## Modul-Architektur

Das gesamte Anwendungssystem wird mit dem Build-Management-System Apache Maven realisiert. Dabei wurde ein Front-End Modul (SnakeFX), ein Back-End Modul (SnakeServer), und ein Core Modul geplant und implementiert welche im Folgenden beschrieben werden.

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

Core

SnakeFX  
Front-End

SnakeServer  
Back-End

Abbildung 1 - 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. Das Core-Modul enthält also Programmteile welche sowohl im Front- als auch im Backend verwendet werden.

## SnakeFX - Frontend

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

## SnakeServer - Backend

Das Modul SnakeServer enthält alle Aspekte des Backend – dazu zählen die Persistenz-Schicht der Anwendung welche mit Spring Data JPA / Hibernate realisiert werden, so wie eine in Java implementierte Datenbank welche direkt mit dem Backend initialisiert wird (H2) – die Vorteile dieses Vorgehens werden weiteren Verlauf des Dokuments dargestellt. Ein weiterer Aspekt des Backend ist die Bereitstellung der REST-Schnittstelle sowie die Auslieferung der Persistenz-Daten darüber. Der letzte Aspekt des Backend ist die direkte bidirektionale Kommunikation mit den angemeldeten Clients über Web-Sockets um Eingaben der Spieler entgegenzunehmen und an die Teilnehmer einer Runde zu replizieren.

### 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 | POST | Liefert einen JSON-Web-Token zurück welcher für die Kommunikation mit anderen Endpoints genutzt werden kann |
| http://localhost:8080/api/register | POST | Registriert einen Spieler mit einem Benutzernamen und Passwort |
| http://localhost:8080/spieler/ | GET | Gibt eine Liste alle Spieler als JSON zurück |
| http://localhost:8080/spiele | GET | Gibt eine Liste aller Spiele als JSON zurück |
| … | … | … |

### Ausstehende Arbeitspakete

Die folgenden Arbeitspakete müssen in den folgenden Wochen realisiert werden.

#### Integration der Design Patterns

Nach Überarbeitung der UML-Diagramme sollen gemäß der Anforderungen Entwurfsmuster implementiert werden. Die Implementierung des Factory-Pattern könnte bspw. Für die Erzeugung der Entitäten (Food, Power-Ups, etc.) auf dem Spielfeld verwendet werden. Das Composite-Pattern könnte ebenfalls für diese Entitäten (Food, Power-Ups, etc.) verwendet werden – da verschiedene Entitäten im Endeffekt dasselbe Verhalten realisieren (ein Spieler sammelt ein solches ein, eine bestimmte Aktion passiert). Das Observer-Pattern wird bereits im Front-End verwendet – es wäre allerdings auch denkbar weitere Punkte in der Architektur mit diesem Pattern zu versehen wie z.B. bei Highscores und auf deren Veränderung zu reagieren.

#### 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.