Dokumentation der Softwarearchitektur des Projekts Explosion Guy

Bösl Florian *f.boesl@oth-aw.de*
Kohl Helge *h.kohl@oth-aw.de*
Anastasia Chernysheva *a.chernysheva@oth-aw.de*
Korinth Patrice *p.korinth@oth-aw.de*
Porsch Philipp *p.porsch@oth-aw.de*

***Zusammenfassung*—In diesem technischem Report wird die Softwarearchitektur des Projekts Explosion Guy vorgestellt. Das Projekt wurde im Rahmen der Vorlesung Big Data und Cloud-basiertes Computing implementiert. Ziel der Implementierung ist es, das allgemein bekannte Spiel „Bomberman " zu adaptieren und es mithilfe einer Cloud-Infrastruktur online spielbar zu machen.**

***Index Terms*—Webdesign, Matchmaking, Cloud Infrastructure**

I. Introduction

Bei der Applikation Explosion Guy geht es darum, das Spiel „Bomberman" zu adaptieren. Dabei werden dem Spieler mithilfe eines graphischen Web-Interfaces basierend auf Phaser anschauliche Informationen über den Verlauf des Spiels mitgeteilt. Das Interface dient auch dazu, die Eingaben des Spielers an das Backend weiterzureichen. Im Backend werden die Informationen aller Spieler verarbeitet und so ein online Spielerlebnis erzeugt.

II. Architektur Allgemein

Als Architektur wird ein Ansatz verfolgt, der sich an einer Client-Server Kommunikation, ähnlich wie in anderen Onlinespielen, orientiert. Dadurch wird eine abgeschlossene Logik von Front- und Backendkomponenten erreicht, die jeweils von einem Subteam entwickelt wird. Diese Einheiten werden jeweils als isolierte Applikation gekapselt. Während das Backend als Anwendung innerhalb der Containervisualisierung Docker betrieben wird, wird das Frontend als allgemeine Webapplikation ausgeliefert. Durch eine zentrale Konfigurationsdatei des Containers kann dieser einfach zwischen den Teammitgliedern ausgetauscht und ausgeführt werden. Damit wird eine hohe Portabilität gewährleistet. Die Kommunikation der Teilsysteme wird durch Websockets angeboten. Der Informationsaustausch über Websockets ermöglicht es, durch die identitätsbasierten Verbindungen die Kommunikation zwischen den Spielern und der zugehörigen Partei im Backend effizient zu gestalten. Dadurch wird es möglich ein Echtzeiterlebnis mit zusätzlicher Überwachung aus dem Backend zu realisieren. Durch die Einbettung der verschiedenen Bestandteile in einen Cloud-Service ist es jederzeit möglich, ein neues Spiel zu starten, oder einem vorhandenen Spiel beizutreten, da durch die nahezu unbegrenzten Ressourcen des Cloud-Services auf Anfrage neue Spielserver erstellt und verwaltet werden können.

III. Frontend: Spielinterface

Mithilfe des Web-Frameworks Phaser ist es möglich, ein graphisch ansprechendes Interface für das Spiel zu erstellen. Dabei werden nicht nur die Ausgangsinformationen in Form von Veränderungen des Spielfeldes für den Spieler bereitgestellt, sondern auch die Eingaben an das System vom Spieler erfasst. Diese Eingaben leitet das Frontend an das Backed weiter, von wo aus sie dann weiter verarbeitet werden können. Um ein möglichst flüssiges und ruckelfreies Spielerlebnis zu erzeugen, wurde das Frontend auch mit Teilen der Spiellogik ausgestattet. So können beispielsweise alle Bewegungen direkt nach Drücken der entsprechenden Eingabe ausgeführt werden ohne die Antwort des Backends abzuwarten, um Verzögerungen zu vermeiden. Sollte der Server nach Kontrolle einen ungültigen Spielzug feststellen, so wird der Zug zurückgesetzt. Dies erfüllt neben der Aufwertung des Gameplays auch eine kleine Anti-Cheat Rolle.

Die Funktionalität des Frontends umfasst folgende API-Zugriffe:

Filmsuche (/db/search-movies)

- Beschreibung: Bei der Filmsuche werden in der Datenbank passende Filme gesucht.
- Parameter:
  - Teiltitel
  - Provider
- Rückgabe: Liste mit Filmen und deren Metadaten

Personensuche (/db/search-persons)

- Beschreibung: Bei der Personensuche werden in der DBpedia Daten zu Filmmitwirkenden eines Filmes gesucht.
- Parameter:
  - Filmtitel
  - Erscheinungsjahr
- Rückgabe: Liste der Filmmitwirkenden und dazugehörigen Metadaten

Gemeinsamkeitensuche (/db/compare-movies)

- Beschreibung: Bei der Gemeinsamkeitensuche werden gemeinsame Merkmale einzelner Filme gesucht.
- Parameter: Liste mit Filmtiteln
- Rückgabe: Liste der Gemeinsamkeiten der Filme

Suche nach ähnlichen Filmen (/db/search-similar-movies)

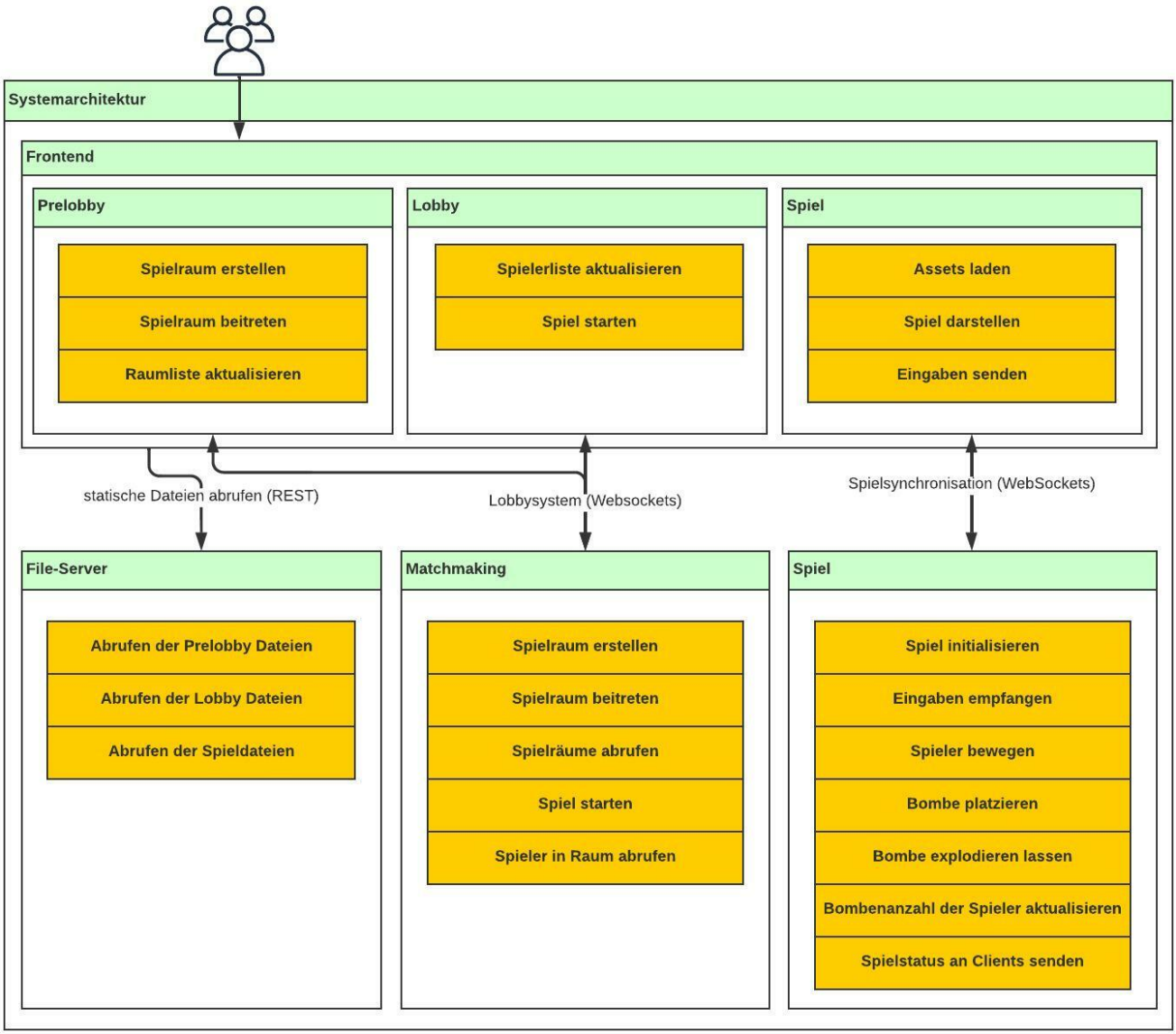

Abbildung 1. Übersicht Gesamtsystem

- Beschreibung: Bei der Suche nach ähnlichen Filmen werden Filme gesucht, die Gemeinsamkeiten mit den angebegeben Filmen aufweisen.
- Parameter:
  - Liste mit Filmtiteln
  - Streamingdienstanbieter der zu vergleichenden Filme
- Rückgabe: Liste mit Filmen

Suche nach IMDB Daten eines Filmes (/imdb/search-imdbdata)

- Beschreibung: Bei der Suche nach einem Film werden die Bewertungen und ein hinterlegtes Bild des Filmes in der IMDB gesucht.
- Parameter: Titel des Filmes
- Rückgabe:
  - Liste mit Bewertungen des Filmes
  - URL eines in der IMDB hinterlegten Bildes

Die Verwaltung der Daten erfolgt mittels einer Virtuoso-Datenbank. Diese enthält Filmdaten der Streamingdienstanbieter Netflix, Amazon-Prime, Disney+ und Hulu, welche aus Kaggle bezogen wurden. Die Datensätze enthalten Filmtitel der jeweiligen Provider, sowie dessen Metainformationen wie Filmmitwirkende, Genre und Erscheinungsjahr. Mithilfe einer in Virtuoso angelegten View ist es möglich mit SparQl-Statements auf die importierten Daten zu zugreifen.

Das Modul FFetch SPARQL Endpointërmöglicht es innerhalb der Filmverwaltung eine Verbindung zwischen dem Node-Server und der Virtuoso-Datenbank sowie der DBpedia aufzubauen. Anschließend können SPARQL-Statements abgesetzt und die Ergebnisse asynchron Verarbeitet werden.

Die IMDB ist eine Datenbank zu Filmen, Fernsehserien, Videoproduktionen und Computerspielen sowie über Personen, die daran mitgewirkt haben. Sie bietet eine API zum Zugriff auf diese Daten. Dafür wird ein API-Schlüssel benötigt, mit diesem sind täglich 100 Anfragen kostenfrei verfügbar. Nach überschreiten dieser Begrenzung sind keine weiteren Anfragen an diesem Tag möglich, die Anfragenbegrenzung kann jedoch kostenpflichtig erhöht werden.

Um die vorgesehene Funktionalität gewährleisten zu können, wurden mithilfe von Jest, einem JavaScript Testframework, Unit- und Integrationstests entworfen. Dadurch kann die interne Integrität, sowie die Anbindung an externe Systeme, sichergestellt werden. Die Tests wurden dabei mit dem Arrange-Act-Assert-Pattern entworfen. Dieses soll eine übersichtliche Struktur und Einheitlichkeit garantieren.

IV. Backend: Spiellogik und Matchmaking

Das Backend besteht aus drei Aufgabenpaketen. Einerseits hat es die Aufgabe, die Informationen aller Spieler entgegenzunehmen, diese zu verarbeiten und sie danach an die jeweils anderen Spieler weiterzuleiten. Dies erfolgt nach dem „Authoritative servers and dumb clients“-Prinzip bei dem der Server immer den tatsächlichen Spielstatus hält und diesen an die Clients weitergibt. Andererseits muss das Backend die Funktionalität des Matchmakings übernehmen. Dazu muss es den Spielern die Möglichkeit geben, eigene Spielräume zu erstellen oder vorhandenen Räumen beizutreten. Die dritte Aufgabe des Backends ist es die nötigen Dateien für das Frontend auszuliefern. Um diese Funktionen optimal abzudecken, wurde das Backend in drei Teilbereiche getrennt, welche im Folgenden ausführlich beleuchtet werden.

A. Spiellogik

Das Paket der Spiellogik bildet das erste wichtige Bestandteil des Backends. Mit Spiellogik werden alle Abläufe, die im Hintergrund zu erfüllen sind, bezeichnet. So beispielsweise die Positionsbestimmung der Spieler nach einer Eingabe oder das Zählen eines Timers der Bomben. Zur Realisierung der Spiellogik wird ein Server mithilfe von Node.js innerhalb eines Docker Containers aufgebaut. Der Server verwaltet die Logik mit verschiedenen Klassen:

- **Spieler** Die Spielerklasse enthält alle nötigen Informationen eines Spielers. Darunter dessen ID zur eindeutigen Identifizierung, seine Position auf dem Spielfeld, die Anzahl seiner Bomben und den ob dieser aktuell noch am Leben ist.
- **Bombe** Die Bombenklasse enthält ähnlich wie die Spielerklasse die Information ob über die Position des Bombenobjekt ihrer Explosionsstärke. Mithilfe eines Timers löst diese nach ablaufen desselben ein Explosionsevent aus, wodurch das Spielfeld angestoßen wird, die Explosion zu verarbeiten.
- **Playground** Die Playgroundklasse spiegelt das Spielfeld wieder. Sie enthält ihre Größe sowie die Spieler und Bombenobjekte. Zusätzlich werden die Positionen von Wänden und zerstörbaren Objekten gespeichert. Mithilfe der Playgroundklasse werden die Spielereingaben auf die umgesetzt und der Spielstatus aktualisiert. Bei der Explosion einer Bombe wird zusätzlich die Kollision mit Objekten verarbeitet.
- **Game** Die Gameklasse enthält die übergeordneten Funktionen für Spiellogik. Sie erstellt den Playground und ruft die nötigen Daten für die Initialisierung des Frontends ab. Zusätzlich leitet diese die Spielereingaben an das Playgroundobjekt weiter.

Der Spielserver hört auf folgendes Websocket-Event von Clientseite:

- Event: **input**
  - Beschreibung: Bei jeder Eingabe auf der Clientseite wird dieses Event ausgelöst. Die Eingabe wird an den Spielserver gesendet und von diesem verarbeitet. Je nach Eingabe wird ein spezifisches Antwort-Event zur Aktualisierung der Clients ausgelöst.
  - Parameter „action“ kann folgende Werte annehmen:
    - left
    - right
    - up
    - down
    - bomb

Der Spielserver löst auf Clientseite folgende Websocket-Events aus:

- Event: **newGameCreated**
  - Beschreibung: Die Clients werden darüber informiert, dass ein neues Spiel gestartet wurde. Sie erhalten alle nötigen Informationen um das Spiel auf Clientseite zu initialisieren.
  - Parameter „mWidth“: Spielfeldbreite
  - Parameter „mHeight“: Spielfeldhöhe
  - Parameter „player“: ID, Name und Position jedes Spielers
  - Parameter „layer1Data“: Matrix des Spielfeldes mit unzerstörbaren Wänden
  - Parameter „layer2Data“: Matrix des Spielfeldes mit zerstörbaren Objekten
- Event: **update**
  - Beschreibung: Antwort auf das „input“-Event eines Clients. Die Clients werden über darüber informiert wie sich das Spielfeld durch den Input verändert hat.
  - Fall 1 Input war Bewegung:
    - Parameter „input“: Enthält den zugehörigen Inputtyp

    - Parameter „data“: Enthält Array Spieler-IDs und deren Position
  - Fall 2 Input war Bombe:
    - Parameter „input“: Enthält den zugehörigen Inputtyp
    - Parameter „PlayerId“: Enthält die ID des Spielers der eine Bombe gesetzt hat
    - Parameter „BombCount“: Enthält den aktualisierten Bombcount des Spielers
    - Parameter „PosX“: Enthält die X-Position der Bombe
    - Parameter „PosY“: Enthält die Y-Position der Bombe
    - Parameter „BombStrength“: Enthält die Explosionsstärke der Bombe
- Event: **explode**
  - Beschreibung: Die Clients werden darüber informiert, dass eine Bombe explodiert. Sie erhalten alle nötigen Informationen um das Spiel auf Clientseite zu aktualisieren.
  - Parameter „bomb“: Position und Stärke der explodierenden Bombe
  - Parameter „hitPlayers“: ID und isAlive Status der getroffenen Spieler
  - Parameter „destroyedObstacles“: Position der zerstörten Hindernisse
  - Parameter „explosionPositions“: Positionen der Felder die Explodieren
- Event: **Refresh**
  - Beschreibung: Die Clients werden darüber informiert, dass ein Spieler eine neue Bombe erhält.
  - Parameter „Id“: ID des Spielers dessen Bombenanzahl aktualisiert wird
  - Parameter „BombCount“: Anzahl der Bomben des Spielers

B. Matchmaking

Das zweite wichtige Paket für die Gesamtfunktion ist das Paket des Matchmakings. Darunter lassen sich alle Mechanismen zusammenfassen, die es den Spielern ermöglichen, einen Raum zu erstellen oder einem Raum beizutreten. Die Spieler werden also bei Verbindungsaufbau zunächst mit dem Server verbunden. Dieser leitet sie dann entweder an einen neu erstellten Spielraum oder an einen bereits vorhandenen Raum weiter. Innerhalb eines Spielraumes übernimmt dann die Spiellogik die Verwaltung, indem sie die für sich relevanten Spieler mithilfe ihrer Socket-ID adressiert. Der Server kennt also alle aktuellen Spielräume mit ihren Spielerzahlen und kann bei Bedarf neue Spielräume erstellen. Wird die Anzahl dieser Räume irgendwann so groß, dass sie ein einzelner Server nicht mehr versorgen könnte, könnte theoretisch ein neuer Server für die Spielräume in der Cloud-Infrastruktur hochgefahren werden (wurde aber nicht mehr implementiert).

- Event: **createGame**
  - Beschreibung: Möchte ein Client in der Prelobby ein Spiel erstellen, drückt er dort den entsprechenden Button und dieses Event wird ausgelöst. Die Übergabeparameter werden validiert. Ist die Validierung erfolgreich, wird eine Spieler-ID generiert, der Spielname und die Spielerdaten in der playersList gespeichert und mittels callback wird die Spieler-ID an den Client gesendet. Der kann sich nun zur Lobby und Spielseite weiterleiten. Sein Zugangsschlüssel ist die Player-ID.
  - Parameter „room“ enthält den Spielnamen und „playername“ den Spielername
  - Callback: „errorCode“ und „status“ als Validierungsinformationen und „playerId“
- Event: **joinGame**
  - Beschreibung: Möchte ein Client in der Prelobby einem Spiel beitreten, wählt er dort das entsprechende Spiel aus und dieses Event wird ausgelöst. Wie bei createGame erfolgt eine Validierung. Die Spielerdaten mit Spieler-ID werden in der playersList beim entsprechenden Spiel hinzugefügt.
  - Parameter „room“ enthält den Spielnamen und „playername“ den Spielername
  - Callback: „errorCode“, „status“ und „playerId“ (wie bei createGame)
- Event: **getGames**
  - Beschreibung: Wenn der Client in der Prelobby die aktuell offenen Spiele aktualisieren möchte, drückt er dort den entsprechenden Button und dieses Event wird ausgelöst. Es werden die aktuell aktiven rooms (entsprechen Spielnamen) abgefragt. Dann werden die schon laufenden Spiele (runningGamesList) von der Liste entfernt und die Liste dem Client zurückgeliefert.
  - Callback: „rooms“ (Array mit Spielen, denen beigetreten werden kann)
- Event: **joinRoom**
  - Beschreibung: Hat ein Client erfolgreich in der Prelobby ein Spiel erstellt bzw. ist einem Spiel beigetreten, wird mit dem Routenwechsel zur Lobby bzw. zum Spiel dieses Event ausgelöst. Der Client übermittelt dabei den Spielnamen, seinen Spielernamen und seine Player-ID. Es wird überprüft, ob die Player-ID hinterlegt ist. Wenn ja wird der Client zum entsprechenden room hinzugefügt bzw. eröffnet diesen und der socket wird in die playersList eingetragen. Des Weiteren wird ein Event an den room ausgelöst, dass die aktuellen Spielernamen des Spiels liefert, damit diese beim Client automatisch aktualisiert werden.
  - Parameter „room“, „playerId“ und „playername“
  - Callback: „errorCode“ und „status“ als Validierungsinformationen
- Event: **startGame**

  - Beschreibung: Drückt ein Spieler in der Lobby auf den Button um das Spiel zu starten, wird dieses Event ausgelöst. Nach erfolgreicher Validierung, dass das Spiel gestartet werden kann, wird die Initialisierungs-Methode des Spiels aufgerufen. Dabei werden der room und die Spielerdaten (inklusive aller Client-Sockets des rooms) übergeben. Außerdem wird der room als aktives Spiel in die runningGamesList eingetragen.
  - Parameter „room“ mit room- bzw. Spielnamen
  - Callback: „errorCode“ und „status“ als Validierungsinformationen

C. File Server

Das letzte Packet ist ist die File-Server-Funktionalität des Backends. Um das Spielen über den Browser des Nutzers zu ermöglichen müssen die nötigen Dateien abrufbar sein. Hierfür wurde ein eigenständiger File-Server aufgesetzt der die abgefragten Dateien ausliefert. Während der Entwicklung liegen diese Dateien auf dem Dateisystem des Entwicklers, im Live-Betrieb sollen diese in einem Amazon S3-Bucket abgelegt werden.

D. Testing

Zum Testen der verschiedenen Klassen und Funktionen des Backends wurde mithilfe von Jest für jede Klasse ein Testfile angelegt. Darin enthalten sind mehrere Testfälle, die teilweise iterativ die einzelnen Komponenten des Systems prüfen. Die Verwendung von Jest ermöglicht zudem eine Coverage-Prüfung, mit der es möglich ist, die genauen Prozentwerte der Testabdeckung sowie nicht abgedeckte Zeilen zu ermitteln. Nach Korrekter Einrichtung der einzelnen Testfälle ist es dann möglich, durch einen einzelnen Befehl die Integrität des gesamten Programmcodes zu verifizieren.

- **Snapshot-Tests:** Die Snapshot-Tests erzeugen eine HTML-Document in welches die zu testende Komponente gerendert wird. Anschließend wird die HTML-Datei mit einer Snapshot-Datei verglichen, um sicher zu stellen das die Komponente wie erwartet dargestellt wird.
- **Funktionstest:** Bei den Funktionstests wird für jede Komponente eine Instanz mit dem Enzyme-Adapter erstellt. Mit dieser Objektinstanz könne verschieden Funktionen der Klasse getestet werden und geprüft werden ob Felder des Objekts richtig gesetzt werden.

LITERATUR

[1] Axios [Online] https://www.npmjs.com/package/axios (visited on Jan. 7, 2022)

[2] Microservice-Frontend-Architekturen [Online] https://www.sigs-datacom.de/uploads/tx_dmjournals/attermeyer_OTS_Microservices_Docker_16.pdf (visited on Jan. 7, 2022)

[3] React-Testing [Online] https://testing-library.com/docs/react-testing-library/intro/ (visited on Jan. 7, 2022)

[4] Enzyme [Online] https://www.npmjs.com/package/enzyme (visited on Jan. 7, 2022)