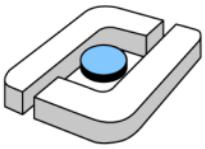


Hochschule Osnabrück

University of Applied Sciences

Fakultät
Ingenieurwissenschaften und Informatik

Schriftliche Projektarbeit zum Thema:

Umsetzung einer Quizwebapp mit Quarkus

im Rahmen des Moduls
Software Architektur - Konzepte und Anwendungen (MI),
des Studiengangs Informatik-Medieninformatik

Autor:	Annette Michel
Matr.-Nr.:	848262
E-Mail:	annette.michel@hs-osnabrueck.de
Dozent:	Prof. Dr. Rainer Roosmann

Abgabedatum: 17.02.2023

Eidesstattliche Erklärung

Hiermit erkläre ich/ Hiermit erklären wir an Eides statt, dass ich/ wir die vorliegende Arbeit selbständig und ohne fremde Hilfe angefertigt habe/ haben. Die aus fremden Quellen direkt oder indirekt übernommenen Gedanken sind als solche einzeln kenntlich gemacht. Es wurden keine anderen als die angegebenen Quellen und Hilfsmittel benutzt. Die Arbeit wurde bisher keiner anderen Prüfungsbehörde vorgelegt und auch nicht veröffentlicht.

Sendenhorst, 17.02.2023 C. Michel

....

Ort, Datum Unterschrift

I Vorwort

Ziel des Projekts ist es, das gelernte Wissen aus den Vorlesungen, sowie vorlesungsbegleitenden Praktika zu vertiefen und anzuwenden. Das zu entwickelnde Resultat ist als Projektarbeit abzugeben.
In der vorliegenden Projektarbeit wird der Entwicklungsprozess der API von der Idee bis hin zur Umsetzung dokumentiert und beschrieben. Hierbei wird auf die Konstruktion der Anwendung, sowie auf die Programmierung in der Programmiersprache Java eingegangen.

II Inhaltsverzeichnis

III Abkürzungsverzeichnis

DDD	Domain Driven Design
SWA	Software Architektur
DTO	Datentransferobjekt

1 Projektbeschreibung

1.1 Beschreibung der Projektarbeit

In dieser Projektarbeit wird die Web-Applikation „QuizTime“ vorgestellt: Eine Quiz-Web-App, die es den Benutzern ermöglicht an Quiz-Spielen teilzunehmen und ihr Wissen in verschiedenen Themenbereichen zu testen.

QuizTime präsentiert den Benutzern eine Vielzahl von Fragen mit verschiedenen Schwierigkeitsgraden. Die Benutzer können aus verschiedenen Kategorien wählen, wie z.B. Tiere, Musik, Filme und Geschichte. Außerdem bietet die Web-App eine Rangliste, auf der die Benutzer ihre Punktzahl und ihren Platz im Vergleich zu anderen Benutzern sehen können.

Des Weiteren bietet die Web-App eine Administratoroberfläche. Damit wird den Administratoren ermöglicht, Quizinhalte und -fragen zu erstellen, zu verwalten und zu bearbeiten. Der Admin kann nach einer bestimmten Quizfrage suchen oder nach Kategorien filtern. Damit wird dem Admin die Möglichkeit geboten, die Quizinhalte zu aktualisieren und zu verbessern, um sicherzustellen, dass die Benutzer stets interessante und aktuelle Fragen erhalten.

Auch die Benutzerverwaltung wird dem Admin zur Verfügung gestellt. Damit kann die Benutzerdatenbank verwaltet werden, um Benutzer zu suchen, Benutzerprofile zu erstellen, zu bearbeiten und zu löschen. Zudem können Administratoren Benutzerrollen vergeben.

1.2 Ziel der Ausarbeitung

Es wird eine Rest-API bereitgestellt, womit verschiedene Quizfragen und Benutzer gelistet, verwaltet und bearbeitet werden können. Diese Informationen werden mithilfe einer Datenbankanbindung bereitgestellt. Des Weiteren werden für die Benutzer und Administratoren eine angepasste Benutzeroberfläche geboten, welche mithilfe von Quarkus Qute umgesetzt wird. Damit wird das Verwalten der Quizfragen und Benutzer für den Admin erleichtert und die Benutzer haben die Möglichkeit Quizfragen zu beantworten und sich mit anderen Benutzern zu messen. Um mit der Web-App zu interagieren, muss sich der Benutzer registrieren bzw. anmelden.

1.3 Aufbau der Hausarbeit

Im ersten Abschnitt werden grundlegende Themen der Software-Architektur beschrieben, die zum Verständnis beitragen sollen. Des Weiteren wird das Konzept mit den grundlegenden Funktionen und Überlegungen der Quiz-Webanwendung aufgezeigt. Dabei wird das Ziel und die zu erforderlichen Funktionen bestimmt und wie die Anwendung im Frontend abgebildet wird. Das Hauptaugenmerk liegt auf der Implementierung, dort wird auf die verwendete Technologie und Programmiersprache eingegangen. Anschließend wird die Testphase beschrieben und zum Schluss werden die Ergebnisse und Erkenntnisse festgehalten.

2 Darstellung der Grundlagen

2.1 Grundlegende Software-Architektur-Prinzipien

In der Softwareentwicklung sind grundlegende Architekturprinzipien von großer Bedeutung, um Systeme auf eine strukturierte und nachhaltige Weise zu gestalten.

2.1.1 SOLID-Prinzipien

Die SOLID-Prinzipien sind grundlegende Architekturprinzipien in der Softwareentwicklung, die helfen, die Komplexität zu reduzieren und die Qualität der Systeme zu verbessern. Sie wurden von Robert C. Martin, einem bekannten Software-Engineer angedacht und sind seitdem zu einem wichtigen Bestandteil der Softwareentwicklung geworden.

Die fünf SOLID-Prinzipien sind:

- Single Responsibility Principle (SRP) - Eine Klasse sollte eine klar definierte Verantwortung haben.
- Open/Closed Principle (OCP) - Softwareentitäten sollten offen für Erweiterungen, aber geschlossen für Modifikationen sein.
- Liskov Substitution Principle (LSP) - Objekte einer abgeleiteten Klasse sollten als Ersatz für Objekte der Basisklasse verwendet werden können.
- Interface Segregation Principle (ISP) – Anbieten von Schnittstellen für Software-Bausteine, die konkret benötigt und erwartet werden.
- Dependency Inversion Principle (DIP) - Abhängigkeiten sollten auf abstrakte statt konkrete Klassen gerichtet sein.

Diese Prinzipien helfen, Code verständlicher, wiederverwendbarer und testbarer zu machen. Sie sind ein wichtiger Bestandteil der Softwareentwicklung. [Roosmann, 2022/23]

2.1.2 Domain Driven Design

Im DDD-Ansatz wird die Geschäftslogik und der Geschäftsprozess in den Vordergrund gestellt. Das System wird in verschiedene Domänen unterteilt, die jeweils ein spezifisches Geschäftsproblem lösen. Innerhalb jeder Domäne werden dann wiederum spezifische Subdomänen mit ihrer eigenen Sprache (Bounded Context) identifiziert, die sich um spezifische Bereiche innerhalb der Hauptdomäne kümmern. Zwischen den Subdomänen gibt es Beziehungen. Mithilfe von Context Maps wird die Art der Abhängigkeit festgelegt.

Insgesamt zielt DDD darauf ab, ein besseres Verständnis der Geschäftsprozesse zu schaffen, die in einem Softwareprojekt umgesetzt werden, um sicherzustellen, dass das System den tatsächlichen Anforderungen und Bedürfnissen des Geschäfts entspricht. [Roosmann, 2022/23]

2.2 Technische Grundlagen

2.2.1 Build- und Dependency-Management mit Maven

Apache Maven ist ein Build-Management-Tool für die Java-Programmierung. Es automatisiert den Build-Prozess von Java-Anwendungen und verwaltet die Abhängigkeiten von Libraries, Frameworks und anderen externen Komponenten. Maven wurde von Jason van Zyl entwickelt und ist ein Open-Source-Projekt der Apache Software Foundation.

Maven verwendet eine XML-basierte Konfigurationsdatei, die die Projektabhängigkeiten, das Build-Verfahren und andere Informationen enthält. Diese Konfigurationsdatei wird "pom.xml" genannt. Das pom.xml-File beschreibt das Projekt in Bezug auf seine Abhängigkeiten, Build-Prozess, Test-Prozess und Deployment-Prozess.

Maven ermöglicht es Entwicklern, Java-Anwendungen schnell und einfach zu erstellen, indem es eine Standard-Ordnerstruktur und -Konfiguration bereitstellt, die sich leicht anpassen und erweitern lässt. Außerdem unterstützt Maven das Dependency-Management, so lassen sich Libraries und Frameworks von einem zentralen Repository zu laden und automatisch die erforderlichen Abhängigkeiten zu verwalten.

2.2.2 Quarkus-Framework

Quarkus ist ein modernes Open-Source-Framework für die Erstellung von Cloud-nativen Anwendungen in der Java-Programmierung. Es ist speziell für Container-basierte Architekturen und Microservices entwickelt worden und bietet eine schnelle Startzeit und einen geringen Ressourcenverbrauch. Quarkus wurde von Red Hat entwickelt und ist ein wichtiges Tool für die Entwicklung von Java-basierten Cloud-Anwendungen. [@Quarkus]

2.2.2.1 Quarkus-Qute

Quarkus-Qute ist eine Template-Engine, die Teil des Quarkus-Frameworks ist und es Entwicklern ermöglicht, dynamische HTML- und Textvorlagen in ihren Anwendungen zu verwenden. Qute wurde entwickelt, um die Erstellung von skalierbaren und leistungsfähigen Webanwendungen zu vereinfachen und die Wiederverwendbarkeit von Templates zu verbessern. Es ist ein Open-Source-Projekt und wird von Red Hat unterstützt. [@Quarkus-Qute]

2.2.3 Hibernate

Hibernate ist ein Open-Source-Framework für die objektorientierte Abbildung von relationalen Datenbanken in Java-Programmierung. Es wurde entwickelt, um die Interaktion mit Datenbanken in Java-Anwendungen zu erleichtern und zu vereinfachen. Hibernate ist ein Teil des JBoss-Frameworks, das von Red Hat unterstützt wird. Hibernate ermöglicht es Entwicklern, Datenbank-Abfragen in objektorientiertem Code auszudrücken. Es übersetzt die Anfragen in die entsprechende SQL-Syntax und führt sie auf der Datenbank aus. Dies macht es einfacher, auf Datenbanken zuzugreife. [@Hibernate]

3 Projektübersicht

3.1 Projektplanung

Das Projekt wurde nach der folgenden Prozesskette umgesetzt und dokumentiert:

Abbildung 1: Prozesskette

3.2 Analyse und Konzeption

3.2.1 Durchführung der Planung

Gestartet wird die Projektarbeit mit der Analyse- und Konzeptionsphase. Innerhalb dieser Phase wird das Ziel des Projekts definiert, ein Überblick der Funktionen erstellt und dementsprechend das Projekt konzeptionell sowie visuell aufbereitet.

3.2.2 Definition des Ziels

Das Ziel dieser Quiz-Webanwendung ist, dass Benutzer ihr Wissen zu bestimmten Themen testen können. Um dieses Ziel zu erreichen, sind folgende Funktionen erforderlich:

- Registrierung und Anmeldung: Benutzer sollten sich registrieren und anmelden können, um Zugang zum Quiz zu erhalten
- Quizfragen und -antworten: Die Quiz-Webanwendung sollte eine Sammlung von Fragen und Antworten haben, die dem Benutzer angezeigt werden.
- Punktevergabe: Die Quiz-Webanwendung sollte die Antworten des Benutzers bewerten und ihm basierend auf seinen Antworten Punkte vergeben.
- Bestenliste: Es sollte eine Bestenliste der Top-Benutzer basierend auf ihren Gesamtpunktzahlen geben, um die Benutzer zu motivieren, sich zu verbessern.
- Benutzerprofil: Jeder Benutzer sollte ein Profil haben, das seine Informationen und Quizergebnisse enthält, sowie das Bearbeiten, Löschen und Abmelden des Profils.
- Administration der Quizfragen: Es sollte eine Administrationsoberfläche geben, um neue Quizfragen und -antworten hinzuzufügen, zu bearbeiten und zu löschen.
- Administration der Benutzer: Es sollte eine Administrationsoberfläche geben, um neue Benutzer und -rollen hinzuzufügen, zu bearbeiten und zu löschen.

3.2.3 Definition der Funktionen

Anhand der Zieldefinition kann ein Plan der Web-Applikationsfunktionen erstellt werden. Der Plan lässt sich in folgende Funktionsanforderungen aufteilen:

1. Startseite:
    a. Erläuterung der Quiz-Webapp
    b. Call-to-Action um das Quiz zu beginnen

2. Quiz:
    a. Auswahl der Quizkategorien
        i. Beantworten der Quizfragen
        ii. Feedback zur beantworteten Quizfrage

3. Highscore:
    a. Tabellarische Auflistung der besten Spieler (Spielername und Punktzahl)

4. Profil (Zugriff: User/Admin):
    a. Anzeigen der Spielerinformationen (Id, Benutzername, E-Mail, Punktzahl)
    b. Bearbeiten der Spielerinformationen (Email,Passwort)
    c. Löschen des Benutzers
    d. Abmelden des Benutzers

5. Anmelden/Registrieren:
    a. Abfragen der Spielerinformationen zur Authentifizierung
    b. Anlegen eines Spielers

6. Spielerübersicht (Zugriff: Admin):
    a. Tabellarische Auflistung aller Benutzer
    b. Suchen eines Benutzers per Benutzername
    c. Anlegen eines Benutzers
    d. Bearbeiten eines Benutzers
    e. Löschen eines Benutzers

7. Quizübersicht (Zugriff: Admin):
    a. Tabellarische Auflistung aller Quizfragen
    b. Suchen einer Quizfrage nach Id
    c. Suchen nach Quizfragen einer Kategorie
    d. Anlegen eine Quizfrage
    e. Bearbeiten einer Quizfrage
    f. Löschen einer Quizfrage

Die aufgelisteten Funktionsanforderungen werden im nächsten Abschnitt, in der Designniederschrift, aufgefasst und umgesetzt.

3.3 Design

Die Schritte zur Erstellung der Benutzeroberfläche, einschließlich der Navigation, des Layouts und des visuellen Designs, werden kurz erläutert und durch Beispiele veranschaulicht.

3.3.1 Konzeption & Design

Layout und Seitenaufbau

Alle wichtigen Informationen sollen für den Nutzer auf einem Blick erfassbar sein.

Startseite

Die Startseite bietet dem Benutzer eine Einführung in die Anwendung und ermöglicht es ihm, auf verschiedene Bereiche der Anwendung zuzugreifen. Die Willkommensseite sollte eine ansprechende Benutzeroberfläche haben, die den Benutzer motiviert, sich mit der Anwendung zu beschäftigen.

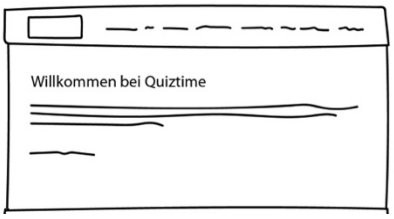


Abbildung 2: Startseite

Login

Die Login-Seite dient dazu, dass Benutzer sich anmelden und Zugang zu ihren Benutzerkonten erhalten können. Um ein Benutzerkonto zu erstellen, müssen die Benutzer auf die Registrierungsseite zugreifen, die unter dem Login-Formular zu finden ist.

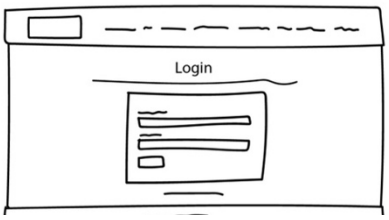


Abbildung 3: Login

Highscore Übersicht

Die Highscore-Übersicht ist eine leicht zu erfassende Oberfläche, die es den Benutzern ermöglicht, schnell und einfach auf ihre Ergebnisse aus einer Tabelle zuzugreifen.

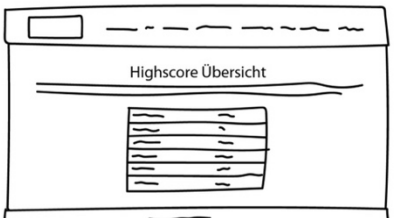


Abbildung 4: Highscore

Quiz

Das Quiz ermöglicht es den Benutzern Fragen zu einer bestimmten Kategorie beantworten und Punkte sammeln können.

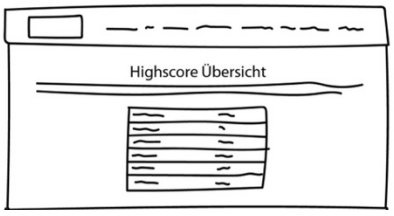


Abbildung 5: Quiz

Quizauflistung/Benutzerauflistung (Admin)

Die Quiz- und Benutzerauflistung für Admins bildet eine wichtige Funktion. Damit haben die Administratoren eine Übersicht über die verfügbaren Quiz und Benutzer und können diese verwalten.

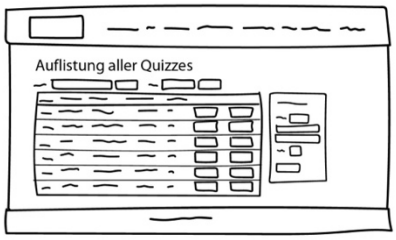


Abbildung 6: Quizauflistung

3.4 Entwicklungsphase

Nachdem alle Vorbereitungen getroffen worden sind, konnte die eigentliche Realisierung der Webapp beginnen.

Die Funktionalitäten und der Aufbau muss logisch strukturiert werden, weshalb ein vereinfachtes Klassendiagram erstellt wird. Diese Struktur wird in die Realisierungsphase übertragen.

3.4.1 Vereinfachte Klassenübersicht

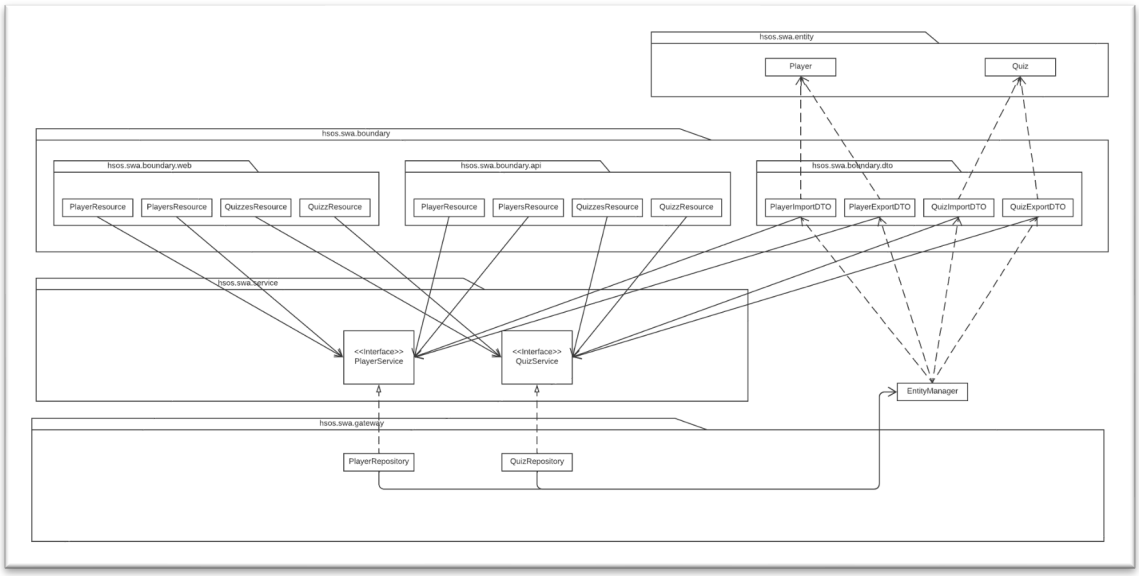


Abbildung 2: Ausschnitt der Klassenübersicht

3.4.2 Realisierung

In der Realisierung dienen die Directorys der besseren Übersicht, bilden eine gute Struktur und fördern damit das bessere Verständnis des Aufbaus der Webapp.

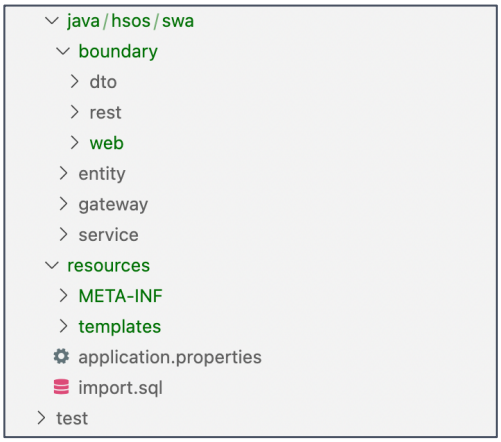


Abbildung 3: Directory-Ansicht

3.4.3 Strukturaufbau

Die Verwendung von Domain Driven Design in der Webapplikation ermöglicht es, die Vielschichtigkeit der Anwendung zu reduzieren und eine klare Trennung zwischen Geschäftslogik und technischer Implementierung zu schaffen.

3.4.4 Entitäten

Die Quizwebapp besteht aus verschiedenen Domänenobjekten, die das Geschäftsmodell der Anwendung repräsentieren. Dazu gehören unter anderem die Entitäten Player, Quiz und Question und das Aggregate Selection.

Die Entität Quiz z. B. repräsentiert ein Quiz mit einem eindeutigen Identifikator Id mit @Id, einer Kategorie (Topic), einer Frage (Question) und einer Sammlung von Antworten (Selection). Jede Frage (Question) wird durch eine eigene Entität repräsentiert, die eine Beschreibung und eine Sammlung von Antworten hat.

3.4.5 DTO

Die Data Transfer Objects (DTOs) werden verwendet, um Daten zwischen verschiedenen Schichten der Anwendung auszutauschen. DTOs sind einfache Objekte, die verwendet werden, um Daten zu transportieren, ohne die Struktur der Domain-Objekte zu zeigen. Zum Beispiel wird ein PlayerExportDTO verwendet, um die Daten eines Player-Objekts abzurufen und an die Benutzeroberfläche zu übergeben. Das PlayerImportDTO wird verwendet, um Daten aus der Webanwendung zu importieren.

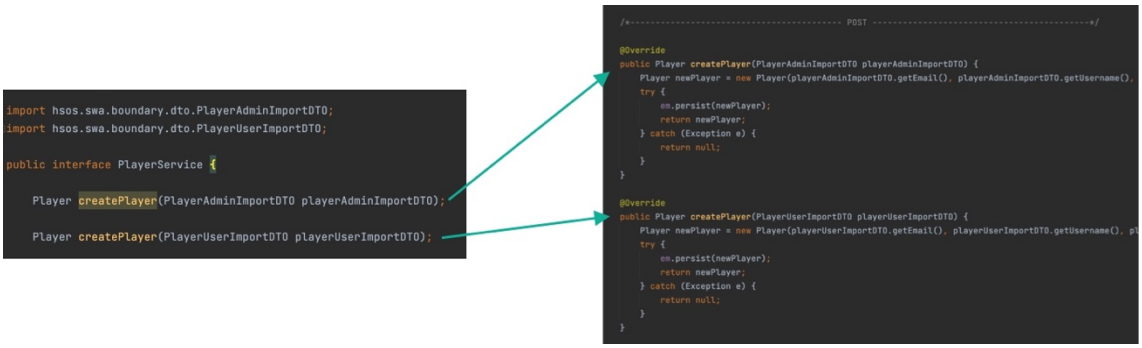
```java
/*----------------------------------------- POST -----------------------------------------*/

@POST
@Operation(summary = "create player", description = "Create new player")
@Parameters({
        @Parameter(name = "id", description = "Player Id", required = true),
        @Parameter(name = "mail", description = "Player email", required = true),
        @Parameter(name = "username", description = "Player username", required = true),
        @Parameter(name = "password", description = "Player password", required = true)
})
@Timeout
@Retry
public Response createPlayer(@QueryParam("mail") String mail, @QueryParam("username") String username, @QueryParam("password") String password) {
    logger.info("Create new player with username: " + username);
    PlayerUserImportDTO playerUserImportDTO = new PlayerUserImportDTO(mail, username, password);
    Player newplayer = playerService.createPlayer(playerUserImportDTO);
    return Response.status(Response.Status.CREATED).entity(newplayer).build();
}
```

In der Abbildung wird gezeigt, dass ein neues Spieler Objekt erstellt wird und die PlayerUserImportDTO dazu verwendet wird, die eingetragenen Parameter aus der Anwendung zu importieren.

3.4.6 Service

In der Quizwebapp werden Services verwendet, um Geschäftsprozesse wie das Erstellen, Bearbeiten und Löschen von Quizze und Players zu implementieren. Zum Beispiel gibt es einen Quiz-Service und Player-Service, der die Erstellung und Aktualisierung von diesen Objekten koordiniert.



Die Methoden aus dem Interface werden in der PlayerRepository implementiert.

3.4.7 Gateway

Im Directory Gateway werden Repositories verwendet, um die Persistenz von Entitäten und den Zugriff auf Daten zu verwalten. Die Repositorys im Projekt speichern und rufen mithilfe des Entity Managers die Quiz- und Playerobjekte auf.

3.4.8 Boundary

In der Boundary-Ebene werden Objekt Resourcen verwendet, um die Eingabe von Daten durch den Benutzer zu verarbeiten (REST) und die Daten zwischen der Benutzeroberfläche (Web) und der Entität auszutauschen. In diesem Projekt gibt es z. B. die QuizResource,die die Eingabe von Daten für die Erstellung und Aktualisierung von Quizzes verarbeitet und das Quiz-DTO an den Quiz-Service übergibt.

Eine gängige Aufgabe für die Boundary ist die Implementierung von CRUD-Funktionen (Create, Read, Update, Delete).

```
/*---------- GET ----------*/

@GET
@RolesAllowed(ADMIN_ROLE)
public TemplateInstance getAllPlayer(){
    List<PlayerExportDTO> playerList = playerService.getAllPlayers() Collection<Player>
            .stream() Stream<Player>
            .map(player -> new PlayerExportDTO(player)) Stream<PlayerExportDTO>
            .toList();
    return Templates.players(playerService.getAllPlayers());
}

/*---------- POST ----------*/

@POST
@RolesAllowed(ADMIN_ROLE)
public TemplateInstance createPlayerAdmin(@FormParam("mail") String mail, @FormParam("username") String user
    PlayerAdminImportDTO playerAdminImportDTO = new PlayerAdminImportDTO(mail,username,password,role);
    playerService.createPlayer(playerAdminImportDTO);
    return Templates.players(playerService.getAllPlayers());
}
```

Create: Die Methode createPlayerAdmin() wird aufgerufen, um ein neues Player-Objekt in der Datenbank zu erstellen. Die Methode nimmt ein PlayerAdminImportDTO-Objekt als Parameter entgegen, dass die erforderlichen Daten für den Player enthält. Das PlayerAdminImportDTO-Objekt wird dann in ein Player- Domänen-Objekt umgewandelt und in der Datenbank gespeichert.

Read: Die Methode getAllPlayer() wird aufgerufen, um alle Player abzurufen. Die Methode ruft dann die entsprechenden Player-Domänen-Objekte ab. Dieses wird dann in ein PlayerIExportDTO-Objekt umgewandelt und an die Präsentationsschicht zurückgegeben.

Update: Die Methode updatePlayer() wird aufgerufen, um ein bestehenden Player zu aktualisieren. Die Methode nimmt ein PlayerImportDTO-Objekt als Parameter entgegen, das die aktualisierten Daten für den Player enthält. Das PlayerImportDTO-Objekt wird dann in ein Player-Domänen-Objekt umgewandelt und in der Datenbank aktualisiert.

Delete: Die Methode deletePlayer() wird aufgerufen, um ein Player anhand seiner ID zu löschen. Die Methode nimmt die ID des Players als Parameter und ruft dann das entsprechende Player-Domänen-Objekt ab. Dieses wird dann aus der Datenbank gelöscht.

3.5 Probleme und Lösungen

Die Implementation eines Logins in einer Quarkus Webanwendung kann je nach Anforderungen und Use Case unterschiedlich aussehen. Ein möglicher Ansatz war die Verwendung eines Security Frameworks wie zum Beispiel Keycloak.

Allerdings war es nicht möglich Benutzerollen einzustellen und diese bei der Anmeldung und Registrierung zu unterscheiden.

Ein möglicher Ansatz wäre die Benutzerrollen in Keycloak zu definieren und richtige Rollennamen zu vergeben. Allerdings war Keycloak für das Quizprojekt zu umfangreich, weshalb die Entscheidung auf eine Formularbasierte Authentifizierung fiel: HTTP Basic Authentication ist mit die einfachste Authentifizierungsmethode. Diese Methode verwendet ein HTML-Formular, um die Benutzeranmeldeinformationen zu erfassen.

Hierzu muss ein Formular mit vordefinierten Attributen (j_security_check, j_username,j_password) erzeugt werden.

```
<form action="/j_security_check" method="post">
    <label for="j_username">Benutzername</label>
    <input class="form-control" type="text" placeholder="user oder admin" id="j_username" name="j_username"
           required>

    <label for="j_password"><b>Password</b></label>
    <input class="form-control" type="password" placeholder="user oder admin" id="j_password" name="j_password"
           required>
    <br>
    <button class="btn btn-primary" type="submit">Anmelden</button>
</form>
```

4 Projektergebnis

4.1 Soll-Ist-Vergleich

Das Anmelden und Registrieren der Benutzer wurde mithilfe eines Anmelde-/Registrierungsformular umgesetzt. Die Unterscheidung von Nutzer und Admin spielt dabei eine relevante Rolle, denn Admin haben noch weitere Interaktionsmöglichkeiten.

Die Administratoren sind in der Lage, Benutzer zu erstellen, zu bearbeiten, zu suchen oder zu löschen. Sie könnten auch Benutzerrollen zuweisen oder Benutzer löschen. Außerdem können Administratoren Fragen und Antworten zu einer bestimmten Quiz Kategorie erstellen, bearbeiten, suchen oder löschen.

Die User können nach Anmeldung eine Kategorie auswählen zu denen sie Quizfragen beantworten möchten. Jedoch ist das Quizspiel nur ein Prototyp und der User hat nicht die Möglichkeit Fragen zu beantworten und Punkte zu erhalten. Die Möglichkeit das eigene Profil anzusehen, zu bearbeiten und zu löschen ist gegeben. Auch das Abmelden des Users, sowie des Admins wird ermöglicht.

Das einsehen der Bestenliste wird für jeden Besucher der Website ermöglicht.

Es werden zwei Schnittstellen zu Interaktion bereitgestellt: Eine prototypische Webapp mit Quarkus Qute und eine REST-API für eine Anwendung für Admins, damit diese Kategorien und Quizze verwalten werden können.

Einschätzungen konnte das Quiz nur prototypisch abgebildet werden. Der User hat hier leider nicht die Möglichkeit das Quiz zu absolvieren und Punkte zu bekommen. Aufgrund der knappen Zeit ist die Entität Quiz relativ grob dargestellt worden. In Kapitel 3 Abschnitt 3.4.4 (Entitäten) wird der richtige Weg beschrieben.

Die Entität Quiz steht nicht in Beziehung zu Question und Selection. Im Projekt sind die beiden Entitäten allerdings aufgelistet.

4.2 Fazit

Aufgrund von Fehleinschätzungen fehlt das Musskriterium, das angemeldete Benutzer das Quiz spielen und Punkte bekommen zu können. Das liegt daran, dass die Quiz Entität grob dargestellt wird ohne weitere Aggregate. Jedoch wurden diese Beispielhaft in der Anwendung angelegt. Jedoch werden diese nicht genutzt. In Kapitel 3 Abschnitt 3.4.4 (Entitäten) wird die richtige Umsetzung. Im Allgemeinen ist Quarkus ein modernes und fortschrittliches Framework. Es ist eine leichtgewichtige Architektur mit nahezu sofortiger Reaktionszeit der Anwendung, weshalb das Umsetzen der Projektes Freude bereitet hat. In Zukunft werden klarere Strukturen definiert, um nicht in Zeitverzug zu kommen.

5 Referenzen

[RoosmannKG, 2022/23]	Prof. Dr. Rainer Roosmann, Software-Archtiektur Konzepte und Anwendungen, 02_SWA_Konzeptionelle_Grundlagen, aufgerufen am 12.02.2023
[RoosmannTG, 2022/23]	Prof. Dr. Rainer Roosmann, Software-Archtiektur Konzepte und Anwendungen, 03_SWA_Technische_Grundlagen, aufgerufen am 12.02.2023
[@Maven]	Apache Maven Project. (2021). About Apache Maven. Abgerufen am 13. Februar 2023, von https://maven.apache.org/what-is-maven.html.
[@Quarkus]	Quarkus.io (2021). What is Quarkus? Abgerufen am 13. Februar 2023, von https://quarkus.io/what-is-quarkus/.
[@Quarkus-Qute]	Quarkus.io (2021). Quarkus-Qute: Lightweight and reactive templating engine. Abgerufen am 13. Februar 2023, von https://quarkus.io/guides/qute.
[@Hibernate]	Hibernate.org (2021). What is Hibernate? Abgerufen am 13. Februar 2023, von https://hibernate.org/orm/what-is-orm/.