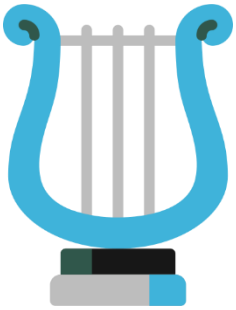


Designbeschreibung

Beschreibung der Softwarestudie

Version X

Team Apollon:

Paul-Benedict Burkard

Florian Albert

Leon Jerke

Daniel Kröker

Alfred Rustemi

Etienne Zink

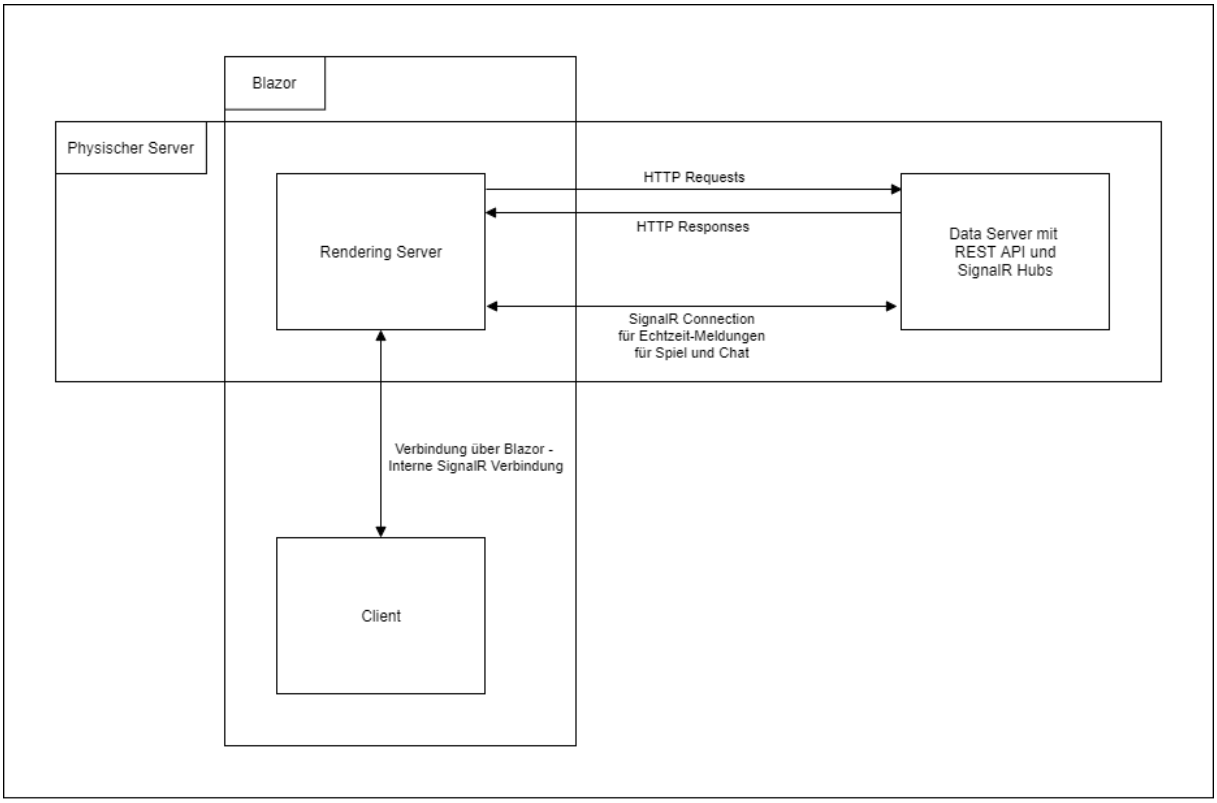
Projektbezeichnung	APOLLON
Projektleiter	Paul-Benedict Burkard
Verantwortlich	Daniel Kröker
Versionsdatum	27.03.2021

1. Einführung

Zur Einarbeitung in das Fachkonzept ist eine kleine Softwarestudie anzufertigen, in der es möglich ist, einige grundlegende Aspekte des Themas prototypisch zu realisieren. Es gibt keine weitergehenden allgemeingültigen Ansätze, um Softwarequalität unabhängig vom Einsatzgebiet und Aufgabenstellung zu sichern. Allerdings existieren für einzelne Anwendungsbereiche Erfahrungen, welche Designprinzipien und Architekturkonzepte im jeweiligen Gebiet besonders erfolgreich waren und deshalb weit verbreitet sind. Design Patterns sind im Rahmen dieses Projektes zu beachten, jedoch nicht ausdrücklich zu erwähnen. Mit der Benutzung der Design Patterns kommt eine zufriedenstellende Qualität der Produkte einher. Zusätzlich sind die Entwickler bereits mit der Art und Weise dieser Arbeit vertraut. Einem unbeteiligtem Softwareentwickler ist es daher möglich, mit diesem Dokument, sich mit dem Thema auseinanderzusetzen und aktiv miteinbringen zu können.

2. Produktübersicht

Die Anwendung soll im Wesentlichen die Möglichkeiten einer Authentifizierung per E-Mail, das Erstellen und Konfigurieren eines Dungeons, das Erstellen eines Avatars und die Kommunikation mit den Spielern über einen Chat bieten. Für detaillierte Beschreibungen wird auf das Lastenheft verwiesen. In diesem Bericht geht es um die Struktur- und Entwurfsentscheidungen. Folgende Gesamtarchitektur ist zur Realisierung des Prototyps dargestellt:



Eine genauere Erläuterung der Architektur erfolgt unter dem Abschnitt „3. Grundsätzliche Struktur- und Entwurfsentscheidungen“. Aufgrund der mangelnden Zeit wurde die obige Darstellung jedoch nur teilweise umgesetzt. Der Data Server wurde deswegen vereinfacht und auf die Benutzung einer REST API verzichtet.

3. Grundsätzliche Struktur- und Entwurfsentscheidungen

Im Folgenden werden die einzelnen Komponenten, welche in der Gesamtarchitektur im Punkt „Produktübersicht“ dargestellt wurden, genauer erläutert.

3.1 Blazor

Blazor ist ein Framework zum Erstellen einer interaktiven clientseitigen Web-Benutzeroberfläche mit .NET. Vorteile der Verwendung von .NET für die clientseitige Webentwicklung sind:

- Schreiben von Code in C# anstatt in JavaScript
- Möglichkeit zur Verwendung des vorhandenen .NET-Ökosystem von .NET-Bibliotheken
- App-Logik server- und clientseitig freigeben

Die Blazor-Apps basieren auf Komponenten. Eine Komponente in Blazor ist ein Element der Benutzeroberfläche, beispielsweise eine Seite, ein Dialogfeld oder ein Formular für die Dateneingabe. Komponenten sind .NET-Assemblys integrierte .NET-C#-Klassen, auf die folgendes zutrifft:

- Definition der Logik für ein flexibles Rendering der Benutzeroberfläche
- Behandlung von Benutzerereignissen
- Möglichkeit von Schachtelung und Wiederverwendbarkeit
- Freigabe und Verteilung als Razor-Klassenbibliotheken oder NuGet-Pakete

Für weitere Informationen zu Blazor wird auf die Dokumentation „Einführung in ASP.NET Core Blazor“ von Microsoft verwiesen. (https://docs.microsoft.com/en-us/aspnet/core/blazor/?view=aspnetcore-5.0)

3.2 SignalR

SignalR ist eine Bibliothek für ASP.NET-Entwickler, die das Hinzufügen von Echtzeit Webfunktionen zu Anwendungen vereinfacht. Echtzeit Webfunktionalität ist die Möglichkeit, Servercode Inhalte sofort an verbundene Clients weiterzuleiten, sobald diese verfügbar sind.

SignalR wird für den Chat der Anwendung verwendet. Genauere Informationen finden sich auf der Dokumentation „SignalR“ von Microsoft (https://docs.microsoft.com/en-us/aspnet/signalr/).

3.3 SQLite

SQLite ist eine C-Sprachen Bibliothek die eine kleine, schnelle, unabhängige, zuverlässige, funktionsumfängliche SQL-Datenbank implementiert. Eine Datenbank ist für die Anmeldungen der Spieler notwendig. Weitere Informationen zu SQLite finden sich unter den Dokumentationen von Microsoft (https://docs.microsoft.com/en-us/windows/uwp/data-access/sqlite-databases; https://docs.microsoft.com/en-us/dotnet/standard/data/sqlite/?tabs=netcore-cli; https://www.sqlite.org/index.html)

3.4 Mailserver

Ein Mailserver ist ein Server, der E-Mails entgegennehmen, weiterleiten, bereithalten und senden kann. Die E-Mail-Adressen, deren E-Mail-Postfächer der Mailserver verwaltet, erben ihren Domain-Part vom Domain-Namen des Mailservers. Der Domain-Name des Mailservers wiederum ist die Domain des E-Mail-Anbieters, der den Mailserver betreibt.

3.5 REST

Die REST-API macht den Austausch von Informationen möglich. Der Informationsaustausch per REST-API wird erst genutzt, wenn es sich um unterschiedliche Systeme handelt. Im Zeitalter von Computer und mobilen Endgeräten ist eine Benutzung von einer REST-API notwendig. Bei einer REST-API handelt es sich um eine Maschine-Maschine-Kommunikation. Verschiedene Systeme und Geräte werden zusammengebracht. Ein einheitlicher Informationsaustausch ist dadurch gewährt. Informationen und Aufgaben verteilen sich auf verschiedene Server. Die Informationen sind mithilfe eines HTTP-Requests anzufordern. Der HTTP-Request setzt sich aus dem Endpoint und den entsprechenden Parametern zusammen.

4. Struktur- und Entwurfsentscheidungen für einzelne Pakete/Komponente

Der Prototyp besteht aus einer Solution, welche sich wiederrum in mehrere Projekte aufteilt. Jedes dieser Projekte stellt eine gekapselte Einheit dar. Nachfolgend werden die wichtigsten Projekte benannt und ihre Funktion erklärt.

4.1 Apollon.MUD.Prototype.Blazor.Domain

Im Projekt Blazor.Domain befindet sich, wie der Name bereits andeutet, die Komponenten für Blazor und somit des Frontends. Die Komponenten teilen sich in die verschiedenen Teile der Website auf und wurden wie bereits erwähnt in HTML und C# Code erstellt. Hierdurch sind eine bessere Wartbarkeit und eine geringere Streuung der Kompetenzen in diversen Programmiersprachen von Nöten. Da der Prototyp hauptsächlich durch die UI gesteuert wird und dieses somit das „Haupt“-Projekt ist, befindet sich hierin auch noch die Programm- und Startup-Konfiguration, wie auch die SQLite Datenbank Datei.

4.2 Apollon.MUD.Prototype.Core.Domain

Das Projekt Core.Domain stellt die Schnittstelle zwischen dem Frontend und dem „Data Server“ (Backend) dar. Durch Klassen wie dem ClientContext oder diversen Hubs, wird die Kommunikation und Weiterleitung der Anfragen des Clients ermöglicht. Außerdem befinden sich hierin die Klassen, welche zur Konfiguration eines Dungeons benötigt werden.

4.3 Apollon.MUD.Prototype.Core.Interfaces

Durch die Interfaces, welche sich im Projekt Core.Interfaces befinden, wurden globale Schnittstellen definiert. Dadurch wird die Zusammenarbeit erleichtert und die Abgrenzung der Funktionalitäten begünstigt. Außerdem bieten die Interfaces die Möglichkeit, selbst im Prototypen Implementierungen auszutauschen. Dies geschieht durch Klassen mit gleicher Basisklasse oder Interface.

4.4 Apollon.MUD.Prototype.Core.Implementation

Die bereits im Projekt Core.Interfaces erwähnten Interfaces werden im Projekt Core.Implementation in konkreten Klassen implementiert. Diese Klassen werden als sogenannte „Beans“ verwendet. Sie besitzen wenig, bis keine funktionale Logik und dienen als reine „Datenspeicher“-Klassen, welche zur Laufzeit verwendet werden.

4.5 Restliche

In den restlichen Projekten befinden sich Klassen, welche übergeordnete Funktionalitäten bereitstellen. Ein Beispiel hierfür wäre z.B. das DungeonRepo, welche die Verbindung zwischen dem ClientContext und dem Dungeon selbst herstellt und somit das Dungeon und die Aufrufe auf dessen in sich kapselt.

Versions-Historie

Version	Datum	Bearbeiter	Art der Änderung	Stand
1.0	27.03.2021	Alfred Rustemi, Daniel Kröker	Erstes Aufsetzen des Dokuments	Fertiggestellt
2.0	28.03.2021	Team Apollon	Besprechung des Entwurfs	Fertiggestellt
3.0	28.03.2021	Etienne Zink	Verbesserung des Dokuments	Fertiggestellt
4.0	28.03.2021	Etienne Zink, Daniel Kröker	Qualitätskontrolle	Fertiggestellt