Projekt-Lösungsdokument   
Snake-Core, SnakeFX, SnakeServer

Ostfalia Fachhochschule für angewandte Wissenschaften

Benjamin Wulfert

Leonard Reidel

Semester: Wintersemester 2020

18. November 2020

Inhaltsverzeichnis

[Projektdokumentation 4](#_Toc56420289)

[Architektur 4](#_Toc56420290)

[Core 4](#_Toc56420291)

[Berechnung der Spieler-Bewegungen & Positions-Berechnung 4](#_Toc56420292)

[SnakeFX 4](#_Toc56420293)

[Login-Screen 4](#_Toc56420294)

[Registrierungs-Prozess 4](#_Toc56420295)

[Login-Prozess 5](#_Toc56420296)

[Register-Screen 5](#_Toc56420297)

[Home-Screen 5](#_Toc56420298)

[Aktive Spieler 5](#_Toc56420299)

[Aktive Spiele 5](#_Toc56420300)

[Spielhistorie-Screen 5](#_Toc56420301)

[New-Game-Screen 5](#_Toc56420302)

[Game-Screen 5](#_Toc56420303)

[SnakeServer 5](#_Toc56420304)

[Persistenz-Layer 5](#_Toc56420305)

[Repository Pattern 6](#_Toc56420306)

[H2 Database Engine 6](#_Toc56420307)

[API-Layer 7](#_Toc56420308)

[JUnit-Tests 7](#_Toc56420309)

[Tasks / Aufgabenstellungen 8](#_Toc56420310)

[Realisiert 8](#_Toc56420311)

[UML-Diagramme 8](#_Toc56420312)

[User-Interface 8](#_Toc56420313)

[Initiale Asynchronität 8](#_Toc56420314)

[Initialer Konsum der Schnittstelle 8](#_Toc56420315)

[Snake-Implementierung 8](#_Toc56420316)

[Snake-Implementierung – Mechanik - Wegfindung 8](#_Toc56420317)

[Ausstehende Arbeitspakete 9](#_Toc56420318)

[Überarbeitung der UML-Diagramme 9](#_Toc56420319)

[Absicherung der Schnittstelle (API) mittels Spring Security und JSON-Web-Tokens 9](#_Toc56420320)

[Vollständige Implementierung der Klassenrelationen 9](#_Toc56420321)

[Vollständiger Konsum der Schnittstellen 9](#_Toc56420322)

[Austausch der Spielereingaben (Implementierung von Web-Sockets) 9](#_Toc56420323)

[Anhang 10](#_Toc56420324)

[Aktivitätendiagramm 10](#_Toc56420325)

[Anwendungsfalldiagramm 10](#_Toc56420326)

[Klassendiagramm 10](#_Toc56420327)

# Projektdokumentation

Dieses Dokument stellt die Dokumentation für das Projekt dar. Im Folgenden werden die verschiedenen Aspekte des Projekts beschrieben. Des Weiteren wird in dieser Lösungsdokumentation dargelegt welche Aspekte des Systems umgesetzt wurden sowie noch ausstehende Punkte erläutert, welche in den folgenden Wochen umgesetzt werden.

Der aktuelle Projektstand kann dem GitHub-Repository der folgenden URL entnommen werden:  
<https://github.com/Bummelnderboris/Patterns_and_Frameworks>

## Architektur

Das folgende Schaubild stellt die Modul-Architektur des Systems dar.

Core

SnakeFX  
Front-End

SnakeServer  
Back-End

Abbildung - Modul-Architektur

## Core

Das Core-Modul enthält die Kern-Aspekte der Anwendung – dazu zählen beispielsweise Ausschnitte der Modelle welche im Klassendiagramm enthalten sind, die gemeinsam genutzten Endpoints der Schnittstelle sowie Konstanten welche sowohl im Backend als auch im Frontend verwendet werden.

## SnakeFX

Das Modul SnakeFX ist das Front-End der Anwendung. Im Front-End sind die User Interfaces (UI) definiert und implementiert. Des Weiteren enthält konsumiert das Front-End mittels Schnittstelle Daten aus dem Backend. Die Implementierung und die gesamten Mechaniken des Snake-Spiels sind ebenfalls Teil des Front-Ends.

### Login-Screen

Der Login-Screen enthält vier Schaltflächen für die Interaktion des Benutzers. Zwei Texteingaben zur Angabe eines Benutzernamen und eines Passworts – und zwei Schaltflächen / Buttons um die Benutzerangaben (Name, Passwort) mittels Login oder Registrierung an das Backend zu übertragen. Beide Prozesse werden im Folgenden näher beschrieben.

#### Registrierungs-Prozess

Das Front-End sendet einen POST-Request an die HTTP-Schnittstelle des Backends. Diese URL lautet <http://localhost:8080/api/login>. Als Header-Daten des Post-Requests werden der Benutzername sowie dessen Passwort (Hash) versendet. Diese Daten werden auf der Seite des Backends empfangen. Die empfangenen Daten werden mittels Unmarshalling1 vom Backend in eine Instanz der Benutzer-Klasse umgewandelt und in die Datenbank persistiert. Die Nutzerdaten des registrierten Benutzers können anschließend für die Anmeldung am System verwendet werden.

1 - https://docs.oracle.com/javase/7/docs/api/javax/xml/bind/Marshaller.html

### Home-Screen

Der Home-Screen stellt die zentrale Benutzeroberfläche der Anwendung dar. Der Benutzer erhält darauf Zugriff nach einem erfolgreichen Anmeldeversuch. Der Home-Screen bietet die Möglichkeit, alle aktiven Spieler und alle aktiven Spiele des Systems zu betrachten. Des Weiteren gelangt der Benutzer über den Home-Screen zur Spielhistorie-Oberfläche. Durch Betätigung der Schaltfläche „Neues Spiel“ ist der Benutzer in der Lage neue Spielrunden zu definieren und in der Lobby zu veröffentlichen. Aktive Spieler können dann, sofern noch genügend Kapazitäten vorhanden sind, dem Spiel beitreten.

#### Aktive Spieler

Zeigt alle aktiven Spieler (am System angemeldete Benutzer) tabellarisch an.

#### Aktive Spiele

Zeigt alle aktiven Spiele (Spielrunden des Systems) tabellarisch an. Ein Spieler kann an einem Spiel teilnehmen (sofern das Spiel noch nicht begonnen wurde oder ein Spieler-Slot verfügbar ist).

### Spielhistorie-Screen

// TODO

### New-Game-Screen

// TODO

### Game-Screen

// TODO

## SnakeServer

Das Snake-Server Modul stellt das Backend der Anwendung dar. Teil des Moduls ist eine Datenbank sowie deren Anbindung an das Backend. Des Weiteren stellt SnakeServer die verschiedenen Schnittstellen zur Verfügung welche zur Kommunikation mit dem Front-End benötigt werden.

Das Backend ist in verschiedene Ebenen / Layer unterteilt.

### Persistenz-Layer

Der Persistenz-Layer verwaltet die Speicherung, Aktualisierung und den Bezug von Daten aus dem relationalen Datenbank Management-System (RDBMS).

Als RDBMS verwenden wir H2 – welche sowohl als In-Memory als auch als File-Storage variante verwendet werden kann. Dieses Vorgehen erweist sich insbesondere für das Aufsetzen der Anwendung als Hilfreich, da die Installation und Konfiguration des RDBMS entfällt. Des Weiteren vereinfacht das Vorgehen die Realisierung und Nutzung von Unit-Tests.

Für die Realisierung mittels der Programmiersprache Java wird das in Spring enthaltene Spring-Data JPA verwendete, welches eine Spezifikation für JPA (Java Persistence Layer) darstellt. Hibernate wird als Implementierung für das ORM-Framework verwendet und direkt von Spring-Data genutzt.

### H2 Database Engine

Ein weiterer Vorteil der Nutzung von H2 ist das in der Abhängigkeit enthaltenen Weboberfläche zur Verwaltung der Datenbank, sodass auch die Installation und Konfiguration eines solchen Tools entfällt:



Abbildung – Login des User-Interfaces zur Verwaltung der In-Memory / File-Storage Datenbank.  
Die URL zur Datenbank lautet: jdbc:H2:mem:testdb.



Abbildung - User-Interface zur Verwaltung der Tabellen

### API-Layer / Schnittstelle

Der API-Layer definiert die vom Backend bereitgestellten Schnittstellen welche zur Kommunikation vom Frontend mit dem Backend benötigt werden. Die Schnittstelle des Backend basiert auf dem HTTP (Hypertext Transfer Protocol) und stellt eine REST-Schnittstelle dar (Representational State Transfer). Dies bedeutet, dass jeder HTTP-fähige Client die Schnittstelle des Backend konsumieren (z.B. auch Internetbrowser, cURL, etc.) kann.

Das Front-End *SnakeFX* verwendet die Java-Bibliothek *Unirest* für die Kommunikation zwischen Front- und Backend. Die Funktionsweise einer REST-Schnittstelle basiert auf dem Gedanken die grundlegenden Operationen des HTTP – wie z.B. GET, PUT, POST, DELETE, … - auf Endpunkte / URLS eines Systems abzubilden. Dabei soll eine HTTP-GET Anfrage (Request) nur für den Bezug von Daten zuständig sein – ein HTTP-POST oder -PUT Request hingegen für die Entgegennahme neuer Daten.

Die Schnittstelle bietet folgende Endpoints für die Kommunikation mit Clients an:

|  |  |  |
| --- | --- | --- |
| URL | HTTP-Methode | Beschreibung |
| http://localhost:8080/api/login |  |  |
| http://localhost:8080/api/register |  |  |
| http://localhost:8080/spieler/ |  |  |
| http://localhost:8080/spiele/login |  |  |
|  |  |  |

## Tasks / Aufgabenstellungen

In den folgenden Auflistungen werden die im Projekt geplanten Aufgabenpakete vorgestellt. Damit sind sowohl bereits implementierte / realisierte Aufgaben als auch noch ausstehende Aufgaben (-blöcke) gemeint.

### Realisiert

Die folgenden Arbeitspakete wurden im Laufe des Moduls bereits realisiert.

#### UML-Diagramme

Zu Beginn des Moduls wurden sowohl Anwendungsfall-, Aktivitäts- als auch Klassendiagramme erdacht und angefertigt. Anhand dieser Diagramme wurden die ersten Aspekte für das Projekt realisiert und implementiert.

#### User-Interface

Alle Screens / Benutzeroberflächen wurden – teilweise prototypisch – realisiert und mit den dazugehörigen Operationen verknüpft (z.B. öffnet ein Klick auf den Button „Spielhistorie“ die Benutzeroberfläche für die Spielhistorie, ein Klick auf „Neues Spiel“ öffnet die Benutzeroberfläche zur Erstellung eines neuen Spiels.

#### Initiale Asynchronität

Die Benutzeroberfläche für den Login und zur Registrierung neuer Spieler verwendet *Tasks* um die Darstellung des User-Interfaces nicht (durch warten oder andere Tätigkeiten des Systems) zu blockieren.

#### Initialer Konsum der Schnittstelle

Das Front-End nutzt bereits erste Endpunkte der Schnittstellen um Beispielsweise mittels POST neue Spieler am System zu registrieren oder bestehende Spieler am System anzumelden. Des Weiteren ist es möglich alle laufenden Spielinstanzen einer Server-Instanz zu beziehen. Die Kommunikation erfolgt dabei auf Basis von HTTP.

#### Snake-Implementierung

- Leo – dein Part kommt hier rein.

#### Snake-Implementierung – Mechanik - Wegfindung

Für die weitere Entwicklung der Spielmechaniken und das Testen verschiedener Aspekte wurde eine Wegfindung in das Spiel realisiert welche auf zufälligen berechneten Eingaben basiert. Dies ermöglicht das Spielen ohne menschliche Gegner (NPCs). Die Wegfindung funktioniert nach dem Prinzip, dass eine Schlange möglichst lange am Spiel teilnimmt ohne mit einer anderen Schlange oder Wänden zu kollidieren.

### Ausstehende Arbeitspakete

Die folgenden Arbeitspakete müssen in den folgenden Wochen realisiert werden.

#### Überarbeitung der UML-Diagramme

Die Entwicklung verschiedener Systemaspekte zeigt auf, dass die UML-Diagramme (Klassendiagramme) noch verfeinert werden müssen, da die aktuelle Implementierung des Systems und die Inhalte der Diagramme auseinander gelaufen ist.

#### Absicherung der Schnittstelle (API) mittels Spring Security und JSON-Web-Tokens

Im aktuellen Stand der Entwicklung ist die Schnittstelle nicht gegen unautorisierte Zugriffe geschützt d.h. jeder http-fähige Client ist in der Lage Informationen aus dem Backend des Systems zu erhalten. Dies soll verhindert werden, in dem Spring Security für die allgemeine Autorisierung, Authentifizierung und Authentisierung implementiert verwendet werden soll. Des Weiteren soll jede Schnittstellen-Anfrage nur mit einem gültigen JSON-Web-Token möglich sein.

#### Vollständige Implementierung der Klassenrelationen

Im aktuellen Stand der Entwicklung sind noch nicht alle Klassen – und deren Relationen zu anderen Klassen – vollständig implementiert, so wie im Klassendiagramm eigentlich ersichtlich.

#### Vollständiger Konsum der Schnittstellen

Im aktuellen Stand der Entwicklung ist noch nicht jede Funktionalität in Anbetracht auf die Nutzung der Schnittstelle vollständig realisiert. Bereits möglich ist die Registrierung neuer Benutzer sowie der Bezug der aktuell laufenden Spiel-Instanzen. Der Bezug der Spielhistorie bspw. ist noch nicht realisiert.

#### Austausch der Spielereingaben (Implementierung von Web-Sockets)

Spiel-Instanzen können aktuell in verschiedensten Szenarien lokal auf einem PC ausgeführt werden. Die Anforderung, dass mehrere Spieler an einem Spiel teilnehmen können und somit Spielereingaben an das Backend gesendet und von dort aus an alle Clients repliziert werden ist noch nicht realisiert worden.

## Anhang

### Aktivitätendiagramm

Das Aktivitätendiagramm …

### Anwendungsfalldiagramm

Das Aktivitätendiagramm …

### Klassendiagramm

Das Aktivitätendiagramm …