Designbeschreibung (V1.4)

Dokumenteninformation

Erstellt von: Linus Kadner
Bearbeitet von: Linus Kadner, Daniel Scherer, Jakub Cielecki, Leon Daskov, Johannes Hüttinger
Dateiname: Designbeschreibung_v1.4.pdf

Versionierung

Version	Datum	Bearbeiter	Änderung	Stand
1.0	18.04.2021	Linus Kadner	Initiale Erstellung	Abgenommen
1.1	23.04.2021	Daniel Scherer	Beschreibungen Aktivitätsdiagramme / Sequenzdiagramm des Konfigurators	Abgenommen
1.2	26.04.2021	Daniel Scherer, Jakub Cielecki, Leon Daskov, Marius Armbruster	Diagramme durch SVGs ersetzt	Abgenommen
1.3	05.05.2021	Johannes Hüttinger	Update Komponentendiagramm, Gamelogik und Main	Abgenommen
1.4	09.05.2021	Leon Daskov	Update Konfiguration zusammengeführt mit Modell	Abgenommen

Inhalt

1. Allgemeines

In dieser Designbeschreibung wird unser Design des MUD-Servers beschrieben, welcher mehrere Komponenten enthält. Die Komponenten sind die folgenden: Webclient, Konfigurator, Benutzer-Authentifizierung und Spielclient. In dieser Designbeschreibung werden unsere Struktur- und Entwurfsentscheidungen festgehalten.

2. Produktübersicht

Das Produkt besteht aus einer Website, über die sich ein Benutzer registrieren und anmelden kann. Wenn sich ein Benutzer angemeldet hat, kann er entweder einem MUD-Spiel beitreten oder ein eigenes MUD-Spiel über den Konfigurator erstellen.
Wenn sich der Benutzer dafür entscheidet, an einem Spiel teilzunehmen, wird er zu dem Charakter-Konfigurator weitergeleitet, falls er noch keinen Charakter in diesem MUD-Spiel besitzt. In diesem Konfigurator kann er dem Charakter einen Namen geben und seine Klasse und Rasse auswählen. Die zur Verfügung stehenden Klassen und Rassen sind von der Konfiguration des MUDs abhängig.
Mit dem erstellten Charakter kann er nun dem MUD-Spiel beitreten.
Wenn ein Charakter neu erstellt wurde, befindet er sich beim Betreten des MUD-Spiels im Startraum, ansonsten ist er dort, wo man das Spiel verlassen hat. Hat er schon einen Charakter kann er entweder mit dem vorhandenen Charakter beitreten oder einen weiteren Charakter erstellen.
Wenn der Benutzer jedoch ein eigenes MUD-Spiel erstellen möchte, wird er zum Konfigurator weitergeleitet. Hier kann er einen MUD frei nach seinem Belieben konfigurieren. Unter den Konfigurationsmöglichkeiten befinden sich: Rassen, Klassen, Räume, NPCs, Equipment, Items und Spieleraktionen. Wenn die Konfiguration beendet ist, kann man dem neu erstellten MUD-Spiel als Spieler beitreten.

3. Grundsätzliche Struktur- und Entwurfsentscheidungen

Das Produkt kann generell in einzelne Komponenten gegliedert werden. Die Client-Komponente, welche dem Benutzer zur Verfügung steht, beinhaltet alle UI-Komponenten sowie die sog. Services zur Kommunikation mit dem Server (weitere Informationen unter 4.1 Webclient).
Die grundsätzliche Hintergrundlogik, befindet sich in der Server-Komponente. Diese Komponente enthält weitere Komponenten, welche das Spielgeschehen sowie die Nutzer verwalten und mit der Datenbank interagieren.
Die Datenbank (PostgreSQL) enthält alle relevanten Daten, wie MUD-Konfigurationen, Spielstände und Benutzerdaten.
Das Produkt ist hauptsächlich in Kotlin programmiert, wobei das Framework Ktor genutzt wird. Ktor bietet hierbei gute Funktionalitäten für Websockets und die Authentifizierung. Zur Datenübertragung zwischen Client und Server werden unter anderem JSON-Daten verwendet. Die Websockets werden unter anderem für den Chat genutzt und im Spiel selbst, um Benutzer-Eingaben an das Spiel zu übermitteln.
Der Webclient ist mit dem Webframework Angular realisiert.
Die vom MUD-Master erstellte Konfiguration wird per JSON-String an den Server gesendet.
Damit ein Benutzer am Spielgeschehen teilnehmen kann, benötigt er einen Benutzeraccount. Diese Authentifizierung geschieht im Server und wird mit JSON Web Tokens verarbeitet.
Die Komponenten sind wie folgt verbunden:

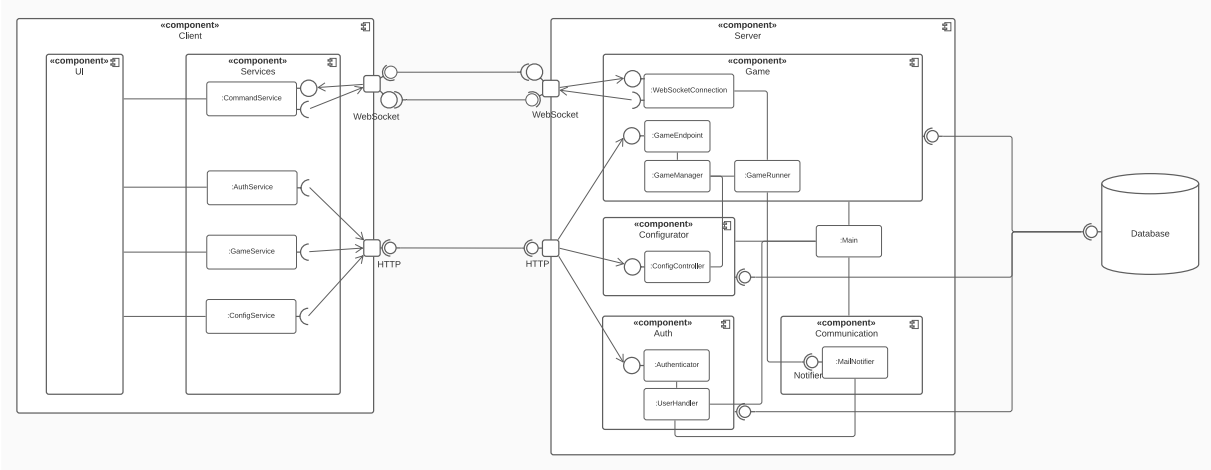

Abbildung 1 – Komponentendiagramm – [Link zur Webansicht]

4. Struktur und Entwurfsentscheidungen der Komponenten

4.1 Webclient

Der Client wird mit dem Webapplikationsframework Angular realisiert.
Die Angular Applikation ist hierarchisch in einzelne Komponenten unterteilt (Angular Components) und analog zu den Mockups aufgebaut.
Unterhalb der Root-Komponente befinden sich die
Komponente AuthentificationPage, DashboardPage, AvatarPage, GamePage und ConfigurationPageei nzelnen Komponenten sind in weitere Unterkomponenten unterteilt, wobei es einfache und wiederverwendbare Komponenten gibt. Die „MapComponent“ wird z.B. in der „GamePage“ Komponente und in der „Rooms“ Komponente verwendet.
Die Services in der Angular-Anwendung dienen zur Kommunikation mit dem Server. Es gibt einen Service für die Authentifikation („AuthService“), die Spiellogik („GameService“), die Spielbefehle („CommandService“) und den Konfigurator („ConfigService“).

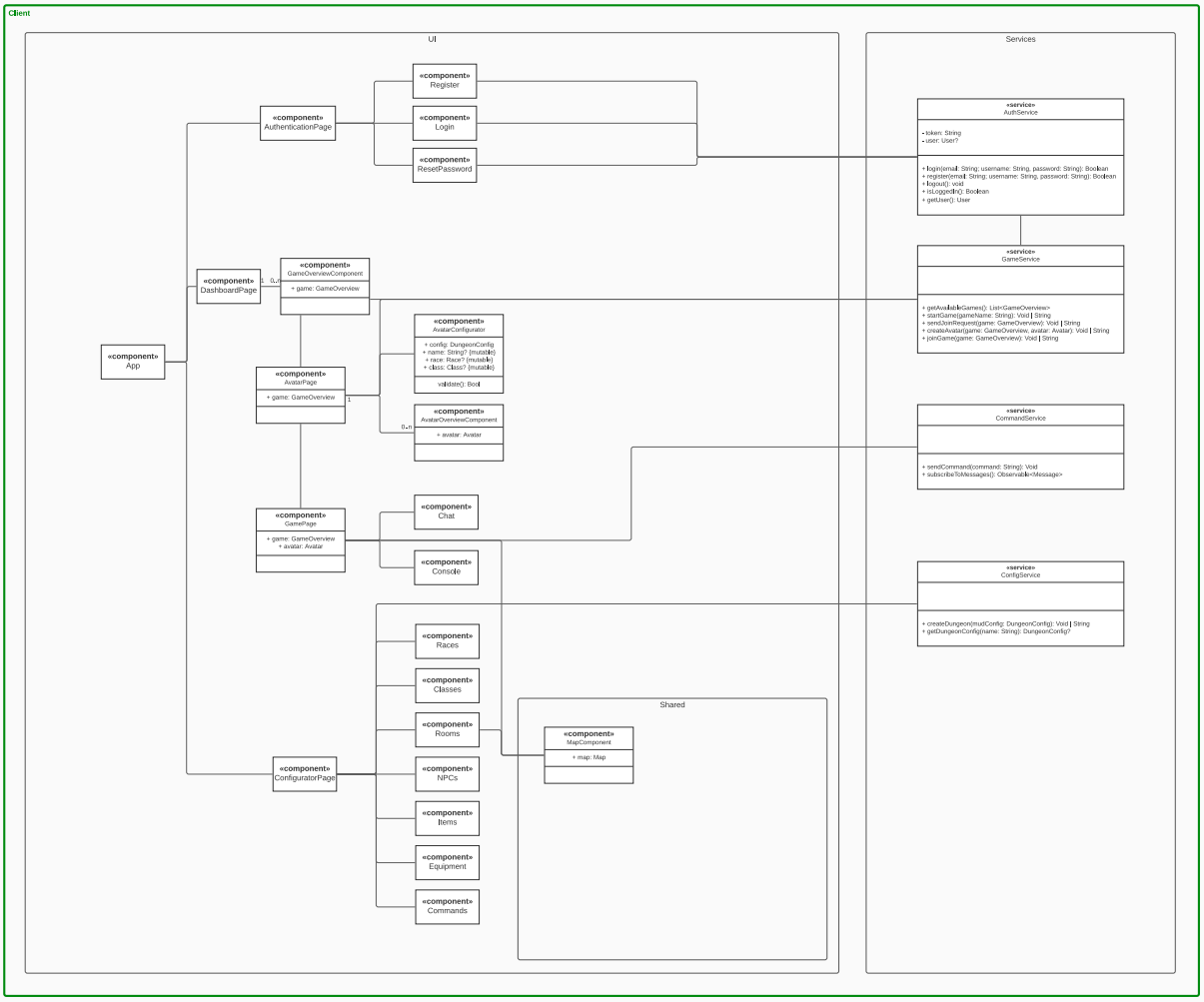

Abbildung 2 – Client - [Link zur Webansicht]

4.2 Konfigurator

Der Konfigurator ist in einem gewissen Rahmen, als separate Anwendung zu sehen, da dieser „nur“ über die erstellten Konfigurationen mit dem eigentlichen Spiel kommuniziert.

Der Konfigurator wird größtenteils im Rahmen der Angular Umgebung befindlich sein. Hier werden durch User-Inputs repräsentative Objekte für die verschiedenen Teilkonfigurationen erstellt. Ein Beispiel für solch eine Teilkonfiguration ist die Item-Konfiguration.

Dem User ist es auch möglich, die einzelnen Teilkonfigurationen nochmal aufzurufen und zu bearbeiten, indem die Teilkonfigurationen kopiert werden und nochmal angepasst werden können.

Teilkonfigurationen wie die Items, welche dann auch in Räumen, Inventaren und weiterem platziert werden können, werden durch Namen ansprechbar sein, damit diese dann auch in der Erstellung weiterer Räume und Inventare verfügbar sind.

Die Teilkonfigurationen verweisen über Relationen aufeinander, damit nicht für jedes Auftreten einer Teilkonfiguration, sie mehrfach vorhanden sein müsste. Über die Relationen werden Redundanzen vermieden.

Ab dem Punkt, an dem der User die Konfiguration vollständig erstellt hat, wird die Konfiguration in eine PostgreSQL-Datenbank abgelegt. Das Spiel kann später auf die Konfigurationen zugreifen, sie verändern und updaten. Somit besitzt jedes Spiel eine Konfiguration.

Ein jedes Modell eines Spiels, basiert auf der am Anfang erstellten Konfiguration. Wiederverwendbare Daten wie Itemdefinitionen wurden somit am Anfang, vor dem Starten des Spiels deklariert.

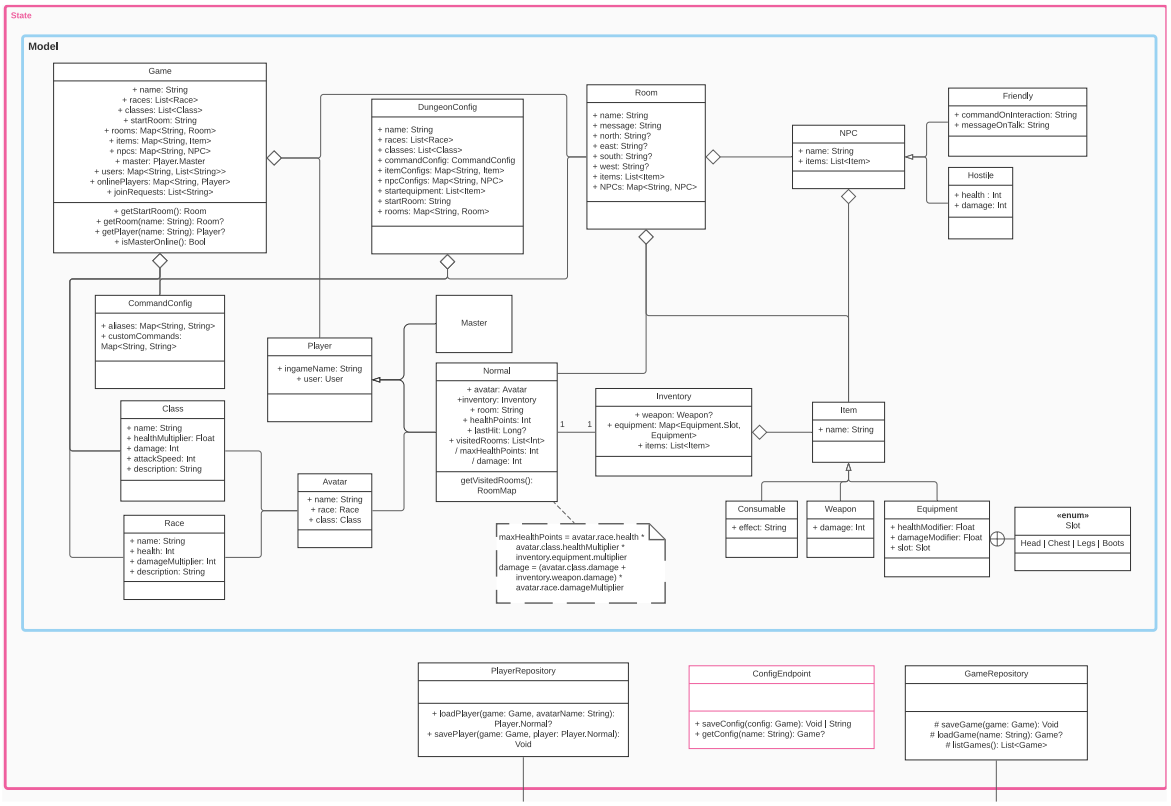

Abbildung 3 – Configurator – [Link zur Webansicht]

4.2.1 Aktivitätsdiagramme

Folgende Aktivitätsdiagramme veranschaulichen den Konfigurationsprozess eines MUDs. Wie auf den Mockups zu sehen ist, geschieht die Konfiguration der einzelnen Einheiten (Rassen, Klassen, Befehle, Items, NPCs, Räume, Equipment) in einzelnen Tabs, d.h. sie finden parallel statt.
Die folgende Abbildung zeigt das Aktivitätsdiagramm des Konfigurationsprozesses.

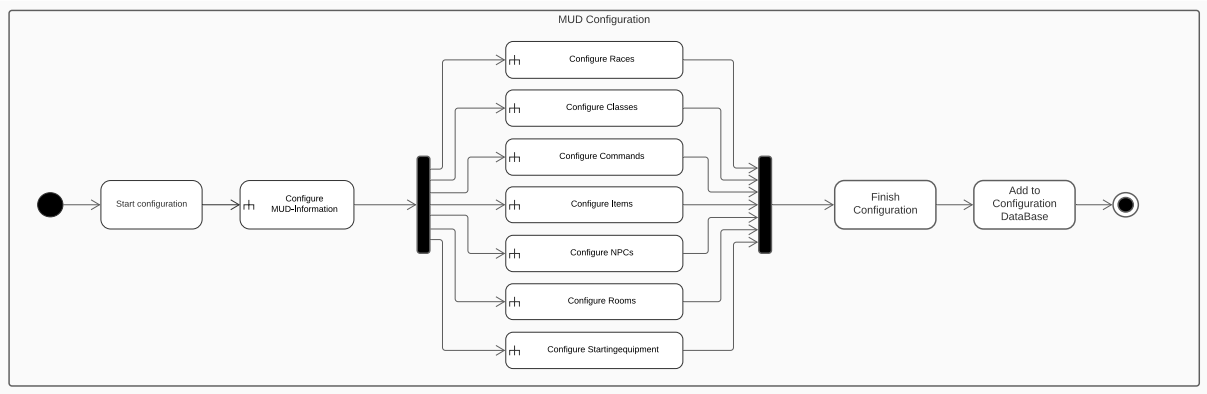

Abbildung 4 – MUD Configuration – [Link zur Webansicht] – [Detailgetreue Webansicht]

Die nachfolgenden Diagramme sind Teil des obigen Aktivitätsdiagramms.
Das Aktivitätsdiagramm “Configure MUD-information" steht am Anfang einer jeden MUD-Konfiguration. Hierbei werden Informationen, wie der MUD-Name und seine Beschreibung aufgenommen.

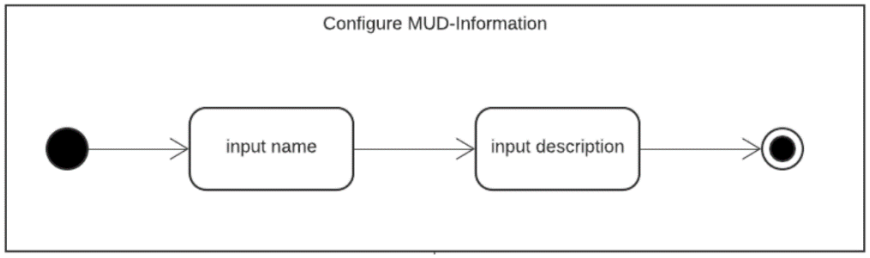

Abbildung 5 – Configure MUD-Information – [Link zur Webansicht]

Wie im allgemeinen Aktivitätsdiagramm („MUD Configuration“, s.o.) zum Konfigurator zu sehen ist, besteht dieser aus einigen Prozessen, welche nachfolgend gesondert dargestellt werden.
Das Aktivitätsdiagramm “Configure Races” zeigt den Konfigurationsprozess für die Charakterrassen, welche im späteren MUD zur Verfügung stehen.
Da dem Benutzer keine Vorgaben gemacht werden, in welcher Reihenfolge er die einzelnen Daten eingeben muss, wird dies hier als “Splitting” und spätere “Synchronisation” dargestellt.

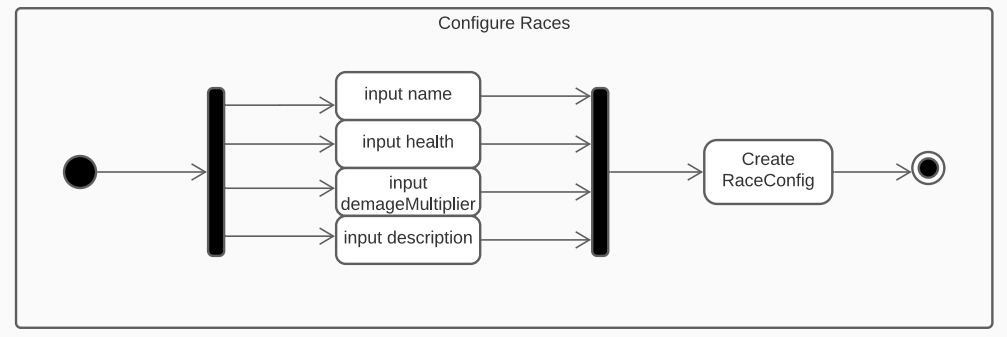

Abbildung 6 – Configure Races – [Link zur Webansicht]

Das folgende Diagramm “Configure Classes” ist äquivalent zum vorangegangenen Diagramm “Configure Races” aufgebaut. Die Reihenfolge der Benutzereingabe ist auch hier wieder dem Benutzer überlassen.

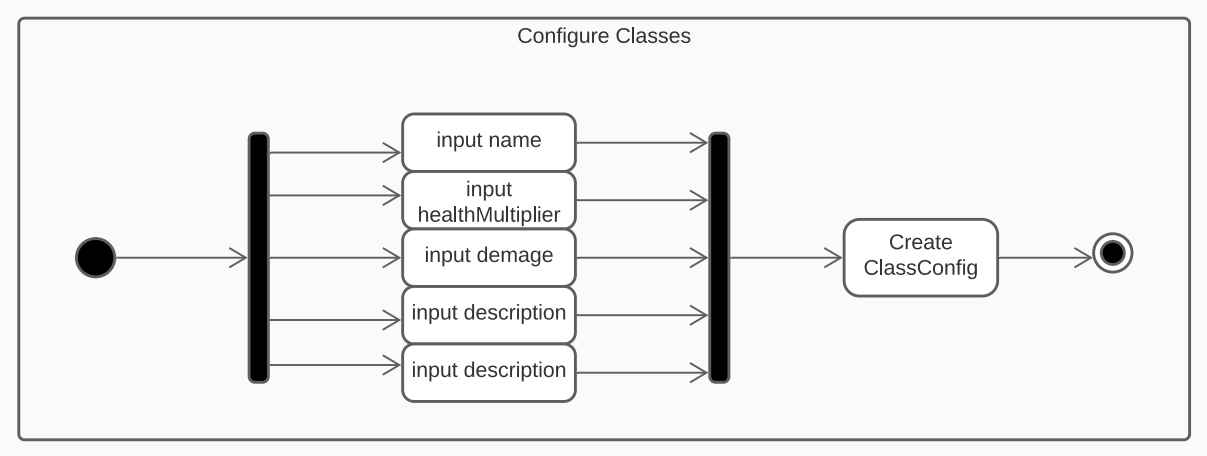

Abbildung 7 – Configure Classes – [Link zur Webansicht]

Dem MUD-Master soll die Möglichkeit gegeben werden, zum einen eigene Spielbefehle zu konfigurieren und zum anderen die Standardspielbefehle umzubenennen.
Für den Fall, dass eigene Spielbefehle erstellt werden, muss der MUD-Master erst eine Syntax für den Befehl definieren und kann anschließend eine Sequenz an Actions auswählen. Diese Actions werden dann ausgeführt, sobald ein MUD-Spieler den Befehl ausführt.
Abschließend müssen den Befehlen Aliase zugeteilt werden (z.B. einem Spielbefehl zum Einnehmen von Konsumitems der Alias “eat”). Auf diese Weise können auch Standardfunktionen umbenannt werden.

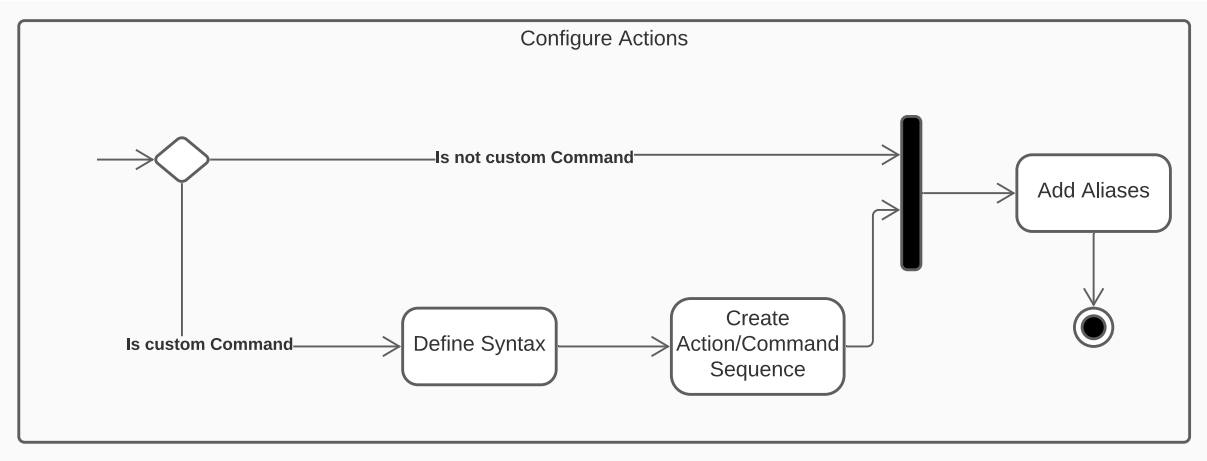

Abbildung 8 – Configure Actions – [Link zur Webansicht]

Das nachfolgende Aktivitätsdiagramm “Configure Items” stellt den Konfigurationsprozess von Items / Objekten dar. Zuerst muss hierbei ein Name für das Objekt vergeben werden. Anschließend muss der Typ des Objektes gewählt werden, da dies entscheidend für die weitere Konfiguration ist. Ist das Objekt keine Ausrüstung (“Equipment”), muss gewählt werden, ob es konsumierbar ist oder nicht. Des Weiteren wird dann ein Command festgelegt, welcher ausgeführt wird, sollte das Objekt konsumiert werden.
Ist das neue Item vom Typ Equipment, werden noch die Attribute “HealthModifier” und “DamageModifier” gesetzt. Das Attribut “DamageModifier” ist dementsprechend entscheidend darüber, ob die Ausrüstung eine Waffe ist oder nur eine Art Rüstung.

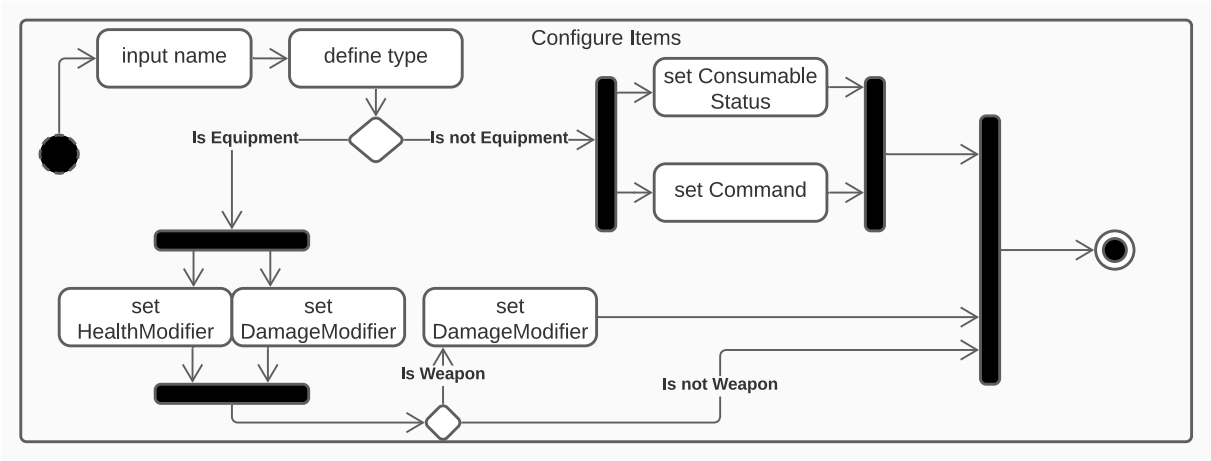

Abbildung 9 – Configure Items – [Link zur Webansicht]

Im Raumsystem des MUDs können beliebig NPCs platziert werden. Diese NPCs werden zuvor vom MUD-Master konfiguriert. Dabei ist vorerst der Name und der Typ des NPCs zu wählen. Der Typ entscheidet darüber, ob der NPC friedlich oder feindlich ist. Sollte der NPC friedlich sein, kann eine Nachricht eingegeben werden, welche der NPC von sich gibt, sobald er von einem Spieler angesprochen wird. Ein feindlicher NPC kann nicht angesprochen werden, hat jedoch die Attribute “Health” und “Damage”. Zum Abschluss kann noch ein “Loottable” definiert werden. Ein “Loottable” ist ein Pool an Items, die der NPC an dem Spieler abgibt (friendly NPC), oder fallen lässt, sobald er besiegt wurde (hostile NPC).

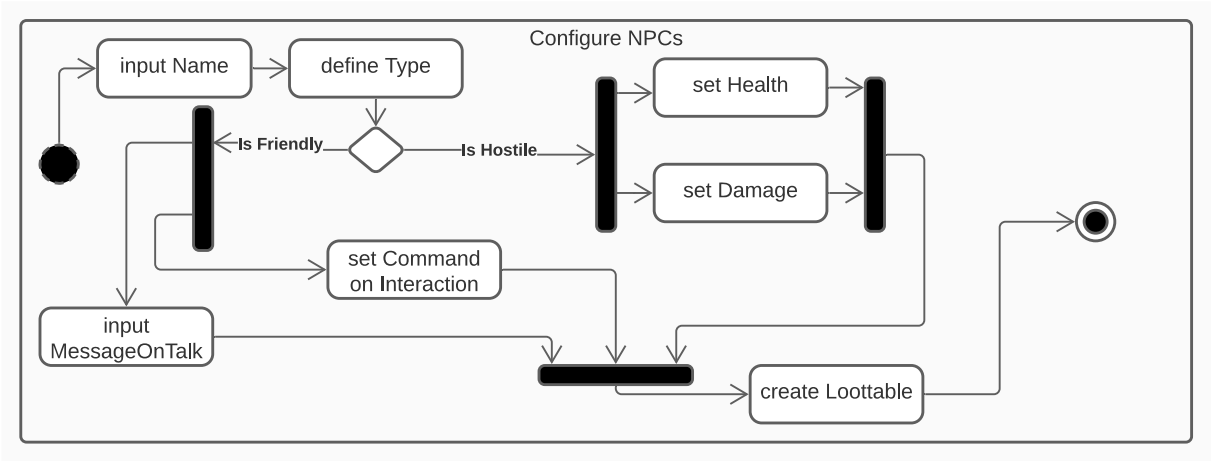

Abbildung 10 – Configure NPCs – [Link zur Webansicht]

Das wichtigste Element der Konfiguration ist die Erstellung der einzelnen Räume und somit das Raumsystem des MUDs. Beim Konfigurieren werden dem Raum Items und NPCs hinzugefügt, welche sich darin befinden. Des Weiteren wird eine Raumeintrittsnachricht definiert und Verbindungen zu anderen Räumen gewählt.

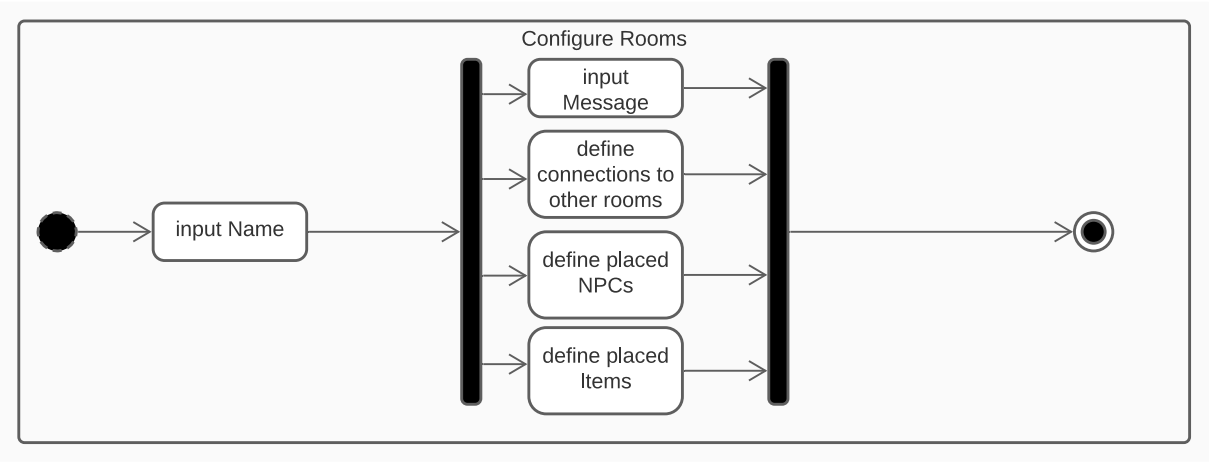

Abbildung 11 - Configure Rooms - [Link zur Webansicht]

Das Aktivitätsdiagramm “Configure Starting Equipment” beschreibt den Vorgang, bei dem das Startequipment gewählt wird, dass jedem MUD-Spieler (bzw. seinem Charakter) zu Spielbeginn bereitsteht.

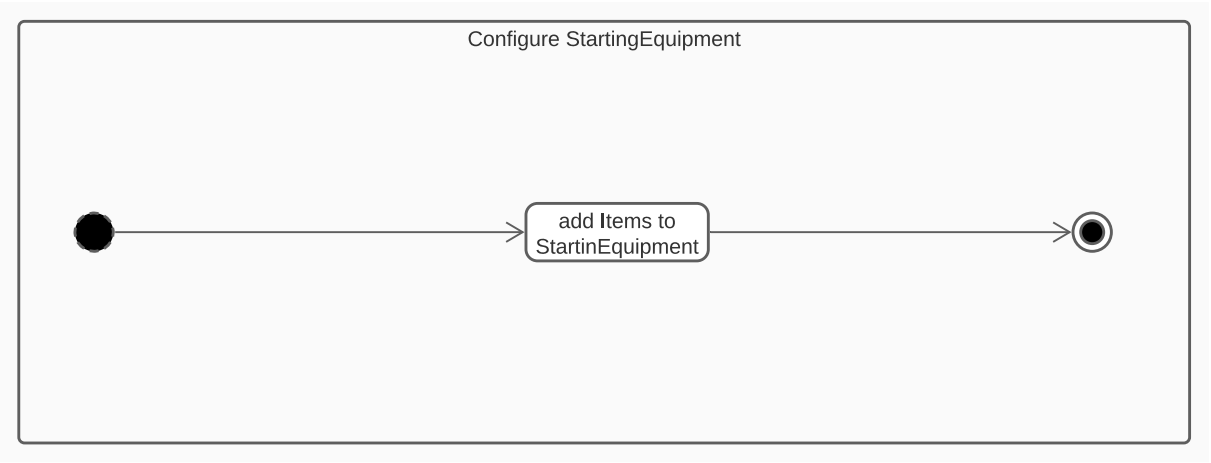

Abbildung 12 - Configure Starting Equipment - [Link zur Webansicht]

4.2.2 Sequenzdiagramme

Das nachfolgende Sequenzdiagramm “validate and submit config” zeigt den Ablauf durch die Programmstruktur, sobald der User (MUD-Master) eine Konfiguration von der Hintergrundlogik im Server validieren lässt und anschließend abspeichert.
Die Klasse “ConfigurationHandler” befindet sich dabei im Client und hat die Aufgabe die jeweiligen Subkonfiguraitonen (Rassen, Klassen etc.) zu validieren und anschließend zusammenzuführen. Über den “ConfigService” der Angular-Anwendung kommuniziert der Client mit dem Server. Die Konfiguration wird vom “ConfigService” aus als JSON-String über einen WebSocket zum “ConfigController” im Server geschickt. Hier wird die Gesamtkonfiguration des MUDs validiert und im erfolgreichen Fall in der Datenbank gespeichert. Sollte hierbei ein Fehler in der Konfiguration gefunden werden, wird über den gleichen Weg eine Nachricht an den Benutzer geschickt, an welcher Stelle sich der Fehler befindet.

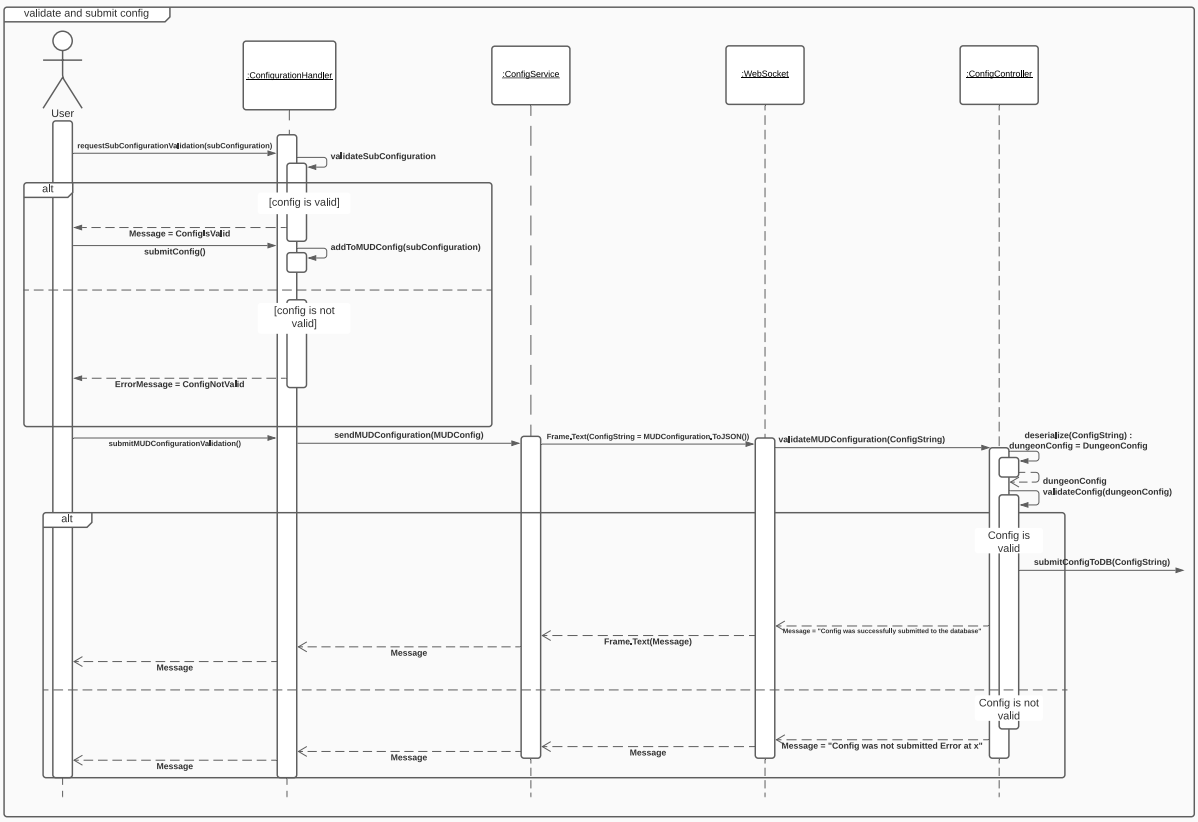

Abbildung 13 - Validate and Submit Config - [Link zur Webansicht]

4.3 Benutzer Authentifizierung

Zur Authentifizierung wird ein JSON Web Token genutzt, mit dem ein Benutzer über seine gesamte Session authentifiziert werden kann.

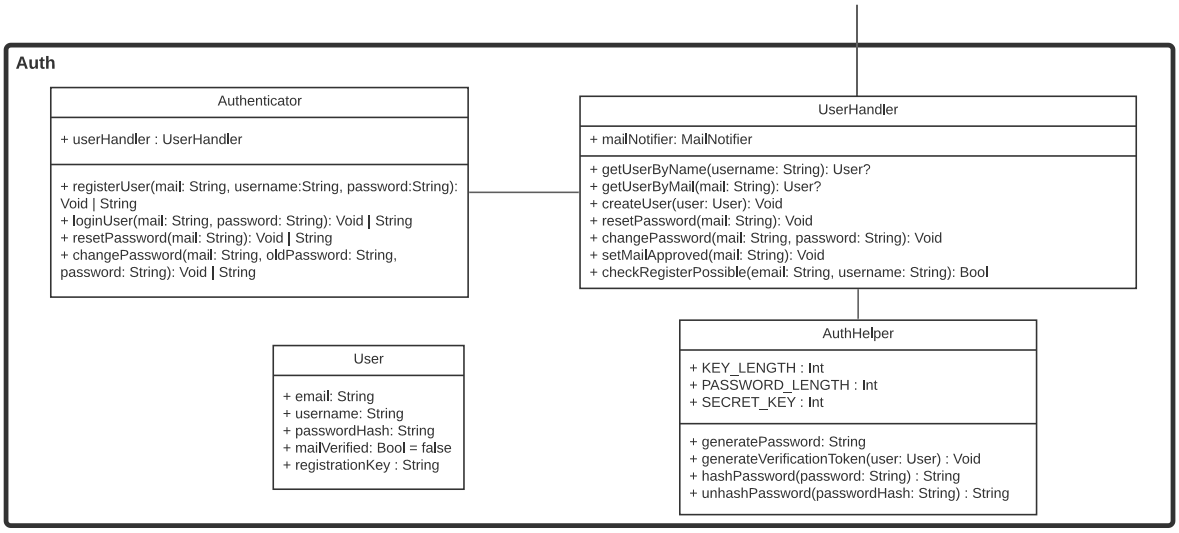

Abbildung 14 - Auth - [Link zur Webansicht]

4.3.1 Sequenzdiagramme

Das folgenden Sequenzdiagramme zeigen den Ablauf durch die Programmstruktur, sobald sich der Spieler am Server anmeldet oder registriert.

Wenn der Benutzer seine Anmeldedaten angegeben hat und den Login-Button betätigt, werden diese Daten über die Login Komponente an die Authentication gesendet und der Login-Prozess startet über die loginUser-Methode.
Hier wird jetzt der UserHandler aufgerufen um den User, auf Basis seiner Mail-Addresse, aus der Datenbank rauszusuchen. Es wird ein User übergeben und das eingegebene Passwort und das Passwort des Users, als auch das Passwort aus dem Anmeldeprozess werden verglichen.
Wenn das Passwort korrekt ist, wird der Benutzer als der gefundene User aus der Datenbank eingeloggt. Falls das Passwort nicht korrekt ist, kann der Prozess von neuem beginnen.

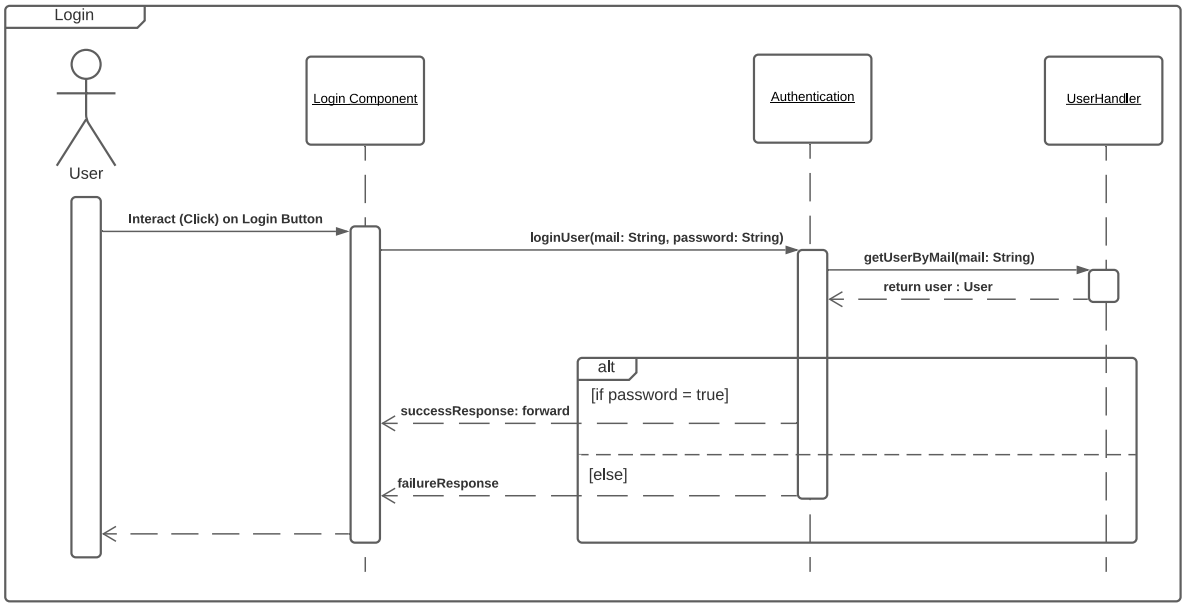

Abbildung 15 - Login - [Link zur Webansicht]

Wenn der Nutzer sich registriert, werden die Daten über die Register-Komponente an die Authentication weitergereicht. Nun überprüft die Authentication über den Userhandler, ob bereits Accounts mit den angegebenen Daten vorhanden sind (E-Mail und Username). Wenn diese User schon vorhanden sind, wird dem Nutzer mitgeteilt, dass er sich mit diesen Daten nicht registrieren kann.
Falls jedoch keine Accounts mit diesen Daten bestehen, wird ein User in der Datenbank angelegt, dessen Passwort mithilfe des AuthHelpers verschlüsselt wird. Danach kriegt der Nutzer eine E-Mail um seinen Account zu verifizieren.

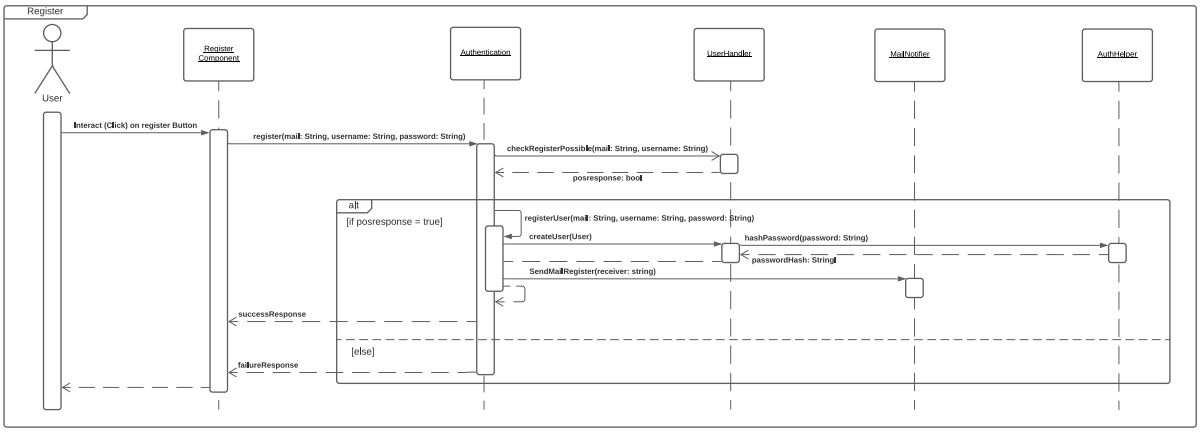

Abbildung 16 - Register - [Link zur Webansicht]

4.4 Spiel

Main

Die Einstiegklasse des Servers startet zentral globale Services wie den GameManager oder den ConfigController.
Auch werden die von diesen Services benötigte Anbindungen wie z.B. E-Mail oder Datenbankverbindung initialisiert.

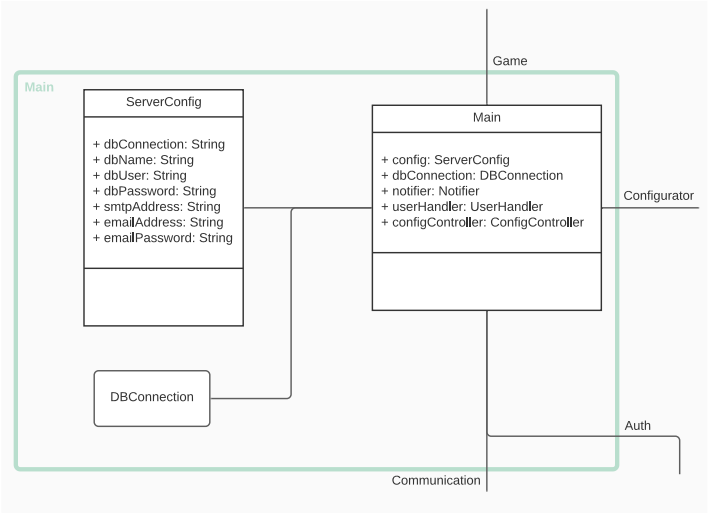

Abbildung 17 – Main – [Link zur Webansicht]

Kommunikation und Benachrichtigungen

Dieses Package ermöglicht die Kommunikation mit Benutzern bei Benachrichtigungen, aktuell über E-Mail. Ein Beispiel dafür ist die Einladung eines MUD-Spielers zum Spiel durch den MUD-Master.

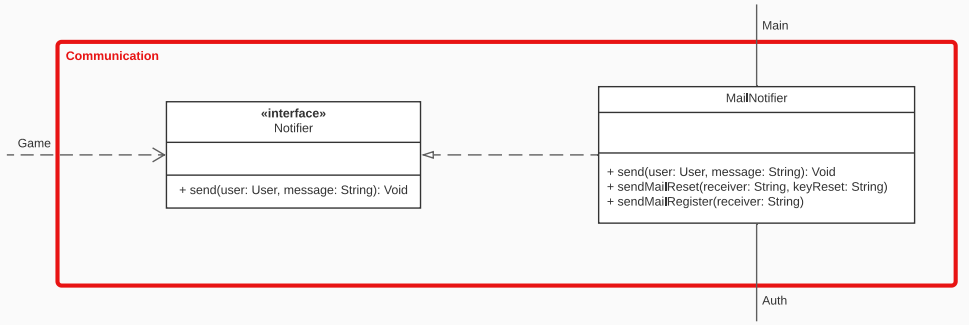

Abbildung 18 - Communication - [Link zur Webansicht]

Spiellogik

Allgemein

Generell sind Felder, die als public markiert sind, readOnly, falls nicht anders angegeben.

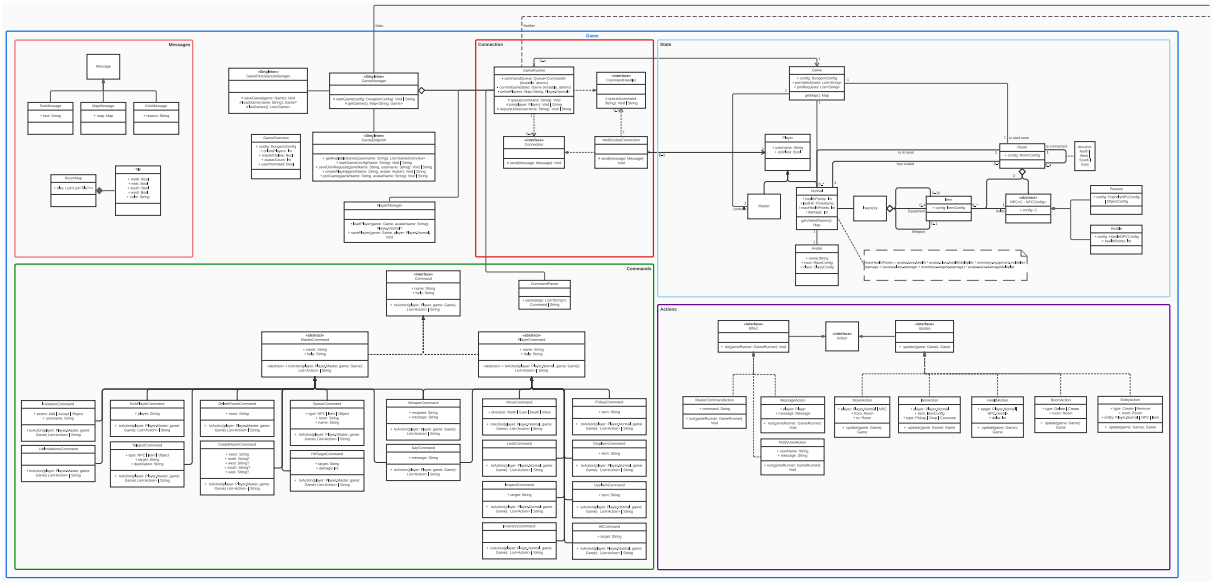

Abbildung 19 - Überblick Game Logik - [Link zur Webansicht]

Gameloop

Die Kommunikation mit der Außenwelt und anderen Komponenten des Servers passiert nur im Top-Level und Connectiomn Package, damit Seiteneffekte minimiert werden. Dabei ist in jedem Spiel der Gamerunner zentral, der die von der WebSocketConnection hinzugefügten Befehle in einer Queue abarbeitet. Hier kann zentral dafür gesorgt werden, dass Änderungen am Gamestate atomar ausgeführt werden, das in einem asynchronen Umfeld wichtig ist, um Race Conditions und invalide Zustände zu vermeiden.

Auch kann wie bei Actions beschrieben das Ausführen von Effekten außerhalb der Gameloop stattfinden. Der Ablauf vom Eingeben eines Commands über die Gameloop bis zur Rückmeldung zum MUD-Spieler ist im Sequenzdiagram “Move Command” näher beschrieben.

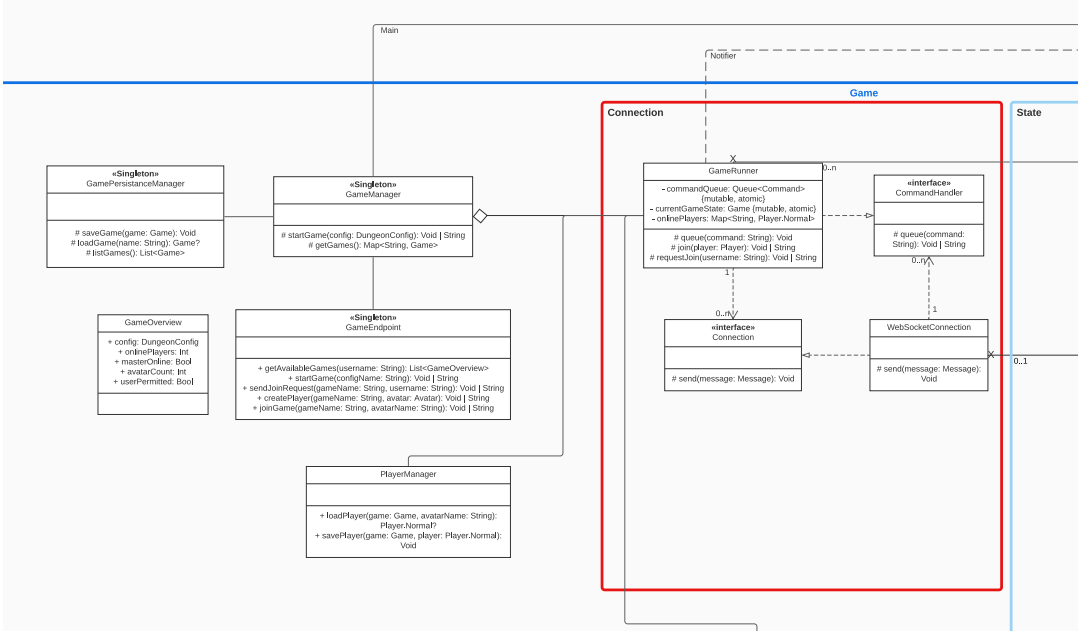

Abbildung 20 – Kommunikation Außenwelt und Gameloop - [Link zur Webansicht]

Gamestate

Der Zustand eines laufenden Spiels wird hier abgebildet, wobei dieser Teil nur Datenklassen und einige Hilfsmethoden enthält.

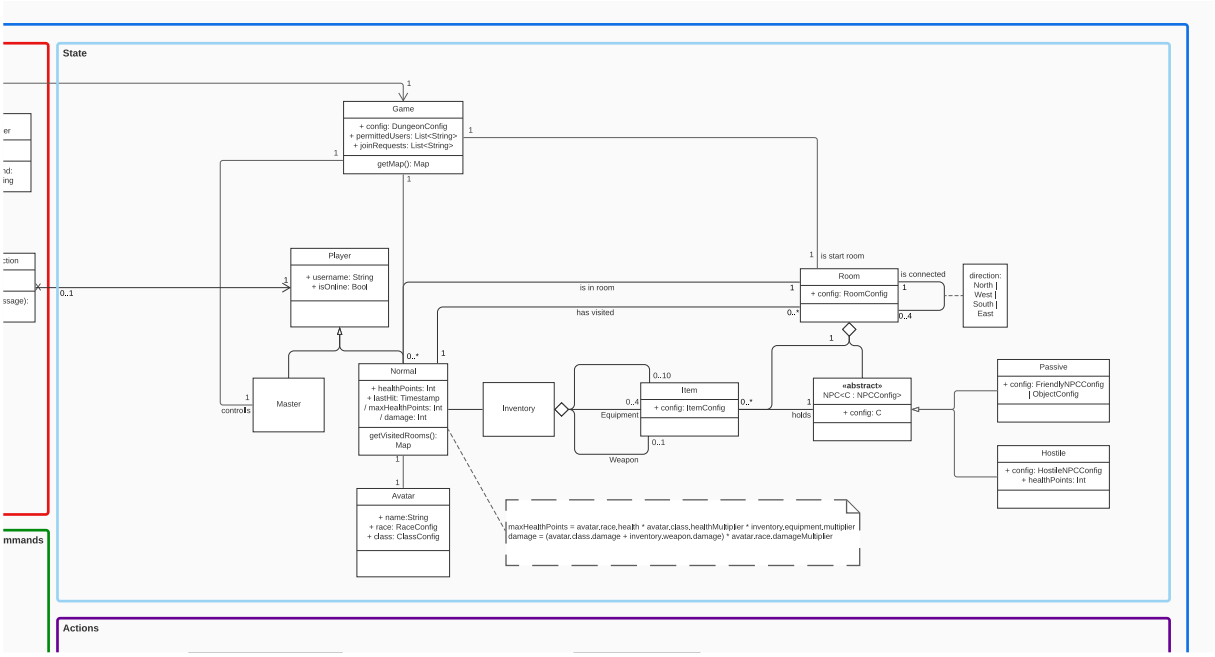

Abbildung 21 – Gamestate – [Link zur Webansicht]

Commands

Alle Commands befinden sich nach Master- und Spielercommands aufgeteilt in diesem Package. Jedes Command implementiert die Methode toAction(Game), wobei entweder eine Fehlernachricht in Form eines Strings oder im Erfolgsfall eine Liste von Actions. Wichtig: Hier werden keine Änderungen am Gamestate oder sonstige Seiteneffekte erzeugt. Das vereinfacht das Testen dieses Teils massiv.

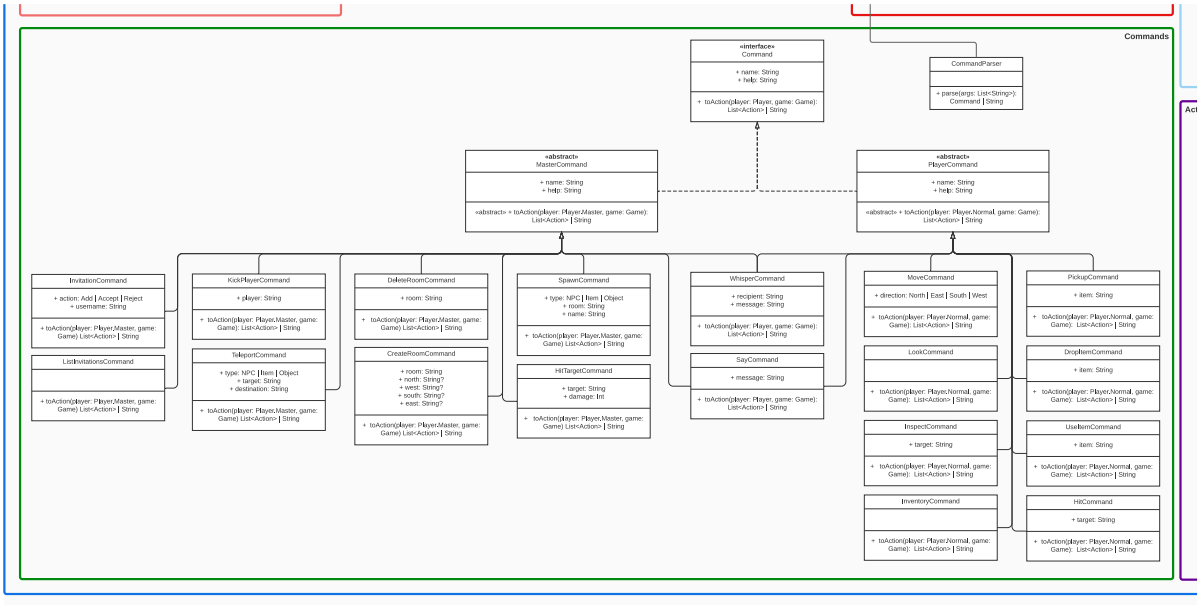

Abbildung 22 – Commands – [Link zur Webansicht]

Actions

Actions sind in Effects und Updates untergeteilt.
Die ersteren implementieren eine Methode run(GameRunner),
die einen Seiteneffekt, beispielsweise das Senden einer Notification (NotifyUserAction), ausführen. Di
ese Methode wird vom GameRunner ausgeführt, sodass dieser über den Ausführungszeitpunkt entsc
heiden kann, wodurch die Gameloop entlastet wird.
Updates dagegen führen keine Effekte aus, sondern geben einfach nur einen neuen, aktualisierten G
amestate zurück. Auch dies vereinfacht das Testen.

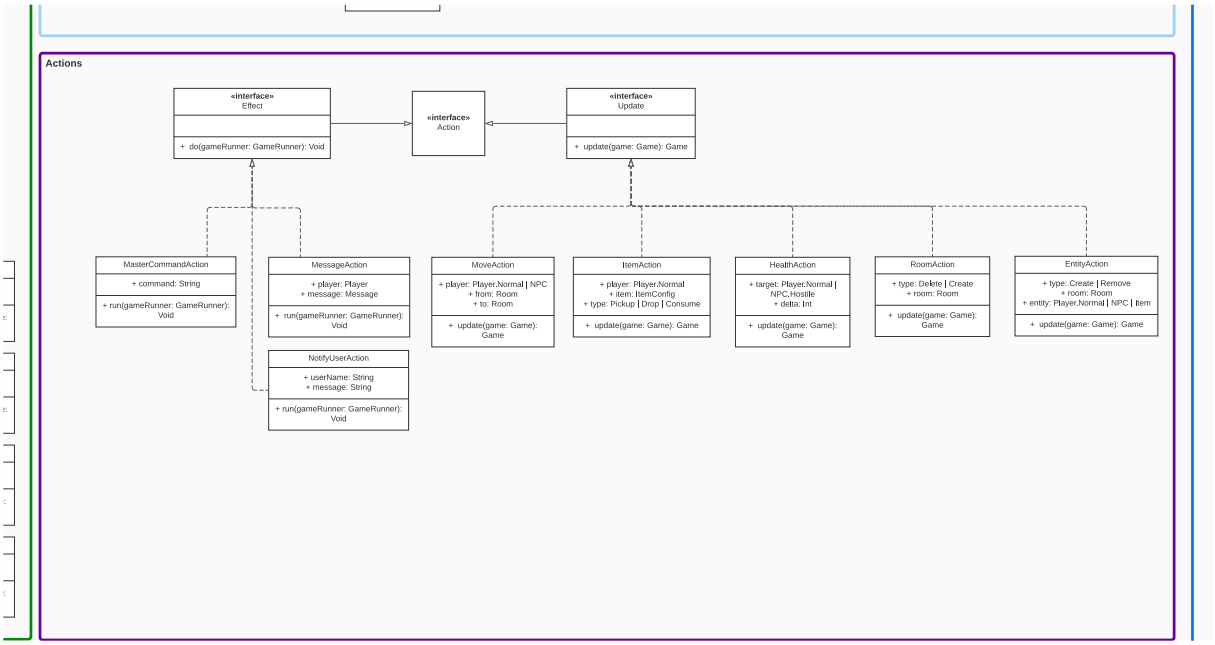

Abbildung 23 - Actions - [Link zur Webansicht]

Messages

Dies sind einfache Datenklassen, die über die Action MessageAction an den Client geschickt werden können. Dabei werden einfache Textnachrichten (TextMessage), Mapaktualisierungen, die bei Bewegung im Raum gesendet werden (MapMessage), und KickMessages, die dem Client das Beenden der Verbindung, ob beim Kicken eines Players durch den Dungeonmaster, oder beim Beenden des Spieles, geschickt werden.

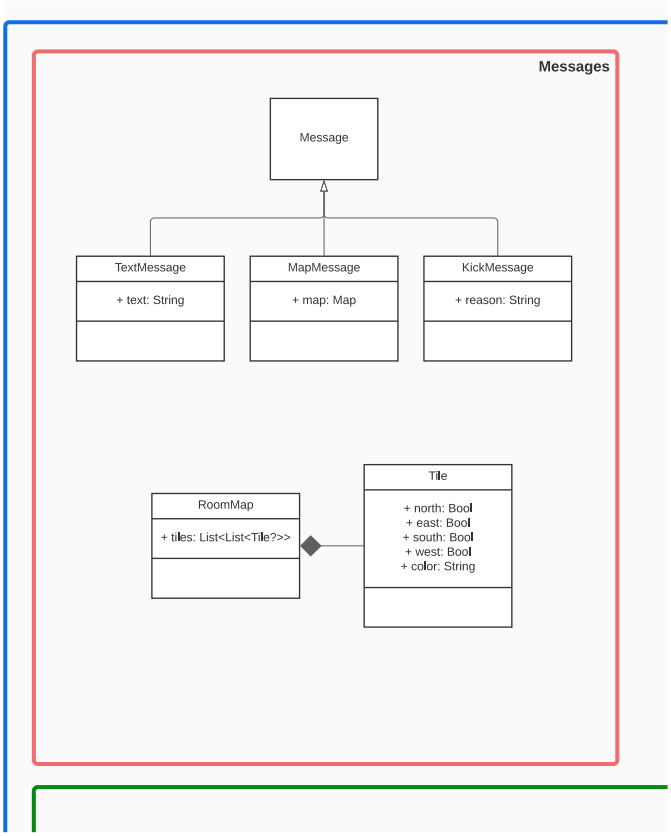

Abbildung 24 - Messages - [Link zur Webansicht]

4.4.1 Sequenzdiagramme

Die folgenden Sequenzdiagramme zeigen den Ablauf eines Move-Commands, wenn ein User einem anderen User schreibt und wenn ein Avatar erstellt wird.

Wenn der Benutzer während des Spiels ein Kommando eingibt, wird dieses über den GameService per Websocket als String an den Server geschickt. Hier versucht die Connection das Kommando zu parsen. Klappt dies, wird dieser in die Command Queue des Gamerunners eingereiht und zu einem späteren Zeitpunkt weiter verarbeitet (Siehe Sequenzdiagramm „Gameloop“). Falls nicht, wird dem Benutzer eine Hilfsnachricht zurückgeschickt.

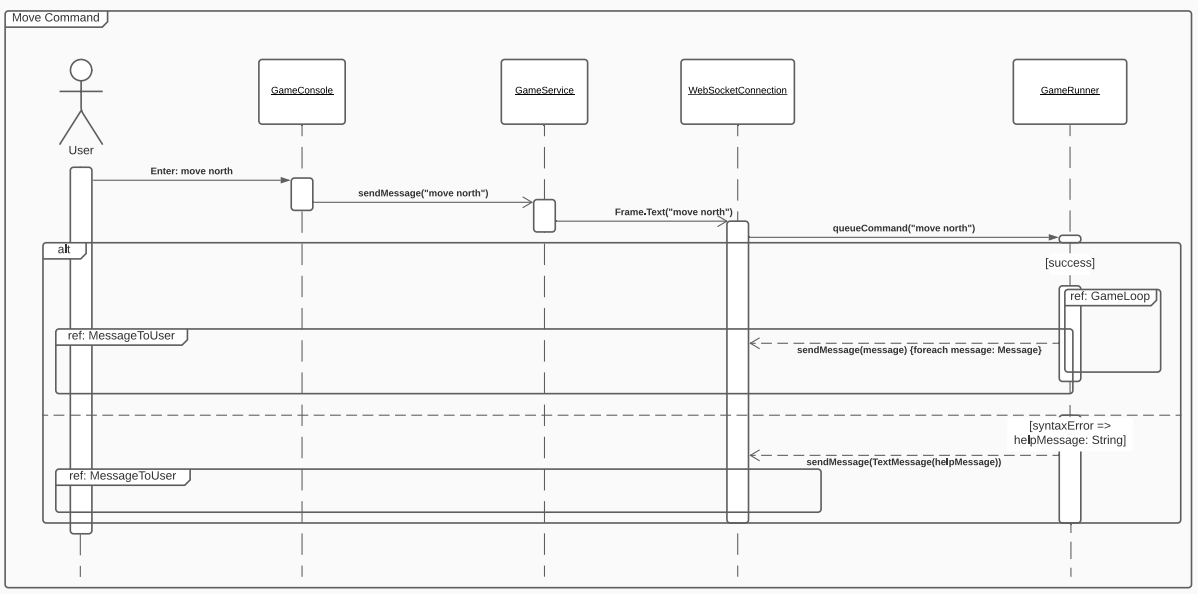

Abbildung 25 – Move Command – [Link zur Webansicht]

Das Senden einer Nachricht zum Nutzer läuft analog zum Empfangen. Da die Nachricht hier kein String sondern ein Objekt vom Typ Message ist, wird dieses vor dem Senden noch in JSON kodiert. Der Client kann dann je nach Typ der Nachricht (z.B. Chat oder Text) verschieden darauf reagieren.

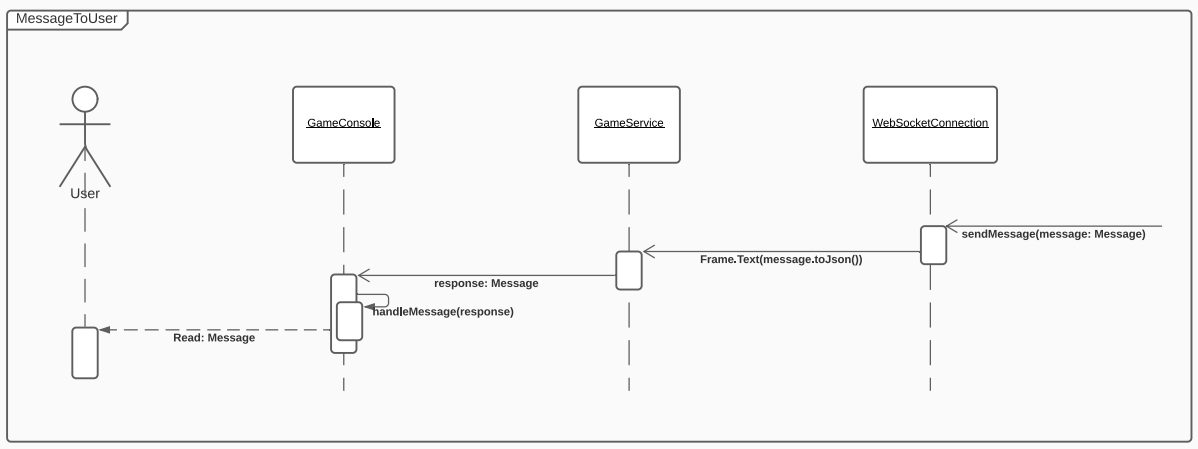

Abbildung 26 – Message To User – [Link zur Webansicht]

Die Gameloop verarbeitet nacheinander alle Kommandos aus der Queue. Dabei wird das Kommando abhängig vom aktuellen Spielstand zuerst in eine Liste von Aktionen umgewandelt. Falls dies fehlschlägt, wird dem Spieler eine Fehlermeldung geschickt. Ansonsten werden die einzelnen Aktionen je nach Typ entweder ausgeführt (Effect) oder der aktuelle Gamestate aktualisiert. Das Ganze läuft atomar ab, sodass die vom Gamestate abhängig erzeugten Actions immer auch auf denselben angewandt werden.

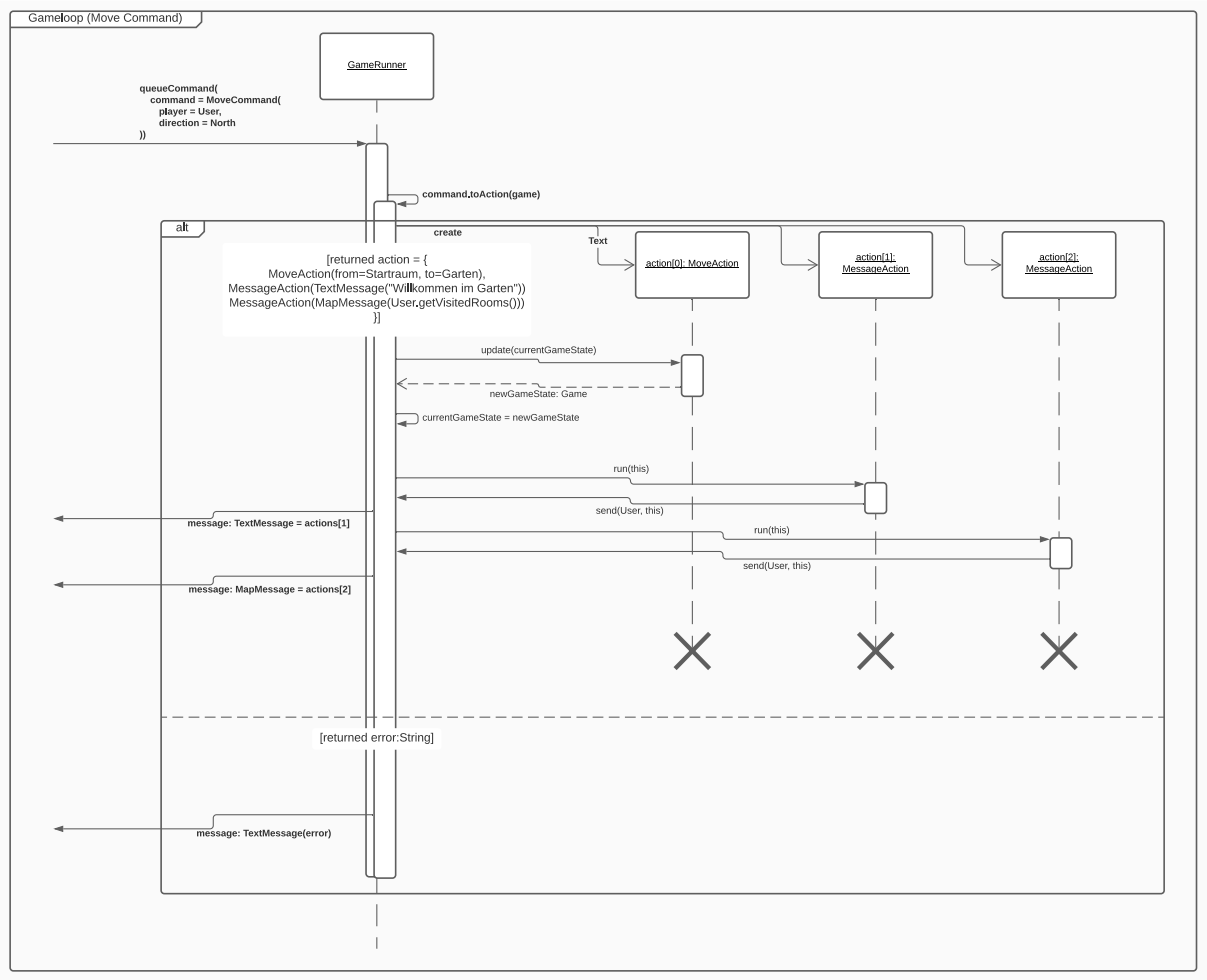

Abbildung 27 – Gameloop (Move Command) – [Link zur Webansicht]

Wenn der Nutzer einen Avatar erstellen möchte, kann er sich einen Namen geben und aus den Klassen und Rassen auswählen.

Der Name wird bei dem Versuch den Avatar zu speichern darauf überprüft, ob er in einem Spiel schon vorhanden ist. Wenn dies der Fall ist wird der Nutzer aufgefordert einen anderen Namen zu nutzen.

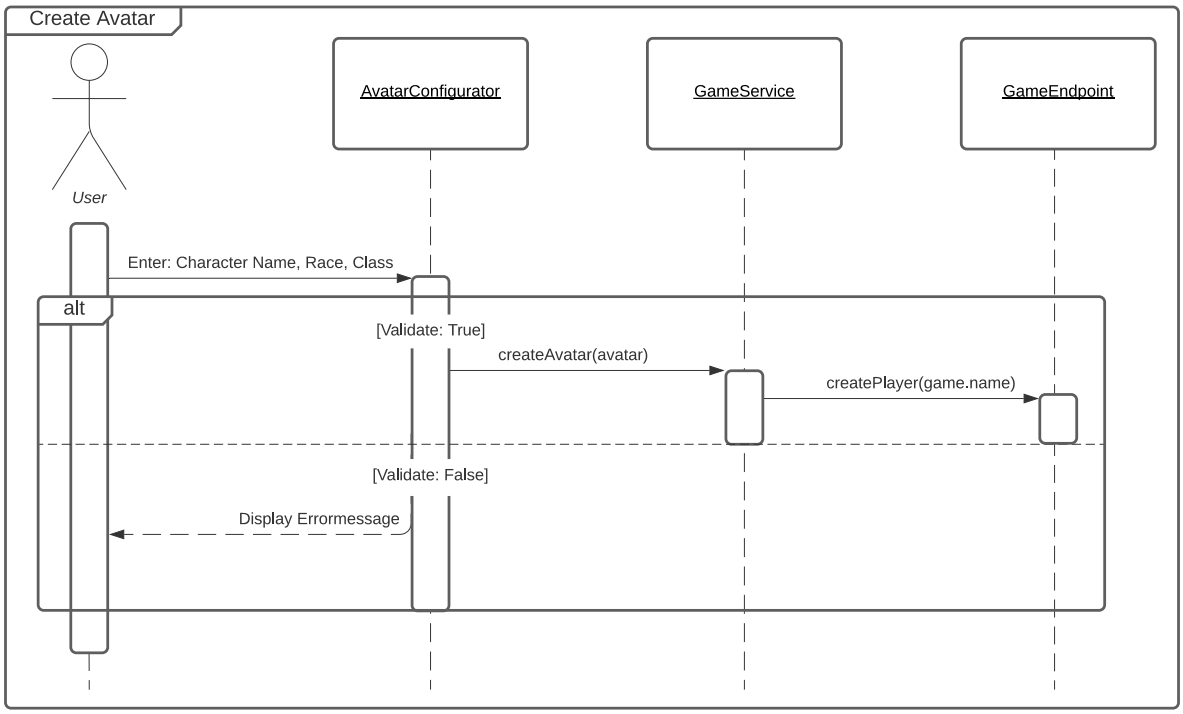

Abbildung 28 – Create Avatar – [Link zur Webansicht]

4.5 Database

Hier eine Darstellung der Daten die in der Datenbank gespeichert werden:

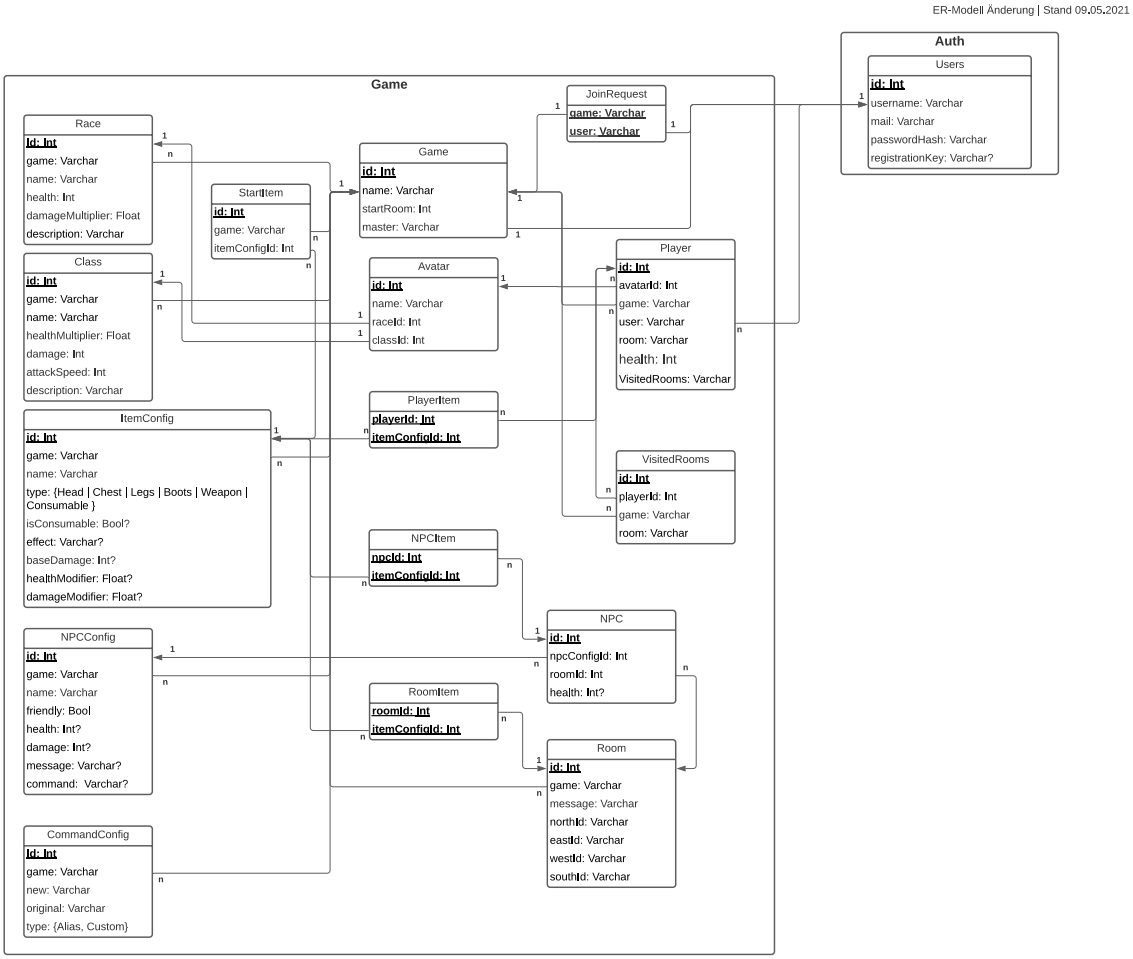

Abbildung 29 - ER-Modell - [Link zur Webansicht]

5. Anhang

Alle hier im Dokument dargestellten Diagramme können auch online in einem Dokument betrachtet werden.

Eine Übersicht des Statischen Modells kann man unter folgendem Link finden:
https://lucid.app/documents/view/e5b4d835-c47a-4aa6-a1a0-8c6d6ddf147d

Das Dynamische Modell kann unter folgendem Link finden:
https://lucid.app/documents/view/deaf86b4-0fea-431f-8cf7-ea3be18137ea