Events sind so definiert, dass sie durch Tastendruck aktiviert werden können.

Diese Buttons sind:

- exit: Dieser Button befindet sich im PlayScreen. Die zugewiesene Event-Taste ist die ESC-Taste. Diese beendet den Zustand PLAY und startet STATE_GAMEOVER.
- music: music befindet sich im PlayScreen mit dem Buchstaben M als EventTaste. Sie deaktiviert beziehungsweise reaktiviert die Hintergrundmusik des Minispiels.
- sound: sound befindet sich ebenfalls im PlayScreen und besitzt den Buchstaben N als EventTaste. Sie deaktiviert beziehungsweise reaktiviert die Soundeffekte des Minispiels.
- beltSlot: Dieser Button befindet sich im PlayScreen. Es besitzt das Attribut n mit 1,2,3,...,n der den beltSlot des Uses definiert. Die EventTasten werden von Q bis R dem entsprechenden Slots zugewiesen, sodass der User die eingebundenen Potions im Spiel benutzen kann.

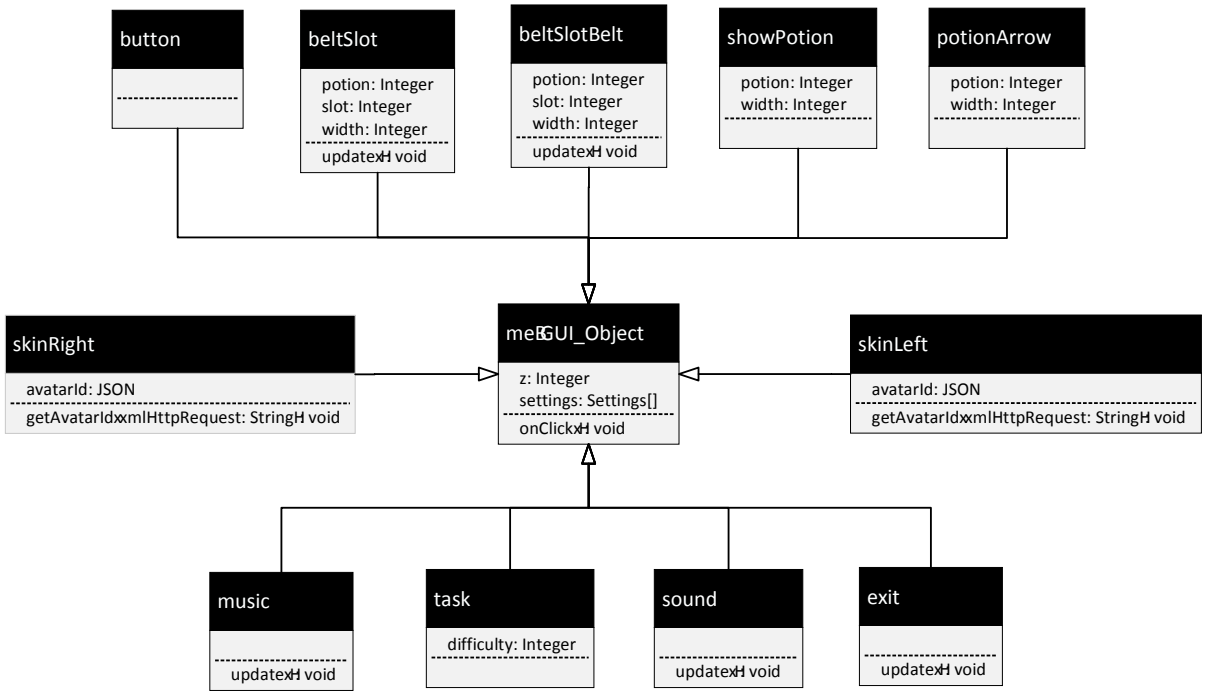

Abbildung 5.2: Klassendiagramm für die verschiedenen Screens

5.1.12 me.Entity

In dem Minispiel befinden sich diverse Entitäten. Diese stehen alle in einer EXTENDS-Beziehung mit me.Entity in Verbindung. me.Entity besitzt folgende Variablen die an alle Entitäten vererbt werden:

- name: Die Variable name vom Typ String beschreibt den Namen der Entität.
- body.collisionType: Dieses Attribut legt fest wie sich die Entitäten verhalten wenn sie kollidieren.
- body.velocity: Dieses Attribut besteht aus zwei Werten x und y vom Typ Integer. Diese beschreiben die Bewegung der Entität in horizontaler (x) beziehungsweise vertikaler (y) Ebene.
- alive: Die Variable alive vom Typ Boolean beschreibt, ob die Entity existiert. Der Default-Wert der Variable ist hierbei immer true.

Außerdem wird die Funktion init(), die dem Konstruktor in Java-Klassen entspricht, an die Entitäten vererbt. Sie definiert die Koordinaten und auf welcher Map die Entitäten erzeugt werden.

Die einzelnen Entitäten im Minispiel sind folgende:

5.1.13 game.PlayerEntity

Die PlayerEntity beschreibt die Spielfigur mit der der User durch das Dungeon läuft. Sie besitzt folgende zusätzliche Variablen und Funktionen:

- alwaysUpdate: Diese Variable vom Typ Boolean wird abgefragt, wenn sich der Spieler außerhalb des Viewports (Bildschirm) befindet. Ist dieser Wert true, wird die update()-Funktion der Entity ausgeführt.
- hurt: Dieses Attribut vom Typ Boolean wird genau dann auf true gesetzt wenn der Spieler mit einem Gegner kollidiert. Um ein erneutes Aufrufen der onCollision()-Funktion zu verhindern, wird nach einiger Zeit das hurt-Attribut auf false gesetzt.
- settings.image: Diese Variable vom Typ String definiert den Namen des Bildes der Spielfigur.
- update(dt): Die Funktion bekommt den Parameter dt übergeben. Dies ist eine Zeitvariable, die definiert in welchen Absänden die update-Funktion ausgeführt werden soll. Sie aktualisiert die Position der Figur abhängig von der body.velocity und fängt das Jump-Event ab (wenn die Spielfigur zum Springen gebracht wurde). Außerdem überprüft die Funktion ob die Hitbox der Spielfigur sich mit der einer anderen Entität schneidet. Sie gibt true zurück, wenn die Verschiebung gerendert werden konnte.
- onCollision(): Die Funktion analysiert wie und mit welcher Entität die PlayerEntity kollidiert ist und reagiert dementsprechend. Sie gibt true zurück, wenn die PlayerEntity mit einem festen Element wie den Böden, den Wänden und der Decke, kollidiert. Das Ergebnis ist false, wenn es stattdessen eine andere Entität war.

5.1.14 game.LevelEntity

Die game.LevelEntity beschreibt das Wechseln der Level am Ende jeder Map. Kollidiert die PlayerEntity mit der LevelEntity so ändert sich die Map. Die zusätzlichen Variablen und Funktionen, die die LevelEntity besitzt, sind:

- goToLevel: Die Variable vom Typ String beschreibt den Namen der nach der Kollision zu ladenden Map.
- settings.to: Diese Variabel vom Typ Integer übergibt die Schwierigkeit des nächsten Levels.
- settings.fade: Dieses Attribut vom Typ String beschreibt die Farbe des Fades in RGB-Werten.
- settings.duration: beschreibt die Länge des Fades in Millisekunden.
- findNextLevel(): Die Funktion sucht zufällig eine Map für das nächste Level eines gewissen Schwierigkeitsgrades aus. Sie gibt den Namen der nächsten Map als String. Dieser Name wird in goToLevel gespeichert.

5.1.15 game.CoinEntity

game.CoinEntity, ist eine "Collectable Entity". Sie erhöht den Score des Users und wird beim Kollidieren mit der PlayerEntity gelöscht. Die zusätzliche Funktion die nicht direkt von me.Entity geerbt wird, ist die onCollision()-Funktion. Sie wird durch das Kollidieren mit der PlayerEntity ausgelöst und erhöht den Score des Users.

5.1.16 game.SpikeEntity

Die game.SpikeEntity ist ebenfalls eine "Collectable Entity". Die beendet den Lauf im Dungeon beim Kollidieren mit der PlayerEntity. Auch diese Entität besitzt die zusätzliche onCollision()-Funktion. Diese beendet den Lauf des Spielers und führt einen Zustandswechsel in den GAMEOVER-State

5.1.17 game.ScrollEntity

Auch die game.ScrollEntity, ist eine "Collectable Entity". Der User sammelt diese ein und sie wird beim Kollidieren mit der PlayerEntity aus der Map entfernt. Die zusätlichen Variablen und Funktionen sind:

- settings.scrollIndex: Diese Variable enthält eine eindeutige ID der Scroll.
- settings.type: Diese Variable vom Typ String beschreibt den Typ der Scroll. Sie ist entweder eine "PotionScroll oder eine ËntchantentScroll
- onCollision(): Diese Funktion wird durch das Kollidieren mit der PlayerEntity ausgelöst. Sie speichert die Scroll für den User und entfernt die Entity.

5.1.18 game.EnemyEntity

Die game.EnemyEntity beschreibt alle Gegner-Entitäten die im Minispiel vorkommen. Die zusätzlichen Attribute und Funktionen die diese Entitäten besitzen sind:

- settings.spritewidth: Dieses Attribut vom Typ Integer beschreibt die Breite des Sprites der die Entität im Spiel darstellt.
- settings.spriteheight: Dieses Attribut beschreibt entsprechend zur Breite des Sprites Die Höhe.
- settings.jump: Das Attribut ist vom Typ Integer und definiert die Geschwindigkeit in vertikaler Ebene.
- settings.speed: Dieses Attribut beschreibt die Geschwindigkeit der Entität in horizontaler Ebene.
- settings.attack: Integer: Diese Eigenschaft definiert die Höhe des Schadens den der Spieler bekommt, wenn er mit der EnemyEntity kollidiert.
- startY: Dieses Attribut vom Typ Integer entspricht der y-Koordinate von me.Entity bei der Initiallisierung.

- startX: Dieses Attribut vom Typ Integer entspricht dementsprechend der x-Koordinate von me.Entity.
- pos.Y: In dieser Variable wird die aktuelle y-Koordinate der Entität gespeichert.
- pos.X: In dieser Variable wird dementsprechend die x-Koordinate der Entität gespeichert.
- endY: endY vom Typ Integer beschreibt die y-Koordinate, welche das Ende des Bereiches, in welchem sich die Gegner-Entität bewegen darf.
- endX: endX ist entsprechend zu endY die x-Koordinate des Bereichs in der sich die Entität bewegen darf.
- jump: Dieses Attribut vom Typ Boolean wird genau dann auf true gesetzt, wenn die Gegner-Entität spring beziehungsweise fliegt und false wenn sie fällt. Der Default-Wert dieser Variable ist false gesetzt.
- walkLeft: Dieses Attribut vom Typ Boolean wird auf Ist true gesetzt, wenn sich die Gegner-Entität nach links bewegt und false wenn sie nach rechts geht. Auch dieses Attribut ist stndartmäßig auf false gesetzt.
- update(): Diese Funktion läst die EnemyEntity nach links laufen und fliegen. Erreicht die Entität endY oder endX kehrt sie um läuft nach rechts und fällt bis zu startY und startX. Die Funktion gibt true zurück wenn das Rendern der Entität auf der neuen Position erfolgreich war.
- body.setVelocity(): Diese Funktion legt horizontale und vertikale Bewegungen fest.

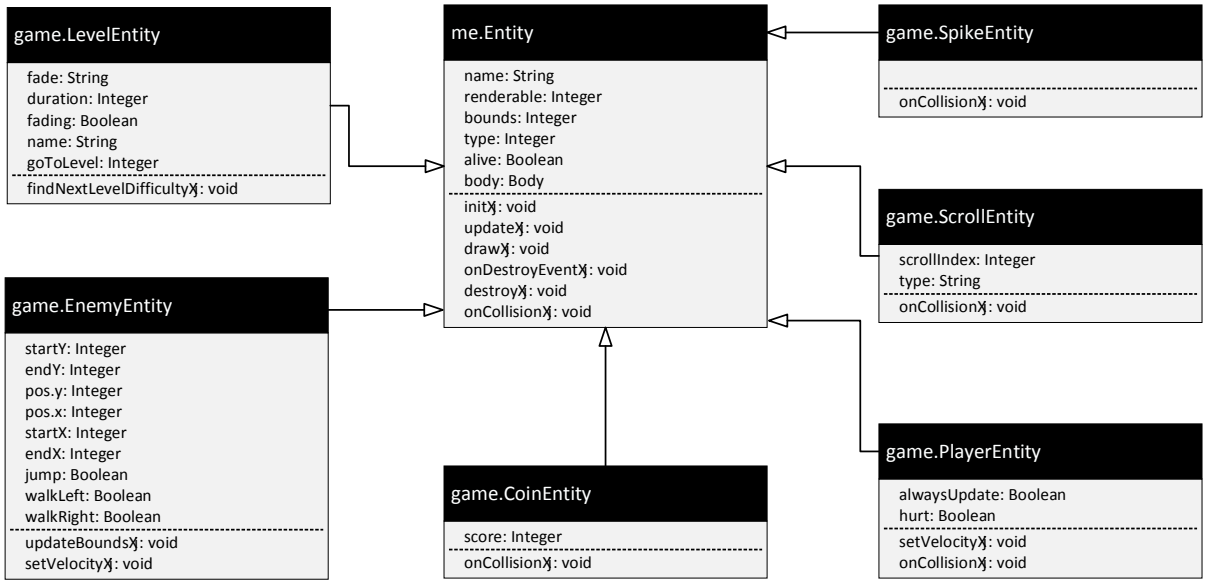

Abbildung 5.3: Klassendiagramm für die verschiedenen Front-End-Entitäten

5.1.19 me.Renderable

Alle HUDs (Head-Up-Displays) stehen in einer EXTENDS-Beziehung zu me.Renderable. HUDs werden in derApplikation dafür verwendet, Texte zu in die Screens zu schreiben und diese zur Laufzeit zu aktualisieren. Sie besitzen folgende Variablen und Funktionen:

- font: Das font-Attribut enthält dem Schrifttyp der in der draw()-Funktion benutzt wird.
- init(): Diese Funktion entspricht dem Konstruktor in Java und initialisiert das HUD.
- update(): Diese Funktion aktualisiert das HUD. Sie gibt true zurück, wenn der eingefügte Text erfolgreich gerendert werden konnte.
- draw(): Diese Funktion enthält den Text der auf das HUD geschrieben werden soll.

Alle HUDs die von me.Renderable erben sind:

- game.HUD.ScoreItem: Dieses HUD schreibt den Score des Spielers das im Minispiel angezeigt wird.
- game.HUD.HealthScore: Dieses HUD schreibt die Leben die der User aktuell im Minispiel besitzt.
- game.HUD.SkinName: Dieses HUD schreibt den Namen des entsprechenden, ausgewählten Avatares. Es wird im SheetScreen verwendet.
- game.HUD.SettingsElements: Dieses HUD schreibt die Einstellungen des User in den game.SettingsScreen.
- game.HUD.GameOver: Dieses HUD schreibt alle Informationen des GameOverScreens.
- game.HUD.Result: Dieses HUD schreibt den Inhalt des ResultScreens.
- game.HUD.Profile: Der Inhalt des game.ProfileScreen werden in diesem HUD geschrieben.
- game.HUD.PotionAmount: Dieses HUD schreibt die Anzahl der vorhanden Potions in den BeltScreen.
- game.HUD.Shop: Dieses HUD schreibt die Anzahl der Lofi-Coins in den ShopScreen.
- game.HUD.OverlayAlert: Dieses HUD schreibt die Fehlermeldungen in den AlertScreen.

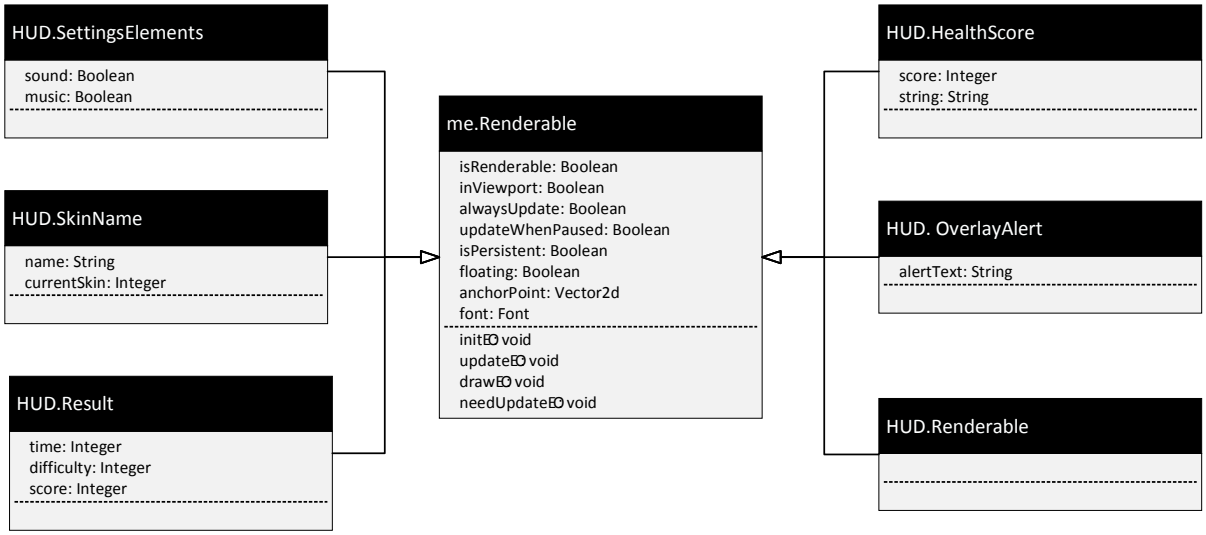

Abbildung 5.4: Klassendiagramm für die Renderables

5.2 Implementierung von Komponente <C20>: Das Back-End

Im folgenden Abschnitt soll detaillierter auf die Implementierung der Back-End-Komponente eingegangen werden.

Das Back-End des SQL-Alchemist hat die Aufgabe, die spielinterne Datenbank zu verwalten. Dabei müssen die anfallenden Daten korrekt gespeichert und dem Front-End auf Anfrage zur Verfügung gestellt werden.

Bei der Entwicklung dieser Komponente wird auf zwei Bibliotheken zurückgegriffen. Zunächst gehört dazu das play!-Framework, welches dazu dient webbasierte Anwendungen zu erstellen. Dabei stellt es zusätzlich auch die im Projekt verwendete Datenbank zur Verfügung. Die zweite verwendete Bibliothek wird vom Teamprojekt entwickelt und prüft SQL-Statements auf ihre Korrektheit. Dies wird beim SQL-Modul des SQL-Alchemisten Anwendung finden.

Im ersten Unterkapitel wird die Komponente durch Klassendiagramme dargestellt. Der Übersicht halber wurden die Klassen dabei auf mehrere Diagramme verteilt, da der Umfang der Komponente den Rahmen einer Seite übersteigen würde. Um den Zusammenhang der einzelnen Diagramme untereinander zu verdeutlichen werden Klassen, die Assoziationen zu Klassen in anderen Diagrammen haben, in diesen ebenfalls, aber ohne Attribute und Methoden, dargestellt.

Im zweiten Unterkapitel werden dann jeweils noch kurz die einzelnen Klassen, sowie deren Attribute und Methoden beschrieben.

5.2.1 Model Classes – Profile

5.2.2 Paket-/Klassendiagramm

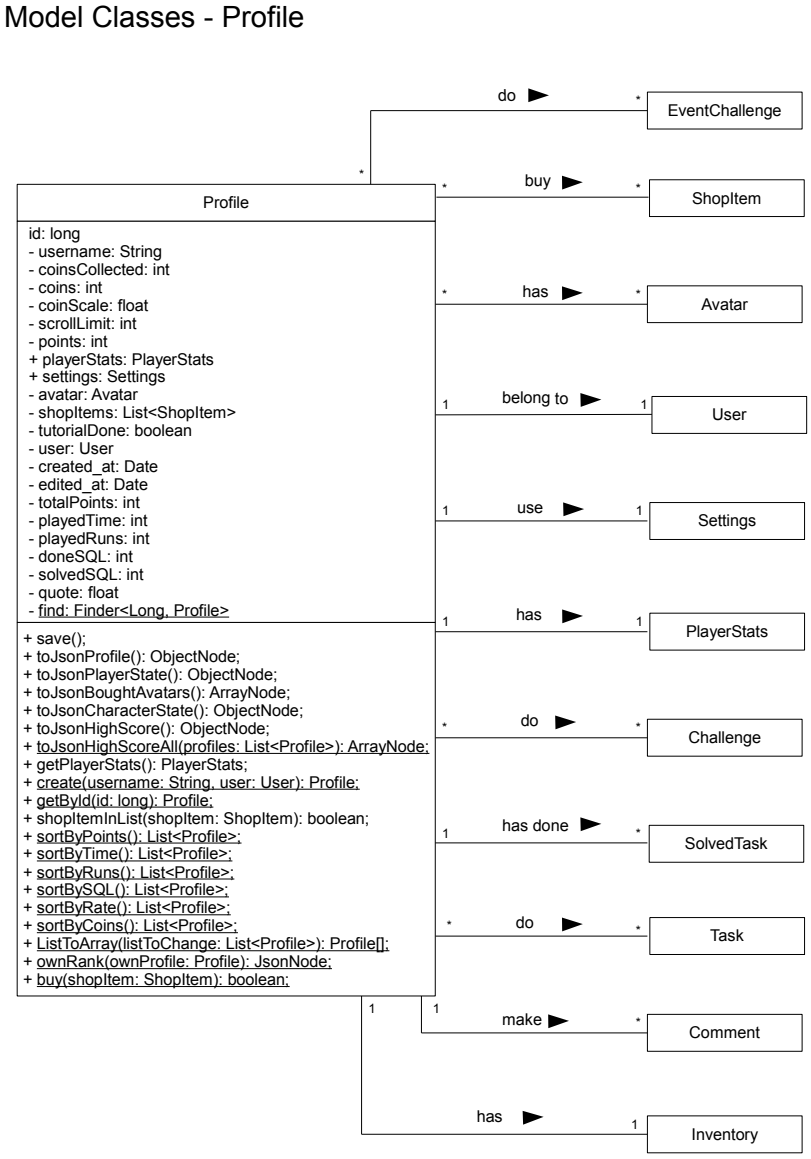

Abbildung 5.5: Klassendiagramm für die Model-Klasse Profile ⟨C20⟩

5.2.3 Erläuterung

Profile⟨$CL10$⟩

Aufgabe

Verwaltung der Nutzerprofile des SQL-Alchemist

Attribute

- id:long – Einzigartige ID
- username: String – Benutzername des zugehörigen Nutzers
- coinsCollected: int – Anzahl der bisher gesammelten Lofi-Coins
- coins: int – Anzahl der momentanen Lofi-Coins des Nutzers
- coinScale: float –
- scrollLimit: int – Anzahl der maximal sammelbaren Schriftrollen pro Zeitraum
- points: int – Die Punkte des Spielers
- playerStats: PlayerStats – Die Attribute der Spielfigur des Nutzers
- settings: Settings – Die vom Nutzer gewählten Einstellungen
- avatar: Avatar – Der vom Nutzer gewählte Avatar
- shopItems: List<ShopItem> – Bisher vom Nutzer gekaufte Gegenstände
- tutorialDone: boolean – Hält fest ob das Tutorial bereits vom Nutzer bearbeitet wurde
- user: User – Der zum Profil gehörige Nutzer
- created_at: Date – Zeitpunkt zu dem das Profil erstellt wurde
- edited_at: Date – Zeitpunkt zu dem das Profil zuletzt bearbeitet wurde
- totalPoints: int – Gesamtzahl der vom Nutzer erspielten Punkte
- playedTime: int – Bisher im Spiel verbrachte Zeit
- playedRuns: int – Bisher gespielte Durchläufe
- doneSQL: int – Bisher bearbeitete SQL-Aufgaben
- solvedSQL: int – Bisher korrekt gelöste SQL-Aufgaben
- quote: float – Rate von gelösten SQL-Aufgaben im Vergleich zu bearbeiteten Aufgaben
- find: Finder<Long, Profile> – Der Finder der Klasse zum Suchen in der Datenbank

Operationen

- save() – Diese Methode speichert das Profil
- toJsonProfile(): ObjectNode – Erzeugt aus einem Profildatensatz ein Json-Objekt
- toJsonPlayerState(): ObjectNode – Erzeugt aus einem Profildatensatz ein auf den PlayerState spezialisiertes Json-Objekt
- toJsonBoughtAvatars(): ArrayNode – Erzeugt ein Json-Array in welchem die gekauften Avatare gespeichert sind
- toJsonCharacterState(): ObjectNode – Erzeugt aus einem Profildatensatz ein auf den CharacterState spezialisiertes Json-Objekt
- toJsonHighScore(): ObjectNode – Erzeugt aus einem Profildatensatz ein für die Ranglisten spezialisiertes Json-Objekt
- toJsonHighScoreAll(profiles: List<Profile>): ArrayNode – Erstellt ein für die Ranglisten spezialisiertes Json-Array, welches mehrere Profileinträge enthält
- getPlayerStats(): PlayerStats – Kombiniert die PlayerStats des Avatars mit denen des Nutzers und gibt diese zurück
- create(username: String, user: User): Profile – Erstellt einen neuen Profildatensatz
- getById(id: long): Profile – Sucht ein Profil anhand dessen ID und gibt es zurück
- shopItemInList(shopItem): boolean – Prüft, ob ein ShopItem bereits gekauft wurde
- sortByPoints(): List<Profile> – Gibt eine Liste der, nach Punkten sortiert, zehn besten Profile zurück
- sortByTime(): List<Profile> – Gibt eine Liste der, nach Spielzeit sortiert, zehn besten Profile zurück
- sortByRuns(): List<Profile> – Gibt eine Liste der, nach Durchläufen sortiert, zehn besten Profile zurück
- sortBySQL(): List<Profile> – Gibt eine Liste der, nach gelösten SQL-Aufgaben sortiert, zehn besten Profile zurück
- sortByRate(): List<Profile> – Gibt eine Liste der, nach Erfolgsquote sortiert, zehn besten Profile zurück
- sortByCoins(): List<Profile> – Gibt eine Liste der, nach gesammelten Lofi-Coins sortiert, zehn besten Profile zurück

- ListToArray(listToChange: List<Profile>): Profile[] – Erzeugt ein Array aus einer Liste
- ownRank(ownProfile: Profile): JsonNode – Erzeugt ein für die Ranglisten spezialisiertes Json-Objekt
- buy(shopItem: ShopItem): boolean – Prüft, ob der Nutzer genügend Coins hat um ein ShopItem zu kaufen und führt dann den Kauf durch

Kommunikationspartner

- EventChallenge
- ShopItem
- Avatar
- User
- Settings
- PlayerStats
- Challenge
- SolvedTask
- Task
- Comment
- Inventory

5.2.4 Model Classes – Part 1

5.2.5 Paket-/Klassendiagramm

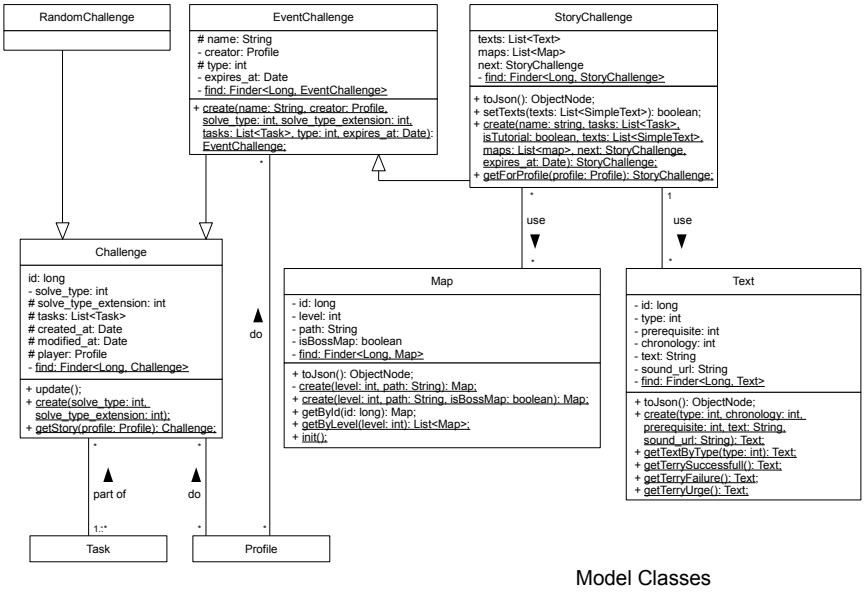

Abbildung 5.6: Klassendiagramm für einige der Model-Klassen $\langle C20 \rangle$

5.2.6 Erläuterung

Challenge⟨$CL20$⟩

Aufgabe

Verwaltung der Aufgabenpakete

Attribute

- id: long – Einzigartige ID
- solve_type: int – Beschreibt die Eigenschaften des Paketes hinsichtlich dessen Lösung
- solve_type_extension: int – Zusatz zu den Eigenschaften des Aufgabenpaketes
- tasks: List<Task> – Eine Liste der im Paket enthaltenen Aufgaben
- created_at: Date – Zeitpunkt zu dem das Aufgabenpaket erstellt wurde
- modified_at: Date – Zeitpunkt zu dem das Aufgabenpaket zuletzt geändert wurde
- player: Profile – Spieler, denen das Aufgabenpaket gestellt wird
- find: Finder<Long, Challenge> – Der Finder der Klasse zum Suchen in der Datenbank

Operationen

- update() – Aktualisiert das Aufgabenpaket
- create(solve_type: int, solve_type_extension: int) – Erstellt ein neues Aufgabenpaket
- getStory(profile: Profile): Challenge – Gibt die Story zurück

Kommunikationspartner

- Task
- Profile
- RandomChallenge
- EventChallenge

RandomChallenge⟨$CL30$⟩

Aufgabe

Verwaltet das zufällige Stellen von Aufgabenpaketen

Attribute

Keine

Operationen

Keine

Kommunikationspartner

- Challenge

EventChallenge⟨$CL40$⟩

Aufgabe

Verwaltung der Aufgabenpakete für spezielle Events

Attribute

- name: String – Name des Aufgabenpaketes
- creator: Profile – Der Nutzer, der das Paket erstellt hat
- type: int – Der Typ des Paketes
- expires_at: Date der Zeitpunkt, wann das Aufgabenpaket nicht mehr gelöst werden kann
- find: Finder<Long, EventChallenge> – Der Finder der Klasse zum Suchen in der Datenbank

Operationen

- create(name: String, creator: Profile, solve_type: int, solve_type_extension: int, tasks: List<Task>, type: int, expires_at: Date): EventChallenge – Erstellt ein neues Aufgabenpaket

Kommunikationspartner

- Profile
- Challenge
- StoryChallenge

StoryChallenge⟨$CL50$⟩

Aufgabe

Verwaltung der Aufgabenpakete für den Story-Modus

Attribute

- texts: List<Text> – Eine Liste der im Aufgabenpaket verwendeten Texte
- maps: List<Map> – Eine Liste der im Aufgabenpaket verwendeten Karten
- next: StoryChallenge – Das Aufgabenpaket, welches auf das aktuelle folgt
- find: Finder<Long, StoryChallenge> – Der Finder der Klasse zum Suchen in der Datenbank

Operationen

- toJson(): ObjectNode – Erstellt aus dem Paket ein Json-Objekt
- setTexts(texts: List<SimpleText>): boolean – Fügt dem Paket neue Texte hinzu
- create(name: String, tasks: List<Task>, isTutorial: boolean, texts: List<SimpleText>, maps: List<map>, next: StoryChallenge, expires_at: Date): StoryChallenge – Erstellt ein neues Aufgabenpaket
- getForProfile(profile: Profile): StoryChallenge – Weist das Aufgabenpaket einem Profil zu

Kommunikationspartner

- Map
- Text
- EventChallenge

Map⟨*CL*60⟩

Aufgabe

Verwaltung der Minispiel-Karten

Attribute

- id: long – Einzigartige ID
- level: int – Gibt den Level dar, in dem die Karte verwendet wird
- path: String – Der Pfad an dem die Karte abgespeichert ist
- isBossMap: boolean – Legt fest, ob die Karte eine Karte mit Endgegner ist
- find: Finder<Long, Map> – Der Finder der Klasse zum Suchen in der Datenbank

Operationen

- toJson(): ObjectNode – Erstellt aus dem Kartendatensatz ein Json-Objekt
- create(level: int, path: String): Map – Erstellt einen neuen Kartendatensatz
- create(level: int, path: String, isBossMap: boolean): Map – Erstellt einen neuen Kartendatensatz
- getById(id: long): Map – Sucht eine Karte anhand ihrer ID
- getByLevel(level: int): List<Map> – Sucht alle Karten, die in einem bestimmten Level vorkommen
- init() – Initialisiert die Karten

Kommunikationspartner

- StoryChallenge

Text⟨$CL70$⟩

Aufgabe

Verwaltung der Texte für den Story-Modus

Attribute

- id: long – Einzigartige ID
- type: int – Art des Textes
- prerequisite: int – Bedingung, die erfüllt sein muss, damit der Text aufgerufen wird
- chronology: int – Reihenfolge, in der die Texte auftreten
- text: String – Der eigentliche Text
- sound_url: String – Der zum Text gehörende Sound
- find: Finder<Long, Text> – Der Finder der Klasse zum Suchen in der Datenbank

Operationen

- toJson(): ObjectNode – Erzeugt aus dem Textdatensatz ein Json-Objekt
- create(type: int, chronology: int, prerequisite: int, text: String, sound_url: String): Text – Erstellt einen neuen Textdatensatz
- getTextByType(type: int): Text – Sucht einen zufälligen Text eines bestimmten Typs heraus und gibt ihn zurück
- getTerrySuccessfull(): Text – Sucht einen Text mit positiven Kommentar von Terry zurück
- getTerryFailure(): Text – Sucht einen Text mit negativen Kommentar von Terry zurück
- getTerryUrge(): Text – Sucht einen Text mit drängendem Kommentar von Terry zurück

Kommunikationspartner

- StoryChallenge

5.2.7 Model Classes – Part 2

5.2.8 Paket-/Klassendiagramm

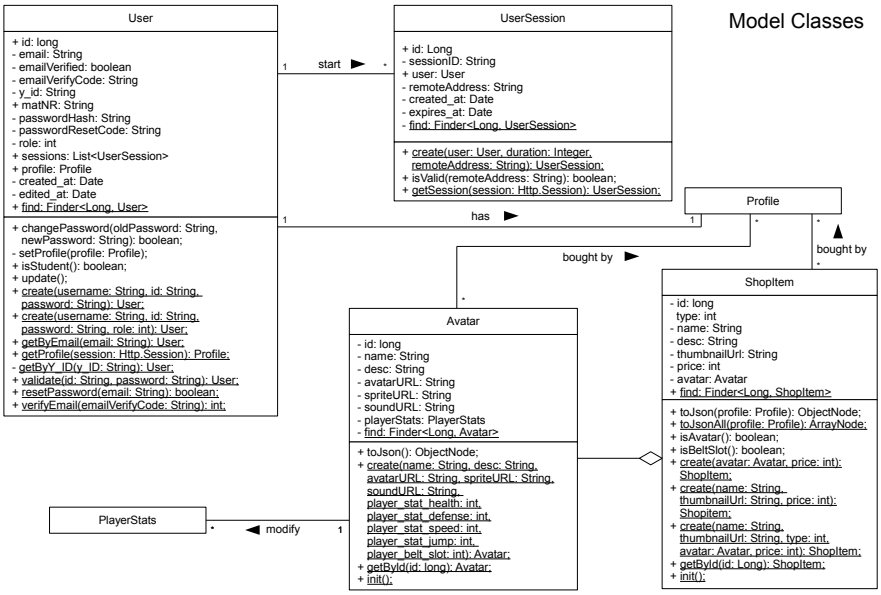

Abbildung 5.7: Klassendiagramm für einige der Model-Klassen $\langle C20 \rangle$

5.2.9 Erläuterung

User⟨*CL*80⟩

Aufgabe

Verwaltung der Nutzer

Attribute

- id: long – Einzigartige ID
- email: String – Email-Adresse des Nutzers
- emailVerified: boolean – Legt fest, ob die Adresse bereits verifiziert ist
- emailVerifyCode: String – Der Verifizierungscode der Adresse
- y_id: String – Die y-Nummer des Nutzers (falls der Nutzer ein Student ist)
- matNR: String – Die Matrikelnummer des Nutzers (falls der Nutzer ein Student ist)
- passwordHash: String – Der Passwort-Hash des Nutzers
- passwordResetCode: String – Der Code zum Zurücksetzen des Passworts
- role: int – Rechtegruppe, welcher der Nutzer angehört
- sessions: List<UserSession> – Liste an Sitzungen, die der Nutzer gestartet hat
- profile: Profile – Das Profil des Nutzers
- created_at: Date – Zeitpunkt zu dem sich der Nutzer registriert hat
- edited_at: Date – Zeitpunkt zu dem zuletzt etwas am Nutzerdatensatz geändert wurde
- find: Finder<Long, User> – Der Finder der Klasse zum Suchen in der Datenbank

Operationen

- changePassword(oldPassword: String, newPassword: String): boolean – Methode zum Ändern des Passworts
- setProfile(profile: Profile) – Weist dem Nutzer ein Profil zu
- isStudent(): boolean – Prüft, ob der Nutzer ein Student ist
- update() – Aktualisiert den Nutzerdatensatz
- create(username: String, id: String, password: String): User – Erstellt einen neuen Nutzerdatensatz

- create(username: String, id: String, password: String, role: int): User – Erstellt einen neuen Nutzerdatensatz
- getByEmail(email: String): User – Sucht einen Nutzer anhand seiner Email-Adresse
- getProfile(session: Http.Session): Profile – Gibt das zu einer Sitzung gehörende Profil zurück
- getByY_ID(y_ID: String): User – Sucht einen Nutzer anhand seiner y-Nummer
- validate(id: String, password: String): User – Validiert einen Nutzer
- resetPassword(email: String): boolean – Setzt das Passwort des Nutzers zurück
- verifyEmail(emailVerifyCode: String): int – Verifiziert eine Email-Adresse

Kommunikationspartner

- UserSession
- Profile

UserSession⟨*CL*90⟩

Aufgabe

Verwaltung der durch die Nutzer gestarteten Sitzungen

Attribute

- id: Long – Einzigartige ID
- sessionID: String – Einzigartige Sitzungs-ID
- user: User – Nutzer der sie Sitzung gestartet hat
- remoteAddress: String – Netzwerkkennung des Nutzers
- created_at: Date – Zeitpunkt zu dem die Sitzung gestartet wurde
- expires_at: Date – Zeitpunkt zu dem die Sitzung geschlossen wird
- find: Finder<Long, UserSession> – Der Finder der Klasse zum Suchen in der Datenbank

Operationen

- create(user: User, duration: Integer, remoteAddress: String): UserSession – Erstellt eine neue Sitzung
- isValid(remoteAddress: String): boolean – Prüft ob die Sitzung gültig ist
- getSession(session: Http.Session): UserSession – Gibt die Sitzung zurück

Kommunikationspartner

- User

Avatar⟨$CL100$⟩

Aufgabe

Verwaltung der Avatare

Attribute

- id: long – Einzigartige ID
- name: String – Name des Avatars
- desc: String – Beschreibung des Avatars
- avatarURL: String – Pfad, an dem der Avatar gespeichert wird
- spriteURL: String – Pfad, an dem der Sprite des Avatars abgespeichert wird
- soundURL: String – Pfad, an dem der Sound des Avatars abgespeichert wird
- playerStats: PlayerStats – Die Attribute des Avatars
- find: Finder<Long, Avatar> – Der Finder der Klasse zum Suchen in der Datenbank

Operationen

- toJson(): ObjectNode – Erzeugt aus dem Datensatz des Avatars ein Json-Objekt
- create(name: String, desc: String, avatarURL: String, spriteURL: String, soundURL: String, player_stat_health: int, player_stat_defense: int, player_stat_speed: int, player_stat_jump: int, player_belt_slot: int): Avatar – Erstellt einen neuen Avatar _getById(id: long): Avatar – Sucht einen Avatar anhand seiner ID _init() – Initialisiert die Avatare

Kommunikationspartner

- PlayerStats
- Profile
- ShopItem

ShopItem⟨*CL*110⟩

Aufgabe

Verwaltung der Shop-Gegenstände

Attribute

- id: long – Einzigartige ID
- type: int – Art des Gegenstandes
- name: String – Name des Gegenstandes
- desc: String – Beschreibung des Gegenstandes
- thumbnailUrl: String – Pfad an dem die Vorschaugrafik des Gegenstandes abgespeichert wird
- price: int – Kaufpreis des Gegenstandes
- avatar: Avatar – Der Avatar, um den es sich bei dem Gegenstand handelt (falls der Gegenstand ein Avatar ist)
- find: Finder<Long, ShopItem> – Der Finder der Klasse zum Suchen in der Datenbank

Operationen

- toJson(profile: Profile): ObjectNode – Erstellt aus dem Datensatz eines Shop-Gegenstands ein Json-Objekt
- toJsonAll(profile: Profile): ArrayNode – Erstellt ein Json-Array, welches Json-Objekte aller verfügbaren Gegenstände enthält
- isAvatar(): boolean – Prüft, ob es sich bei dem Gegenstand um einen Avatar handelt
- isBeltSlot(): boolean – Prüft, ob es sich bei dem Gegenstand um einen Gürtelplatz handelt
- create(avatar: Avatar, price: int): ShopItem – Erstellt einen neuen Shop-Gegenstand
- create(name: String, thumbnailUrl: String, price: int): ShopItem – Erstellt einen neuen Shop-Gegenstand
- create(name: String, thumbnailUrl: String, type: int, avatar: Avatar, price: int): ShopItem – Erstellt einen neuen Shop-Gegenstand
- getById(id: Long): ShopItem – Sucht einen Gegenstand anhand seiner ID
- init() – Initialisiert die Shop-Gegenstände

Kommunikationspartner

- Profile
- Avatar

5.2.10 Model Classes – Part 3

5.2.11 Paket-/Klassendiagramm

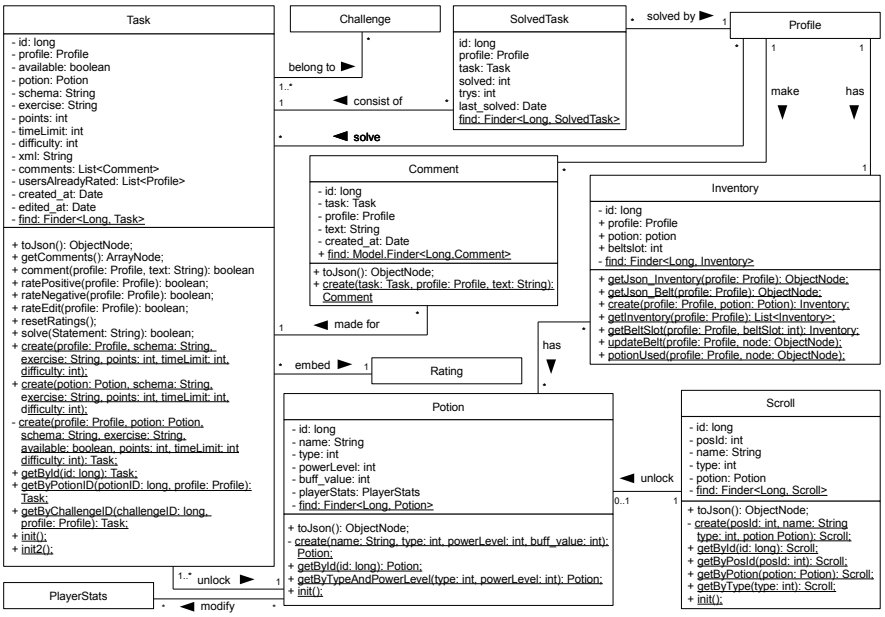

Abbildung 5.8: Klassendiagramm für einige der Model-Klassen $\langle C20 \rangle$

5.2.12 Erläuterung

Task⟨$CL120$⟩

Aufgabe

Verwaltung der Aufgaben inkl. Erstellung und Bewertung

Attribute

- id: long – Identität zur eindeutigen Identifizierung der Aufgabe
- profile: Profile – Profil, das der Aufgabe zugeordnet ist
- available: boolean – Verfügbarkeit der Aufgabe für den User
- potion: Potion – Trank, den der Spieler durch die erfolgreiche Bearbeitung der Aufgabe erhält
- schema: String – Verwendetes Datenbankschema
- exercise: String – Aufgabenstellung
- points: int – Punkte der Aufgabe
- timeLimit: int – Zeitlimit der Aufgabe
- difficulty: int – Schwierigkeit der Aufgabe
- xml: String – XML-Datei, die die Aufgabe enthält
- comments: List<Comment> – Liste mit Kommentaren, die Nutzer für diese Aufgabe abgegeben haben
- usersAlreadyRated: List<Profile> – Liste mit den Profilen der Nutzer, die diese Aufgabe bereits bewertet haben
- created_at: Date – Erstellungsdatum
- edited_at: Date – Datum der letzten Bearbeitung
- find: Finder<Long, Task> – Finder zum Suchen von Objekten in der Datenbank

Operationen

- toJson(): ObjectNode – Erzeugung und Rückgabe der Aufgabe als Json-Objekt
- getComments(): ArrayNode – Rückgabe der Kommentare als JsonNode-Objekte
- comment(profile: Profile, text: String): boolean – Bewerten einer Aufgabe. Das Profil sowie der Kommentar als string werden übergeben

- ratePositive(profile: Profile): boolean – Aufgabe positiv bewerten
- rateNegative(profile: Profile): boolean – Aufgabe negativ bewerten
- rateEdit(profile: Profile): boolean – Aufgabe zur Nachbearbeitung markieren
- resetRatings() – Zurücksetzen der Bewertungen
- solve(Statement: String): boolean – Aufgabe lösen
- create(profile: Profile, schema: String, exercise: String, points: int, timeLimit: int, difficulty: int) – Aufgabe erstellen
- create(potion: Potion, schema: String, exercise: String, points: int, timeLimit: int, difficulty: int) – Aufgabe erstellen
- create(profile: Profile, potion: Potion, schema: String, exercise: String, available: boolean, points: int, timeLimit: int difficulty: int): Task – Aufgabe erstellen
- getById(id: long): Task – Rückgabe der Aufgabe der entsprechenden Identität
- getByPotionID(potionID: long, profile: Profile): Task – Rückgabe der Aufgabe anhand des Tranks
- getByChallengeID(challengeID: long, profile: Profile): Task – Rückgabe der Aufgabe anhand der ChallengeID
- init() – Initiierung der Aufgaben
- init2() – Initiierung der Aufgaben

Kommunikationspartner

- SolvedTask
- Comment
- Challenge
- Rating
- Potion
- Profile

SolvedTask⟨$CL130$⟩

Aufgabe

Verwaltung der gelösten Aufgaben

Attribute

- id: long – Identität zur eindeutigen Identifizierung des Objekts
- profile: Profile – Profil des Users, der die Aufgabe gelöst hat
- task: Task – Verweis auf die Aufgabe
- solved: int – Anzahl, wie oft der User diese Aufgabe bereits gelöst hat
- trys: int – Anzahl der Versuche, die nötig waren, die Aufgabe zu lösen
- last_solved: Date – Datum des letzten richtigen Lösens der Aufgabe
- find: Finder<Long, SolvedTask> – Finder zum Suchen von Objekten in der Datenbank

Operationen

Keine

Kommunikationspartner

- Task
- Profile

Comment⟨$CL140$⟩

Aufgabe

Verwaltung der Kommentare zu den Aufgaben

Attribute

- id: long – Identität zur eindeutigen Identifizierung des Objekts
- task: Task – Die zum Kommentar zugehörige Aufgabe
- profile: Profile – Profil des Users, der die Bewertung abgegeben hat
- text: String – Kommentarinhalt
- created_at: Date – Erstellungsdatum des Kommentars
- find: Finder<Long, Comment> – Finder zum Suchen von Objekten in der Datenbank

Operationen

- toJson(): ObjectNode – Rückgabe des Objekts als JsonNode
- create(task: Task, profile: Profile, text: String): Comment – Erstellung eines Kommentars

Kommunikationspartner

- Task
- Profile

Potion⟨$CL150$⟩

Aufgabe

Verwaltung der Tränke

Attribute

- id: long – Identität zur eindeutigen Identifizierung des Objekts
- name: String – Name des Tranks
- type: int – Art des Tranks
- powerLevel: int – Stärke des Tranks
- buff_value: int – Wert, um den das entsprechende Attribut verändert wird
- playerStats: PlayerStats – Objekt mit den Eigenschaften des Avatars
- find: Finder<Long, Potion> – Finder zum Suchen von Objekten in der Datenbank

Operationen

- toJson(): ObjectNode – Rückgabe des Objekts als JsonNode
- create(name: String, type: int, powerLevel: int, buff_value: int): Potion – Erstellen neuer Tränke
- getById(id: long): Potion – Rückgabe eines Tranks anhand der ID
- getByTypeAndPowerLevel(type: int, powerLevel: int): Potion – Rückgabe des Tranks anhand des Typs und des PowerLevels
- init() – Initiierung der Tränke

Kommunikationspartner

- Task
- PlayerStats
- Scroll
- Inventory

Inventory⟨*CL*160⟩

Aufgabe

Inventar des Spielers

Attribute

- id: long – Identität zur eindeutigen Identifizierung des Objekts
- profile: Profile – Profil des Users
- potion: potion – Trank des Users
- beltslot: int – Gürtelslot
- find: Finder<Long, Inventory> – Finder zum Suchen von Objekten in der Datenbank

Operationen

- getJson_Inventory(profile: Profile): ObjectNode – Rückgabe des Inventars als JSonNode-Objekt
- getJson_Belt(profile: Profile): ObjectNode – Rückgabe des Gürtels als JSonNode-Objekt
- create(profile: Profile, potion: Potion): Inventory – Trank dem Inventar hinzufügen
- getInventory(profile: Profile): List<Inventory> – Rückgabe der Inventare
- getBeltSlot(profile: Profile, beltSlot: int): Inventory – Rückgabe des Gürtelslots
- updateBelt(profile: Profile, node: ObjectNode) – Update des Gürtels
- potionUsed(profile: Profile, node: ObjectNode) – Benutzte Tränke aus dem Inventar löschen

Kommunikationspartner

- Task
- PlayerStats
- Scroll

Scroll⟨$CL170$⟩

Aufgabe

Verwaltung der im Spiel verwendeten Schriftrollen

Attribute

- id: long – Identität zur eindeutigen Identifizierung des Objekts
- posId: int – Positions-ID
- name: String – Name der Schriftrolle
- type: int – Art der Schriftrolle
- potion: Potion – Angabe des Tranks, der durch die Schriftrolle freigeschaltet werden kann
- find: Finder<Long, Scroll> – Finder zum Suchen von Objekten in der Datenbank

Operationen

- toJson(): ObjectNode – Rückgabe der Schriftrolle als Json-Objekt
- create(posId: int, name: String type: int, potion Potion): Scroll – Schriftrolle erstellen
- getById(id: long): Scroll – Rückgabe der Schriftrolle mit der entsprechenden ID
- getByPosId(posId: int): Scroll – Rückgabe der Schriftrolle entsprechend der Positions-ID im Spiel
- getByPotion(potion: Potion): Scroll – Rückgabe der Schriftrolle entsprechend des freischaltbaren Tranks
- getByType(type: int): Scroll – Rückgabe der Schriftrolle entsprechend des Typs
- init() – Initiierung der Schriftrollen

Kommunikationspartner

- Potion

5.2.13 Helper and Secured Classes

5.2.14 Paket-/Klassendiagramm

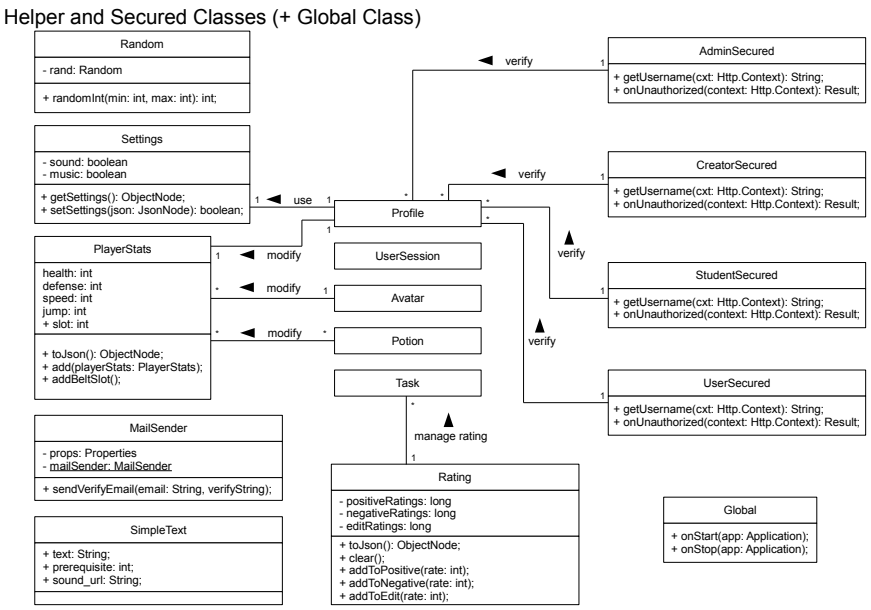

Abbildung 5.9: Klassendiagramm für die Helper- und Secured-Klassen $\langle C20 \rangle$

5.2.15 Erläuterung

Random⟨$CL180$⟩

Aufgabe

Generierung von Zufallszahlen

Attribute

- rand: Random – Instanz der Klasse java.util.Random()

Operationen

- randomInt(min: int, max: int): int – Generierung einer Zufallszahl im angegebenen offenen Intervall

Kommunikationspartner

Keine

Settings⟨$CL190$⟩

Aufgabe

Verwaltung der Benutzereinstellungen

Attribute

- sound: boolean – Soundeffekte sind entweder ein- oder ausgeschaltet
- music: boolean – Hintergrundmusik ist entweder ein- oder ausgeschaltet

Operationen

- getSettings(): ObjectNode – Rückgabe der Einstellungen als Json-Objekt
- setSettings(json: JsonNode): boolean – Einstellungen setzen

Kommunikationspartner

- Profile

PlayerStats⟨*CL*200⟩

Aufgabe

Statistiken des Spielers

Attribute

- health: int – Gesundheit
- defense: int – Verteidigung
- speed: int – Geschwindigkeit
- jump: int – Sprungkraft
- slot: int – Anzahl der Taschenslots

Operationen

- toJson(): ObjectNode – Rückgabe der Statistik als Json-Objekt
- add(playerStats: PlayerStats) – Hinzufügen der Statistik
- addBeltSlot() – Taschenslot hinzufügen

Kommunikationspartner

- Profile
- Avatar
- Potion

MailSender⟨*CL*210⟩

Aufgabe

Versenden von Emails

Attribute

- props: Properties – Properties-Objekt mit den Einstellungen
- mailSender: MailSender – Instanz der Klasse für die gesamte Anwendung

Operationen

- sendVerifyEmail(email: String, verifyString) – Versenden der Verifizierungsnachricht

Kommunikationspartner

Keine

SimpleText⟨$CL220$⟩

Aufgabe

Verwaltung der Texte

Attribute

- text: String – Text
- prerequisite: int – Voraussetzung
- sound_url: String – Referenz auf den Sound

Operationen

Keine

Kommunikationspartner

Keine

Rating⟨$CL230$⟩

Aufgabe

Bewertung der Aufgaben

Attribute

- positiveRatings: long – Anzahl der positiven Bewertungen
- negativeRatings: long – Anzahl der negativen Bewertungen
- editRatings: long – Anzahl, wie oft die Aufgabe zur Nachbearbeitung empfohlen wurde

Operationen

- toJson(): ObjectNode – Rückgabe als Json-Objekt
- clear() – Bewertung zurücksetzen
- addToPositive(rate: int) – Positive Bewertung hinzufügen
- addToNegative(rate: int) – Negative Bewertung hinzufügen
- addToEdit(rate: int) – Anzahl der Nutzer, die die Aufgabe zur Überarbeitung empfohlen haben

Kommunikationspartner

- Task

AdminSecured⟨$CL240$⟩

Aufgabe

Berechtigung für Admintool

Attribute

Keine

Operationen

- getUsername(cxt: Http.Context): String – Rückgabe des Benutzernamens
- onUnauthorized(context: Http.Context): Result – Result-Objekt mit einer Fehlermeldung bei unberechtigem Zugriff

Kommunikationspartner

- UserSession

CreatorSecured⟨$CL250$⟩

Aufgabe

Berechtigung geschützten Bereich

Attribute

Keine

Operationen

- getUsername(cxt: Http.Context): String – Rückgabe des Benutzernamens
- onUnauthorized(context: Http.Context): Result – Result-Objekt mit einer Fehlermeldung bei unberechtigem Zugriff

Kommunikationspartner

- UserSession

StudentSecured⟨$CL260$⟩

Aufgabe

Berechtigung geschützten Bereich

Attribute

Keine

Operationen

- getUsername(cxt: Http.Context): String – Rückgabe des Benutzernamens
- onUnauthorized(context: Http.Context): Result – Result-Objekt mit einer Fehlermeldung bei unberechtigem Zugriff

Kommunikationspartner

- UserSession

UserSecured⟨*CL*270⟩

Aufgabe

Berechtigung geschützten Bereich

Attribute

Keine

Operationen

- getUsername(cxt: Http.Context): String – Rückgabe des Benutzernamens
- onUnauthorized(context: Http.Context): Result – Result-Objekt mit einer Fehlermeldung bei unberechtigem Zugriff

Kommunikationspartner

- UserSession

Global⟨$CL280$⟩

Aufgabe

Verwaltung der Methoden zum Starten und Beenden

Attribute

Keine

Operationen

- onStart(app: Application) – Aufruf beim Starten
- onStop(app: Application) – Aufruf beim Beenden

Kommunikationspartner

Keine

Application⟨$CL290$⟩

Aufgabe

Die Methoden dieser Klasse liefern die Content-Seiten mit einem Authorisierungsschlüssel an den Benutzer. Der Schlüssel legt dabei die Berechtigungen fest

Attribute

Keine

Operationen

- admin(): Result – Rückgabe eines Result-Objekts mit dem Inhalt Ädmin"
- init(): Result – Initiierungsmethode

Kommunikationspartner

- All model classes

5.2.16 Controller Classes

5.2.17 Paket-/Klassendiagramm

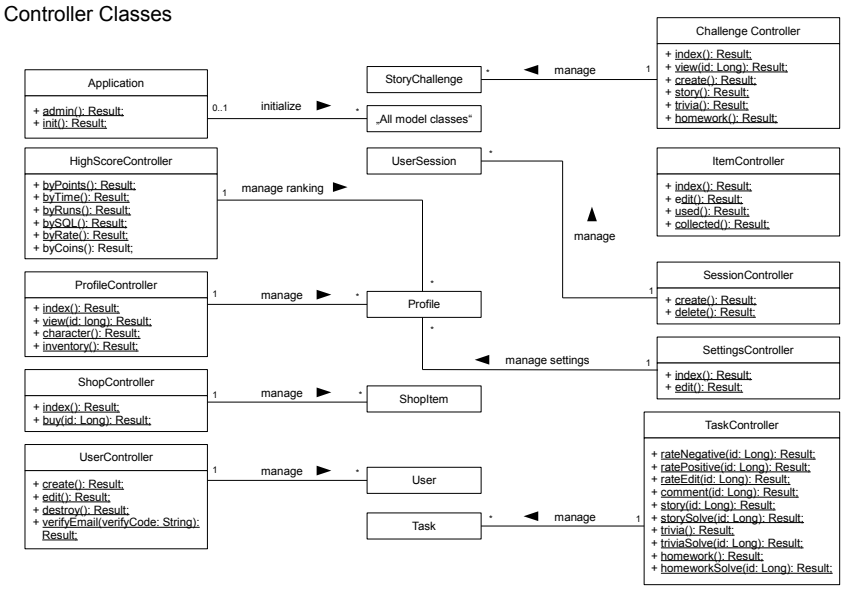

Abbildung 5.10: Klassendiagramm für die Controller-Klassen $\langle C20 \rangle$

5.2.18 Erläuterung

HighScoreController⟨$CL300$⟩

Aufgabe

Methoden zur Sortierung der HighScore-Listen

Attribute

Keine

Operationen

- byPoints(): Result – Sortierung nach Punkten
- byTime(): Result – Sortierung nach Zeit
- byRuns(): Result – Sortierung nach Runs
- bySQL(): Result – Sortierung nach gelösten SQL-Aufgaben
- byRate(): Result – Sortierung nach Bewertung
- byCoins(): Result – Sortierung nach Münzen

Kommunikationspartner

- Profile

ProfileController⟨$CL310$⟩

Aufgabe

Methoden zur Benutzung der Profile

Attribute

Keine

Operationen

- index(): Result – Rückgabe der Spielstatistik als Json-Objekt
- view(id: long): Result – Rückgabe des Profils entsprechend der ID
- character(): Result – Rückgabe des Charakterzustands als Json-Objekt
- inventory(): Result – Rückgabe des Inventars

Kommunikationspartner

- Profile

ShopController⟨*CL*320⟩

Aufgabe

Methoden zur Benutzung der Objekte aus dem Shop

Attribute

Keine

Operationen

- index(): Result – Rückgabe aller Shop-Gegenstände als Result-Objekt
- buy(id: Long): Result – Kauf eines Gegenstands

Kommunikationspartner

- ShopItem

UserController⟨$CL330$⟩

Aufgabe

Methoden zur Benutzung des Benutzerobjekts

Attribute

Keine

Operationen

- create(): Result – Erstellen eines Benutzerobjekts
- edit(): Result – Ändern des Passworts
- destroy(): Result – Löschen des Benutzerobjekts
- verifyEmail(verifyCode: String) – Emailverifizierung Result;

Kommunikationspartner

- User

ChallengeController⟨$CL340$⟩

Aufgabe

Methoden zur Benutzung des Herausforderungsobjekts

Attribute

Keine

Operationen

- index(): Result – Rückgabe der Herausforderungen
- view(id: Long): Result – Ansicht der Herausforderung
- create(): Result – Erstellen der Herausforderung
- story(): Result – Herausforderung dem Profil zuordnen
- trivia(): Result – Rückgabe einer Triviaaufgabe
- homework(): Result – Rückgabe einer Hausaufgabe

Kommunikationspartner

- StoryChallenge

ItemController⟨$CL350$⟩

Aufgabe

Methoden zur Benutzung der Gegenstände

Attribute

Keine

Operationen

- index(): Result – Rückgabe der Gegenstände
- edit(): Result – Gegenstand bearbeiten
- used(): Result – Gegenstand benutzt
- collected(): Result – Gegenstand aufgesammelt

Kommunikationspartner

SessionController⟨$CL360$⟩

Aufgabe

Methoden zur Verwaltung der aktuellen Sitzung

Attribute

Keine

Operationen

- create(): Result – Sitzung erstellen
- delete(): Result – Sitzung löschen

Kommunikationspartner

- UserSession

SettingsController⟨*CL*370⟩

Aufgabe

Methoden zur Verwaltung der aktuellen Einstellungen

Attribute

Keine

Operationen

- index(): Result – Rückgabe der Einstellungen
- edit(): Result – neue Einstellungen übernehmen

Kommunikationspartner

- Profile

TaskController⟨$CL380$⟩

Aufgabe

Methoden zur Verwaltung der aktuellen Aufgabe

Attribute

Keine

Operationen

- rateNegative(id: Long): Result – Aufgabe positiv bewerten
- ratePositive(id: Long): Result – Aufgabe negativ bewerten
- rateEdit(id: Long): Result – Aufgabe zur Nachbearbeitung markieren
- comment(id: Long): Result – Aufgabe kommentieren
- story(id: Long): Result – Aufruf der Aufgabe aus dem Story-Modus
- storySolve(id: Long): Result – Lösen der Aufgabe
- trivia(): Result – Aufruf der Trivia-Aufgabe
- triviaSolve(id: Long): Result – Lösen einer Trivia-Aufgabe
- homework(): Result – Aufruf der Hausaufgabe
- homeworkSolve(id: Long): Result – Lösen der Hausaufgabe

Kommunikationspartner

- Task

6 Datenmodell

Das folgende Kapitel beschreibt die Datensätze, welche der SQL-Alchemist dauerhaft oder teilweise auch nur temporär abspeichert. Dazu werden zuerst einige Erläuterungen zu den einzelnen Beziehungen angegeben und zum Ende des Kapitels ist ein Klassen-Diagramm zur Veranschaulichung dieses Sachverhaltes abgebildet.

6.1 Erläuterung

Avatar $\langle E10 \rangle$

Beziehung	Kardinalität	Erwartete Datenmenge	Beschreibung
Profile	0 ... *	Min: 1kB, Max: 100kB	Der Avatar wird vom Profil als Erkennungsmerkmal und Spielfigur verwendet.
Profile_Avatar	1 ... *	-	Verweis auf alle vom Nutzer gekauften Avatare.

Die Entitäten vom Typ „Avatar“ beschreiben die verschiedenen vom Spiel zur Verfügung gestellten Spielfiguren. Diese werden sowohl im Minispiel als auch als Erkennungsmerkmal der Profile der Nutzer verwendet. In der Datenbank werden unter anderem die zugehörigen Eigenschaften, sowie die Darstellungsmerkmale gespeichert. **Hinweis:** In der Relation Profile_Avatar sind die Avatare mit den Profilen verknüpft, das Profil verfügt jedoch zusätzlich über das Attribut „Avatar“. Der im Profil gespeicherte Avatar ist dabei der aktuell verwendete.

Bag ⟨*E*20⟩

Beziehung	Kardinalität	Erwartete Datenmenge	Beschreibung
Potion	1	Min: 1 kB, Max: 16 kB	In Bagslots werden die bereits gesammelten Tränke gespeichert.
Profile	1	Min: 1 kB, Max: 100 kB	Ein Bagslot gehört einem Profil.

Bei den Entitäten des Typs „Bag“ handelt es sich um die einzelnen Plätze innerhalb des Inventars der Nutzer, in denen alle hergestellten Tränke abgelegt werden. Dabei nimmt jeder Slot genau einen Trank auf.

Challenge ⟨*E*30⟩

Beziehung	Kardinalität	Erwartete Datenmenge	Beschreibung
Map_In_Challenge	0 ... *	-	Eine Challenge kann mit mehreren Maps verknüpft sein.
Text_In_Challenge	0 ... *	-	Story-Texte können Challenges zugeordnet werden, um diese in den Kontext der Handlung einzubinden.
Task_In_Challenge	1 ... *	8 B	Eine Challenge besteht aus mindestens einer Aufgabe.

Die „Challenge“-Objekte beschreiben die Aufgabenpakete, welche zum Beispiel im Laufe der Story oder als Hausaufgaben an die Nutzer gestellt werden. Jedes dieser Pakete besteht aus verschiedenen Aufgaben. Als Attribute werden alle Informationen, die zur Beschreibung eines solchen Paketes benötigt werden in der Datenbank gespeichert.

Map $\langle E40 \rangle$

Beziehung	Kardinalität	Erwartete Datenmenge	Beschreibung
Map_In_Challenge	0 ... *	-	Eine Map kann in mehreren Challenges verwendet werden.

Bei den „Map“-Entitäten handelt es sich um die verschiedenen Levels, aus denen das Minispiel besteht.

Map_In_Challenge $\langle E50 \rangle$

Beziehung	Kardinalität	Erwartete Datenmenge	Beschreibung
Challenge	1	ca. 28 byte	Verweis auf die Challenge
Map	1	Min: 512 byte, Max: 16 kB	Verweis auf die Map

In diesen Objekten werden die Beziehungen der „Maps“ und der „Challenges“ festgehalten. Dies umfasst, welche Level des Minispiels für die Lösung bestimmter Aufgabenpakete abgeschlossen werden müssen und in welcher Reihenfolge diese auftreten.

Potion $\langle E60 \rangle$

Beziehung	Kardinalität	Erwartete Datenmenge	Beschreibung
Bag	0 ... *	16 byte	Die Potion wird in einem Bagslot abgelegt.
Scroll	1	Min: 50 byte, Max: 500 byte	Die Potion ist einer Scroll zugeordnet.
Task	1 ... *	Min: 300 byte, Max: 5 kByte	Zum Erzeugen der Potion muss eine Aufgabe gelöst werden.

Die „Potion“-Entitäten beschreiben die Tränke, welche die Spieler im Laufe des Spiels herstellen. Dazu werden die Auswirkungen der Tränke als Attribute gespeichert.

Profile $\langle E70 \rangle$

Beziehung	Kardinalität	Erwartete Datenmenge	Beschreibung
Avatar	1	Min: 500 byte, Max: 32 kByte	Jedes Profil besitzt einen Avatar, der dieses repräsentiert.
Bag	0 ... *	16 byte	Bagslots, die dem Profil gehören.
Profile_Avatar	0 ... *	-	Dem Profil zur Verfügung stehende Avatare.
Comment_Task	0 ... *	-	In dieser Relation wird auf die vom Nutzer kommentierten Aufgaben verwiesen.

Rate_Task	0 ... *	-	In dieser Relation wird auf die vom Nutzer bewerteten Aufgaben verwiesen.
Profile_Scroll	0 ... *	8 byte	Das Profil sammelt Schriftrollen in der Scrollcollection.
Profile_Shop_Item	0 ... *	8byte	Relation mit gekauften Items im Shop.
Solve_Task	0 ... *	20 byte	Tasks, zu denen der Nutzer Zugang hat.
User	1	Min: 250 byte, Max: 150 kByte	Jedes Profil gehört einem Nutzer.

Die Entitäten des Typs „Profile“ stellen die zentrale Verwaltungsstelle der Nutzer dar. Über diese wird der Fortschritt der Spieler abgewickelt, deren Einstellungen gespeichert und die Eigenschaften der Spielfiguren festgehalten.

Profile_Avatar ⟨*E*80⟩

Beziehung	Kardinalität	Erwartete Datenmenge	Beschreibung
Avatar	1	Min: 500 byte, Max: 32 kByte	Verweis auf ein Avatar
Profile	1	Min: 1 kByte, Max: 100 kByte	Verweis auf ein Profil

Diese Entitäten halten die durch ein Profil gekauften Avatare fest und speichert dazu wann die Transaktion durchgeführt wurde.

Comment_Task ⟨*E*90⟩

Beziehung	Kardinalität	Erwartete Datenmenge	Beschreibung
Profile	1	Min: 1 kByte, Max: 100 kByte	Verweis auf das Profil
Task	1	Min: 300 byte, Max: 5 kByte	Verweis auf die Aufgabe

Über die „Comment_Task"-Objekte werden die durch die Nutzer kommentierten Aufgaben gespeichert. Als Attribut wird dazu unter anderem das jeweilige Kommentar festgehalten.

Rate_Task ⟨*E*100⟩

Beziehung	Kardinalität	Erwartete Datenmenge	Beschreibung

Profile	1	Min: 1 kByte, Max: 100 kByte	Verweis auf das Profil
Task	1	Min: 300 byte, Max: 5 kByte	Verweis auf die Aufgabe

Über die „Rate_Task“-Objekte werden die durch die Nutzer bewerteten Aufgaben gespeichert. Als Attribut wird dazu unter anderem die jeweilige Bewertung festgehalten.

Profile_Scroll ⟨*E*110⟩

Beziehung	**Kardinalität**	**Erwartete Datenmenge**	**Beschreibung**
Profile	1	Min: 1 kByte, Max: 100 kByte	Verweis auf das Profil
Scroll	1	Min: 50 byte, Max: 500 byte	Verweis auf die Scroll

Hier werden die von den Nutzern gesammelten Scrolls gespeichert.

Profile_Shop_Item ⟨*E*120⟩

Beziehung	**Kardinalität**	**Erwartete Datenmenge**	**Beschreibung**
Profile	1	Min: 1 kByte, Max: 100 kbyte	Verweis auf das Profil
Shop_Item	1	Min: 100 byte, Max: 1 kByte	Verweis auf das Shop-Item

Die Entitäten des Typs „Profile_Shop_Item“ halten fest, welche Spielgegenstände die Spieler bisher bereits erworben haben.

Solve_Challenge ⟨*E*130⟩

Beziehung	Kardinalität	Erwartete Datenmenge	Beschreibung
Challenge	1	ca. 28 byte	Verweis auf die Challenge
Profile	1	Min: 1 kByte, Max: 100 kByte	Verweis auf das Profil

Diese Objekte halten den Bearbeitungsfortschritt der Nutzer bezüglich der Aufgabenpakete fest. Es werden dabei sowohl die bereits gelösten Aufgabenpakete, als auch die noch nicht abgeschlossenen Aufgabenpakete abgespeichert.

Solve_Task ⟨*E*140⟩

Beziehung	Kardinalität	Erwartete Datenmenge	Beschreibung
Profile	1	Min: 1 kByte, Max: 100 kByte	Verweis auf das Profil
Task	1	Min: 300 byte, Max: 5 kByte	Verweis auf die Aufgabe.

Diese Entitäten sorgen für die Speicherung des Fortschritts bei der Lösung der einzelnen Aufgaben durch die verschiedenen Nutzer. Dabei werden unter anderem die benötigte Zeit, sowie das Lösungsdatum als Attribute gespeichert.

Scroll ⟨*E*150⟩

Beziehung	Kardinalität	Erwartete Datenmenge	Beschreibung
Potion	0 .. 1	Min: 1 kByte, Max: 16 kByte	Jede Schriftrolle schaltet eine oder keine Potion frei.
Profile_Scroll	0 ... *	8 byte	Eine gesammelte Schriftrolle wird in der Scrollcollection des Profils gespeichert.

Die „Scrolls“ beschreiben die im Spiel verteilten Schriftrollen, welche durch die Spieler eingesammelt werden um neue Tränke oder temporäre Verbesserungen der Eigenschaften der Spielfigur (Buffs) freischalten. **Hinweis:** Wird durch die Scroll keine Potion freigeschaltet, erhält der Nutzer einen Buff.

Shop_Item ⟨*E*160⟩

Beziehung	Kardinalität	Erwartete Datenmenge	Beschreibung
Profile_Shop_item	0 ... *	8 byte	Relation zwischen Shop-Items und den Profilen der Nutzer, die das Shop-Item gekauft haben.

Die Objekte des Typs „Shop_Item“ stellen die im Shop zum Kauf zur Verfügung stehenden Gegenstände dar. Daher wird unter anderem der Kaufpreis als Attribut gespeichert.

Story_Text ⟨*E*170⟩

Beziehung	Kardinalität	Erwartete Datenmenge	Beschreibung
Text_In_Challenge	1	-	Story-Texte beschreiben Challenges handlungsbezogen.
Text	0 ... *	Min: 50 byte, Max: 500 byte	Texte beschreiben die Spielsituation.

Diese Entitäten beschreiben die verschiedenen Texte, welche in der Story aufgerufen werden. Dazu werden die Bedingungen, welche erfüllt sein müssen um den Text aufzurufen, sowie die Reihenfolge, in der diese auftreten und deren Inhalt abgespeichert.

Text_In_Challenge ⟨*E*180⟩

Beziehung	Kardinalität	Erwartete Datenmenge	Beschreibung
Challenge	1	ca. 28 byte	Verweis auf die Challenge.

An dieser Stelle wird festgehalten, welche Texte in welchem Aufgabenpaket auftreten und in welcher Reihenfolge dies passiert.

Task ⟨*E*190⟩

Beziehung	Kardinalität	Erwartete Datenmenge	Beschreibung
Potion	1	Min: 1 kByte, Max: 16 kByte	Eine gelöste Aufgabe schaltet einen neuen Trank frei.
Comment_Task	0 ... *	-	Kommentare der Aufgabe.
Rate_Task	0 ... *	-	Bewertungen der Aufgabe.
Solve_Task	0 ... *	20 byte	Relation mit Profilen von Nutzern, die Zugang zu dieser Aufgabe haben.
Task_In_Challenge	0 ... *	8 byte	Aufgaben können Teil einer Challenge.

Die „Task“-Entitäten beschreiben die einzelnen Aufgaben, welche in den Aufgabenpaketen enthalten sind und von den Nutzern gelöst werden müssen um neue Tränke freizuschalten. Als Attribute werden dafür unter anderem deren Schwierigkeit, die Bewertung, die Zeitbegrenzung und der dazugehörige Trank abgespeichert.

Task_In_Challenge ⟨*E*200⟩

Beziehung	Kardinalität	Erwartete Datenmenge	Beschreibung
Task	1	Min: 300 byte, Max: 5 kByte	Verweis auf die Aufgabe
Challenge	1	ca. 28 byte	Verweis auf die Challenge.

Hier wird festgehalten in welchen Aufgabenpaketen die verschiedenen Aufgaben enthalten sind und in welcher Reihenfolge diese auftreten.

Text ⟨*E*210⟩

Die Entitäten vom Typ "Text"beschreiben die verschiedenen Texte, welche innerhalb des SQL-Alchemist auftreten.

„entfernt“’

User ⟨*E*220⟩

Beziehung	**Kardinalität**	**Erwartete Datenmenge**	**Beschreibung**
Profile	1	Min: 1 kByte, Max: 100 kByte	Jeder Nutzer hat ein eigenes Profil.
User_Session	0 ... *	Min: 250 kByte, Max: 16 kByte	Die Anzahl der User-Sessions gibt an, wie oft der einzelne Benutzer gleichzeitig im System angemeldet ist.

Die „User“-Objekte stellen die Nutzer dar, die sich bisher für das Spiel registriert haben. Dazu werden an dieser Stelle alle Nutzerdaten, die für die Erkennung und die Anmeldung von Bedeutung sind als Attribute gespeichert.

User_Session ⟨*E*230⟩

Beziehung	Kardinalität	Erwartete Datenmenge	Beschreibung
User	1	Min: 250 byte, Max: 150 kByte	Verweis auf den entsprechenden User, der die User-Session gestartet hat.

Die „User_Session“-Objekte stellen die einzelnen durch die Nutzer gestarteten Sitzungen dar. Das bedeutet, dass jedes mal, wenn sich ein Nutzer zum Spiel anmeldet wird ein neues „User_Session“-Objekt erzeugt. Als Attribute werden die Verbindungsdaten festgehalten.

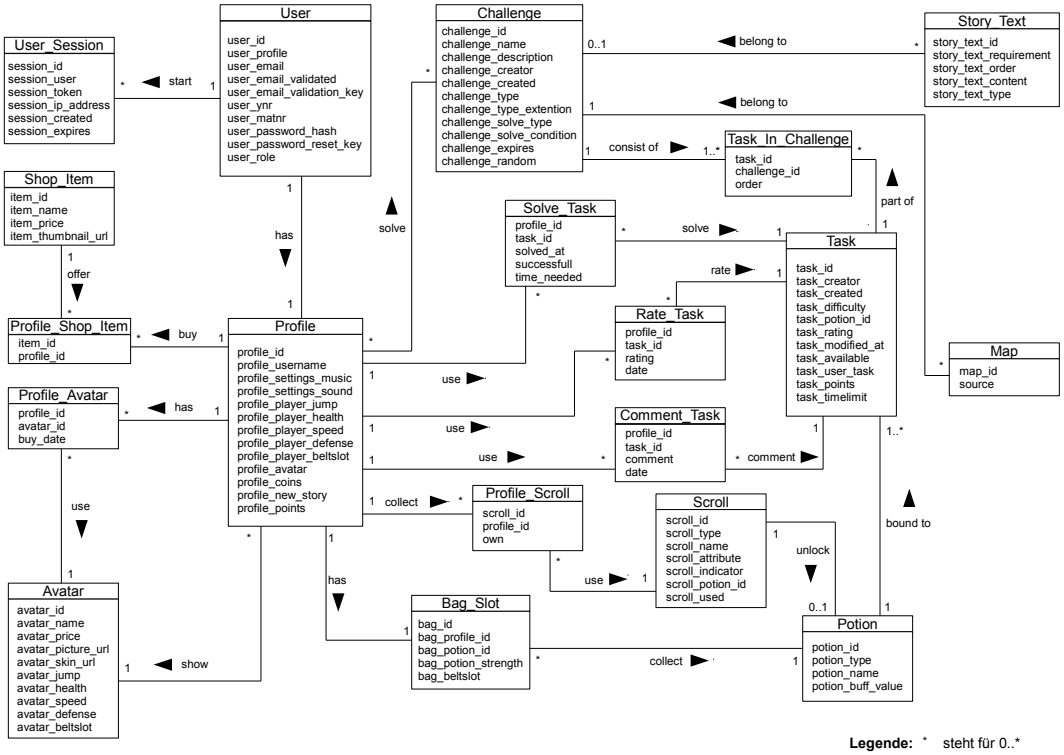

Abbildung 6.1: Klassendiagramm zum SQL-Alchemist

7 Konfiguration

Für den SQL-Alchemist benötigen wir weder einen Rechner, noch einen Server mit einer bestimmten Konfiguration. Aus diesem Grund wird an dieser Stelle nicht weiter auf diesen Punkt eingegangen.

Außerdem existieren auch keine config-Dateien auf die an dieser Stelle hingewiesen werden müsste.

8 Änderungen gegenüber Fachentwurf

Im Gegensatz zum Fachentwurf wurde nur das Datenmodell geändert. Dabei wurden die Entitäten „Story_Text“, „Text_In_Challenge “ und „Solve_Task“ entfernt und ihre Inhalte in anliegende Entities integriert.

9 Erfüllung der Kriterien

In diesem Kapitel wird beschrieben, wie die einzelnen Kriterien, die im Pflichtenheft genannt sind, in der Software umgesetzt wurden. Dabei wird auf die Nutzung bei der Umsetzung eingegangen. Da in der Software zwei Komponenten vorhanden sind und diese dadurch sehr oft auftauchen werden, wird das Front-End nachfolgend mit seiner Bezeichnung „C10“ und das Back-End mit seiner Bezeichnung „C10“ genannt.

9.1 Musskriterien

⟨$RM1$⟩ Das geforderte Minispiel wird in C10 angezeigt und die Daten dafür in C20 gespeichert
⟨$RM2$⟩ Das grafisch aufbereitete Webinterface ist durch C10 komplett realisiert.
⟨$RM3$⟩ Die Fenster für Aufgabenstellung und eingaben sind Teil von C10. Dabei werden die Daten für die Aufgabenstellung vom C20 angefordert und die Daten aus dem Eingabefeld werden zur Kontrolle zurück an C20 gesendet
⟨$RM4$⟩ C20 besteht zum Großteil aus einer Datenbank die die anfallenden Nutzerden speichert.
⟨$RM5$⟩ C20 stellt eine Benutzeroberfläche zur Verfügung, die administrative Tätigkeiten ermöglicht
⟨$RM6$⟩ Die in RM5 Benutzerobfläche beinhaltet auch die Möglichkeit, Aufgaben zu erstellen.
⟨$RM7$⟩ C10 bietet drei Spielmodi (Trivia-, Story und Homework-Mode).
⟨$RM8$⟩ C20 hat eine Anbindung an das LDAP der Technischen Universität Braunschweig.
⟨$RM9$⟩ Die Game-Engine ist in C20 integriert.

9.2 Sollkriterien

⟨$RS1$⟩ Die gestellten SQL-Aufgaben haben fünf Schwierigkeitgrade.
⟨$RS2$⟩ Die Gesamtsoftware ist mit leichten Leistungseinbußen auf moblien Endgeräten funktionabel.
⟨$RS3$⟩ Die Software bietet verschiedene Avatare mit verschiedenen Eigenschaften die im Shop erworben werden können.
⟨$RS4$⟩ Das Spielerprofil speichert verschiedene Höchstleistung des Spielers.

⟨$RS5$⟩ Es werden verschiedene Ranglisten unterstützt.

9.3 Kannkriterien

⟨$RC1$⟩ Mit den entsprechenden Rechten kann jeglicher Aufgabentyp erstellt werden.
⟨$RC2$⟩ Das Nutzen der Software von anderen Universitäten ist nicht vorgesehen, kann jedoch besprochen werden.
⟨$RC3$⟩ Es könnte ein abgespecktes Tutorial für den Trivia-Mode geben. Dies wurde nicht umgesetzt.
⟨$RC4$⟩ Im Shop können, gegen Ingame-Währung, Avatare und „Belt-Slots“ erworben werden.
⟨$RC5$⟩ Im Minispiel erfolgen Sprünge über die Leertaste oder die linke Maustaste. Die Beltslots zum Nutzen der darin befindlichen Tränke werden mit den Tasten 1 bis 7 angewählt.
⟨$RC6$⟩ Freundeslisten wird es vorerst nicht geben.