Designbeschreibung

Version 3.0

Lukas Klein

Dokumentenhistorie

Datum	Version	Beschreibung	Autor
26.03.2021	Version 1.0	Erstellung des Dokuments	Thomas Zimmermann
17.04.2021	Version 2.0	Erweiterung UML	Lukas Klein
30.04.2021	Version 3.0	Neuauflage aufgrund von nichtbestehen	Lukas Klein

Inhaltsverzeichnis

1. Allgemeines

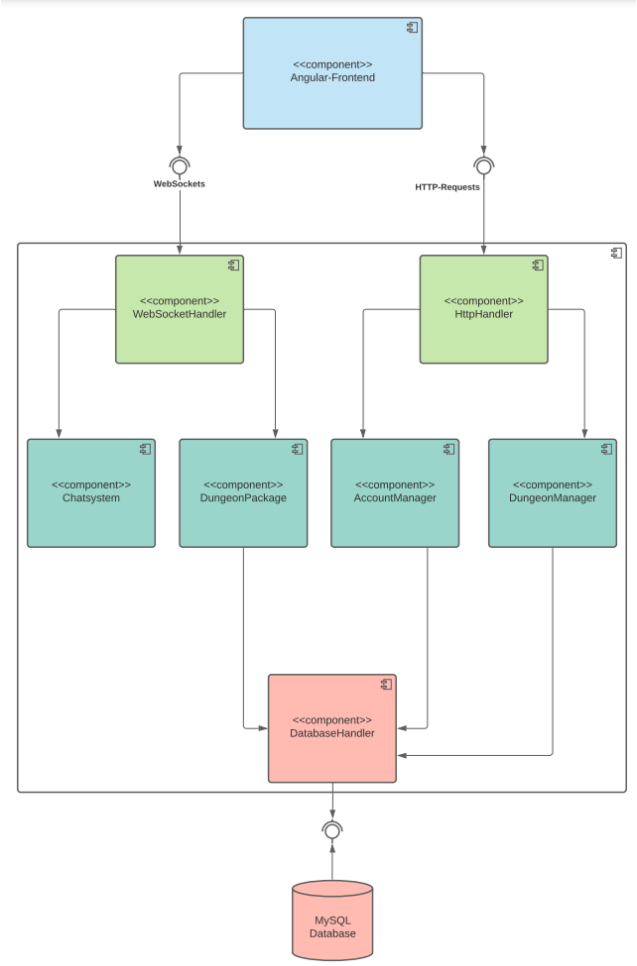
Diese Designbeschreibung enthält alle wichtigen Informationen zu den Struktur- und Entwurfsprinzipien der Software.

Dem Kunden soll durch den Einsatz des Produktes die Möglichkeit gegeben werden einen Server zu hosten welcher ein Multi-User-Dungeon Spiel und Konfigurator für seine Nutzer bereitstellt.
Dieses Produkt ist ein solcher Konfigurator und bietet dem Kunden eine Möglichkeit Dungeons möglichst frei zu gestalten und seine Kreativität umzusetzen. Die konfigurierten Dungeons können von anderen Nutzern nach der Erstellung und Veröffentlichung gespielt werden.

Das Produkt basiert auf einer Web-Architektur und utilisiert WebSockets sowie http-Requests für die bidirektionale Übermittlung von Daten von der Web-Client/-Server-Architektur zum Anwendungs-Server. Der Anwendungs-Server, welcher für die Steuerung von Geschäftsprozessen zuständig ist, ist in Python verfasst und bietet ebenfalls eine Anbindung an den Daten-Server in Form einer MySQL-Datenbank.

1.1. Architektur

Das Frontend ist per Angular Anwendung dargestellt. Es kommuniziert mit dem Backend via Websockets und Http-Requests. Im Backend werden die Anfragen der Schnittstellen über jeweilig ein WebsocketHandler und ein Http-Handler. Diese beiden Handler empfangen die Anfragen der Schnittstellen und verteilen sie auf die entsprechenden Komponenten im Backend. Teilweise sprechen Backendkomponenten auch die Datenbank an. Dies ist allerdings über eine eigene ausgelagerte Klasse geregelt.



2. Produktübersicht

2.1. Äußere Funktionsmerkmale

Dieses Produkt ist eine Browser Anwendung, welche eine grafische Oberfläche bietet.
Bevor der Nutzer mit der grafischen Oberfläche interagieren kann, muss er sich registrieren. Hat der Nutzer sich bereits registriert, muss er sich nur einloggen. Der Nutzer kann jederzeit sein Passwort zurücksetzen, oder seinen Account unwiderruflich löschen.
Sobald der Nutzer eingeloggt ist, hat er die Möglichkeit sich einen Dungeon auszusuchen, indem er eine Liste durchsucht oder nach bestimmten Kriterien filtert.

Zusätzlich kann ein Nutzer neue Dungeons erstellen. Hierbei kann er die Größe des Dungeons festlegen und beliebig viele Räume hinzufügen. Neben der Kartenkonfiguration hat der Nutzer die Möglichkeit, Aspekte wie beispielsweise Klassen, Rassen, Items und NPCs zu erstellen.
Ist der Dungeon erstellt, so kann der Nutzer den Dungeon speichern und anschließend veröffentlichen. Dabei kann er eine Zugangsbeschränkung festlegen, wobei er hierbei die Auswahl zwischen einem öffentlichen, beziehungsweise einem privaten Dungeon hat. Öffentlich bedeutet, dass jeder Nutzer beitreten kann, sofern die maximale Spieleranzahl nicht erreicht ist. Privat legt fest, dass der Dungeon Master bestimmen kann, ob ein Nutzer beitreten kann oder nicht. Dies geschieht über eine Beitrittsanfrage.
Jeder Nutzer kann seine selbst erstellten Dungeons kopieren und gegebenenfalls bearbeiten, oder löschen. Der Dungeon Master kann dies auch zur Laufzeit des Spiels, was zur Folge hat, dass vorhandene Charaktere und ihre Daten (Klasse, Rasse, Beschreibung des Charakters und Items) unwiderruflich gelöscht werden.

Sobald sich ein Nutzer für einen Dungeon entschieden hat, muss er sich zuerst einen Charakter konfigurieren. Dabei muss er sich eine definierte Klasse, Rasse aussuchen sowie einen Namen und eine Beschreibung zu seinem Charakter hinterlegen. Wenn der Nutzer das Spiel verlässt, kann er danach immer wieder beitreten, ohne sich einen neuen Charakter anzulegen und wird an der zuletzt hinterlegten Position in das Spiel gelassen. Fortan nimmt er aktiv als Spieler teil und wird von dem Dungeon Master durch das Geschehen geführt. Der Spieler kann durch vordefinierte Befehle, die er in die Konsole eingibt, bestimmte Aktionen ausführen. Unter diesen Befehlen gibt es bestimmte Basisaktionen, welche in jedem Spiel verfügbar sind.
Zu Basisaktionen zählen Befehle, wie beispielsweise sich im Raum umzuschauen, den Raum in die vier Himmelsrichtungen zu verlassen, sofern ein Raum in dieser Richtung vorhanden ist. Spieler können ebenfalls Gegenstände aufnehmen, welche sie in einem Inventar ihres Charakters lagern können.
Eine weitere wichtige Basisaktion ist das Chatten. Der Spieler hat hierbei drei Möglichkeiten, um zu kommunizieren:
1. Innerhalb eines Raumes mit allen lokalen Spielern.
2. Mit dem Dungeon Master, unabhängig von der Position.
3. Mit einem spezifischen Spieler privat, welcher sich im selben Raum befindet.

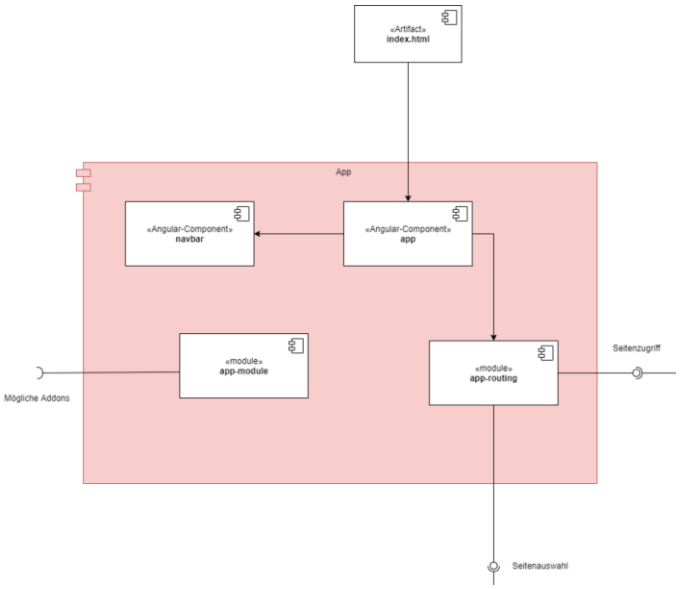
Wenn eine Aktion ausgeführt wird, führt dies zu einer Benachrichtigung bei dem Dungeon Master, welche er beantworten muss, damit die Aktion als abgeschlossen angesehen wird. Das Chatten ist von dieser Regelung ausgeschlossen.

Die Antwort des Dungeon Masters beinhaltet eine Beschreibung der Ausgangssituation und kann die Lebenspunkte des Spielers beeinflussen. Hierbei kann der Dungeon Master Aspekte des Charakters (Klasse, Rasse, Beschreibung des Charakters und Items) in Betracht ziehen, um den Ausgang zu spezifizieren. Wenn die Lebenspunkte eines Charakters nach einer Aktion auf null fallen sollten, so wird der Charakter aus dem Dungeon gelöscht und der Spieler muss sich einen neuen Charakter anlegen.

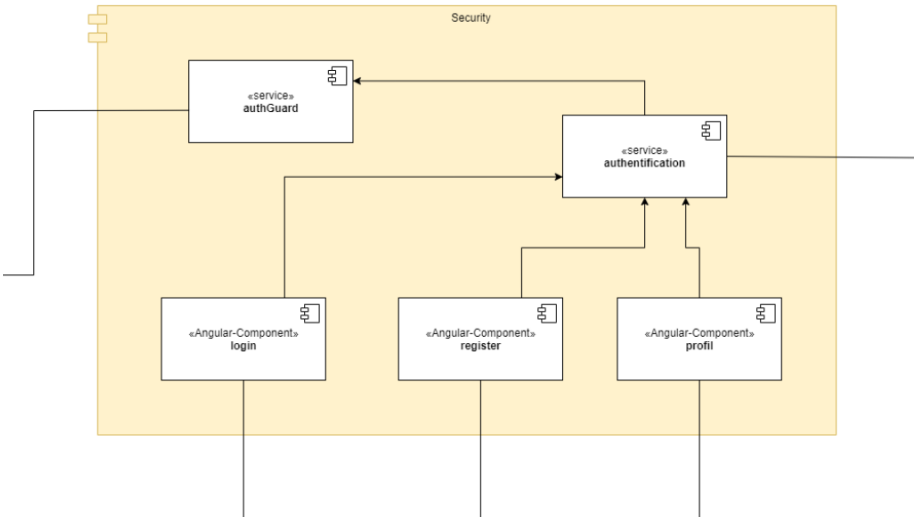
Wenn der Dungeon Master das Spiel verlässt wird das Geschehen unterbrochen. Dies bedeutet, dass dieses pausiert wird und die Spieler innerhalb des Dungeons keine Aktionen

mehr ausführen können. Ein Dungeon Master kann nach Belieben in ein Spiel wiedereintreten. Der Wiedereintritt des Dungeon Masters hebt jegliche Pausierung des Spielgeschehens auf. Damit der Dungeon Master mit den Spielern direkt kommunizieren kann, hat er die Möglichkeit Spieler privat über eine Chatfunktion anzuschreiben. Darüber hinaus kann er eine Rundnachricht an alle Spieler in einem Spiel senden.

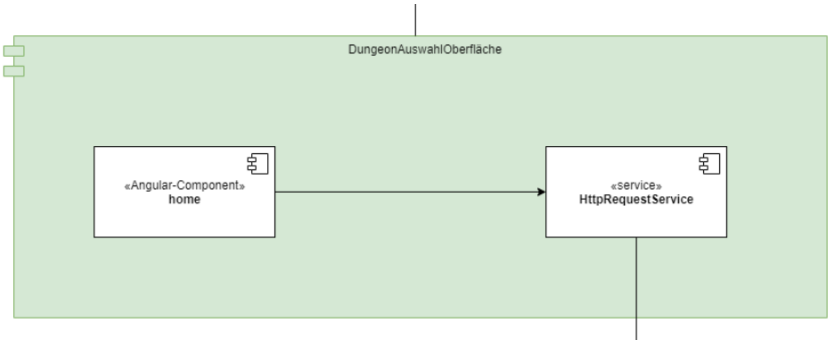
3. Aufbau des Frontends



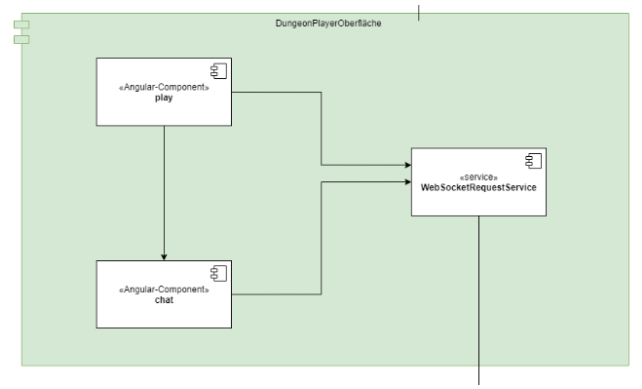
Unsere Anwendung wird über die auf dem Server liegende Index.html Datei aufgerufen. Diese repräsentiert unsere Angular Applikation welche sich aus der NavBar-Komponente (auch als Header bezeichnet) und der durch das Routing-Module ausgewählten Komponente besteht. Zusätzlich können über die sogenannten App-Modules zusätzliche Bibliotheken eingebunden werden. Wie schon erwähnt, steuert das Routing-Module jeweils anzuzeigende Komponenten.



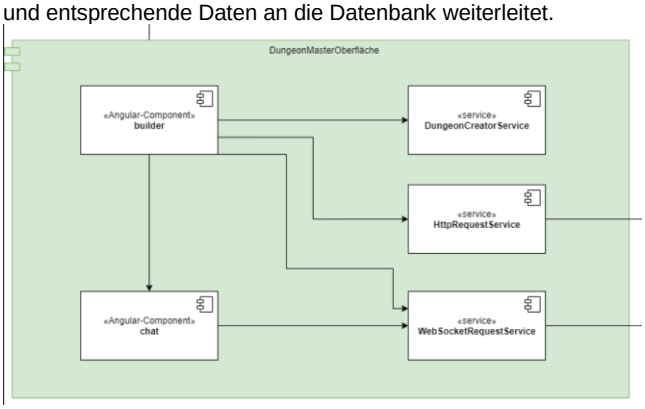
Zusätzlich wird dieser Vorgang noch durch einen AuthGuard Service unterstützt, welcher checkt, ob ein Nutzer die Berechtigung hat, entsprechende Inhalte wie zum Beispiel die DungeonPlayerOberfläche darzustellen. Der AuthGuard Service steht in Verbindung mit dem Authentifikation Service, welcher für die Registrierung und Anmeldung von Nutzer sorgt und somit auch die zu überprüfenden Daten für den AuthGuard zur Verfügung stellt.



Die DungeonAuswahlOberfläche Komponente ist die Landing Page Komponente. Sie kann ohne jegliche Authentifikation immer aufgerufen werden. Über Sie ist es möglich, die über den AuthGuard überwachten Komponenten wie DungeonMasterOberfläche oder DungeonPlayerOberfläche aufzurufen. Jede einzelne Oberflächen Komponente steht mit einem zu ihr passenden Service in Verbindung, welcher die Anfrage an unser Backend steuert.



Zum Speichern und Aufrufen von Daten in Verbindung mit der Datenbank wird hierbei immer das http Backend benutzt, welches http Anfragen wie Option, Post und Get bearbeiten kann und entsprechende Daten an die Datenbank weiterleitet.

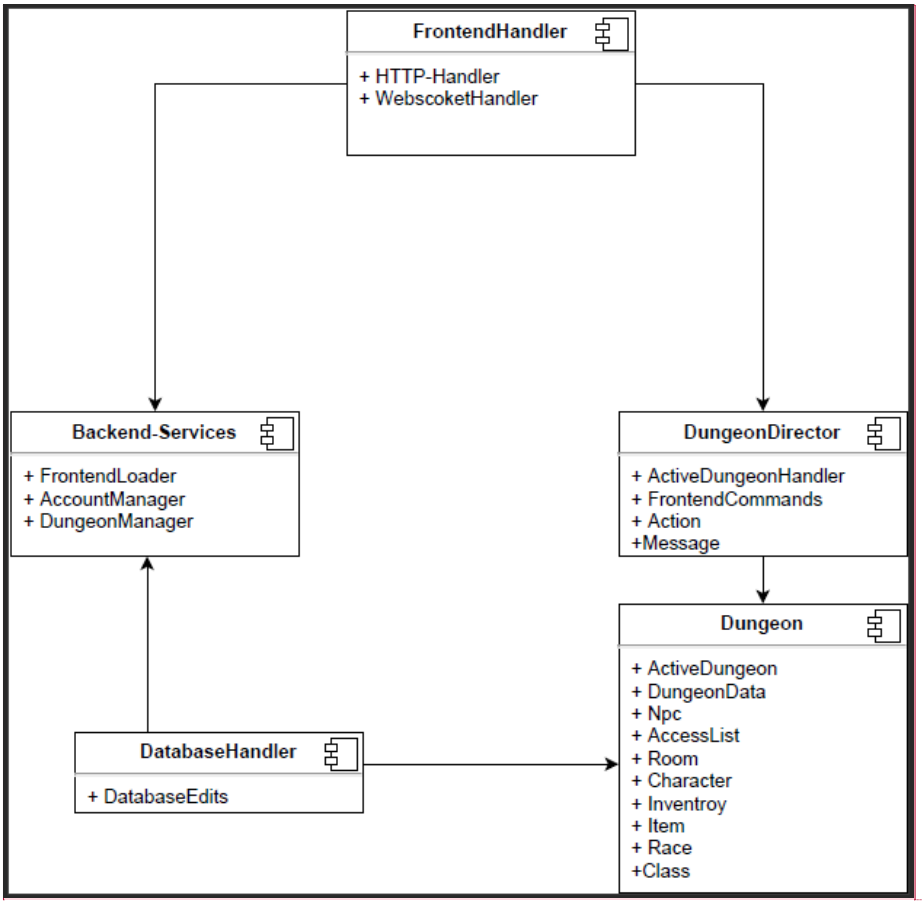


Für das generelle Spielgeschehen wird zum Beginn einmal der jeweilige Datenbestand aus der Datenbank entnommen und kann nun über den WebSocket Server dynamisch bearbeitet werden. Erst bei Beendigung des Spieles gehen die Daten wieder zurück in die Datenbank.

Siehe **Komponentendiagramm MUDCAKE.pdf**

4. Aufbau des Backends

4.1. Komponentendiagramm

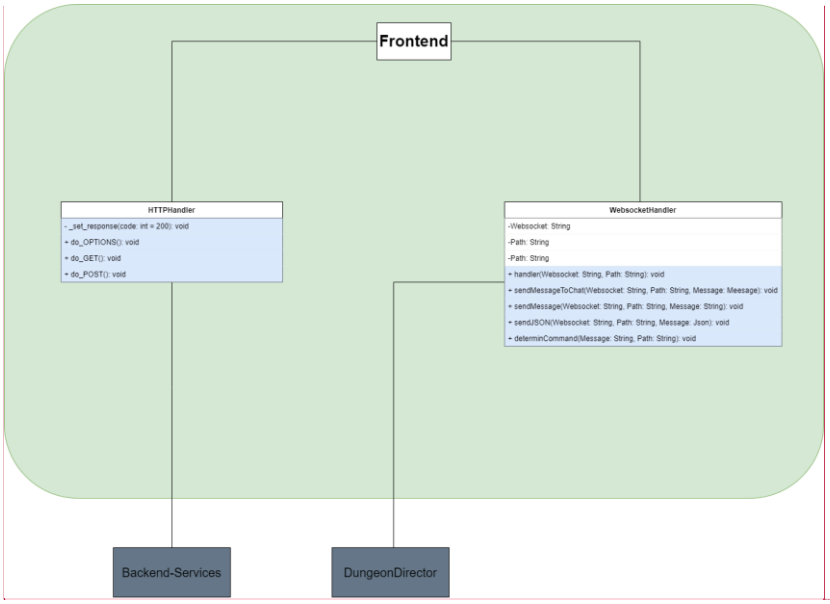


Kommentiert [JG1]: FrontendHandler und Backend Services nutzen nun auch Dungeon

Das Klassendiagramm ist in 5 Hauptkomponente unterteilt. Der Frontend Handler ist zuständig für die Kommunikation zwischen Backend und Frontend. Hier sind also der HTTP sowie der WebsocketHandler enthalten, die wiederum die Kommunikation weiterleiten und verarbeiten. Die Backend Services regeln die HTTP Anfragen die vom Frontend kommen. Außerdem ist sie die Komponente die für den Registrierungsprozess verantwortlich ist und arbeitet entsprechend mit der Klasse Email Service zusammen. Über die Klasse Backend Services wird zudem gesteuert wie relevante Dungeondaten aus der Datenbank geladen und in die Datenbank gespeichert werden (siehe Sequenzdiagramm: Save Dungeon). Der Email Service, welcher von den entsprechenden Backend-Services verwendet wird, ist sowohl bei der Registrierung eines Users als auch wenn ein User sein Passwort vergessen hat dafür verantwortlich, dass entsprechende Emails versendet werden. Der JSON Serializer ist für die Serialisierung und Deserialisierung von JSONs verantwortlich. Bei der Deserialisierung einer JSON nimmt die entsprechende Methode eine JSON auseinander und verteilt die Daten an die entsprechenden Objekte. Bei der Serialisierung geschieht das Gegenteil,

JSONs werden aus den bestehenden Objekten erstellt. Die Komponente Dungeon Director stellt den Hauptkommunikationsweg zwischen Frontend und dem Spiel zur Verfügung. Die Komponente enthält die aktiv gespielten Dungeons. Außerdem werden alle Chateingaben im Spiel über die Komponente gesteuert. Über die Klasse Frontend Commands in der entsprechenden Komponente holt sich das Frontend die Daten die es für das Spiel benötigt. Das DungeonPackage enthält prinzipiell alle Relationen die auch in der Datenbank vorkommen, allerdings sind diese weiter aufgeteilt und haben teils andere Eigenschaften. Da ein Dungeonpackage immer eine Instanz darstellt, gelingt es somit auf den verschiedenen Instanzen zu spielen. Per lazy-loading wird sichergestellt, dass immer nur das geladen wird was auch benötigt wird. Der Database Handler enthält eine Klasse, den Database Handler und die Datenbank selbst. Der DatabaseHandler ist die einzige Klasse, die auf die Datenbank zugreift. Sie schreibt und lädt Daten und gibt diese wiederum an das DungeonPackage und die BackendServices weiter.

4.2. Frontend Handler



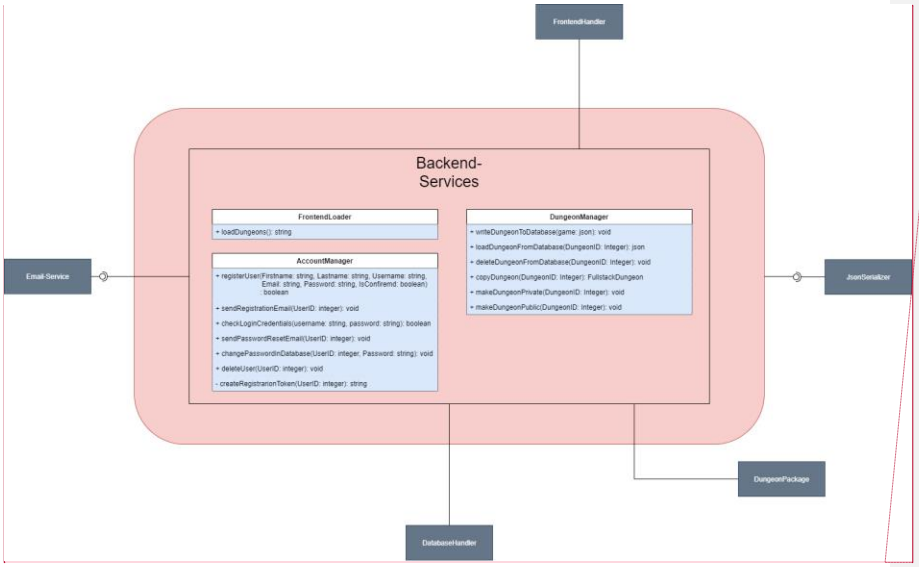
Kommentiert [JG2]: WebsocketHandler und HTTPHandler wurden um alle benötigten Methoden ergänzt

Der Frontendhandler besteht aus mehreren Klassen. Diese Klassen sind der HTTP und der Websocket-handler.

http-Handler
Der http-Handler Empfängt und bearbeitet die HTTP-Requests so weit, dass erkenntlich wird, welcher Command genau angefragt wurde. Hierfür wird der empfangene String geparst und anhand dessen der Entsprechende Befehl innerhalb des Backend-Services aufgerufen.

Websocket-Handler
Der Websocket-Handler Empfängt ausschließlich Aktionen während des Ausführens des aktuellen Spielgeschehens. Hierfür besitzt er den handler, welche Nachrichten empfängt, sendMessageToChat, welches eine Nachricht während des Spiels ich Chat verteilt, send Message, welches eine Nachricht mit Dateninhalt an das Frontend weitergibt, „sendJSON“, welches eine Nachricht an das Frontend per JSON weitergibt, und „determineCommand“, welches Filtert ob der eingegangene String ein Userbefehl per Chat oder ein Befehl des Frontends ist. In beiden Fällen wird im DungeonDirector eine entsprechende Klasse aufgerufen.

4.3. Backend-Services



Kommentiert [JG3]: FrontendLoader existiert so nicht. AccessManager kam dazu. JsonSerializer wird nicht verwendet.

Der Backend-Service besteht aus mehreren Klassen. Diese Klassen sind einmal der FrontendLoader, der Dungeon Manager und der AccountManager.

Dungeon Manager
Der Dungeon Manager ist dafür zuständig per http-Anfragen gestellte Befehle eine entsprechende Methode aufzurufen. Hierfür wird gegebenenfalls der JSON-Serializer verwendet, welcher JSONs trivial serialisiert und deserialisiert. Die deserialisierten JSONs können innerhalb der Methoden von Dungeon Manager weiterverarbeitet werden. Zusätzlich verwendet der Dungeon Manager die Schnittstelle der Datenbank.

Innerhalb der Methode „writeDungeonToDatabase“ wird das Dungeon über den DatabaseHandler in die Datenbank übertragen zusätzlich wird über den ActiveDungeonManager überprüft ob das jetzt gespeicherte Dungeon aktuell bespielt wird, wenn das der Fall ist muss zusätzlich das aktuell bespielte Dungeon überschrieben werden.
Innerhalb der „loadDungeonFromDatabase“ Methode wird ein Dungeon aus der Datenbank geladen und ans Frontend weitergegeben, dies kann nützlich sein, wenn Beispielsweise ein noch nicht veröffentlichtes Dungeon weiterbearbeitet werden soll.
Die Methode „deleteDungeon“ löscht per ID innerhalb der Datenbank das angegebene Dungeon und gegebenenfalls auch das Dungeon, wenn es aktuell bespielt wird.

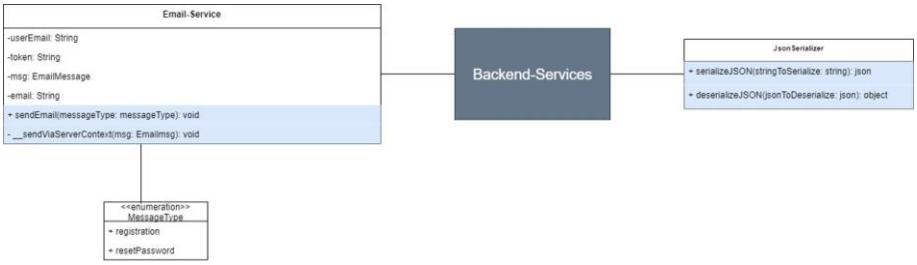
„CopyDungeon „ist eine Methode, welche ein Dungeon per ID aus der Datenbank abruft und eine neue entsprechende Instanz erstellt (selbstverständlich mit geänderter ID).
„MakeDungeonPrivate“ ändert den private Status eines Dungeons, welches per ID angegeben ist, auf true. True repräsentiert hier in diesem Fall, dass das Dungeon nun privat ist.
„MageDungeonPublic“ änder den private Status eines Dungeons, welches per ID angegeben ist, auf false. False repräsentiert in diesem Fall, dass ein Dungeon nun öffentlich zugänglich ist.

AccountManager
Der AccountManager ist zuständig für das Management einer Nutzer Accounts. Hierfür benötigt dieser eine Schnittstelle an die Datenbank und eine Schnittstelle an das Email-Service Modul.
Der AccountManager besitzt eine Methode namens „registerUser.“ Diese Methode bekommt alle Eingaben mittgeteilt, welche der Nutzer bei der Erstellung seines Accounts angibt. Diese Eingaben werden in die Datenbank geschrieben. Zusätzlich wird ein RegistrationToken durch die private Methode „createRegistrationToken“ erstellt und mit in die Datenbank geschrieben. Anschließend ruft „registerUser“ eine weiter Methode auf, welche „sendRegistrationEmail“ heißt, diese ist public, weil sie auch außerhalb des ursprünglichen Registrierungsprozesses aufgerufen werden soll, wenn Beispielsweise auf den „Email erneut senden“ Knopf gedrückt werden würde. „sendRegistrationEmail“ nutzt das Email-Service Modul um innerhalb dessen Emails versenden zu können.
„CheckLoginCredentials“ ist eine Methode, welche die angegebenen Daten mit denen aus der Datenbank vergleicht und unter Bedingung der Gleichheit True zurückgibt, andernfalls False. Hierdurch wird dem Frontend mitgeteilt ob ein Nutzer die korrekten Daten angegeben hat.
Die Methode „sendPasswordResetEmail“ sendet eine Email an einen Nutzer der sein Passwort zurücksetzen möchte. Dementsprechend gibt es auch eine Methode die „changePasswordInDatabase“ heißt, diese ist dafür zuständig das angegebene Passwort bei dem angegebenen UserID zu ändern.
„DeleteUser“ ist für entsprechendes löschen eines Nutzer Accounts zuständig und löscht alle entsprechenden Einträge in der Datenbank.

FrontendLoader
Der FrontendLoader ist für das laden der Dungeondaten auf der Startseite zuständig. Hierfür wird über den DatenbankHandler eine Anfrage geschickt, welche alle relevanten Dungeondaten lädt. Zu den relevanten Dungeondaten gehören: Dungeonname, Dungeonbeschreibung, MaxPlayers und den Private-Status.

4.4. Additional Classes

Kommentiert [JG4]: JsonSerializer wurde entfernt

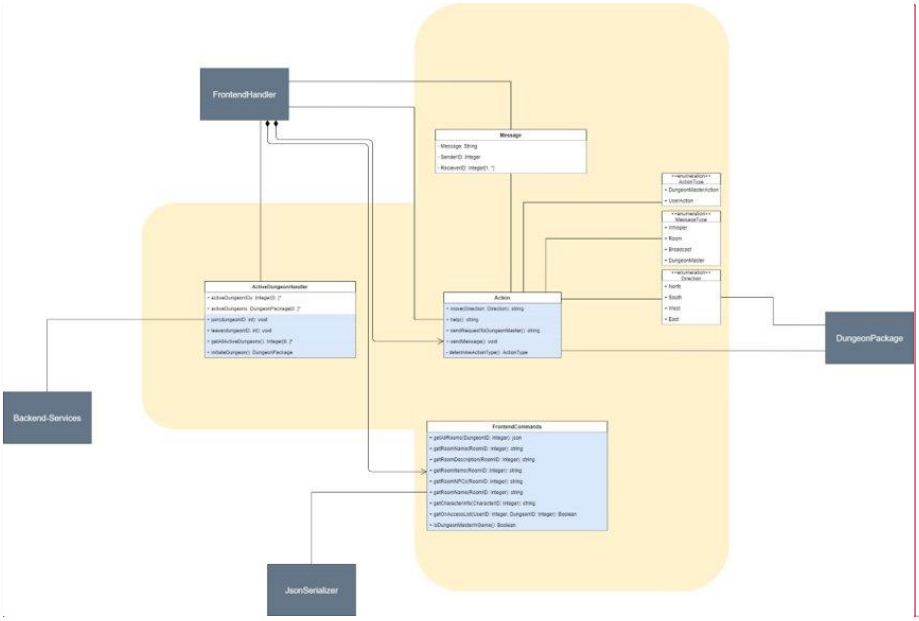


Die Additional Classes unterteilen sich in „Email-Service“ und in „JSON-Serializer“. Der Email-Service ist dafür zuständig vorgefertigte Emails mit spezifischen Variablen zu versenden.
Wohingegen der JSON-Serializer dafür Zuständig ist trivial JSON Dateien zu serialisieren und zu deserialisieren.

Email-Service
Der Email-Service besitzt 4 Attribute, welche bei Initialisierung des Objektes gegeben sein müssen. Zunächst muss die Email des Empfängers angegeben sein, damit diese innerhalb der Klasse als Empfänger Email genutzt werden kann. Anschließend wird ein „MessageType“ angegeben, welcher die Art der vorgefertigten Email bestimmt welche gesendet wird. Zusätzlich wird ein Token mitgegeben, welcher relevant für die Bestätigung der Email und Zurückstetzen des Passworts ist.
Es wird ausschließlich von außen die Methode sendEmail aufgerufen, wobei hier innerhalb intern die Methode „sendViaServerContext“ verwendet wird.

JSON-Serializer
Der JSON Serializer besitzt zwei Methoden. Einmal „serializeJSON“, welche einen String als Parameter bekommt und als Ausgabe eine dem String entsprechende JSON-Datei liefert.
Die Zweite Methode ist „deserializeJSON“, welche als Parameter eine JSON-Datei bekommt und dementsprechend einen String zurück gibt.

4.5. Dungeon Director



Kommentiert [JG5]: Es kamen weitere Methoden hinzu

Die Komponente Dungeon-Director dient primär als Hauptkommunikationsweg zwischen Frontend und Spiel. Die Klasse ActiveDungeonHandler beinhaltet die aktuell aktiven Dungeons sowie in einer getrennten Liste die entsprechenden DungeonIDs. Die Klasse kommuniziert mit dem WebsocketHandler, dem Dungeon-Package und den Backend Services und dient somit als zentrale Zwischeninstanz. Die Klasse FrontendCommands dient dazu dem Frontend je nach Bedarf Daten aus der Komponente DungeonPackage zur Verfügung zu stellen. Um es an einem Beispiel festzumachen, wird die Methode „getRoomName“ gewählt. Das Frontend möchte dem Spieler den Namen des Raums anzeigen in diesem er sich momentan befindet und holt über die entsprechende Methode den Namen aus der Klasse Raum.
Die Klasse Action innerhalb der Komponente DungeonDirector nimmt vom WebsocketHandler Chateingaben entgegen und bearbeitet diese entsprechend. Chateingaben können Basisaktionen (move oder help), Chatnachrichten an andere Spieler oder eine Aktionsanfrage/antwort sein.
Der Aktions/MessageType der Nachricht wird über die Enums des Pakets mitgegeben. Aufgrund der Aktionen vom Spieler zum DungeonMaster und zurück wird von der Klasse Action eine Verbindung zum DungeonPackage erforderlich.
Eine Message besteht immer aus einem Sender, gegebenenfalls mehreren Empfänger und einer Textnachricht.

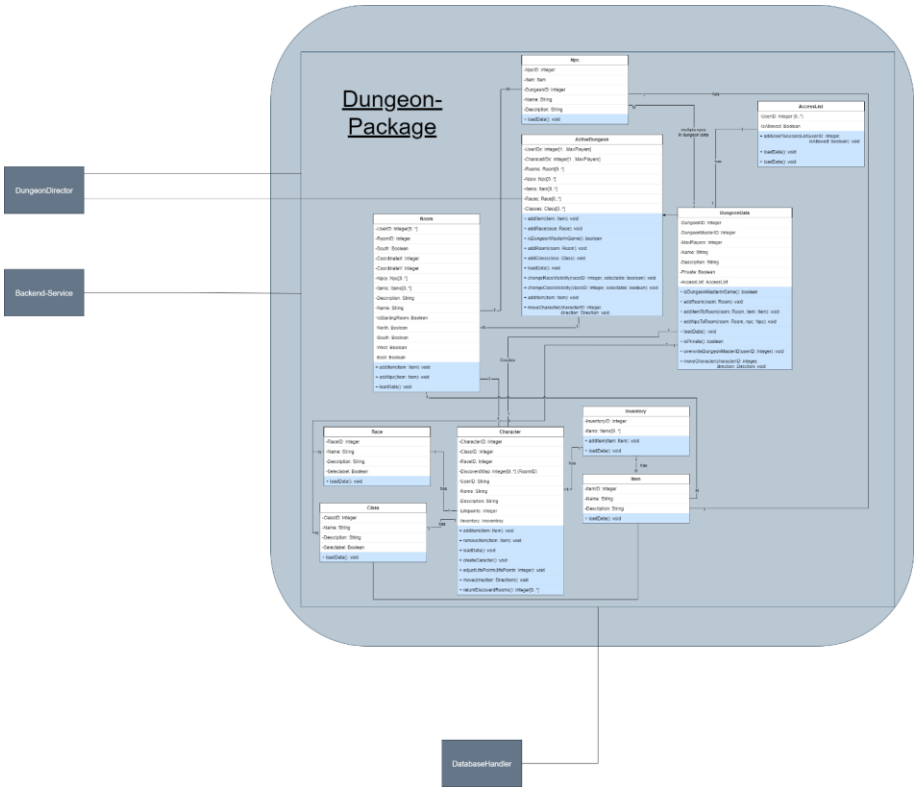
AcitveDungeonHandler
Der ActiveDungeonHandler verfügt über eine join-Methode, welche dafür zuständig ist die ID eines Dungeons, welches aktuell bespielt wird in eine Liste einzutragen um kontrollieren zu können, welche Dungeon aktuell online sind. Zusätzlich wir auch eine Instanz des entsprechenden Dungeon gespeichert.

Außerdem verfügt der AcitveDungeonHandler über eine leave Methode, welche die entsprechende ID aus der ID Liste löscht und die entsprechende Instanz aus dem aktiven Dungeons terminiert.
Zusätzlich ist es möglich alle IDs der aktuellen Dungeons auszugeben.

Action
Die Action Klasse ist eine Klasse welche zuständig für das be- und verarbeiten des Userinputs ist.
Die „move“ Methode leitet mittels eines Enumerators (Directions) die Art der Bewegung an das DungeonPackage weiter.
Die Action Klasse besitzt eine weitere Methode namens „help“, diese gibt einen einfachen String zurück, welcher alle Aktionen beinhaltet die der Nutzer zum aktuellen Zeitpunkt ausführen kann.
„SendRequestToDungeonMaster“ ist eine Methode, welche über ein separates Message Objekt eine Nachricht über den Websocket weitergeben kann. Diese Message besteht immer aus einem String als Nachricht, einer SenderID, welche die ID des Senders darstellt und einer RecieverID, welche die gegebenenfalls mehren IDs der Nutzer darstellen.

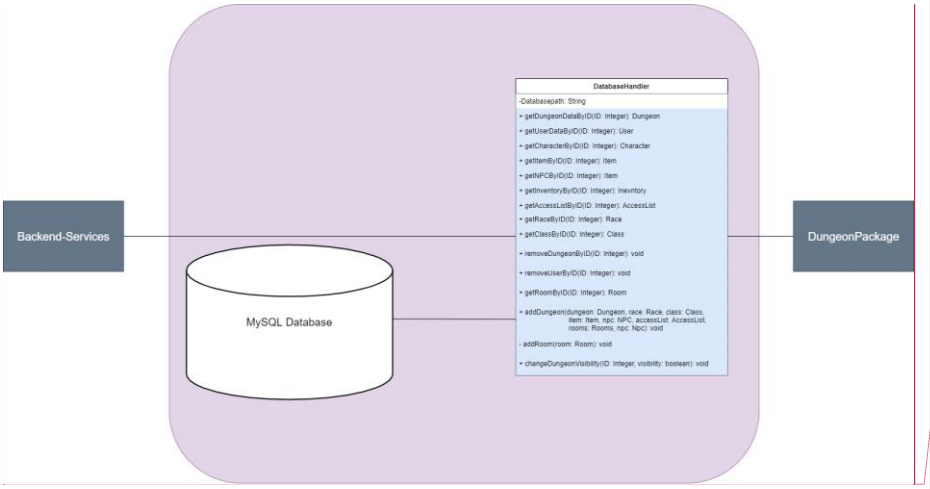
FrontendCommands
FrontendCommands ist eine Klasse, welche alle abfragen abdeckt die unser Frontend unabhängig der Nutzereingabe stelle kann. Als Beispiel das Abfragen eines Raumnamens, welches nach ausführen der Methode das entsprechende Dungeon-Package Objekt nach dem Namen anfragt.

4.6. Dungeon-Package



Das „DungeonPackage“ enthält alle Klassen, die für den Dungeon in erster Linie relevant sind. Es hat direkten Zugriff auf den „DatabaseHandler“ und kann sich somit alle nötigen Daten aus der Datenbank holen sie bzw. dort speichern. Während des gesamten Spielverlaufes existiert nur eine einzige Instanz, pro aktivem Spiel, des „DungeonPackage“. Die Klasse „AktiveDungeon“ hält alle nötigen Informationen, um die aktive Spielinstanz bei laufendem Betrieb zu verändern. Alle Daten, die für den aktiven Spielbetrieb nicht gebraucht werden, werden von den „DungeonData“-Klasse bereitgehalten. Jede Klasse im Dungeonpackage hat die Methode „loadData()“ mit der die entsprechenden Daten für das entsprechende Objekt vorbereitet werden können. Somit werden nur dann Daten von der Datebank geladen, wenn sie auch wirklich benötigt werden.

4.7. DatabaseHandler



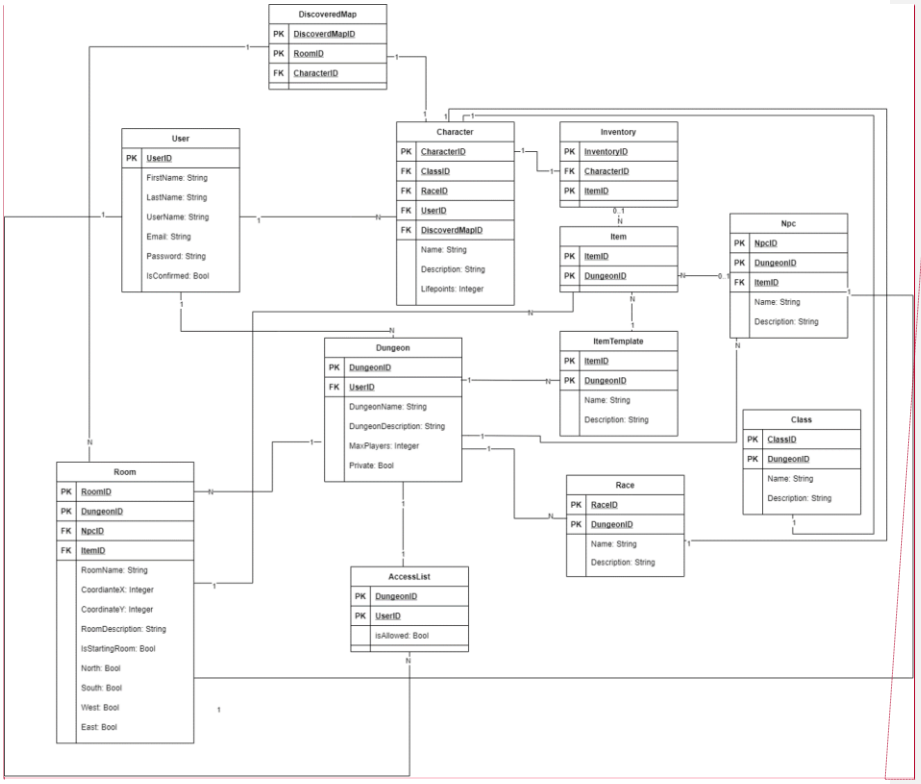
Kommentiert [JG6]: Es kamen einige methoden hinzu

Der Databasehandler fungiert als Dataaccess und die einzige Klasse, die auf die Datenbank zugreifen kann. Sowohl Dungeonpackage als auch die Backendservices müssen den Databasehandler nutzen, um Daten in die Datenbank zu schreiben und um Daten aus der Datenbank auszulesen. Somit werden redundante Klassen vermieden da mehrere Services dieselbe Klasse im Databasehandler verwenden können, wenn sie dieselben Daten benötigen. Sollte zum Beispiel in Zukunft die Datenbank gewechselt werden, so muss dementsprechend nur der Databasehandler gewechselt bzw. modifiziert werden. Der DatabaseHandler ist zusätzlich ein Singelton Objekt, was bedeutet, dass es in unserem gesamten Service nur eine Instanz dieses Objektes gibt.

DatabaseHandler-Methoden

Der DatabaseHandler besitzt Methoden wie Beispielsweise „getDungeonByID“, nach einem ähnlichen Schema sind die meisten Datenbank abfragen aufgebaut. Es gibt also mehrere Anfragen, welche innerhalb des DatabaseHandlers per SQL ausformuliert sind.
Außerdem ist es möglich mit dem DatabaseHandler Objekte wie ein gesamtes Dungon in die Datenbank zu schreiben. Dies geschieht zunächst über eine Öffentliche Methode, welche anschließend für Listen-Objekte, wie beispielsweise mehrere Räume in einem Dungeon,eine private Methode aufruft, welche dann alle diese Räume einzeln innerhalb der Datenbank plaziert

5. Aufbau der Datenbank



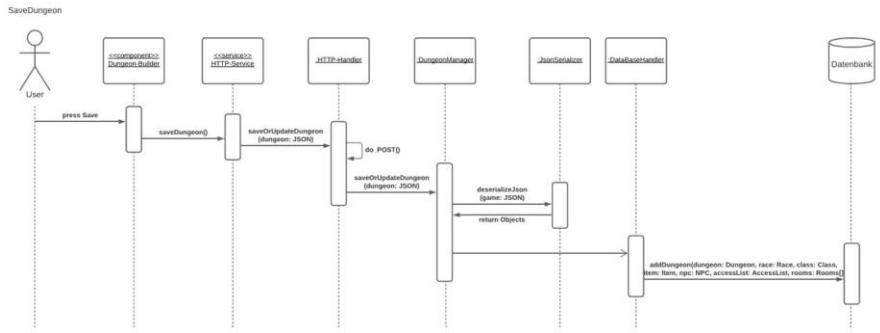
Kommentiert [JG7]: Item Template wurde rausgenommen. Bei character wurde der pk auf user id und dungeon id verändert. In inventory wurde characterid mit den anderen zum pk, discovered room hat jetzt userid dungeon id und room id.

Um die Daten eines Dungeons zu persistieren und abrufbar zu machen, wird eine Datenbank benötigt. In diesem Fall wurde die relationale Datenbank MySQL gewählt. Da die Daten hauptsächlich in JSONs an das Frontend geliefert werden, müssen im Backend Umformungen für die Datenbank geschehen, damit die erste Regel der Normalform des relationalen Modells eingehalten wird. Der User besitzt den Primärschlüssel UserID und Attribute wie Vorname, Nachname, die hinterlegte E-Mail, ein Passwort und ein Attribut isConfirmed vom Typ Boolean, um festzustellen, ob der User seinen Account bereits bestätigt hat. Der User kann beliebig viele Dungeons erstellen, jedoch ist ein Dungeon genau einem User zugewiesen. In der Dungeon Relation selbst befindet sich Attribute wie Dungeonname, Dungeonbeschreibung, Anzahl der Maximalen Spielerkapazität, und ob der Dungeon privat oder öffentlich sein soll. Als Fremdschlüssel hat der Dungeon eine UserID und als Primärschlüssel eine entsprechende DungeonID. Betrachtet man nun die Konfigurationsdaten wie Item, ItemTemplate, Npc, Class, Race und Room, so muss angemerkt werden, dass es sich hier um schwache Entitätstypen handelt. Schwache Enitätstypen sind Entitäten die nicht allein durch die eigenen Attribute, sondern nur durch eine zusätzliche Beziehung zu Entitäten eines übergeordneten Entitätstyps genau definiert werden. Aufgrund dessen müssen in den zuvor genannten Relationen der Fremdschlüssel DungeonID zum individuellen Primärschlüssel hinzugefügt werden. Aufgrund dieser Relation zwischen Dungeon und den Konfigurationsdaten, hat der Dungeon zu allen schwachen Entitätstypen eine 1:N Beziehung. Die Relation AccessList hingegen darf nur ein einziges Mal innerhalb eines Dungeons vorkommen. Bei der AccessList wird mittels

eines Boolean festgelegt, ob ein User direkt in einen Dungeon eintreten darf oder ob ihm der Zugriff auf den Dungeon verweigert wird. Weitere besondere Relationen sind DiscoveredMap und ItemTemplate. Die DiscoveredMap Relation dient dazu die bereits entdeckten Räume eines Charakters zu erfassen. Hierbei werden beliebig viele Räume für genau einen Charakter abgespeichert. Die ItemTemplate Relation wird benötigt, um bestimmte Item Konfigurationen zu löschen, ohne die tatsächlichen Instanzen zu löschen, damit diese nicht unangekündigt von einem Charakter-Inventar entfernt werden. Ein Charakter besitzt mehrere Items. Dies wird dargestellt durch die Inventory Relation. Das Inventar dient als Liste, in welcher die gesammelten Items hinterlegt werden. Ein Inventar hat somit beliebig viele Items. Ein NPC kann ebenfalls ein Item haben, falls der Dungeon Master dies als nötig sieht. Aufgrund dessen braucht ebenfalls der NPC die ItemID als Fremdschlüssel. Ein Character wird definiert durch eine Klasse, Rasse, den zugehörigen User und der bisher erkundeten Räumen. Diese Eigenschaften werden dargestellt durch entsprechende Fremdschlüssel zu anderen Relationen. Der Charakter hat weitergehend immer einen Namen, eine Beschreibung und eine Anzahl an momentanen Lebenspunkten. Letztlich gibt es noch die Raum Relation. In einem Raum kann es mehrere Items geben sowie einen NPC geben. In einem Raum können Ausgänge in die vier Himmelsrichtungen festgelegt werden, welche die Verbindung zwischen mehreren Räumen bildet. Einem Raum kann zugewiesen werden, ob dieser ein Startraum sein kann. Neben einer Beschreibung und einem Namen, gibt es X und Y Koordinaten die die Position des Raumes innerhalb des Dungeons definieren.

6. Sequenzdiagramme

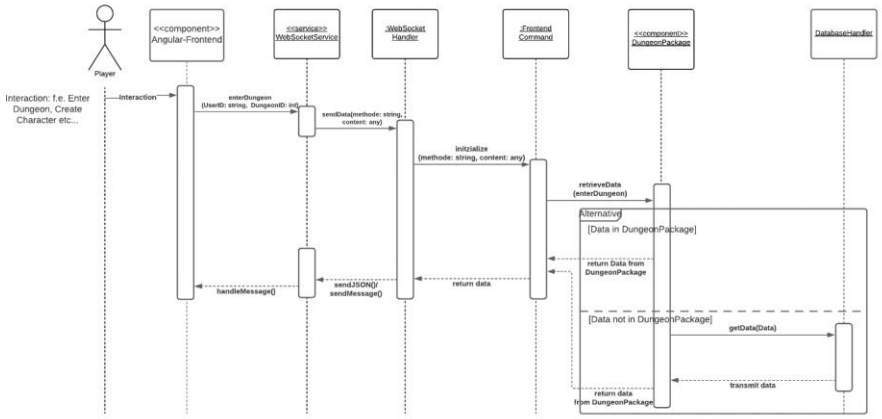
6.1. http: SaveDungeon



Der User welcher als Dungeon Master seine Dungeons konfiguriert hat die Möglichkeit diesen nach Belieben zu speichern. Dies geschieht durch den Klick auf einen entsprechenden Button. Dieser löst in der Dungeon-Builder Komponente eine Methode aus, welche den HTTP-Service aktiviert. Im HTTP-Service wird das generierte JSON Objekt hinterlegt und an das Backend weitergeleitet. Der HTTP-Handler hat die wesentliche Aufgabe festzumachen was für ein Request eingetroffen ist. Sobald diese evaluiert ist, wird die entsprechende Methode ausgeführt. In diesem Fall wäre das die Methode saveOrUpdateDungeon. Diese nimmt ein JSON Objekt entgegen und instanziiert im selben Zug den DungeonManager. Der DungeonManager hat die Aufgabe das JSON Objekt, mithilfe einer zusätzlichen Schnittstelle, namens JsonSerializer in die einzelnen Konfigurationsdaten aufzuspalten. Die Objekte, welche in einer JSON mit Dungeon Konfigurationsdaten enthalten sind, sind Klassen, Rassen, Items, ItemTemplates, Räume,

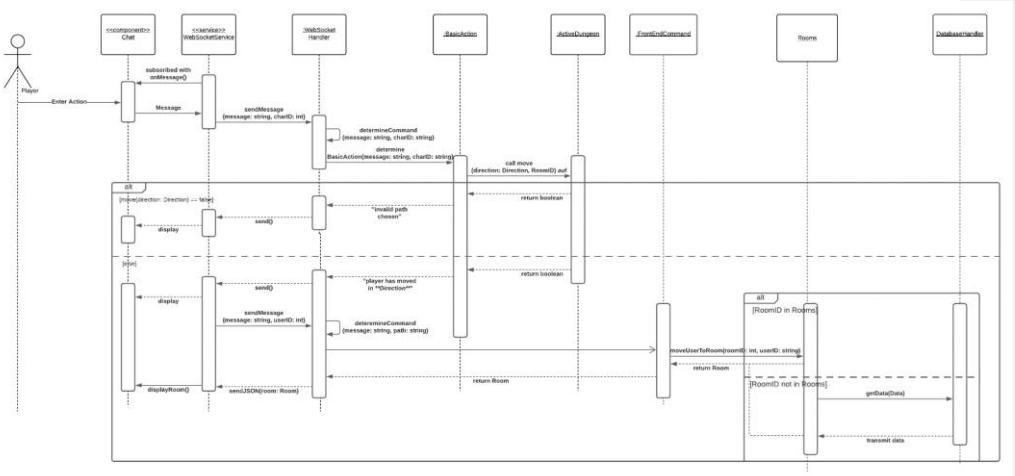
Charaktere, NPCs, sowie grundlegende Daten über den Dungeon selbst. Die geparsten Objekte werden nun dem DataBaseHandler übergeben, welcher dann durch SQL-Befehle in die entsprechenden Tabellen schreibt.

6.2. Websocket: EnterDungeon



Wenn der Spieler in einem Spiel mit der Oberfläche interagiert, werden bestimmte Prozesse ausgelöst, um Daten per Lazy Loading zu laden. Wenn der Spieler beispielsweise einen Dungeon betritt, können mehrere Daten, wie Räume oder anderes angefordert werden. Ausgehend vom Angular-Frontend werden dann die UserID und die DungeonID übermittelt und verpackt an den WebSocketHandler im Backend geschickt. Der WebSocketHandler ruft dann die FrontEndCommand-Klasse auf und die Daten werden entsprechend aus dem DungeonPackage geladen, sollten die Daten bereits vorhanden sein. Ist dies nicht der Fall so wird eine Datenbank-Anfrage getätigt und die gewünschten Daten werden dann geladen und dem User übermittelt.

6.3. Websocket: BasicAction



Der Spieler innerhalb eines Dungeons kann Basisaktionen in den Chat eingeben. Eine hiervon wäre das Bewegen zwischen den Räumen eines Dungeons. Gibt der Spieler nun den entsprechenden Befehl und die Richtung ein, so wird eine Kette von Aktionen ausgelöst. Der WebSocketService bekommt über eine OnMessage Methode mit, ob es eine neue Nachricht innerhalb des Chats gibt. Ist diese Bedingung erfüllt, wird die eingegebene Nachricht und die Charakter-ID an den WebSocketService weitergeleitet und dem WebSocket-Handler im Backend übermittelt. Innerhalb des WebSocket-Handlers wird festgestellt um was für eine Art von Nachricht es sich handelt. In diesem Fall wird die Aktion BasicAction ermittelt. Innerhalb der BasicAction Klasse wird dann nochmals unterschieden zwischen mehreren Basisaktionen. Da es sich um die Bewegungsaktion handelt, wird in der Klasse ActiveDungeon eine move Methode aufgerufen, welche die tatsächliche Bewegung mithilfe eines Enums und der Entsprechenden RaumID feststellt. Die Methode liefert einen Boolean zurück, und übermittelt somit über den WebSocket-Handler und dem WebSocketService entweder eine Erfolgsnachricht oder eine Fehlermeldung im Chat. Anschließend müssen die Daten des neuen Raumes in dem sich der Nutzer nun befindet abgefragt werden. Hierfür sendet der Frontend WebsocketService eine Anfrage, welche von dem WebsocketHandler erneut aufgenommen wird. Dieser filtert und stellt fest, dass es sich um ein FrontEndCommand handelt, dieser wird dann in der Methode der FrontEndCommands ausgeführt. Dementsprechend wird FrontEndCommands nun versuchen auf den entsprechenden Raum zuzugreifen in dem sich der Spieler nun befindet. Wenn der Raum noch nicht in einem aktiv bespielten Dungeon geladen wurde führt das Dungeon nun den Befehl des Abrufens an der Datenbank aus. Nachdem nun die Daten des Raumes vorhanden sind kann FrontEndCommand mit dem entsprechenden Return dem Frontend über den WebsocketHandler bescheid geben, was denn nun die Beschreibung des Raumes ist. Zuletzt wird diese Beschreibung im Frontend angezeigt.