Design-Dokumentation

Inhalt

[Glossar 1](#_Toc36982700)

[Design der Anwendung 2](#_Toc36982701)

[Client 2](#_Toc36982702)

[Server 3](#_Toc36982703)

[Kommunikation zwischen Client und Server 3](#_Toc36982704)

[Datenhaltung 3](#_Toc36982705)

[Funktionsabdeckung Prototyp 4](#_Toc36982706)

[Übernahme des Prototyps 5](#_Toc36982707)

[Benutzerführung 6](#_Toc36982708)

[Registrierung 6](#_Toc36982709)

[Anmeldung 6](#_Toc36982710)

[Sessionauswahl 6](#_Toc36982711)

[Spielkonfiguration 6](#_Toc36982712)

[Charaktererstellung 7](#_Toc36982713)

[Spielbildschirm 7](#_Toc36982714)

[UML-Diagramm 8](#_Toc36982715)

# Glossar

Im Folgenden werden die Begriffe erläutert, die im weiteren Verlauf dieses Dokuments genutzt werden.

|  |  |
| --- | --- |
| Begriff | Bedeutung |
| MUD, Multi User Dungeon | Ein meist textbasiertes Rollenspiel, in dem mehrere Spielercharaktere ein Abenteuer erleben können. |
| DM, Dungeon Master | Person, die die Geschichte erzählt und die Spielergruppe durch den Spielinhalt leitet. Der DM besitzt nahezu völlige Kontrolle auf das Spielgeschehen, und kann auf den Chat der Spieler, mit Ausnahme des Flüsterchats, zugreifen und daran teilnehmen. |
| Spieler, User | Ein Teilnehmer an einer Session auf einem MUD, bei dem es sich nicht um einen Dungeon Master handelt. Ein Spieler in einer Session kann der Dungeon Master in einer anderen Session sein. |
| Responsive | Die Webseite soll sich dynamisch an verschiedene Browsergrößen anpassen. |
| Accountname | Name des Nutzeraccounts. Im Gegensatz zu den Nicknamen, die für die jeweiligen Sessions gewählt werden, ist der Accountname für andere Nutzer nicht einsehbar. |
| Rasse, Klasse | Beides sind Konfigurationsmöglichkeiten eines Spielercharakters. Sie beeinflussen die Fähigkeiten eines Charakters und seine Attribute. Im Verlauf des Spiels kann ein der Spieler abhängig von seiner Rasse bzw. Klasse zusätzliche Fähigkeiten erhalten. |
| Spiel | Ein eingeloggter User kann jederzeit ein neues Spiel konfigurieren. Dieser User wird zum DM dieses Spiels. Danach kann mit diesem Spiel eine Session gestartet werden, in der Spieler dieser beitreten und ihren Charakter einrichten können. |
| Session | Nachdem ein Spiel konfiguriert wurde, kann der DM eine Session starten. Diese kann von dem DM jederzeit beendet oder neu gestartet werden. |
| NPC | Non-Player Characters; Charaktere, die nicht von Spielern gesteuert werden |

# Design der Anwendung

Die Anwendung basiert auf einer Client-Server Struktur. Daraus entsteht eine strikte Trennung zwischen Front- und Backend.

Durch diese Aufteilung wird eine Modularität geschaffen, die später eine größere Flexibilität ermöglicht. Beispielsweise ist eine Einbindung eines alternativen Frontends (z.B. eine App oder ein Messaging-Bot) einfach umzusetzen.

Der Benutzer innerhalb der Anwendung mit *du* angesprochen. Neben der Spielwebseite gibt es eine Online-Dokumentation, die die für den Benutzer wichtigsten Funktionen erklärt und einen kleinen Überblick über die verwendete Technik gibt. Der vollständige Funktionsumfang wird im Lastenheft beschrieben.

### Client

Der Client entspricht größtenteils dem Frontend in Form einer Webseite. Er übernimmt keine relevante Logik, da dieser Programmteil lokal auf dem Benutzergerät läuft und dementsprechend leicht manipulierbar wäre. Der Kunde tritt nur mithilfe der Webseite mit dem Produkt in Kontakt

Die Umsetzung des Frontends erfolgt in HTML, CSS und JavaScript.

HTML und CSS wurden bis jetzt vollständig selbst entwickelt, auf Frameworks wurde verzichtet. Im weiteren Verlauf wird über die Benutzung eines CSS-Frameworks entschieden.

Die Umsetzung mit JavaScript erfolgt mithilfe des Frameworks Vue.js.

### Server

Der Server umfasst den Großteil des Backendes.

Das Backend beinhaltet die serverseitige Verarbeitung der Daten, die vom Frontend geliefert werden. Die gesamte Verifikation der Eingaben wird wegen Sicherheitsaspekten auf dem Server vorgenommen.

Die Umsetzung wird in der objektorientierten Sprache Java realisiert. Weiterhin ist aufgrund der Modularität und Wartbarkeit die Benutzung von Microservices geplant.

### Kommunikation zwischen Client und Server

Die Kommunikation zwischen Client und Server wird größtenteils über ReST erfolgen. ReST bedeutet Representational State Transfer. Durch die REST-API können verteilte Systeme (z.B. Server und Client) miteinander kommunizieren. Die Verwendung der ReST-API basiert auf bekannte Verfahren, wie HTTP/S, URI, JSON.

Der Informationsaustausch zwischen Server und Client ist in diesem Projekt essentiell, weshalb die Umsetzung der Schnittstelle von großer Bedeutung ist. Realisiert wird die REST-Schnittstelle in diesem Projekt mit Jersey und Tomcat. Jersey ist ein Standard, um REST-Services in Java zu implementieren. Mit Tomcat wird eine Umgebung bereitgestellt, um Webbasierten Java-Code auf einem Webserver auszuführen.

Zur Umsetzung des Chats ist eine stabile bidirektionale Verbindung ohne Verzögerungen für ein ungehindertes Spielerlebnis von Bedeutung. Sollte sich herausstellen, dass dies mit den vorhandenen Techniken nicht gewährleistet ist, wird die Chatkommunikation über Websockets realisiert.

### Datenhaltung

Wenn ein Spiel erstellt wurde, soll dieses für spätere Sessions wiederverwendet werden können. Dafür ist eine persistente Speicherung notwendig. Aufgrund der Möglichkeit von vielen Clients auf die Daten zuzugreifen und diese zu bearbeiten, ist eine widerspruchsfreie Speicherung unabdingbar.

Folgende Daten sollen persistent gespeichert werden:

* Anmeldedaten (E-Mail, Benutzername, Passwort)
* Charaktere der Spieler
* Spielkonfiguration
* Aktueller Stand der Session
* Fortschritt der Spiele

Um zu gewährleisten, dass diese Daten auch bei vielen Benutzern problemlos gespeichert werden können, ist eine Skalierbarkeit notwendig.

Eine effiziente Lösung, die alle diese Kriterien erfüllt, ist eine relationale Datenbank.

Die Vorauswahl ist bei uns auf MySQL und SQLite gefallen. Da die Datenmenge nicht erheblich groß ist und keine komplexeren Abfragen an die Datenbank gestellt werden, dürfte die schlanke SQLite Datenbank genügen. Sollte sich herausstellen, dass nützliche Funktionen nur in MySQL vorhanden sind, wird eine Migration auf diese Technik stattfinden.

# Funktionsabdeckung Prototyp

Um dem Kunden die späteren Produktfunktionen gut präsentieren zu können, eignet sich die teilweise Umsetzung des Frontends. Wie oben erwähnt wird, ist geplant dieses im Endprodukt mit HTML, CSS und JavaScript umzusetzen. Da diese Techniken keinen Server benötigen und rein im Webbrowser funktionieren, wird im Prototyp dieselbe Technik benutzt. Für den Prototyp wird mit Vue.js dasselbe Framework benutzt, das auch im Endprodukt zum Einsatz kommen soll.

Das Registrieren sowie das Zurücksetzten des Passworts wird im Prototyp nur simuliert, da der Mailversand aus dem Webbrowser nicht möglich ist. Deshalb ist ein Standardbenutzer vorkonfiguriert: Der Benutzername lautet „d6Prototyp“ und das Passwort ist „allesgrau“.

Die Kernfunktionen wie die Sessionauswahl, das Konfigurieren eines Spiels, die Auswahl beziehungsweise Erstellung eines Charakters und natürlich die Spieloberfläche werden bereits im Prototyp – selbstverständlich nicht in vollem Funktionsumfang – implementiert.

Da für viele dieser Aktionen im Endprodukt eine Kommunikation mit dem Server notwendig ist, muss im Prototyp ein Teil des Backends simuliert werden. Dies wird mit JavaScript gelöst, somit ist eine Erstellung eines temporären Servers nicht notwendig.

Eine persistente Speicherung wird deshalb im Prototyp nicht umgesetzt. Stattdessen werden die Spieldaten temporär in der Sitzung gespeichert. Ein Neu laden der Webseite löscht diese Werte.

### Übernahme des Prototyps

Da die Architektur des Prototyps ähnlich wie die des geplanten Endproduktes, ist geplant, einen Großteil des Prototyps weiter zu verwenden.

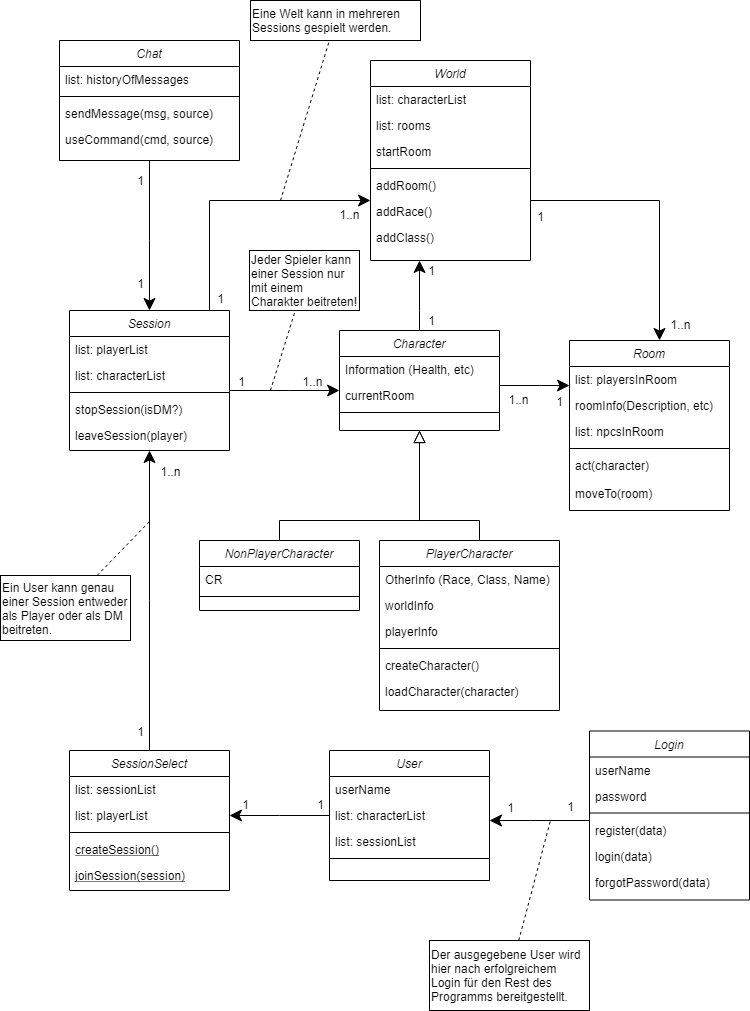
Die HTML und CSS-Elemente des Prototyps können fast ohne Anpassungen übernommen und weiterentwickelt werden. Da im Prototyp nicht großen Wert auf die farbliche Gestaltung gelegt wurde, wird eine ist eine teilweise Überarbeitung der CSS-Elemente notwendig, jedoch ist dies für die Funktionalität nicht notwendig und besitzt daher keine Priorität.

Die JavaScript-Teile wurden so geschrieben, dass diese weiterverwendet werden können. Jedoch sind die Teile, die das Backend simulieren sollen für die weitere Entwicklung nicht mehr notwendig.

Im Allgemeinen dürften etwas über zwei Drittel des Prototyps weiterverwendet werden können.

# Aufbau des Prototyps

Im folgenden Klassendiagramm ist die Struktur des Programms zu sehen. Da JavaScript keine objektorientierte Programmiersprache ist, sind die Klassen im Prototyp als abstrakte Darstellung zu sehen. Die Umsetzung der Reihenfolge sowie der Verbindungen ist jedoch genau so umgesetzt wie im Klassendiagramm.

Im Endprodukt wird die Überführung in Java wie im Klassendiagramm erfolgen, deshalb ist die Struktur des Prototyps darauf ausgelegt.

# Benutzerführung

### Registrierung

Vor der ersten Nutzung muss sich der Benutzer registrieren. Dafür wird eine E-Mailadresse benötigt, außerdem muss ein Benutzername und ein Passwort erstellt werden. Der Benutzername muss eindeutig sein. Der Benutzer bekommt eine E-Mail mit einem Bestätigungslink. Danach wird der Account freigeschalten.

Im Prototyp wird dieser Schritt nur simuliert, ein Mailversand findet nicht statt.

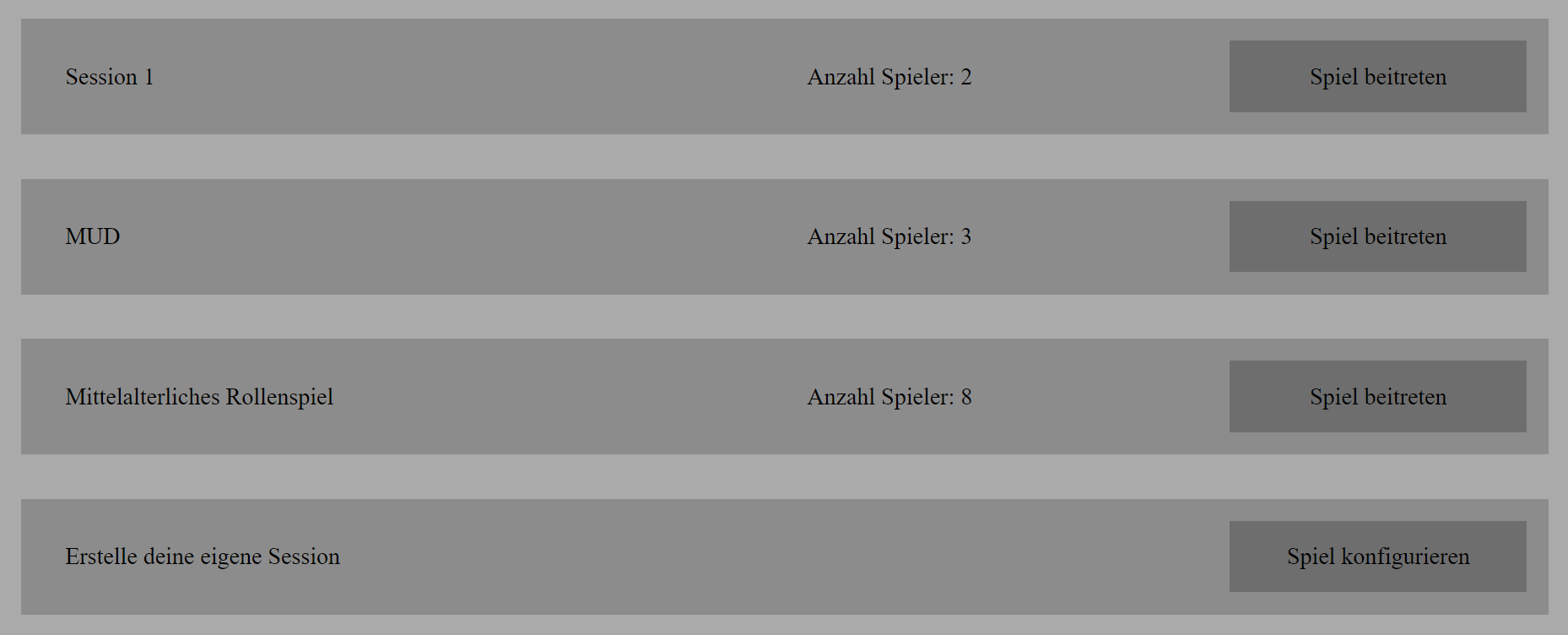
### Anmeldung

Nach erfolgreicher Registrierung kann sicher der Benutzer mit seinem Benutzernamen und Passwort anmelden. Hat er letzteres vergessen, kann er mithilfe der E-Mailadresse sein Passwort zurücksetzen.

Auch hier wird der Mailversand nur simuliert, zum Anmelden können die vorkonfigurierten Anmeldedaten verwendet werden (siehe oben).

### Sessionauswahl

Sobald die Anmeldung erfolgt ist kann der Benutzer einer Session beitreten oder eine neue Session erstellen. Im Bild ist zu sehen, wie dies im Prototyp umgesetzt ist.



### Spielkonfiguration

Entscheidet sich der Benutzer sein eigenes Spiel zu erstellen, hat er diverse Möglichkeiten, das Spiel zu konfigurieren. Dabei können zum Beispiel Räume erstellen, Verbindungen zwischen diesen konfiguriert und Gegenstände in diesen platziert werden. Außerdem kann dem Spiel ein Name gegeben werden sowie Rassen und Klassen erstellt werden.

Startet der Benutzer sein Spiel ist er automatisch der Dungeon Master. Er kann seiner eigenen Session auch als Spieler beitreten. Seine Dungeon Master-Rechte verliert er dabei nicht.

Bis auf die Verbindungen zwischen den Räumen und den Gegenständen sind diese Funktionalitäten im Prototyp enthalten, aber nicht immer vollständig grafisch dargestellt.

### Charaktererstellung

Bevor ein Spieler einem Spiel beitreten kann, muss er erst einen Charakter erstellen. Dabei kann er zwischen Rassen und Klassen wählen, die der Dungeon Master in der Spielkonfiguration erstellt hat. Im Laufe des Spiels kann die Spielfigur im Level steigen.

Einmal erstellte Charaktere können bei späteren Spielfortsetzung wieder geladen werden.

Im Prototyp ist dieses Feature, bis auf das Aufsteigen im Level, bereits vollständig umgesetzt.

### Spielbildschirm

Tritt ein Spieler oder ein Dungeon Master einem Spiel bei, kann er in diesem Programmteil mit anderen Spielern interagieren und Spielbefehle ausführen. Die Ansicht ist für normale Spieler und DMs gleich, jedoch hat der DM mehr Rechte und zusätzliche Befehle. Damit kann der DM entscheidend in das Spielgeschehen eingreifen.

Wird dieser Bildschirm verlassen, wird der Spieler automatisch aus der Session entfernt.

Im Prototyp ist aufgrund des Fehlens des Servers keine Interaktion mit anderen Spielern möglich. Eigene Chatnachrichten können versandt werden, auch einige Befehle funktionieren bereits. Auch DM-exklusive Befehle, wie das Entfernen von Räumen, sind bereits enthalten.