Designbeschreibung

Inhalt

1. Allgemeines

In dieser Designbeschreibung wird der Prototyp eines MUD-Servers beschrieben, welcher die Komponenten eines Konfigurators und das Hosten von mehreren MUD-Spielen beinhaltet. Dieser Prototyp dient dem Test der verschiedenen Struktur- und Entwurfsentscheidungen, um danach ein fertiges Produkt optimal erstellen zu können.

2. Produktübersicht

Der Prototyp besteht aus einer Website, über die sich ein Benutzer registrieren und anmelden. Sobald sich ein Benutzer angemeldet hat, hat er die Möglichkeit bestehenden MUD-Spielen beizutreten oder ein eigenes MUD-Spiel mit dem Konfigurator zu erstellen.

Wenn sich ein Benutzer dafür entscheidet, an einem Spiel teilzunehmen, wird er aufgefordert einen Charakter zu erstellen. Dafür muss er einen Namen eingeben und eine Klasse und eine Rasse auswählen. Nachdem der Charakter erstellt wird, befindet sich der Benutzer in einem Startraum und kann das Spiel spielen.

Wenn der Benutzer jedoch ein eigenes Spiel erstellen möchte, wird der Benutzer zum Konfigurator weitergeleitet, in diesem kann er nach eigenem Ermessen ein MUD-Spiel konfigurieren. Unter den Dingen die Konfiguriert werden müssen, sind unter anderem: Rassen, Klassen, Räume, NPCs und Equipment. Wenn die Konfiguration abgeschlossen ist, können andere Benutzer dem erstellten Spiel als Charaktere beitreten, während der Ersteller als MUD-Master am Spiel teilnimmt.

3. Grundsätzliche Struktur- und Entwurfsentscheidungen

Der Prototyp ist hauptsächlich in Kotlin geschrieben, da man damit Ktor nutzen kann, ein Webframework welches Websockets und Authentifizierung unterstützt. Des Weiteren werden JSON-Daten genutzt, um Informationen vom Benutzer an den Server zu schicken, welche unter anderem mit JavaScript erstellt werden. Die Websockets werden unter anderem für den Chat genutzt und im Spiel selbst, um Benutzer-Eingaben an das Spiel zu übermitteln.

Allgemein ist der Prototyp in den Konfigurator und das Spiel selbst unterteilt, welche beide unterschiedliche Wege nutzen, um Daten zu übertragen. Während der Konfigurator JSON-Daten empfängt und in verschiedene DataClasses umwandelt, nutzt das Spiel selbst Websockets um Daten zu übergeben.

4. Struktur- und Entwurfsentscheidungen der Pakete

4.1 F10 - Konfigurator

Die Erstellung der MUD-Konfiguration geschieht durch das Erstellen und Zusammenführen der einzelnen Elemente des MUDs. Die Konfiguration selbst und diese Elemente sind über IDs ansprechbar und suchbar.

Die UI:

Die UI wurde mit der Kotlin HTML DSL implementiert. Sie besteht aus Input-Feldern, in denen die Eigenschaften der zu konfigurierenden Elemente eingegeben werden können.

Der Konfigurator ist in einzelne Untergruppen unterteilt. Zu Beginn kann dem MUD eine ID gegeben werden. Danach gibt jeweils einen Bereich zur Konfiguration der Items, der NPCs, der Räume, der Charakterklassen und den Charakterrassen. Anschließend gibt es noch Inputs zur Konfiguration des Startequipments und des Startraums.

Jeder Bereich besitzt einen Submit-Button, mit welchem die Konfiguration des jeweiligen Bereichs bestätigt werden kann.

Über den Button „Speichern und zurück zur Startseite“ wird die Konfiguration des MUDs abgeschlossen und man gelangt zurück zur Startseite.

Übergabe der Daten an die Logik-Schicht:

Jeder Submit-Button besitzt ein eigenes onClick Event, welches eine Funktion in einem Skript-Block aufruft. Die Funktionen der einzelnen Submits generieren aus den eingegebenen Daten ein JavaScript Objekt. Dieses Objekt wird in einen JSON-String umgewandelt und über einen POST-request an den Server gesendet.

Der Server empfängt die Daten und wandelt den JSON-String in ein Objekt der jeweiligen Data-Class um. Anschließend werden die einzelnen Daten an die Logik-Schicht des Konfigurators übergeben.

Erstellung der Items:
Die Items werden in der „ItemConfigurator“-Klasse erstellt. Hier wird auf Basis von den Parametern „ID“ und „Name“ ein Item erstellt, welches danach in eine Sammlung aller Items des MUDs übernommen wird. Diese Sammlung an Items kann während der Konfiguration verwendet werden, wird aber am Ende nicht in der eigentlichen Konfigurationsdatei ausgegeben.

Erstellung von NPCs:
NPCs werden in der Klasse „NPCConfigurator“ erstellt. Alle NPCs besitzen eine Sammlung an Items (Inventar), einen Typen, einen Namen und eine Begrüßung. Die NPCs befinden sich in den Räumen und werden immer vor dem Raum konfiguriert in welchem sie sich befinden, denn bei der Erstellung des Raumes werden alle NPCs, welche seit der Erstellung eines letzten Raumes/seit dem Anfang erstellt worden sind in den neuen Raum hinzugefügt.

Der Ablauf der Erstellung der NPCs spaltet sich in drei Schritte auf:
Das Erstellen des Inventars des jeweiligen NPCs, die Erstellung des eigentlichen NPCs und dem Hinzufügen des NPCs zu der Sammlung, welche zur Verwendung in der Erstellung der Räume verwendet wird.

Die Erstellung des Inventars der NPCs geschieht dadurch, dass ein String mit allen gewünschten IDs übergeben wird. Dieser String wird aufgeteilt, und nun werden alle Items mit den genannten IDs aus der globalen Itemsammlung in das Inventar des NPCs kopiert.

Erstellung der Räume:
Die Erstellung der Räume findet in der „RoomConfigurator“-Klasse statt. Alle Räume besitzen eine Sammlung an NPCs und Relationen, basierend auf den 4 Himmelsrichtungen, zu anderen Räumen.

Die Sammlung der NPCs, welche in einem Raum existieren, wird vor der Erstellung des Raumes getätigt. Nachdem ein NPC erstellt worden ist, wird dieser in diese Sammlung übertragen und beim eigentlichen Erstellen des Raumes, wird diese Sammlung in das Raumobjekt gespeichert. Nach dieser Speicherung der Sammlung der NPCs im Raumobjekt, wird die NPC-Liste geleert, damit diese für weitere Räume neu definiert werden kann.

Die Verbindung zu anderen Räumen wird über Felder, welche die IDs anderer Räume besitzen gespeichert. Jeder Raum besitzt die Eigenschaften „north“, „east“, „south“, „west“. Diese Eigenschaften werden mit IDs von anderen Räumen versehen, um die Relationen zwischen den Räumen darzustellen. Wenn diese Eigenschaften mit einem leeren String versehen werden, heißt dies, dass an dieser Seite kein Raum ist.

Erstellung der Rassen für die Charaktererstellung:
Die Rassen werden über IDs in der „RaceConfigurator“-Klasse konfiguriert. Die Rassen werden einer Rassensammlung hinzugefügt, welche dann in der Konfigurationsdatei des MUDs hinterlegt ist. Die IDs der Rassen sollen auch als Name funktionieren, da jede Rassenbennenung

Erstellung der Klassen für die Charaktererstellung:
Die Klassen, welche zur Erstellung von Charakteren genutzt werden, werden durch IDs definiert. Diese IDs fungieren auch als Name der Klasse, da diese einzigartig sein sollten.
Die Klassen werden in der „ClassConfigurator“-Klasse definiert und danach in eine Sammlung an Klassen übergeben, welche dann in der Konfiguration des kompletten MUDs vorkommt.

Erstellung der Starterausrüstung:
Die Starterausrüstung ist eine Sammlung an Items. Die Starterausrüstung wird in der Klasse „StartingEquipmentConfigurator“ erstellt, auf Basis von den IDs, die der MUD-Ersteller übergibt. Die übergebenen IDs werden genutzt, um die Items mit den passenden IDs in die Starterausrüstung zu übergeben. Die Starterausrüstung wird später in der Konfiguration des kompletten MUDs aufgelistet.

Erstellung der eigentlichen MUD-Konfiguration:
Die Erstellung der eigentlichen MUD-Konfiguration geschieht in der Klasse „ConfiguratorMain“. Hier wird das Objekt der MUD-Konfiguration, auf Basis der Einzelkonfigurationen der genannten Elemente erstellt. Die MUD-Konfiguration umfasst: eine ID, eine Liste aller Räume, die ID des Startraumes, eine Liste aller Rassen, eine Liste aller Klassen und eine Sammlung der Starterausrüstung.

Um die MUD-Konfiguration später nutzen zu können, wird das Objekt, welches die Konfiguration enthält, mithilfe der GSON-Library zu einem Json-String serialisiert und in einer Map gespeichert, um sie später vom eigentlichen Spiel aus nutzen zu können.

4.2 F20+F30 – Plattform & Game-State/Savegame Zusammenhänge

Als grundlegende Plattform wird auch hier Kotlin genutzt. Über eine Startseite wird dem Benutzer die Möglichkeit geboten sich anzumelden, um das Spiel zu spielen. Dafür muss dieser mit dem dortigen Link interagieren.

Realisiert wurde dies mit OAuth von Ktor, welches einen Mechanismus zur sicheren Authentifizierung über externe Anbieter definiert. Das heißt, dass der User zu einer Autorisierung-URL des von uns angegebenen Anbieters weitergeleitet wird unter der Angabe einer clientid sowie einer gültigen Weiterleitungs-URL. Dort wird dem User nun die Möglichkeit geboten sich anhand seines Username & Passwort bzw. E-Mail & Passwort anzumelden. Sollte er noch keinen Account besitzen, so kann dort auch ein neuer erstellt werden unter Angabe der E-Mail, dem Passwort sowie eines einzigartigen Usernames. Das heißt, dass der Username nur 1-mal im System vorhanden sein kann. E-Mail-Bestätigungen zur Anmeldung bzw. das zurücksetzen des Passwortes wird auch über diesen Anbieter gehandelt.

Sobald nun die Anmeldung bzw. Registrierung korrekt ist, generiert der Anbieter einen Auth-Token unter der Verwendung eines clientSecret, welches mit genau dieser clientId verknüpft ist. Dann wird der Client an eine gültige und zuvor vereinbarte Anwendungs-URL mit einem Auth-Token umgeleitet, welches mit diesem clientSecret signiert ist.

Die oAuth-Funktion von Ktor verifiziert nun diesen Token und erzeugt ein Principal OAuthAccessTokenResponse (Access Token). Anhand dieses Tokens können wir nun Informationen des Users abfragen wie bspw. seine eindeutige ID, seine E-Mail und einiges mehr. Um jedoch seinen einzigartigen Username abzufragen nutzen wir die davor abgefragte eindeutige ID und fragen diese mit einem zweiten Request über die ManagementAPI ab. Dort können wir nun den eindeutigen Username des Users auslesen.

Um den Client nun durchgehend weiter zu identifizieren, nutzen wir einen weiteren Mechanismus von Ktor mit dem Namen Sessions. Dies bietet und die Möglichkeit Daten zwischen verschiedenen http-Anfragen aufrechtzuerhalten. Das heißt, dass wir dem User ein Cookie generieren mit dem Namen UserIdentifier, welcher jedoch nur eine Sitzungs-ID speichert. Die Session-Daten selbst werden im Speicher des Servers gespeichert und anhand der Sitzungs-ID, welcher zwischen dem Server und dem Client weitergeben wird, abgerufen.

Jegliche weiteren Aktionen, Mechanismen bzw. Savegames können nun stets einem User anhand des eindeutigen/einzigartigen Usernames zugeordnet werden.

4.3 F40 – Charakter konfigurieren

Die Charakter Konfiguration wird beim Anmelden in ein MUD-Spiel gestartet. Die Konfiguration beinhaltet drei Möglichkeiten, welche wie folgt sind: Name, Rasse und Klasse.

Dem Benutzer werden für jede Möglichkeit eine Frage gestellt, auf welche er antworten kann. Je nach Möglichkeit wird auf eine Korrektheit der Antwort überprüft, z.B. kann es nur einen Charakter mit einem Namen geben. Das bedeutet, wenn es schon einen Charakter mit dem eingegebenen Namen in der Spielerliste gibt, wird die Eingabe nicht akzeptiert. Für jeweils Rasse und Klasse wird nur eine Antwort akzeptiert, welche mit der Liste der möglichen Rassen und Klassen übereinstimmt. Diese Listen wurden vom MUD-Master im Konfigurator angelegt.

Die Antworten des Benutzers werden in einer Liste gespeichert und diese wird nachdem alle Fragen beantwortet sind genutzt, um den Character zu erstellen.

Dieser Charakter wird nun dem Username des Benutzers zugewiesen, damit nur dieser Benutzer ihn benutzen kann und somit ist die Charakterkonfiguration abgeschlossen. Dieser Charakter wird in die Collection der onlinePlayers aufgenommen, damit er für zukünftiges Betreten gespeichert ist.

4.4 F50 Spiel betreten und verlassen

Wenn sich ein Benutzer in ein MUD-Spiel einloggt, in dem er schon einen Charakter hat, wird er direkt in den Startraum gebracht und kann das Spiel spielen. Der Charakter wird von der Collection offlinePlayers, in die Collection onlinePlayers verschoben.

Wenn der Benutzer das MUD-Spiel verlässt oder schließt, wird sein Character wieder in die offlinePlayers Collection verschoben.

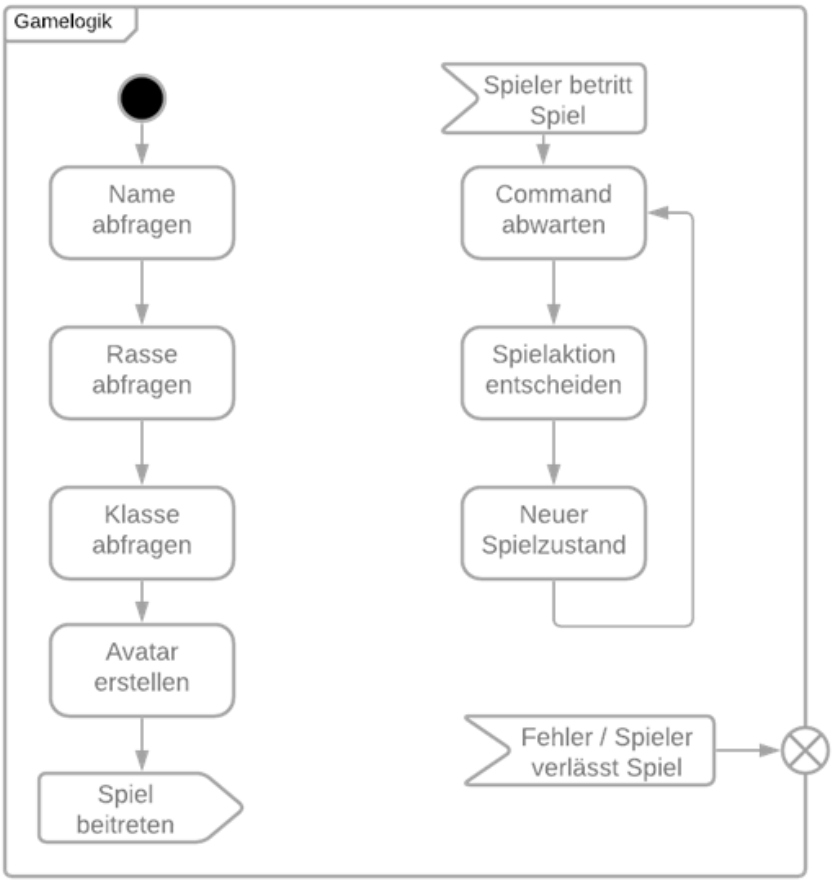
4.5 F60-70 Chat und Aktion durchführen

User Interface

Sobald der User authentifiziert ist, kann er die Seite "/dashboard" sehen, auf der zum einen eine Liste mit den verfügbaren Spielen, als auch ein Link zum Konfigurator zu sehen ist. Sobald er ein Spiel auswählt, wird er auf eine Seite mit einer Konsole weitergeleitet. Diese ist per WebSocket mit dem Server verbunden. Die Eingaben des Users werden "roh" zum Server geschickt und dort erst geparst.

Gamelogik

Sobald ein Spieler ein MUD-Spiel betritt, durchläuft er folgendene Aktionen.



In der eigentlichen Spielschleife wird basierend auf dem vorherigen Spielstand sowie einem Spielerkommando ein neuer Spielstand sowie eine Liste von Aktionen generiert, wodurch die Spiellogik von der Implementierung der Netzwerkanbindung getrennt bleibt:

```
doGameTick :: (Game, Command, Player) -> ([GameAction], Game)
```

Für die Demo gibt es aktuell nur die Aktion Message, mit der an beliebige Spieler im Spiel eine Textnachricht geschickt werden kann. Diese Aktion wird dann von der WebSocket-Implementierung den jeweiligen Netzwerkverbindungen zugeordnet und geschickt. Auch wird hier threadsicher der aktuelle Spielstand gehalten.