



Mettmann

Studienarbeit

Entwicklung eines Softwareprojektes

Spiele Lizenzschlüssel Preisvergleich

Prüfer:

Dr. Thomas Ströder

Verfasser:

Niklas Hardes

Nicolas Groß

Am Wildgatter 24

Winkelstraße 66

45891 Gelsenkirchen

45966 Gladbeck

Matrikelnummer: 101671

Matrikelnummer: 101669

Robert Hesselmann

Kronprinzenstraße 83

40217 Düsseldorf

Matrikelnummer: 101672

Studiengang : Angewandte Informatik

Abgabetermin:

16. Januar 2023

Vorbemerkung

In dieser Arbeit wird aus Gründen der besseren Lesbarkeit das generische Maskulinum verwendet. Weibliche und anderweitige Geschlechteridentitäten werden dabei ausdrücklich mitgemeint, soweit es für die Aussage erforderlich ist.

Inhaltsverzeichnis

Vorbemerkung	II
Abbildungsverzeichnis	VI
Tabellenverzeichnis	VII
Quelltextverzeichnis	VIII
1 UserStories & Use Cases	1
2 Technologieentscheidungen	3
3 Entwicklungsprinzipien	6
4 Entwicklungsprinzipien-Verstöße	8
5 Visualisierte Sichten auf das Gesamtsystem	9
6 Entwurfsmuster	10
7 Architekturstile	12
Anhang	14
Quellenverzeichnis	15
Ehrenwörtliche Erklärung	16

Abkürzungsverzeichnis

CI/CD Continious-Integration/Continious-Deployment

Abbildungsverzeichnis

Abbildung 1: Use Case Authentifizieren	1
Abbildung 2: Use Case Streamer	2
Abbildung 3: Use-Case Diagramm Gesamt System	9

Tabellenverzeichnis

Quelltextverzeichnis

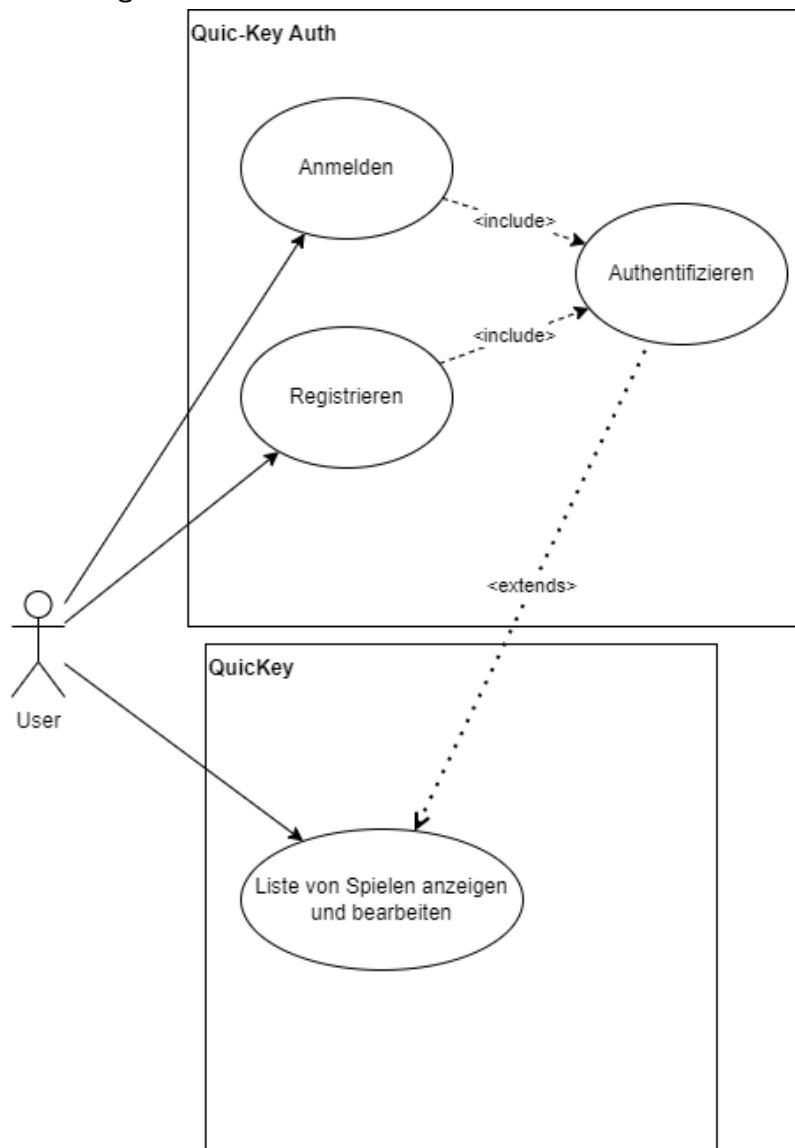
Quelltext 1: Codeausschnitt des Tabs Routingmodule	10
Quelltext 2: Codeausschnitt von CustomHttpRequest (1)	11
Quelltext 3: Codeausschnitt von CustomHttpRequest (2)	11

1 UserStories & Use Cases

Niklas Hardes

Ein Kunde möchte eine persönliche Liste von Spielen anlegen können, mit den für ihn beliebtesten Spielen. Dadurch kann er mit wenig Aufwand diese schnell auf Ihren aktuellen Preis überprüfen.

Abbildung 1: Use Case Authentifizieren

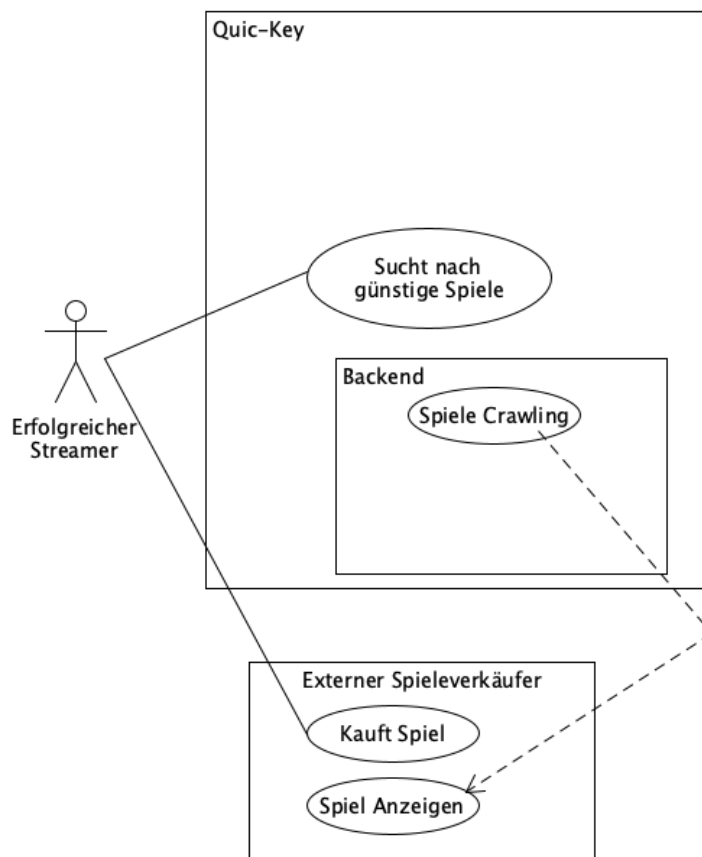


Quelle: Eigene Darstellung

Robert Hesselmann

Ein erfolgreicher Streamer auf der Livestreaming Plattform Twitch sucht nach neuen Spielen, die er in seinen Livestreams spielen kann. Um seinen Gewinn zu maximieren sucht er nach einem Spiel zu einem günstigen Preis.

Abbildung 2: Use Case Streamer



Quelle: Eigene Darstellung

Nicolas Groß

Hallo Welt

2 Technologieentscheidungen

Nicolas Groß

Bei der Entwicklung des Frontend haben wir uns für das Ionic-Framework entschieden. Das Ionic-Framework ist ein Open-Source UI Toolkit für die Entwicklung von Hybrid-Apps. Hybrid-Apps sind Anwendungen, welche auf Web-Technologien wie HTML, CSS und JavaScript basieren und die nativen APIs und Funktionen des Betriebssystems nutzen können.¹ Ionic bietet Vorteile wie Performance-Optimierung, Cross-Plattform und die automatische Bereitstellung einer Hellen und Dunklen Ansicht. Ionic bietet zahlreiche anpassbare UI-Komponenten, welche helfen die Strukturierung und Gestaltung der Webanwendung vorzunehmen. Zudem bietet Ionic eine ausführliche Dokumentation und eine große Community, wodurch das Einbauen unbekannter Komponenten schnell und einfach ist. Ionic unterstützt neben dem klassischen JavaScript die Frameworks Angular, React und Vue.²

Hierbei fiel unsere Entscheidung aus erfahrungsgründen auf Angular. Angular ist ein von Google entwickeltes auf TypeScript basierendes Open-Source Webframework zur Entwicklung von Mobile und Desktop Webanwendungen. Angular stellt ein Command-Line-Interface bereit, welches Entwicklern bei der Erstellung und Entwicklung der Projekte unterstützt. Die Entwicklung von Single-Page-Apps wird durch Angular Routing ermöglicht, wodurch Webanwendungen schneller auf Benutzereingaben reagieren und die Ladezeiten zwischen einzelnen Seiten stark vermindern. Zudem können Inhalte der Webseite dynamisch geladen und angepasst werden. Angular bietet genau wie Ionic eine ausführliche Dokumentation, Lehrunterlagen und eine aktive Community. Durch Angular ergibt sich die Möglichkeit umfangreiche Bibliotheken einzubinden und deren Ressourcen für die Entwicklung des Projektes zu nutzen.³

Docker (Robert Hesselmann)

«««< Updated upstream Bei der Planung unseres Softwaresystems haben wir uns für Docker als Laufzeitumgebung entschieden. Dadurch kann unsere Software auf allen System laufen die Docker unterstützen. Der Ablauf vom Push eines Commits bis zum Deploy der Software, geht die Software durch die Schritte des Bau

¹Anonymus, n.d. .vgl.

²Drifty Co., 2013a, .vgl.

³Google LLC., 2016, .vgl.

Datenbank (Niklas Harges)

Bei der Datenbank gab es diverse Möglichkeiten zu betrachten. Einerseits existieren die relationalen Datenbanken unter denen am bekanntesten die MySQL, PostgreSQL, MariaDB, Microsoft SQL Server und die Oracle Database sind. Eine relationale Datenbank ist dabei eine Sammlung von Informationen, welche Ihre Daten in Form von Tabellen mit Spalten und Zeilen speichert. Beziehungen sind hier vorab definiert. Neben den relationalen Datenbanken existieren die nicht relationalen Datenbanken. Diese werden auch NoSQL Datenbanken genannt. Die bekanntesten sind hier Apache Cassandra, Riak, MongoDB, Redis und CouchDB. Die relationalen Datenbanken eignen sich gut bei fest definierten Schemas und einer gewollten Stabilität und Sicherheit. Wohingegen nicht relationale Datenbanken kein fest definiertes Schema besitzen und so ein hohes Maß an Flexibilität bei Datenmodellen gewähren. Des Weiteren sind sie besser geeignet für riesige Datensätze und außerdem kostengünstiger im Aufbau und dem Betrieb.

Im ersten Schritt haben wir uns dafür entschieden, eine nicht relationale Datenbank zu wählen. Dies einmal aus dem Grund, dass davon auszugehen ist, bei dem Sammeln aller aktuellen Preise von möglichst vielen Spielen bei möglichst vielen Händlern eine sehr große Menge an Daten anfällt. Der andere Grund ist eine gewollte Flexibilität. Eine detaillierte Datenbankmodellierung war zu Beginn nicht möglich. Die genauen Daten, welche wir von allen Händlern bekommen würden, mussten erst noch in Erfahrung gebracht werden. Mit der Zeit und dem dazu kommen von immer mehr Händlern muss die Datenbank sich daran anpassen können und gleichzeitig ermöglichen, dass sich nicht zu viel damit beschäftigt werden muss ein gesamtes Modell für die Datenbank anzulegen.

Unter den relationalen Datenbanken haben wir uns für eine MongoDB entschieden. Der Grund dafür ist, dass hier bereits eine große Instanz existierte, an die sich angeschlossen werden konnte. Diese Instanz besteht aus mehreren Nodes, was für Datensicherheit sorgt, und aus SSDs mit einem sehr hohen Durchsatz. Dadurch konnten wir ohne jedwede Kosten eine hochperformante Lösung benutzen. ===== Bei der Planung unseres Softwaresystems haben wir uns für Docker als Laufzeitumgebung entschieden. Dadurch kann unsere Software auf allen System laufen die Docker unterstützen. Der Ablauf vom Push eines Commits bis zum Deploy der Software, geht durch unsere Continuous-Integration/Continuous-Deployment (CI/CD), die Software wird Schritt für Schritt Compiliert, Analysiert und in ein Docker-Image verpackt, welches im letzten Schritt auf unseren Server ausgespielt wird.

Der Vorteil von Docker ist, dass Docker in Isolierten Containern arbeitet. Somit

bekommen andere Container nicht davon mit wird ein andere Container erstellt oder das der Container eine Berechnung macht. Ein weiterer Vorteil von Docker sind die universell einsetzbaren Images. Diese Images bilden einen Standard ab. Somit kann ein beliebiges Image auf jeder Plattform eingesetzt werden. »»»> Stashed changes

3 Entwicklungsprinzipien

Nicolas Groß

Modularisierung ist die Unterteilung eines Systems in kleinere Module. Die Modularisierung bietet Eigenschaften, welche bei der Entwicklung und Wartung von Software, Vorteile einbringen. Zu diesen Vorteilen zählen Verständlichkeit, Kombinierbarkeit, Lokalität und die parallele Entwicklung. Die Verständlichkeit einer Software wird durch die Modularisierung verbessert, da einzelne Bestandteile weitgehend unabhängig von anderen Bestandteilen gekapselt und verständlich sind. Die einzelnen Module einer Software können auf unterschiedliche Arten miteinander kombiniert werden, um neue Systeme zusammenzufügen. Dies bietet den Vorteil das Module idealerweise unabhängig vom restlichen System funktionieren und wiederverwendet werden können. Durch die Lokalität zeichnet sich aus, dass Änderungen an einzelnen Modulen keine größeren Änderungen im Gesamtsystem zufolge haben. Ein entscheidender Vorteil für die Entwicklung im Team ist die Möglichkeit zur parallelen Entwicklung. Hierbei können einzelne Teammitglieder an unterschiedlichen Modulen arbeiten, ohne mit den Entwicklungen der anderen Mitglieder zu kollidieren. Das System wird dann zu einem späteren Zeitpunkt zusammengesetzt.⁴

Durch Angular ist eine hohe Modularisierung bereits von Beginn des Projekts gegeben, da die Angular-CLI Komponenten und Services in einzelnen Dateien erzeugen. Auch die Projektstruktur wird von der Angular-CLI angepasst, um Komponenten und Services anzulegen. Die einzelnen Komponenten können durch die Verwendung von Variablen wiederverwendbar gestaltet werden, um diese beliebig zu kombinieren. Die Änderungen an einer Komponente haben hierbei in der Regel keinen Einfluss auf andere Komponenten oder Services, wodurch Änderungen meist leicht umgesetzt werden können und dennoch Projektweit geltend sind.

Single-Responsibility Prinzip (Niklas Hades)

Bei dem Single-Responsibility Prinzip lautet die Kern-Definition „Ein Modul sollte einem, und nur einem, Akteur gegenüber verantwortlich sein.“. Es ist ein wichtiger Bestandteil der SOLID-Prinzipien der objektorientierten Programmierung. Es bedeutet, dass jede Klasse, Methode oder Funktion in einem Programm nur eine einzige Verantwortung hat.

⁴Schmidauer, 2002, .vgl.

Dem entgegen steht ein sogenanntes God-Object. Es hat viele Verantwortungen und wirkt sich auf unterschiedliche Bereiche aus. Das macht es schwierig, dieses zu verstehen, zu testen und zu warten. Des Weiteren kann es aufwendig sein, alle Auswirkungen im Blick zu behalten, die eine Änderung an diesem mit sich bringen würde. Insofern ist es sehr zu empfehlen, die Verantwortlichkeiten gering zu halten bzw. im Idealfall nur eine einzige zu haben.

Ebenfalls gibt es Auswirkungen auf Abhängigkeiten zwischen Klassen. Wenn eine Klasse mehrere Verantwortungen hat, kann es viele Abhängigkeiten zu anderen Klassen geben, die die Wartbarkeit des Codes beeinträchtigen können. Eine Klasse mit einer einzigen Verantwortung hat jedoch in der Regel weniger Abhängigkeiten, was die Wartbarkeit des Codes verbessert.

Zusammenfassend, unter Einhaltung des SRP sollte jede Klasse, Methode oder Funktion nur eine einzige Verantwortung haben. Dies erleichtert es, Änderungen an einer Klasse vorzunehmen, ohne Auswirkungen auf andere Teile des Systems zu haben, und die Abhängigkeiten zwischen Klassen zu minimieren.

Das beste Beispiel liefert hierzu unsere Implementation des ThreadedWebSocket im Namespace QuicKey.Business.Web. Es besitzt eine geringe Anzahl an Abhängigkeiten, welche ebenfalls über Interfaces ausgetauscht werden können und somit gut testbar sind. Seine einzige Verantwortlichkeit liegt dabei, die Kommunikation über ein WebSocket zu ordnen, zu steuern und anderen Klassen zur Verfügung zu stellen.

4 Entwicklungsprinzipien-Verstöße

Open-Closed Prinzip (Niklas Harges)

Das Open-Closed-Prinzip ist ein Prinzip der Softwareentwicklung, das besagt, dass ein Modul oder eine Klasse geöffnet sein sollte für Erweiterungen, aber geschlossen für Änderungen. Dies bedeutet, dass neue Funktionalitäten hinzugefügt werden können, ohne bestehenden Code zu ändern.

Ein wichtiger Aspekt des Open-Closed-Prinzips ist die Verwendung von Abstraktionen. Durch die Verwendung von Abstraktionen können neue Funktionalitäten implementiert werden, ohne die bestehenden Klassen oder Module zu ändern.

Das Open-Closed-Prinzip ist Teil der SOLID-Prinzipien, einer Sammlung von fünf Prinzipien der objektorientierten Softwareentwicklung, die entwickelt wurden, um Code wartbar und erweiterbar zu machen. Es ist ein grundlegendes Konzept in der objektorientierten Programmierung und hat großen Einfluss auf die Architektur und das Design von Software-Systemen.

Es ist wichtig zu beachten, dass das Open-Closed-Prinzip nicht immer einfach umzusetzen ist und dass es in manchen Fällen notwendig sein kann, bestehenden Code zu ändern, um neue Anforderungen zu erfüllen.

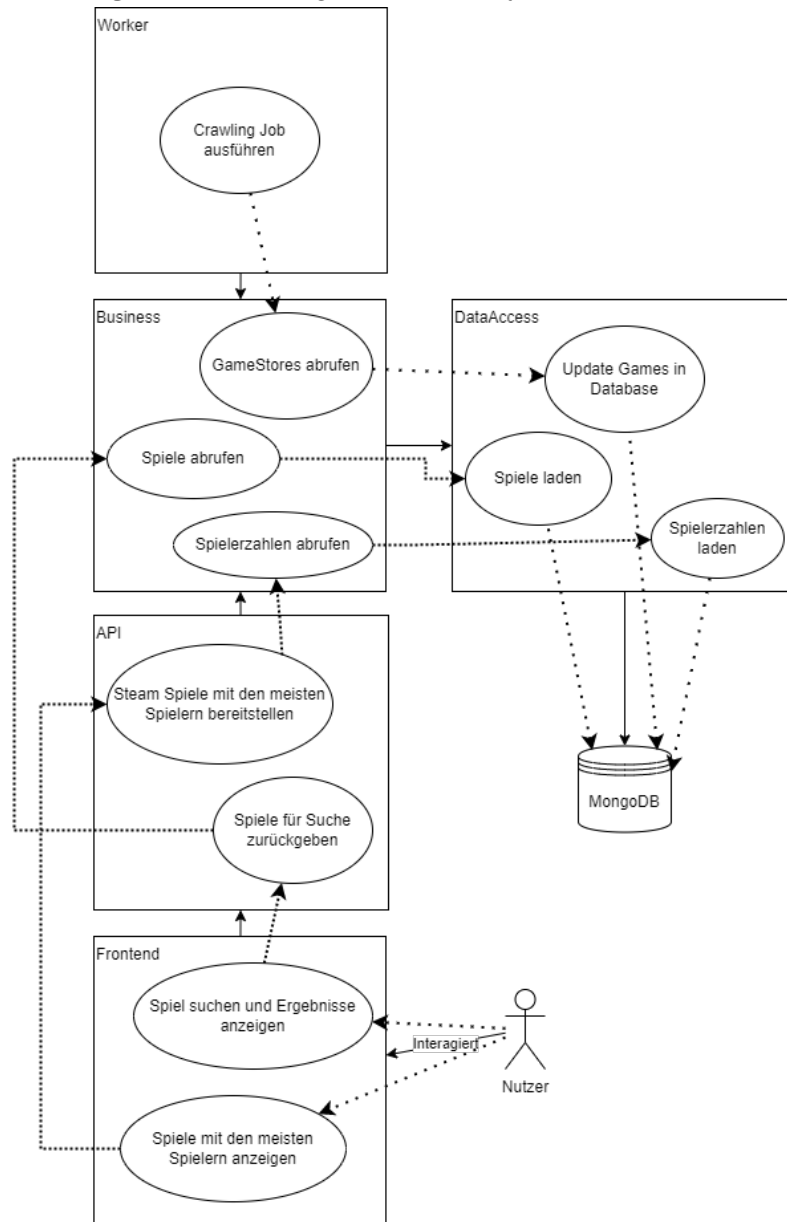
In unserem Projekt haben wir gegen dieses Prinzip absichtlich verstoßen. In dem Wrapper für die MongoDB haben wir neben höheren Funktionen für z.B. GetFirst und GetAll auch einen direkten Zugang zum Objekt der MongoDB Lib Public gemacht, wodurch ein direkter Zugang ermöglicht wird. Das hat die Entwicklung deutlich erleichtert, bringt aber den Nachteil eines nicht verwalteten Zugriffs ohne Beschränkungen.

5 Visualisierte Sichten auf das Gesamtsystem

Gesamtsystem Use-Case Diagramm (Niklas Hardes)

Im Folgenden ein Use-Case Diagramm als Visualisierung zum Gesamtsystem:

Abbildung 3: Use-Case Diagramm Gesamt System



Quelle: Eigene Darstellung

6 Entwurfsmuster

Nicolas Groß

Bei Anwendungen wie in unserem Fall, einer Single-Page-Application (SPA), werden alle Teile auf einmal geladen, um dem Nutzer ein möglichst flüssiges Erlebnis zu bieten. Dadurch werden häufig auch unbenutzte Module der Anwendung geladen. Bei kleineren Projekten stellt dies keine Probleme da, allerdings kann es bei größeren Projekten zu langen Ladezeiten führen. Um dem entgegenzuwirken, bietet sich Lazy Loading. Beim Lazy Loading werden Inhalte einer Anwendung erst geladen, sobald sie benötigt werden. Dadurch werden Wartezeiten beim Start der Anwendung verkürzt. In Angular haben Module die Möglichkeit via Lazy Loading geladen zu werden. Durch den entsprechenden Code im Routing-Module wird das Lazy Loading auf die Module angewendet.

Quelltext 1: Codeausschnitt des Tabs Routingmodule

```
1 {  
2   path: 'home',  
3   loadChildren: () => import('../tab1/tab1.module')  
4                               .then(m => Tab1PageModule),  
5   data: {  
6     title: 'Home',  
7     icon: 'home'  
8   }  
9 }
```

Wie in diesem Codeausschnitt zu sehen, wird das Modul erst beim Aufruf der Route importiert und geladen. Somit wurde Lazy Loading in Angular-Routing implementiert.⁵

Neben dem Routing bedienen wir uns noch zwei weitere Male den Lazy Loading. Der von Ionic bereitgestellte HTML-Tag „<ion-img>“, welche zum Anzeigen Bildern verwendet wird, lädt Bilder standardmäßig über Lazy Loading. Den deutlichsten unterschied der Ladezeiten weist allerdings die InfinityScroll der „Search“-Seite auf. Bei einer Suche werden die Einträge paketweise der Liste hinzugefügt. Weiter Einträge werden erst der Liste erst hinzugeladen, sobald der Nutzer bis kurz vorm unteren Ende der Liste scrollte.⁶

⁵Arjav Dave, 2021, .vgl.

⁶Drifty Co., 2013b, .vgl.

Fluent-Interfaces (Niklas Harges)

In der Softwareentwicklung sind Fluent-Interfaces ein Entwurfsmuster, das darauf abzielt, den Code lesbarer und verständlicher zu machen, indem es die Verkettung von Methoden ermöglicht. Sie sind besonders nützlich für die Erstellung von komplexen und hierarchischen Objekten.

In unserem Projekt haben wir für das Backend die Sprache C# gewählt. In dieser Sprache werden diese Ketten dadurch ermöglicht, dass Methoden das eigene Objekt zurückgeben, wodurch weitere Methoden direkt hintereinander aufgerufen werden können.

Wir haben dieses Muster unter anderem benutzt, um HTTP Anfragen auf einfache Art konstruieren zu können. Dazu haben wir die Klasse CustomHttpRequest angelegt, welche dann Methoden wie die SetHeader besitzt.

Quelltext 2: Codeausschnitt von CustomHttpRequest (1)

```
1 public CustomHttpRequest SetHeader(string key, string value)
2 {
3     this._headers ??= new Dictionary<string, string>();
4     this._headers[key] = value;
5     return this;
6 }
```

Diese Methode konfiguriert bzw. modifiziert die Header des Requests und gibt sich selber zurück, wodurch weitere Methoden aufgerufen werden können. Ein vollständiger Aufruf könnte dann wie folgt aussehen.

Quelltext 3: Codeausschnitt von CustomHttpRequest (2)

```
1 _requester.BuildRequest(HttpMethod.Get, "https://api.steampowered.
    com/ISteamApps/GetAppList/v2/")
2     .SetHeader("User-Agent", "QuicKey Worker")
3     .SetCookie("testCookie", "testValue")
4     .Request()
```

In dem Konstruktor bzw. der Factory Methode werden die benötigten Attribute wie die Methode und die Uri, welche zwingend benötigt werden direkt übergeben. Diese Methode gibt dann ein Objekt der Klasse CustomHttpRequest zurück. Bei diesem Objekt wird dann mithilfe eines Fluent-Interfaces ein exemplarischer Header und Cookie gesetzt. Am Ende wird dann die Request Methode aufgerufen, wodurch die Kette abgeschlossen, die Anfrage gestellt und das Ergebnis der Anfrage zurückgegeben wird. Dies ist dann kein CustomHttpRequest mehr, sondern eine HttpResponseMessage.

7 Architekturstile

Master-Slave-Architektur / Client-Server-Architektur (Niklas Hardes)

Bei dem Abfragen von Spiele-Händlern ergab sich das Problem, dass diese Limitierungen eingebaut haben, wonach die Anfragen pro IP-Adresse eingeschränkt sind. Dafür hatten wir 2 Lösungen. Die Erste wäre die Geschwindigkeit stark zu reduzieren, was allerdings dennoch schnell zu einem Ban führen würde, da das Abfragen dennoch überwacht wird und ein Sequenzielles Abfragen sehr auffällig ist. Die 2. Option ist es die Anzahl an IP-Adressen zu erhöhen und dann wie sehr viele Nutzer auszusehen.

Diese 2. Option wurde dann gewählt. Dazu baut der Worker eine WebSocket Verbindung mit einem Vermittlungsserver auf. Dieser Server nimmt die Anfragen entgegen, leitet sie an hunderte Slaves weiter, welche diese Anfrage durchführen und dann die Antwort an den Vermittlungsserver zurückgeben. Dieser gibt die Antworten dann über das WebSocket an den Worker zurück, wo dann diese verarbeitet und in die Datenbank eingepflegt werden.

Einerseits stellt dies eine Client-Serververbindung dar zwischen unserem Worker und dem Vermittlungsserver. Andererseits eine Master-Slave-Architektur, da die Anfragen auf viele kleine Slaves verteilt werden. Im Allgemeinen gibt es in dieser einen Master, der die Kontrolle über eine Gruppe von Slaves hat. Der Master ist dafür verantwortlich, die Anfragen der Slaves zu verwalten und sicherzustellen, dass die Slaves die Anforderungen erfüllen. Dies passiert in unserem Fall noch mit einer Zwischenstelle.

Harmonisierung vs. Abstraktion (Niklas Harges)

Harmonisierung und Abstraktion sind zwei verschiedene Konzepte in der Software-Entwicklung.

Harmonisierung bezieht sich auf die Anpassung von Daten oder Prozessen, um sie konsistent und einheitlich zu machen bzw. an Standards anzupassen. Das Ziel der Harmonisierung ist es, Inkonsistenzen oder Unstimmigkeiten in Daten oder Prozessen zu beseitigen, um sicherzustellen, dass sie miteinander kompatibel und leicht zu verarbeiten sind.

Abstraktion hingegen bezieht sich auf die Verringerung der Komplexität von etwas, indem man nur die wesentlichen Eigenschaften behält und die unbedeutenden Details ignoriert. Das Ziel der Abstraktion ist es, ein Problem oder eine Aufgabe in kleinere, einfachere Teile zu zerlegen, um es leichter zu verstehen und zu lösen.

Kurz gesagt, Harmonisierung sorgt dafür, dass Daten und Prozesse miteinander kompatibel sind, während Abstraktion dafür sorgt, dass Probleme und Aufgaben leichter verstanden und gelöst werden können.

Anhang

Anhangsverzeichnis

Quellenverzeichnis

Internetquellen

Anonymus (n.d.). *Mobile App: Native App vs Hybride App, Web App & PWA*. URL: <https://www.brightsolutions.de/blog/native-vs-hybride-vs-web-app/> (besucht am 13. Jan. 2023).

Arjav Dave (2021). *Lazy Loading in Angular – A Beginner’s Guide to NgModules*. URL: <https://www.freecodecamp.org/news/lazy-loading-in-angular-intro-to-ngmodules> (besucht am 13. Jan. 2023).

Drifty Co. (2013a). *Ionic Framework*. URL: <https://ionicframework.com/> (besucht am 13. Jan. 2023).

Drifty Co. (2013b). *Ionic Framework*. URL: <https://ionicframework.com/docs/api/img> (besucht am 13. Jan. 2023).

Google LLC. (2016). *Angular*. URL: <https://angular.io/> (besucht am 13. Jan. 2023).

Schmidauer, Helmut (2002). *Modularisierung*. URL: <https://ssw.jku.at/Teaching/Lectures/Sem/2002/reports/Schmidauer/> (besucht am 13. Jan. 2023).

Ehrenwörtliche Erklärung

Hiermit erklären wir, dass wir die vorliegende Studienarbeit selbständig angefertigt haben. Es wurden nur die in der Arbeit ausdrücklich benannten Quellen und Hilfsmittel benutzt. Wörtlich oder sinngemäß übernommenes Gedankengut haben wir als solches kenntlich gemacht. Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Mettmann, 16. Januar 2023

Niklas Harges

Nicolas Groß

Robert Hesselmann