

Entwicklung einer mobilen Applikation zur Suche und Verwaltung von Brettspielen

Software Engineering

erstellt am 9. Februar 2024

Fakultät Wirtschaft und Gesundheit

Studiengang Wirtschaftsinformatik

Kurs WWI2021F

von

SIMON BURBIEL

LUKAS GROSSERHODE

TIM KEICHER

SIMON SPITZER

Inhaltsverzeichnis

Abkürzungsverzeichnis	IV
Abbildungsverzeichnis	V
Tabellenverzeichnis	VI
1 Einleitung	1
1.1 Motivation	1
1.2 Zielsetzung	1
1.3 Arbeitspakete	2
2 User Interface Konzept	3
2.1 Allgemeines und Funktionalität	3
2.2 HTML-Dokumentation	3
2.2.1 Einführung	3
2.2.2 Allgemeine Beobachtungen	4
2.2.3 Details zu den Seiten	4
2.2.4 Besondere Merkmale	4
2.3 SCSS-Dokumentation	4
2.3.1 Einführung	4
2.3.2 Allgemeine Designentscheidungen	5
2.3.3 Spezifische Designelemente	5
2.3.4 Begründung der Designentscheidungen	5
2.4 TypeScript-Dokumentation	6
2.4.1 Einführung	6
2.4.2 Komponenten und Importe	6
2.4.3 Komponentendefinition	6
2.4.4 Funktionalität	6
2.4.5 Zusätzliche Hinweise	6
2.4.6 Erweiterungen	6
2.4.7 Beispiel	7
2.5 Beschreibung der erstellten Mockups	7
2.5.1 Änderungen und Anpassungen	7
2.5.2 Unveränderte Elemente	8
2.5.3 Fazit	8
3 Technische Umsetzung	10
3.1 Beschreibung der App	10
3.1.1 Funktionen der jeweiligen Seiten	10
3.1.2 Genutzte Interfaces und Modelle	12
3.2 Beschreibung der verwendeten Services	14
3.3 Beschreibung der verwendeten APIs	15
3.3.1 BoardGameGeek XML API	15
3.3.2 MongoDB API	17
3.3.3 Verwendung der APIs im Projekt selbst	17
3.4 Beschreibung der Docker-Umgebung	18

3.4.1	MongoDB-Datenbank	18
3.4.2	Mongo-Express	18
3.4.3	Server	19
3.4.4	Proxy-Server	19
3.4.5	Volumes	19
3.4.6	Zusammenfassung	19
3.5	Beschreibung des Proxys und des externen Routings	19
3.6	Anleitung zum Starten der Applikation	20

Abkürzungsverzeichnis

API Application Programming Interface

BIOS Basic Input/Output System

FAB Floating Action Button

VM Virtuelle Maschinen

Abbildungsverzeichnis

1	Erster Entwurf eines Konzeptes für die Benutzeroberfläche.	3
2	Optimierte Detailseite eines Spiels.	8
3	Mockup-Vergleich (alt vs. neu).	9
4	Ergebnis der ‘/xmlapi/search’-Funktion bei Suchterm Frika.	15
5	Docker-Container nach dem Starten.	21
6	Emulator nach dem Starten der App.	21

Tabellenverzeichnis

1	Zuweisung der Arbeitspakete.	2
2	Funktionen der Homepage.	10
3	Funktionen der Searchpage.	11
4	Funktionen der Infopage.	11
5	Eigenschaften des Boardgame Interfaces.	13
6	Eigenschaften des Game Interfaces.	13
7	Eigenschaften des Randomtopgame Interfaces.	14
8	Funktionen des BoardgameService	14
9	Die zurückgegebenen detaillierten Informationen.	16

1 Einleitung

1.1 Motivation

Die fortschreitende Digitalisierung hat in den letzten Jahren zu einem starken Wandel in der individuellen Freizeitgestaltung geführt. So konnten sich Streaming-Dienste und Videospiele als eine beliebte Form der Freizeitbeschäftigung etablieren. Dennoch erfreuen sich auch Brett-, Karten- und Würfelspiele nach wie vor einer großen Beliebtheit. Gründe hierfür liegen vor allem darin, dass analoge Spiele soziales Miteinander fördern und eine willkommene Abwechslung zu digitalen Medien darstellen. Dabei kann es jedoch schwierig sein, ein passendes Spiel zu finden, welches den eigenen Vorlieben entspricht. Ebenso kann es bei zunehmendem Bestand an Spielen kompliziert werden, den Überblick zu behalten, welche Spiele derzeit in Besitz sind und welche für einen künftige Anschaffung in Frage kommen.

1.2 Zielsetzung

Zur Unterstützung soll daher im Rahmen des Moduls „Software Engineering“, welche als Prüfungsleistung die Entwicklung einer mobilen Applikation vorsieht, eine Ionic-App erstellt werden, welche die Suche und Verwaltung von Brettspielen ermöglicht. Im Kern soll die App über folgende Funktionalitäten verfügen:

- **Suche nach Spielen:** Eine Suchleiste ermöglicht die Suche nach einer Vielzahl von Brett-, Karten- und Würfelspielen.
- **Anzeige detaillierter Informationen zu Spielen:** Hierzu gehören unter anderem Name, Hersteller, Erscheinungsjahr, Anzahl der Spieler sowie eine Beschreibung des Spiels. Diese sollen über ein „Application Programming Interface (API)“ eingespielt werden.
- **Verwaltung von Spielen in einer Wunschliste:** In der Wunschliste können Spiele hinterlegt werden, welche der Nutzer sich zu einem späteren Zeitpunkt kaufen möchte.
- **Verwaltung von Spielen im eigenen Inventar:** Das Inventar bietet eine Übersicht über die Spiele, welche der Nutzer bereits besitzt.
- **Vergabe von Bewertungen zu Spielen:** Dem Nutzer soll es auch ermöglicht werden, eine eigene Bewertung für jedes Spiel abgeben zu können. Diese Funktion soll anhand eines 5-Sterne-Systems realisiert werden.

Der Nutzer soll sich zwischen den Seiten der App frei bewegen können. Die App soll dabei eine übersichtliche und intuitive Benutzeroberfläche bieten, welche die Funktionalitäten der App klar darstellt und eine einfache Bedienung ermöglicht.

1.3 Arbeitspakete

Um eine bessere Organisation des Entwicklungsprozesses zu gewährleisten, wird das Projekt in verschiedene Arbeitspakete unterteilt. Zu diesen zählen:

1. **UI-Design:** Hier soll das Design der App gestaltet werden. Hierunter fallen bspw. die Gestaltung der Farbgebung, Schriftarten und Icons.
2. **API-Integration:** In diesem Paket soll die Anbindung einer externen API realisiert werden, welche die Daten zu den Spielen bereitstellt. Die Formatierung der Daten soll dabei der Struktur der Datenbank entsprechen.
3. **MongoDB-Setup:** In diesem Paket soll die Datenbank für die App eingerichtet werden. Hierunter fallen primär die Erstellung der Datenbank, die Einrichtung der Collections und das Schreiben von Funktionen, welche Abrufe (GET), Einfügungen (POST) und Löschungen (DELETE) von Daten ermöglichen.
4. **Suchfunktions-Implementierung:** Hierbei wird die Suchfunktion für die App konzipiert und implementiert. Die Suchfunktion soll dabei die durch die API bereitgestellten Informationen nach den eingegebenen Suchbegriffen durchsuchen und die passenden Ergebnisse zurückgeben. Selbiges soll auch für die Elemente in der eigenen Datenbank ermöglicht werden.

Um eine annähernd gleichmäßige Verteilung der Arbeitspakete zu gewährleisten, wird jedem Teammitglied ein Arbeitspaket zugewiesen (siehe Tab. 1). Für größere Arbeitspakete ist zudem jeweils ein weiteres Teammitglied zur Unterstützung vorgesehen. In allen anderen Fällen erfolgt die Beihilfe zu den Tätigkeiten der hauptverantwortlichen Teammitglieder abhängig von der aktuellen Auslastung des jeweiligen Unterstützers.

Arbeitspaket	Teammitglied	Unterstützung
UI-Design	Tim Keicher	auslastungsabhängig
API-Integration	Simon Spitzer	Simon Burbiel
MongoDB-Setup	Lukas Großerhode	Simon Spitzer
Suchfunktions-Implementierung	Simon Burbiel	auslastungsabhängig

Tab. 1: Zuweisung der Arbeitspakete.

2 User Interface Konzept

2.1 Allgemeines und Funktionalität

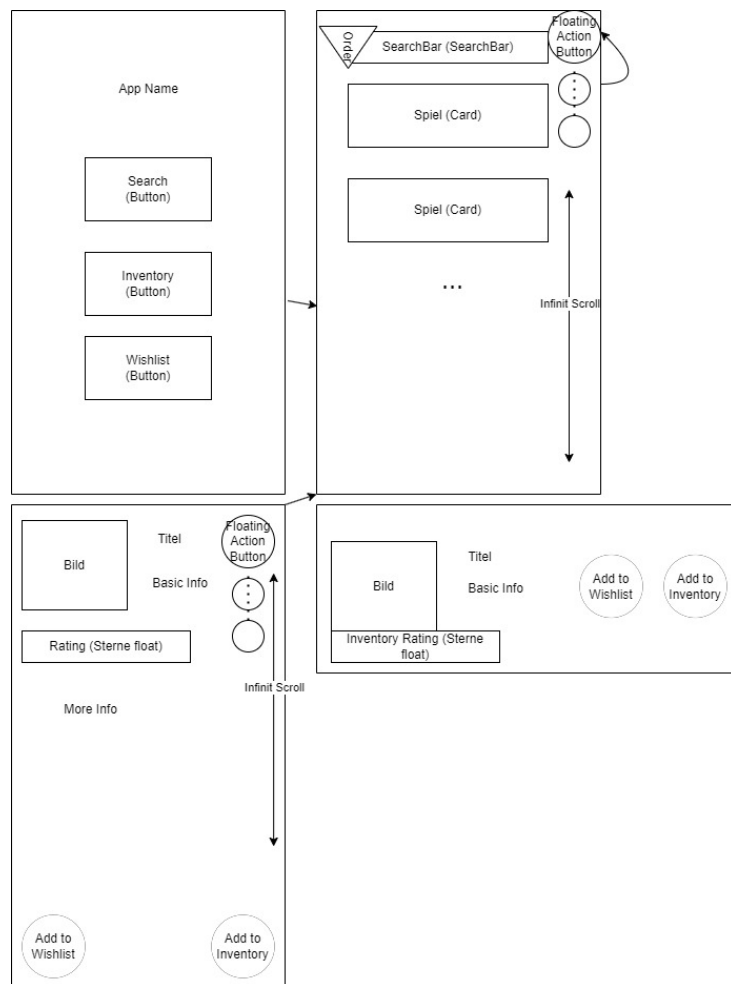


Abb. 1: Erster Entwurf eines Konzeptes für die Benutzeroberfläche.

2.2 HTML-Dokumentation

2.2.1 Einführung

Diese HTML-Dokumentation bietet einen Überblick über die Architektur und die verschiedenen Seiten unseres Projekts, welche mithilfe des Ionic Frameworks entwickelt wurde. Die Anwendung besteht aus einem intuitiven Design, welches auf die Präsentation von Spielen fokussiert ist und verschiedene Funktionen wie Suche, Inventarverwaltung und Wunschliste bietet.

2.2.2 Allgemeine Beobachtungen

Unsere Anwendung nutzt das Ionic Framework und besteht aus einem Header- und einem Content-Bereich. Ein „Floating Action Button (FAB)“ mit Navigationsoptionen befindet sich in der rechten oberen Ecke. Die Inhalte der Seiten werden mithilfe von Ion-Komponenten wie ion-card und ion-searchbar gestaltet.

2.2.3 Details zu den Seiten

1. `Home.page.html` - Die Startseite präsentiert ein zufälliges Spiel des Tages mit einem minimalistischen Design und drei Buttons für die Suche, das Inventar und die Wunschliste.
2. `Info.page.html` - Diese Detailseite zeigt Informationen zu einem einzelnen Spiel an, wie Name, Bild, Bewertung und Beschreibung. Es gibt auch Optionen zum Hinzufügen / Entfernen des Spiels aus der Wunschliste und dem Inventar.
3. `Inventory.page.html` - Hier werden alle Spiele im Inventar des Benutzers aufgelistet, wobei ion-card-Elemente verwendet werden, um jedes Spiel mit Bild und Name anzuzeigen. Das Scrollen lädt weitere Spiele nach.
4. `Search.page.html` - Diese Seite ermöglicht die Suche nach Spielen anhand von Keywords mit dynamischen Vorschlägen und einer Liste der gefundenen Spiele.
5. `Wishlist.page.html` - Ähnlich als die InventorySeite listet diese Seite alle Spiele in der Wunschliste des Benutzers auf, wobei die Struktur und Funktionen identisch sind.

2.2.4 Besondere Merkmale

Die Anwendung bietet dynamische Inhalte, da Spiele auf den Seiten „Inventory“, „Search“ und „Wishlist“ dynamisch aus dem Datenbestand geladen werden. Infinite Scroll wird unterstützt, um nahtloses Laden weiterer Inhalte zu ermöglichen. Zudem passt sich die Anwendung mit responsive Design automatisch an verschiedene Bildschirmgrößen und Geräte an.

2.3 SCSS-Dokumentation

2.3.1 Einführung

Die SCSS-Dokumentation bietet einen umfassenden Überblick über das Design und die Gestaltungsentscheidungen für das vorliegende Projekt. Dabei wird besonderen Wert auf eine elegante Farbpalette, klare Typografie und ein flexibles Layout gelegt, um ein professionelles und modernes Erscheinungsbild zu erzeugen.

2.3.2 Allgemeine Designentscheidungen

Für die Farbpalette wurde sich für dunkle und elegante Töne entschieden, die eine professionelle Atmosphäre schaffen sollen. Akzentfarben wie Gelb und Hellblau bringen Leichtigkeit und Abwechslung ins Design, während eine harmonische Abstimmung der Farben für ein angenehmes Nutzererlebnis sorgt.

Für die Typografie wurde die gut lesbare Schriftart „Nyala“ gewählt und es wurden die Größen sowie der Zeilenabstand entsprechend für verschiedene Bildschirmgrößen angepasst.

Das Layout basiert auf Flexbox für Flexibilität und zentrierte Elemente sowie Media Queries zur Anpassung an verschiedene Bildschirmgrößen.

2.3.3 Spezifische Designelemente

Der Header präsentiert sich mit einem dunkelgrauen Hintergrund und weißem Text, um den Fokus auf den Inhalt zu lenken. Ähnlich minimalistisch ist auch der Content gestaltet, mit einem gut lesbaren Grau und Weiß.

Buttons heben sich durch einen dunkelgrauen Hintergrund und weißen Text ab, wobei unterschiedliche Farben je nach Funktion verwendet werden. Sie zeigen Interaktivität durch leichte Animationen beim Hover-Effekt.

Karten und Icons folgen ähnlichen Designrichtlinien, mit klaren Farben und gut sichtbaren Symbolen, die zur jeweiligen Funktion passen. Hier wurde auch besonders bei den Karten darauf geachtet, welche Informationen für den Nutzer relevant sind und welche Informationen erst bei genauerer Betrachtung des Spiels relevant sind, zu denen bspw. die Spieldauer gehört.

Der FAB setzt auf Kontrast mit einem schwarzen Button und weißem Icon, um Aufmerksamkeit zu erregen.

2.3.4 Begründung der Designentscheidungen

Die Wahl der Farben und Formen dient nicht nur der Ästhetik, sondern auch der Nutzererfahrung wie z.B. das große anzeigen des Spielbildes auf der Infopage für den Nutzer. Durch die Verwendung von dunklen Farben wird eine elegante und professionelle Atmosphäre geschaffen, während Akzentfarben wichtige Elemente hervorheben und für visuelle Abwechslung sorgen. Die Schriftart und das Layout wurden mit Blick auf Lesbarkeit durch das Beispielsweise anzeigen der wichtigsten Spiele Infos als Tabelle und Anpassungsfähigkeit entwickelt, um sicherzustellen, dass die App auf verschiedenen Geräten gut aussieht und einfach zu bedienen ist. Insgesamt wurde der SCSS-Code sorgfältig gestaltet, um ein konsistentes und ansprechendes Design zu gewährleisten, das den Anforderungen an eine moderne und nutzerfreundliche Anwendung gerecht wird.

2.4 TypeScript-Dokumentation

2.4.1 Einführung

Die TypeScript-Dokumentation bietet einen detaillierten Einblick in die Komponenten und Funktionalitäten des auf dem Angular-Framework basierenden Projekts. Es werden die Importe, die Komponentendefinition und die Funktionalität der Hauptkomponente „app-home“ erläutert, welche grundlegende Funktionen für die Startseite der Anwendung bereitstellt. Zusätzlich werden Erweiterungen und Beispielhinweise für eine bessere Verständlichkeit des Codes bereitgestellt.

2.4.2 Komponenten und Importe

Die Importe umfassen verschiedene Module wie `@angular/core` und `@angular/router`, die für die Komponentenerstellung und die Navigation innerhalb der Anwendung benötigt werden. Zusätzlich werden Services und Modelle importiert, um Daten abzurufen und zu strukturieren.

2.4.3 Komponentendefinition

Die „app-home“ Komponente wird durch den Selektor „app-home“ referenziert und besteht aus einer Template-Datei und einem Stylesheet, die das Aussehen und Verhalten der Komponente definieren.

2.4.4 Funktionalität

Im Konstruktor werden erforderliche Services und Router für die Navigation injiziert. Die `ngOnInit()`-Methode wird verwendet, um bei jeder Navigation ein zufälliges Top-Spiel abzurufen und die Daten zu aktualisieren.

2.4.5 Zusätzliche Hinweise

Der Code verwendet das `Randomgame`-Interface, um die Struktur der abgerufenen Spieldaten festzulegen. Die Navigation erfolgt über relative Pfade für eine konsistente und interne Routenführung.

2.4.6 Erweiterungen

Es werden Funktionen wie `goToSearch()`, `goToInventory()` und `goToWishlist()` beschrieben, die zur Navigation zu bestimmten Seiten innerhalb der App dienen. Jede Funktion wird erklärt und ihr Zweck sowie das Ziel der Navigation angegeben.

2.4.7 Beispiel

`goToSearch()`: Navigiert zur „/searchpage“-Route, um die Suche nach Spielen zu ermöglichen.

2.5 Beschreibung der erstellten Mockups

In diesem Kapitel wird ein Blick auf die Entwicklung der App vom anfänglichen Mockup bis zum fertigen Endprodukt geworfen. Es werden die wichtigsten Änderungen und Anpassungen erläutert, die während des Entwicklungsprozesses vorgenommen wurden, während zeitgleich hervorgehoben wird, welche Elemente des Mockups unverändert geblieben sind.

2.5.1 Änderungen und Anpassungen

1. **Detaillierte Spielinformationen:** Die Detailseiten der Spiele sind um zusätzliche Informationen erweitert worden. Videos, Bildschirmfotos, Rezensionen und Spielerbewertungen sind nun ebenfalls einsehbar (siehe Abb. 2).
2. **Verbessertes Design:** Das Design der App ist im Laufe des Entwicklungsprozesses kontinuierlich optimiert worden. Insbesondere die Gestaltung der Benutzeroberfläche und die Farbgebung sind fortlaufenden Verbesserungen unterzogen worden.

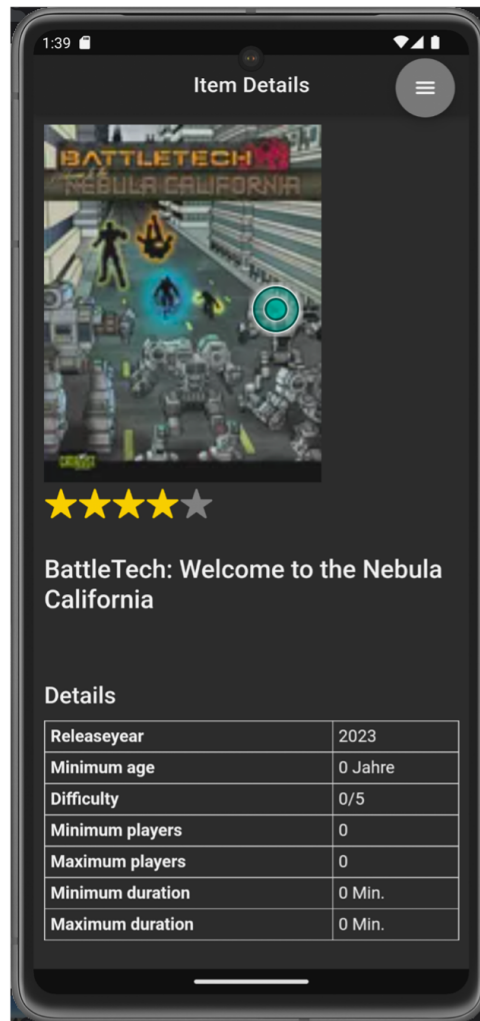


Abb. 2: Optimierte Detailseite eines Spiels.

2.5.2 Unveränderte Elemente

1. **Grundlegende Struktur:** Die zugrundeliegende Struktur, zu welcher die Hauptnavigation und die Anordnung der Hauptkomponenten gehören, ist im Wesentlichen unverändert geblieben.
2. **Kernfunktionen:** Die Kernfunktionen der App, wie die Suche nach Spielen, die Anzeige detaillierter Informationen zu Spielen sowie die Verwaltung von Spielen in Wunschliste und Inventar, haben auch nach Erstellung des Mockups noch Bestand.

2.5.3 Fazit

Die Entwicklung der App vom Mockup bis zum Endprodukt kann als iterativer Prozess angesehen werden, welcher von ständigen Verbesserungen und Anpassungen geprägt war. Hierbei

konnten sich die Kernfunktionen der App als stabil erweisen, während für Design und Benutzeroberfläche kontinuierliche Optimierungspotentiale bestanden (siehe Fig. 3). In dem Zuge sind auch neue Funktionen dem Projekt hinzugefügt worden, auch wenn diese nicht im ursprünglichen Mockup enthalten waren. Ziel war eine kontinuierliche Verbesserung der Benutzererfahrung und Erweiterung des Funktionsumfangs.

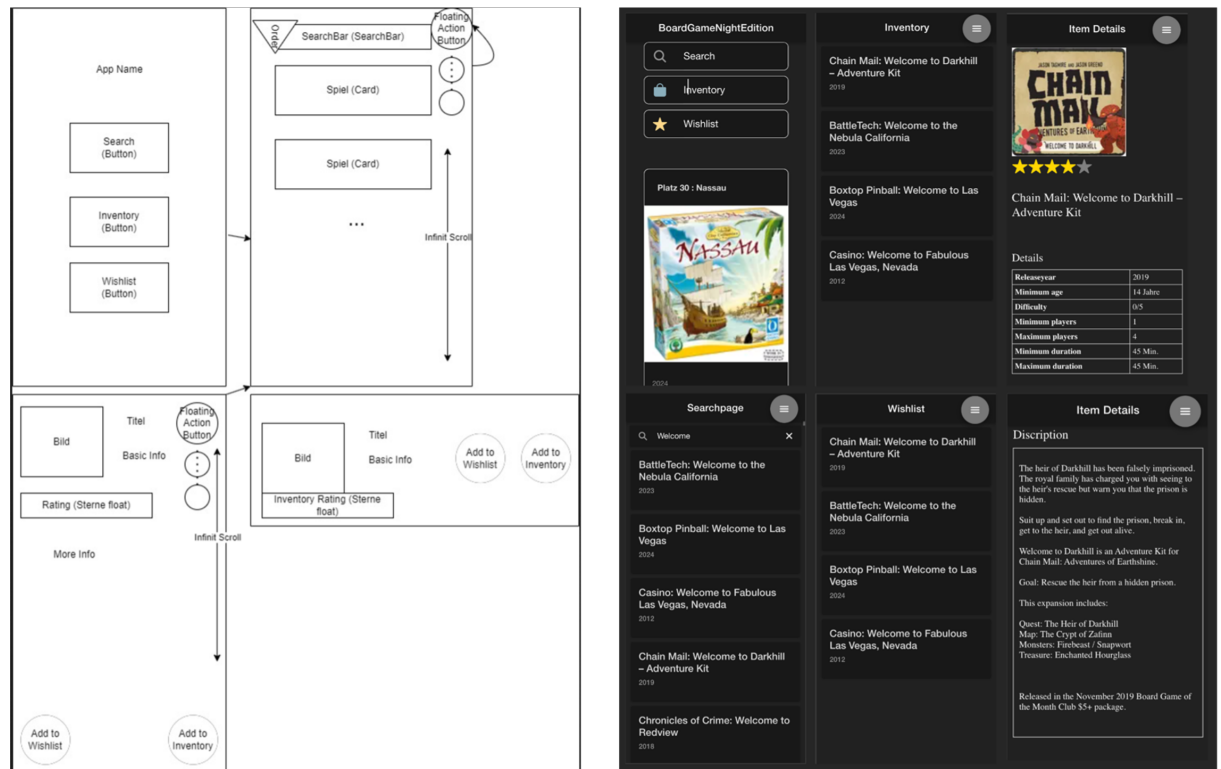


Abb. 3: Mockup-Vergleich (alt vs. neu).

3 Technische Umsetzung

3.1 Beschreibung der App

3.1.1 Funktionen der jeweiligen Seiten

Homepage

Die Homepage der App zeigt ein zufälliges und derzeit besonders beliebtes Spiel an, welches von der BoardGameGeek API bezogen wird. Die Generierung erfolgt automatisch und bei jedem Aufruf der Seite neu. Daher wird dieser Fähigkeit in der `ngOnInit()`-Funktion auf der Homepage aufgerufen. Darüber hinaus dient die Homepage primär als Ausgangspunkt für die Navigation zu den anderen Seiten (siehe Tab. 2).

Funktion	Beschreibung
<code>goToHomepage()</code>	Navigiere zur Homepage.
<code>navigateToHome()</code>	Navigiere zur Homepage.
<code>navigateToInventory()</code>	Navigiere zum Inventar.
<code>navigateToWishlist()</code>	Navigiere zur Wunschliste.
<code>navigateToSearch()</code>	Navigiere zur Suchseite.
<code>navigateToItemDetail()</code>	Navigiere zur Infopage des ausgewählten Spiels.

Tab. 2: Funktionen der Homepage.

Searchpage

Die Searchpage repräsentiert die Suchleiste der App. Hier wird es dem Nutzer ermöglicht, Spiele anhand eines Suchbegriffs zu finden. Die Suchfunktion wird durch die API bereitgestellt und liefert eine Liste von Spielen, welche dem übergebenen Suchbegriff entsprechen. Anhand einer Liste von Spielen kann der Nutzer dann ein Spiel auswählen, um daraufhin zur Infopage weitergeleitet zu werden. Auch hier ist es also wieder möglich, auf verschiedene Seiten zu navigieren. Darüber hinaus existiert eine Funktion, welche unendlich viele Spiele nachlädt, sobald der Nutzer das Ende der Liste erreicht hat (siehe Tab. 3).

Funktion	Beschreibung
<code>search()</code>	Suche nach Spielen anhand eines Suchbegriffs.
<code>onIonInfinite()</code>	Lade unendlich viele Spiele.
<code>goToHomepage()</code>	Navigiere zur Homepage.
<code>goToWishlist()</code>	Navigiere zur Wunschliste.
<code>goToInventory()</code>	Navigiere zum Inventar.
<code>openGameDetails()</code>	Navigiere zur Infopage des ausgewählten Spiels.

Tab. 3: Funktionen der Searchpage.

Infopage

Die Infopage dient der detaillierten Anzeige von Informationen zu einem ausgewählten Spiel. Hierunter fallen bspw. der Name, das Erscheinungsjahr, die minimal erforderliche und maximal mögliche Anzahl an Spielern sowie die durchschnittliche Spieldauer. Genauere Informationen hierüber sind auch den Interfaces zu entnehmen. Darüber hinaus kann der Nutzer das der jeweiligen Infopage zugrundeliegende Spiel zu seiner Wunschliste oder seinem Inventar hinzufügen und es auch aus diesem wieder entfernen. Die Infopage verfügt im Allgemeinen über eine Vielzahl von Funktionen, da sie die zentrale Anlaufstelle für die Verwaltung von Spielen darstellt. So muss hier bspw. geprüft werden, ob ein Spiel bereits in der Wunschliste oder im Inventar enthalten ist, um die entsprechenden Buttons anzuzeigen (siehe Tab. 4).

Funktion	Beschreibung
<code>addToWishlist()</code>	Füge das Spiel zur Wunschliste hinzu.
<code>loadGameDetails()</code>	Lade die detaillierten Informationen des Spiels.
<code>toggleWishlist()</code>	Ändere den Besitzstatus für die Wunschliste.
<code>toggleInventory()</code>	Ändere den Besitzstatus für das Inventar.
<code>addToWishlist()</code>	Füge das Spiel der Wunschliste hinzu.
<code>removeFromWishlist()</code>	Entferne das Spiel aus der Wunschliste.
<code>addToInventory()</code>	Füge das Spiel dem Inventar hinzu.
<code>removeFromInventory()</code>	Entferne das Spiel aus dem Inventar.
<code>navigateToHome()</code>	Navigiere zur Homepage.
<code>navigateToInventory()</code>	Navigiere zum Inventar.
<code>navigateToWishlist()</code>	Navigiere zur Wunschliste.
<code>navigateToSearch()</code>	Navigiere zur Suchseite.
<code>onIonInfinite()</code>	Lade unendlich viele Spiele.

Tab. 4: Funktionen der Infopage.

3.1.2 Genutzte Interfaces und Modelle

In der Welt der Softwareentwicklung ist es entscheidend, dass Daten, die zwischen verschiedenen Systemen ausgetauscht werden, klar definierte Strukturen haben. TypeScript-Interfaces sind hierbei äußerst nützlich, insbesondere wenn es um die Interaktion mit APIs geht. Das Boardgame Interface beschreibt alle wesentlichen Informationen eines Brettspiels, die über die BGG-API abgerufen werden und in einer MongoDB gespeichert werden. Dazu gehören Details wie der eindeutige Identifikator des Spiels, das Veröffentlichungsjahr, die Spieleranzahl, die Spieldauer, das empfohlene Alter, eine Beschreibung und ein Vorschaubild. Dieses Interface stellt sicher, dass die von der API gelieferten Daten vollständig und in einer korrekten Form sind, die direkt in der Datenbank gespeichert werden kann (so dass auch Falls kein Bild in der Api Verfügbar ist dies abgespeichert wird). Das Game Interface hingegen ist eine reduzierte Version des Boardgame Interfaces. Es kann in Situationen eingesetzt werden, in denen nicht alle Informationen eines Brettspiels benötigt werden, wie zum Beispiel in einer Übersichtsliste von Spielen. Durch das Weglassen von Details wie Spielzeit oder Altersangabe wird die übertragene Datenmenge verringert, was die Effizienz verbessern und die Ladezeiten verkürzen kann. Das Randomgame Interface ist spezialisiert auf die Darstellung eines zufällig ausgewählten Spiels aus den Top 50 der am besten bewerteten Brettspiele. Neben der Standardinformation wie Identifikator, Name, Veröffentlichungsjahr und einem Bild des Spiels, speichert dieses Interface zusätzlich den Rang des Spiels, also seine Position in der Rangliste. Diese Information ist besonders wertvoll, um auf einen Blick die Beliebtheit eines Spiels einschätzen zu können, diese wird basierend auf ihrer Position in der Top 50 eingelesen. Diese Interfaces sind essenzielle Bausteine für die zuverlässige Datenarchitektur. Sie sorgen nicht nur für eine korrekte Typisierung und eine klare Vertragsgestaltung zwischen dem Backend (der API) und dem Frontend, sondern tragen auch dazu bei, den Datenfluss übersichtlich und wartbar zu halten. So kann die Software sich an verändernde Anforderungen anpassen, ohne dass es zu einem Durcheinander in der Datenstruktur kommt.

Boardgame

Das **Boardgame** Interface repräsentiert die Struktur zur Speicherung von Informationen über ein Brettspiel. Die Eigenschaften sind wie folgt:

Eigenschaft	Typ	Beschreibung
objectId	string	Dient der eindeutigen Identifikation eines Spiels.
yearPublished	string	Das Jahr, in dem das Spiel veröffentlicht wurde.
minPlayers	string	Die minimal erforderliche Anzahl an Spielern.
maxPlayers	string	Die maximal mögliche Anzahl an Spielern.
playingTime	string	Die durchschnittliche Spieldauer.
minPlayTime	string	Die minimal mögliche Spieldauer.
maxPlayTime	string	Die maximal mögliche Spieldauer.
age	string	Die empfohlene Altersangabe.
description	string	Eine Beschreibung des Spiels.
name	string[]	Eine Liste von Namen, unter denen das Spiel bekannt ist.
publisher	string[]	Eine Liste von Verlagen, die das Spiel veröffentlicht haben.
averageWeight	string	Die durchschnittliche Komplexität des Spiels.
averageRating	string	Die durchschnittliche Bewertung des Spiels.
thumbnail	string	Ein Link zu einem Vorschaubild des Spiels.
usersRated	string	Die Anzahl der Benutzer, die das Spiel bewertet haben.

Tab. 5: Eigenschaften des **Boardgame** Interfaces.

Game

Das **Game** Interface repräsentiert eine reduzierte Version des **Boardgame** Interfaces. Es wird verwendet, um die Datenmenge zu verringern, wenn nicht alle Informationen eines Spiels benötigt werden. Die Eigenschaften sind hierbei wie folgt:

Eigenschaft	Typ	Beschreibung
objectId	string	Dient der eindeutigen Identifikation eines Spiels.
name	string	Gibt den Namen des Spiels an.
yearPublished	string	Das Jahr, in dem das Spiel veröffentlicht wurde.
thumbnail	string	Ein Link zu einem Vorschaubild des Spiels.

Tab. 6: Eigenschaften des **Game** Interfaces.

Hinweis: Es gibt zwei Versionen des **Game** Interfaces, eine mit der Eigenschaft „thumbnail“ und eine ohne. Es ist wichtig sicherzustellen, dass die richtige Version verwendet wird, um der erforderlichen Datenstruktur für die Anwendung zu entsprechen.

Randomtopgame

Das **Randomtopgame** Interface repräsentiert die Struktur zur Speicherung von Informationen über ein zufällig ausgewähltes Spiel aus den Top 50 der am besten bewerteten Brettspiele. Hierfür

werden folgende Eigenschaften definiert:

Eigenschaft	Typ	Beschreibung
id	string	Dient der eindeutigen Identifikation eines Spiels.
name	string	Der Name des Spiels.
yearPublished	string	Das Jahr, in dem das Spiel veröffentlicht wurde.
thumbnail	string	Ein Link zu einem Vorschaubild des Spiels.
rank	string	Die Position des Spiels in der Rangliste.

Tab. 7: Eigenschaften des `Randomtopgame` Interfaces.

3.2 Beschreibung der verwendeten Services

Als primärer Service für die Anwendung dient der `BoardgameService`. Dieser Service ist für die Kommunikation mit der BoardGameGeek API verantwortlich und ermöglicht den Zugriff auf die MongoDB-Datenbank. Der Service stellt viele Funktionen bereit, die überwiegend `Observable`-Objekte zurückgeben und Änderungen, Ergänzungen und Löschungen dort befindlicher Einträge ermöglichen. Im Rahmen dieses Projektes existieren zwei Collections, die von diesem Service verwaltet werden: `Wishlist` und `Inventory`. Die Funktionen des `BoardgameService` sind in Tab. 8 aufgeführt.

Funktion	Beschreibung
<code>getWishlist</code>	GET-Anfrage für die Wunschliste.
<code>addToWishlist</code>	POST-Anfrage für das Hinzufügen zur Wunschliste.
<code>addToInventory</code>	POST-Anfrage für das Hinzufügen zum Inventar.
<code>removeFromWishlist</code>	DELETE-Anfrage für das Entfernen aus der Wunschliste.
<code>removeFromInventory</code>	DELETE-Anfrage für das Entfernen aus dem Inventar.
<code>searchGames</code>	GET-Anfrage für die Suche nach Spielen.
<code>getRandomTopGame</code>	GET-Anfrage für ein zufälliges Top-Spiel.
<code>searchGamesWishlist</code>	Suchfunktion für die Wunschliste.
<code>searchGamesInventory</code>	Suchfunktion für das Inventar.
<code>getDetailInformation</code>	Abrufen detaillierter Informationen zu einem Spiel.

Tab. 8: Funktionen des `BoardgameService`.

Vorab werden die benötigten URLs instanziiert, um Anfragen an die API senden zu können. Darüber hinaus werden zwei String-Arrays für die Wunschliste und das Inventar erstellt, um den aktuellen Stand hinsichtlich des Vorhandenseins von Spielen zu speichern. Die Funktionen `searchGames()`, `getRandomTopGame()` und `getDetailInformation` senden Anfragen an die BoardGameGeek API und konvertieren die erhaltenen Daten von XML in JSON. Die eindeutige Identifikation der Spiele erfolgt dabei über die `objectId` des `Boardgame` Interfaces.

3.3 Beschreibung der verwendeten APIs

3.3.1 BoardGameGeek XML API

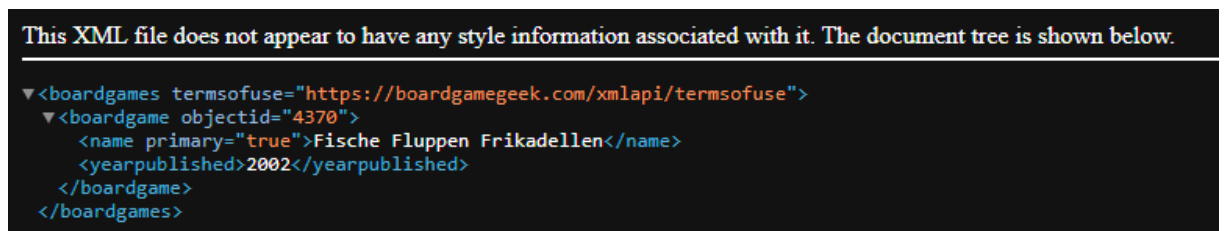
Die BoardGameGeek XML API bietet eine Schnittstelle zum Zugriff auf eine Vielzahl von Informationen rund um Brettspiele, die auf BoardGameGeek.com, einer umfangreichen Datenbank und Community für Brettspiel-Enthusiasten, verfügbar sind. Diese API ermöglicht es Entwicklern durch verschiedene Endpoints auf Spielinformationen, Benutzerkollektionen und Forendiskussionen zuzugreifen. Dabei zu beachten ist, dass die API die Antwort als XML-Format weitergibt. Da häufig mit JSON gearbeitet wird ist eine mögliche Umformung der Daten sinnvoll.

Suchfunktion

Eine der drei benutzten Endpunkte ist die „/xmlapi/search“-Funktion, bei der Nutzer als Input einen spezifischen Suchterm eingeben. Als Ergebnis erhält man eine Liste an Brettspielen in XML Format, deren Namen oder Alias im Suchbegriff enthalten war. Die Suchfunktion-Antwort enthält folgende Informationen:

- `objectId` des Brettspiels
- Name des Brettspiels
- Erscheinungsjahr des Brettspiels

Die reale Ausgabe für den fiktiven Suchterm „Frika“ würde somit Folgendes zurückgeben:

The image shows a screenshot of a web browser displaying an XML response from the BoardGameGeek API. At the top, a message states: "This XML file does not appear to have any style information associated with it. The document tree is shown below." Below this, the XML structure is visible, starting with a root element <boardgames>. Inside, there is a <boardgame> element with attributes <termsfuse="https://boardgamegeek.com/xmlapi/termsfuse"> and <objectid="4370">. The <name> element has a primary attribute set to "true" and contains the text "Fische Fluppen Frikadellen". The <yearpublished> element contains the value "2002". The XML structure is closed with </boardgame> and </boardgames>.

```
<?xml version="1.0" encoding="UTF-8" ?>
<boardgames termsfuse="https://boardgamegeek.com/xmlapi/termsfuse">
  <boardgame objectid="4370">
    <name primary="true">Fische Fluppen Frikadellen</name>
    <yearpublished>2002</yearpublished>
  </boardgame>
</boardgames>
```

Abb. 4: Ergebnis der ‘/xmlapi/search’-Funktion bei Suchterm Frika.

Detaillierte Spielinformationen

Der zweite Endpunkt ist die „/xmlapi/boardgame/<gameid>“-Funktion und dient dazu, detaillierte Informationen zu einem Brettspiel zu erhalten. Nutzer könnten hier noch einige weitere Parameter mitgeben um spezifische Informationen zu erhalten, jedoch ist dies in unserem Projekt nicht notwendig, da die Ergebnisse schon detailliert genug sind. Die `gameId` ist hierbei der Input, ist gleichzusetzen mit der `objectId` und kann aus des oben erhaltenen Ergebnisses der

Suchfunktion entnommen werden. Die detaillierten Spielinformationen enthalten folgende Informationen:

Name	Formattyp
objectId	String
yearPublished	String
minPlayers	String
maxPlayers	String
playingTime	String
minPlayTime	String
maxPlayTime	String
age	String
description	String
name	Array (String)
publisher	Array (String)
averageWeight	String
averageRating	String
thumbnail	String (URL to thumbnail image)
usersRated	String

Tab. 9: Die zurückgegebenen detaillierten Informationen.

Es ist zu erkennen, dass der Name und der Publisher Arrays sind. Dies lässt sich darauf zurückführen, dass es mehrere Namen für dasselbe Spiel gibt (teilweise auch in anderen Sprachen) und mehrere Entwickler beteiligt sein können.

Game of the Day

Der dritte Endpunkt im Bunde ist die 'xmlapi2/hotoverall'-Funktion. Sie gibt die top 50 Spiele mit besonders guter Bewertung und Beliebtheit zurück. Dabei werden genau fünf unterschiedliche Informationen aufgeführt:

- id: String (objectId von vorher earlier)
- rank: String (Ranking Position der Top 50 Spiele)
- thumbnail: String (URL to thumbnail image)
- name: String
- yearpublished

Im Anschluss kann ebenfalls mit der ObjectId eine Anfrage für detaillierte Informationen gestartet werden. Einsatz findet dieses Verfahren wenn mit einem Klick das Spiel des Tages ausgewählt

wird und somit die zurückgegebene ObjectId auf die Infopage weitergeleitet wird. Somit konnte eine weitere Seite wiederverwendet werden und zudem ein neues Feature für die App ermöglicht.

3.3.2 MongoDB API

In diesem Teil wird nur der externe Aufruf auf den selbsterstellten MongoDB API-Endpunkt behandelt. Der interne Aufbau der MongoDB wird in der Beschreibung der App aufgeführt. Es gibt insgesamt drei Zugriffsmethoden für jede Collection auf die MongoDB:

- `searchGamesWishlist()`
- `searchGamesInventory()`
- `addToWishlist(gameData)`
- `addToInventory(gameData)`
- `removeFromWishlist(objectId)`
- `removeFromInventory(objectId)`

Die GET-Methode, bezieht sich auf den HTTP Endpoint `http://localhost:8999/wishlist` bzw. `/inventory` und liefert eine Liste aus JSON-Elementen zurück. Für das Hinzufügen von Brettspielen wird die HTTPClient Methode POST benutzt und die gameData übergeben, wobei Gamedata ein Objekt des Modells Boardgame.ts ist. Dieses Objekt wird dann in eine JSON Form in die ausgewählte Collection hinzugefügt. Bei der Entfernung eines Spiels wird als Parameter die ObjectId des Spiels übergeben. Dieser einzigartige Klassifikator sorgt dafür, dass wenn ein Spiel in der Collection enthalten ist, dass diese ObjectId als Klassifikator verwendet, wird dieses Brettspiel entfernt. Des Weiteren ist dies aus Effizienzgründen vorteilhaft, da kein komplettes Objekt übergeben wird, geparsed werden muss und schlussendlich verglichen. Eine Überprüfung kann direkt stattfinden.

```
1      this.http.delete(`${this.inventoryUrl}/${objectId}`)
```

Listing 3.1: Löschaufruf der MongoDB API

3.3.3 Verwendung der APIs im Projekt selbst

Die Definition der Methoden zum Aufrufen der API-Endpunkte ist in der Datei `board-game.service.ts` des `services`-Ordner zu finden. Dazu wird das HTTPClient Modul importiert und es wird ein `http.get` Aufruf mit den jeweiligen Parametern durchgeführt. Da das Ergebnis in Form einer XML-Datei vorliegt und es vorteilhafter ist mit einem JSON-Format zu arbeiten, wird zu Beginn eine Transformation des Formats in die JSON-Form durchgeführt. Dies geschieht konkret durch das importierte Modul „`xml2json`“.

Die vorliegende JSON-Datei kann nun dem Modell `game.ts`, `boardgame.ts` oder `randomtopgame.ts` zugewiesen werden. Also eine Initialisierung einer Liste von Objekten dieser Modelle basierend auf dem spezifischen API-Aufruf. Im Folgenden wird dann ein Array als

`return`-Element der Funktion definiert. Dies ermöglicht es erstmalig ein subscribe auf die Funktion zu legen und auf der benötigten Seite anzuzeigen.

Die konkrete Verwendung der APIs ist auf den folgenden Pages ist wie folgt:

- Homepage
 - Game of the Day mit `‘xmlapi2/hotoverall‘`-Funktion
- Searchpage
 - Suchfunktion mit `‘/xmlapi/search‘`-Funktion
- Inventory, Wishlist
 - MongoDB GET mit `“http://localhost:8999/inventory/”`-Funktion
- Infopage
 - Detaillierte Spielinformationen mit `‘/xmlapi/boardgame/<gameid>‘`-Funktion
 - MongoDB mit `“http://localhost:8999/inventory?objectId”`-DELETE Funktion
 - MongoDB mit `“http://localhost:8999/inventory?gameData”`-ADD Funktion
 - MongoDB mit `“http://localhost:8999/wishlisht?objectId”`-DELETE Funktion
 - MongoDB mit `“http://localhost:8999/wishlisht?gameData”`-ADD Funktion

3.4 Beschreibung der Docker-Umgebung

In der Docker-Umgebung werden vier Services für die Anwendung rund um die Verwaltung von Brettspielen definiert. Diese Anwendung verwendet Docker, um die Entwicklung und Bereitstellung zu vereinfachen und zu standardisieren. Die Konfiguration umfasst vier Hauptdienste: MongoDB-Datenbank, Mongo-Express für die Datenbankverwaltung, einen Server für die Kommunikation zwischen Frontend und Datenbank sowie einen Proxy-Server für den Zugriff auf die BGG (BoardGameGeek) API, um CORS-Probleme zu vermeiden.

3.4.1 MongoDB-Datenbank

Dieser Dienst verwendet das offizielle MongoDB-Image und ist so konfiguriert, dass er automatisch neu startet, falls es zu einem Fehler kommt. Die Umgebungsvariablen `MONGO_INITDB_ROOT_USERNAME` und `MONGO_INITDB_ROOT_PASSWORD` sind auf `admin` gesetzt, wodurch die Anmeldedaten für den Datenbankzugriff definiert sind. Der Dienst nutzt ein Volume `MONGO_DATA`, um Daten dauerhaft zu speichern und macht den Standard-MongoDB-Port 27017 für andere Dienste zugänglich.

3.4.2 Mongo-Express

Als webbasiertes Verwaltungstool für MongoDB erleichtert Mongo-Express die Verwaltung der Datenbank über einen Webbrowser. Es ist ebenfalls so eingestellt, dass es immer neu startet und verwendet die gleichen Anmeldeinformationen wie die MongoDB. Dieser Dienst hängt von `mongo_boardgame` ab und kommuniziert über den internen Netzwerknamen. Mongo-Express ist über den Port 9001 zugänglich.

3.4.3 Server

Dieser Dienst baut einen Server aus einem Dockerfile, das sich im Verzeichnis `../assignment_gruppe_2_server` befindet. Der Server dient als Schnittstelle zwischen dem Frontend und der MongoDB-Datenbank. Er ist über den Port 8999 erreichbar.

3.4.4 Proxy-Server

Der Proxy-Server ermöglicht den sicheren Zugriff auf externe APIs, in diesem Fall die BGG API, entweder die `xmlapi` oder die `xmlapi2` dieser Webseite (Doku hier abrufbar). Der Container wird aus einem Dockerfile gebaut, welches im Verzeichnis des Proxy-Servers liegt. Der Dienst ist über den Port 8888 erreichbar und so konfiguriert, dass er stets neu startet.

3.4.5 Volumes

Ein definiertes Volume `MONGO_DATA` wird verwendet, um die Datenbankdaten dauerhaft zu speichern und sicherzustellen, dass diese über den Neustart von Containern hinweg erhalten bleiben.

3.4.6 Zusammenfassung

Zusammenfassend stellt dieses Docker Compose-Projekt eine vollständige Umgebung für die Entwicklung und Bereitstellung der Brettspiel-App bereit. Es automatisiert die Konfiguration und Verwaltung von Diensten, die für die Speicherung von Daten, die Verwaltung der Datenbank, die Kommunikation mit dem Frontend und den sicheren Zugriff auf externe APIs erforderlich sind. Die Server sind intern so konfiguriert, dass sie vollen Zugriff zulassen, sodass keine CORS-Probleme auftreten auch durch das Ansprechen auf dem lokalen Gerät durch den Emulator.

3.5 Beschreibung des Proxys und des externen Routings

Der Docker-Container mit dem Proxy ist zuständig für die Kommunikation zwischen der App und der externen API. Dies ist notwendig, da der Server hinter der API eine direkte Anfrage nicht akzeptiert und CORS-Probleme auftreten. Deshalb hilft der Proxyserver, indem dieser seine Herkunft verschleiert und Anfragen von der App an die API weiterleitet. Dabei gibt es drei Suchanfragen, welche die App verwendet:

- Diese Anfrage gibt in XML-Format die 50 bestbewerteten Brettspiele zurück.
- Diese Anfrage kann nach Spielen suchen, indem hinter den `search`-Teil der URL ein oder mehrere Buchstaben bzw. Worte gesetzt werden, wodurch nach einem Spiel gesucht werden kann. Es wird eine XML-Datei zurückgegeben, die alle Brettspiele enthält, auf welche die Beschreibung des Searchstring zutrifft. Je ungenauer die Anfrage gestellt wird, desto länger wird auf die Rückmeldung des Servers gewartet, um alle Spiele darzustellen. Ein Beispiel kann hier geöffnet werden.

- Mit dieser Anfrage können alle für die App wichtigen Informationen eines Brettspiels aufgerufen werden (inklusive der Statistiken und Bewertungen). Für das Aufrufen dieser Seite ist die ID des Objektes von höchster Relevanz, in dem Beispielpfad sieht man die erste 1, welche auf die ID des Brettspiels verweist. Diese wird durch die ID des gesuchten Spiels ersetzt. Diese ID wird genommen von der Auswahl von den durch die Suche gefundenen Daten oder den Daten, die in Wishlist/Inventory gespeichert sind.

Das Routing funktioniert ausgehend von der `boardgame.service.ts` dort wird vom System unterschieden ob momentan von einem Emulator/einem Mobilgerät `http://10.0.2.2:...` oder über den Lokalthost `http://localhost:...` kommuniziert wird. Dadurch werden die gesamten Pfade an das Herkunftssystem angepasst. Eine Anfrage an den Proxy wird dabei mit `/redirect` durchgeführt. Der Proxy hört darauf wenn er angesprochen wird und entfernt das `/redirect` aus dem Suchstring um die Anfrage an die BGG-API weiterzuleiten. Dabei wird der Proxy als Übermittler verwendet und dieser gibt beidseitig die Ergebnisse der Anfragen weiter und ermöglicht dadurch eine indirekte Kommunikation zwischen Applikation und API-Server.

3.6 Anleitung zum Starten der Applikation

Zuerst muss von Git das gesamte Assignment-Gruppe-2 Projekt auf das lokale Gerät gezogen werden. Auf diesem lokalen Gerät werden einige Bedingungen vorausgesetzt:

1. Docker muss bereits existieren.
2. Virtuelle Maschinen (VM) müssen im Basic Input/Output System (BIOS) aktiviert sein.
3. Für das Starten der emulierten App auf einem emulierten Endgerät wird ein Emulator vorausgesetzt.
4. Eine Internetverbindung ist erforderlich.

Wenn all diese Bedingungen erfüllt sind und das Projekt vorliegt, muss eine Kommandozeile/Terminal (mit Adminrechten) gestartet werden.

1. Navigiere auf das Projekt:
`cd <derPfad>/assignment-gruppe-2.`
2. Installiere die relevanten Paketmodule für die Node-Umgebung: `npm install.`
3. Navigiere auf das Docker-Compose um alle notwendigen Container zu starten:
`cd <derPfad>/assignment-gruppe-2/assignment_gruppe_2_mongo_DB.`
4. Führe das Docker-Compose permanent aus um automatisiert die vier Images zu bilden, die für die App benötigt werden: `docker compose up -d`. Dabei werden die Images gebaut und dann in Docker bereits gestartet, dies sollte für die folgenden Schritte im Terminal oder über eine graphische Ansicht der Docker-Container überprüft werden (siehe Fig. 5).
5. Navigiere zurück auf das Gesamtprojekt:
`cd <derPfad>/assignment-gruppe-2.`

6. Möglichkeit 1: Zum Starten von der Applikation im Browser nutze: `ionic serve`. Dies sollte ein Fenster öffnen mit `http://localhost:8100/home`, welches die App anzeigt. Für eine sinnvolle Anzeige lohnt sich das Aktivieren der Mobilgerätesicht.
7. Möglichkeit 2: Zum Starten der App auf einem emulierten Endgerät nutze, öffne ein Emulator und starte ein Gerät und spreche es mit dem folgenden Befehl an, um das Spiel auf diesem Gerät zu starten:
`ionic cordova run android` (siehe Fig. 6).

<input type="checkbox"/>	<div> assignment_gruppe_2_mongo_db</div>	Running (4/4)	N/A	22 hours ago		:		
<input type="checkbox"/>	<div><div> server-1</div><div>81e283d22935 </div></div>	assignment_gruppe_2_mongo_db-server	Running	N/A 8999.8080 	22 hours ago		:	
<input type="checkbox"/>	<div><div> proxy_server-1</div><div>d979830acd17 </div></div>	assignment_gruppe_2_mongo_db-proxy_servi	Running	N/A 8888.8888 	22 hours ago		:	
<input type="checkbox"/>	<div><div> mongo-express_boardgame-1</div><div>4c4a8559bee3 </div></div>	mongo-express:latest	Running	N/A 9001.8081 	22 hours ago		:	
<input type="checkbox"/>	<div><div> mongo_boardgame-1</div><div>ef59a5496318 </div></div>	mongo:latest	Running	N/A 27017.27017 	22 hours ago		:	

Abb. 5: Docker-Container nach dem Starten.

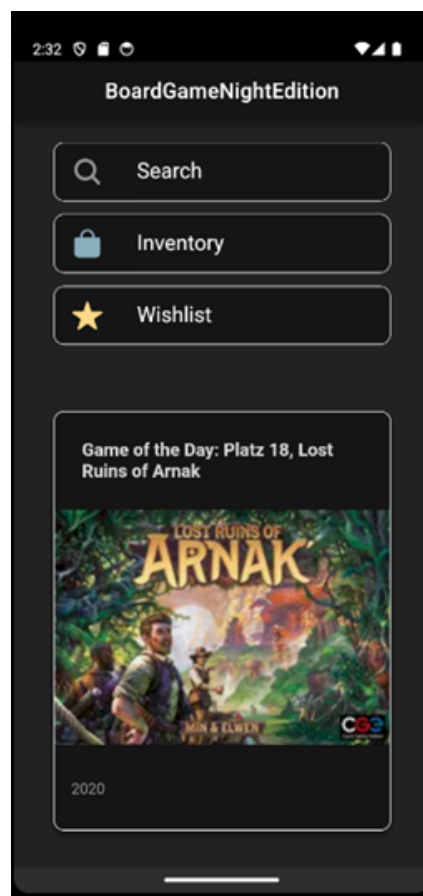


Abb. 6: Emulator nach dem Starten der App.