Projektdokumentation   
Snake-Core, SnakeFX, SnakeServer

Ostfalia Fachhochschule für angewandte Wissenschaften

Benjamin Wulfert

Leonard Reidel

Semester: Wintersemester 2020

23. Dezember 2020

Inhaltsverzeichnis

[Projektdokumentation 3](#_Toc60231786)

[STOMP-Server 3](#_Toc60231787)

[StompClient 3](#_Toc60231788)

[StompSessionHandler 3](#_Toc60231789)

[ChatController 3](#_Toc60231790)

[WebSocketConfig 3](#_Toc60231791)

# 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 (Dokumente, UML-Diagramme, gesamter Projektcode, etc. ) kann dem folgenden GitHub-Repository entnommen werden:

<https://github.com/benjaminfoo/SnakeFX>

<https://github.com/Bummelnderboris/Patterns_and_Frameworks>

# STOMP-Server

Der folgende Abschnitt beschreibt die Integration des STOMP-Services sowie dessen Bereitstellung. Des Weiteren wird erläutert wie mithilfe eines STOMP-fähigen Clients die Kommunikation zwischen dem Front- und dem Backend hergestellt und verwendet werden kann.

ws://localhost:8080/chat

## StompClient

Der StompClient …

*CONNECTION\_URL* -> Die URL zum STOMP-Service.

## StompSessionHandler

Der StompClient …

## ChatController

Der StompClient …

## WebSocketConfig

Der StompClient …

Was ist ein Broker?

Ein Broker

Was ist ein Endpoint?

Neues Spiel in der Lobby

Authentisierter Client erzeugt neue Spiel definition mittels „Neues Spiel -> Erstellen“ schaltflächen. Im Anschluss dessen wird

/snakeserver  
Endpoint für http-Handshake und STOMP-Upgrade

/topic/  
Broker-Topic für Broadcast-Messages

/topic/messages  
Dient dem Austausch von Chat-Nachrichten

/topic/games  
Informationen über alle (laufenden) Spiele in der Lobby

/app/games/{gameId}

Ein Aufruf an /app/games/1 mit einer GameInputMessage hat zufolge, dass alle Clients welche bspw. die URL „ws://localhost:8080/app/game/1“ subscribed sind alle folgenachrichten der Clients erhalten – somit ist die Eingabe für Multiplayer gesichert =)

# STOMP / Websockets – Use Cases

Der folgende Abschnitt beschreibt die verschiedenen Anwendungsfälle in denen STOMP / WebSocket Nachrichten ausgetauscht werden.

## Neues Spiel wird erstellt

Spieler -> NewGameScreen -> Sendet LobbyMessage an Backend -> /app/games

LobbyMessage enthält(erstellender Spieler („admin“), spielDefinition)

Backend -> empfängt nachricht auf /app/games -> ruft broadcastGames() auf

Backend überprüft ob Spiel in LobbyController vorhanden

Wenn nicht -> trage Spiel als runningGame ein, setze den erstellenden Spieler als Admin

Backend -> Broadcasted LobbyMessage an subscribte Clients

Client -> onLobbyMessageReceived() wird aufgerufen -> update der UI

## Spieler tritt Spiel bei

1. Spieler wählt in ListView ein RunningGame aus
2. Spieler klickt auf „Trete Spiel bei“ / joinGame-Button
3. Client -> Sendet PlayerJoinsGameMessage
   1. PlayerJoinsGameMessage enthält (spieler, runningGame)
4. Backend empfängt nachricht in StompLobbyController
5. Backend ruft broadcastPlayerJoinedGameToClients auf
6. Backend trägt Client als Spieler des RunningGames ein, teilnehmer ++
7. Broadcast an subscribte Clients
8. Client erhält Callback auf onPlayerJoinedGameMessageReceived
9. Client aktualisiert UserInterface mit LobbySpielen
10. Client subscribed sich auf Spiel mit ausgewählten GameTopic
    1. = Subscribtion auf Stomp-Pfad – z.b. /app/games/1/<SpielerName>

## Admin startet Spiel

1. Admin-Client -> Sendet GameSessionMessage an Backend
2. GameSessionMessage enthält STARTING, userName, runningGame