

## Projekt: Software Engineering

---

<b>Projekt-Titel:</b>	Manga-Melody
<b>Name:</b>	Rahil Chughtai
<b>Matrikelnummer:</b>	IU14099395
<b>Abschluss:</b>	Informatik, M. Sc.
<b>E-Mail:</b>	<a href="mailto:rahil.chughtai@iu-study.org">rahil.chughtai@iu-study.org</a>
<b>GitHub-Link:</b>	<a href="https://github.com/rahilchughtai/manga-melody">https://github.com/rahilchughtai/manga-melody</a>
<b>Website Domain</b>	<a href="https://manga-melody.web.app/">https://manga-melody.web.app/</a>
<b>Kursnummer:</b>	DLMCSPSE01_D
<b>Dozent:</b>	Prof. Dr. Markus Kleffmann

---

# Contents

<b>Projektdokumentation</b>	<b>1</b>
Einleitung . . . . .	1
Projektübersicht . . . . .	1
Hauptfeatures - Kurzüberblick . . . . .	1
Risikomanagement . . . . .	2
Risiko 1: Ausfall der Manga-Schnittstelle: . . . . .	2
Risiko 2: Zeitüberschreitung: . . . . .	2
Risiko 3: Herausforderungen bei der Integration des Logins: . . . . .	2
Risikomatrix . . . . .	2
Zeitplanung . . . . .	3
Gantt-Diagramm . . . . .	4
Benutzeranleitung . . . . .	4
Lokale Ausführung . . . . .	4
Nutzung der Anwendung . . . . .	5
Hinweis zu Browser-Support . . . . .	7
<b>Anforderungsdokument</b>	<b>8</b>
Stakeholder-Analyse . . . . .	8
Ziel- und Benutzergruppen . . . . .	8
Zusammenfassung der Hauptinteressen . . . . .	10
Funktionale Anforderungen . . . . .	10
Manga-Suche . . . . .	10
Favoriten . . . . .	10
Sign Up/Log in . . . . .	10
Kauffunktion . . . . .	11
Use-Case Diagram . . . . .	11
Nichtfunktionale Anforderungen . . . . .	13
Benutzerfreundlichkeit und Design . . . . .	13
Performance und Robustheit . . . . .	13
Responsiveness . . . . .	13
Sicherheit . . . . .	13
Glossar . . . . .	13
<b>Spezifikationsdokument</b>	<b>15</b>
Datenmodell . . . . .	15
Geschäftsprozesse . . . . .	17
Favorisierung . . . . .	17
Login-Prozess . . . . .	17
Bestellungsprozess . . . . .	18
Systemschnittstellen . . . . .	20
Manga-Schnittstelle . . . . .	20
Firebase Datenbank . . . . .	20
Geschäftsregeln . . . . .	20
Nutzer . . . . .	20

Bestellungen . . . . .	20
Preise . . . . .	20
Benutzerschnittstellen . . . . .	20
<b>Architekturdokument</b>	<b>28</b>
Technologieübersicht . . . . .	28
Backend . . . . .	29
Programmiersprachen . . . . .	29
Sonstige Tools, Pattern und Paradigmen . . . . .	29
Architekturübersicht . . . . .	31
Struktur . . . . .	32
Verhalten . . . . .	33
<b>Testdokument</b>	<b>36</b>
Teststrategie . . . . .	36
Unit-Tests . . . . .	37
End-to-End-Tests . . . . .	38
Testprotokoll . . . . .	39
Order Service . . . . .	39
Manga-API Service . . . . .	40
Favorites Service . . . . .	40
Cart Service . . . . .	41
Snackbar Service . . . . .	41
Homepage . . . . .	43
Search-Page . . . . .	45
Favorisierung . . . . .	46
Login/Register . . . . .	51
Cart . . . . .	52
Bestellungsprozess . . . . .	53

# **Projektdokumentation**

## **Einleitung**

Mangas sind eine besondere Form von Comics und Graphic Novels, die ihren Ursprung in Japan haben und durch ihre einzigartigen künstlerischen Stile sowie innovativen Erzähltechniken weltweit an Popularität gewonnen haben. Sie beeinflussen zahlreiche andere Kunstformen und Medienkulturen. Mangas dienen nicht nur der Unterhaltung, sondern bieten häufig tiefgründige Reflexionen über kulturelle, soziale und philosophische Themen. In ihren facettenreichen Geschichten verbinden sie komplexe Charakterentwicklungen mit gesellschaftsrelevanten Fragen und sprechen dadurch ein breites Publikum an – von Kindern bis zu Erwachsenen.

Getrieben durch Faktoren wie der zunehmenden Globalisierung, dem wachsenden Einfluss der Popkultur und technologischen Fortschritten im Bereich des digitalen Publizierens und der Distribution, erreichte der globale Manga-Markt im Jahr 2023 einen Wert von 14,7 Milliarden USD. Prognosen zufolge soll dieser Markt bis 2030 auf 42 Milliarden USD anwachsen ([Manga Global Market Report, 2024](#)).

Ein solches Wachstum an Popularität benötigt Plattformen, die dieser Nachfrage gerecht werden. Das folgende Projekt dient als ein Prototyp für eine Applikation für Manga Fans, die dieser Nachfrage gerecht wird, indem eine Plattform bereitgestellt wird, mit der Nutzer ihre Lieblingsmangas einfach finden und erwerben können.

## **Projektübersicht**

Ziel des Projektes ist die Erstellung einer modernen, E-Commerce basierten Web-Applikation “MangaMelody”, die eine Online-Plattform für das Entdecken, Suchen und Kaufen von Mangas bieten soll. Die Applikation soll öffentlich zugänglich sein und vor allem mit einer intuitiven User Experience, einem modernen Look, und mit einer großen Auswahl an Mangas dem Nutzer in die faszinierende Welt der kunstvollen japanischen Illustrationen einladen.

Es folgt eine knappe Übersicht der Hauptfunktionalitäten der Web-Anwendung, die in dem Anforderungsdokument detailliert beschrieben werden:

### **Hauptfeatures - Kurzüberblick**

- Manga Suche
- Manga Favoriten speichern
- Registrierung/Einloggen
- Kauffunktion mit Warenkorb, Bestellung und Rechnung

## Risikomanagement

Zunächst werden die Risiken ermittelt und erläutert. Anschließend werden die Risiken in einer Risikomatrix mit einer Eintrittswahrscheinlichkeit und einer Schadenshöhe eingeordnet. Zuletzt werden für jedes Risiko Aktionen dokumentiert, mit denen das Risiko mitigiert werden kann.

Die folgenden Risiken wurden ermittelt:

### Risiko 1: Ausfall der Manga-Schnittstelle:

Die Applikation besitzt eine Abhängigkeit zu einer Externen API, die die Manga-Daten liefert. Es besteht ein Risiko, dass diese API temporär oder auch langfristig ausfällt.

### Risiko 2: Zeitüberschreitung:

Es besteht ein Risiko, dass die Zeitplanung nicht eingehalten wird, und es somit zu einer Zeitüberschreitung kommt.

### Risiko 3: Herausforderungen bei der Integration des Logins:

Da Login-Funktionalitäten oft komplex sind und viele Abhängigkeiten haben, besteht ein Risiko, dass die Implementierung des Logins länger dauert als geplant.

## Risikomatrix

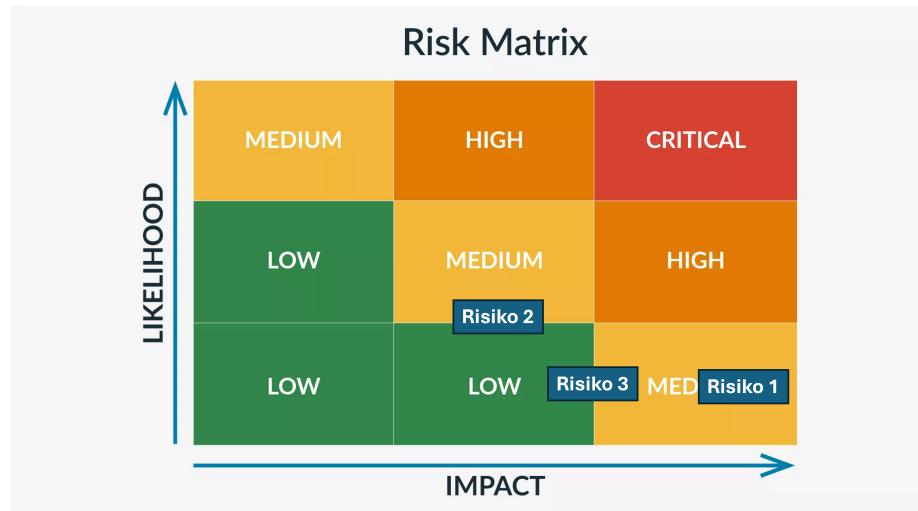


Figure 1: Risikomatrix mit den ermittelten Risiken.

**Risiko 1** wurde als relativ unwahrscheinlich bewertet, da es sich um eine wohl dokumentierte und populäre Schnittstelle handelt. Ein Ausfall ohne Mitigation wäre jedoch schwerwiegend, da ein Großteil der Applikation auf die API angewiesen ist.

**Mitigation:** Bei der Implementierung der Anbindung der Schnittstelle wird ein Mechanismus eingebaut, der im Falle einer Fehler-Response der Schnittstelle eine lokal abgespeicherte Untergruppe der Manga Daten zurückgibt. Somit wäre die Applikation noch eingeschränkt nutzbar, bis der Ausfall vorbei ist. Außerdem soll bei der Implementierung auf Error-Handling geachtet werden, um transparent gegenüber dem Nutzer zu sein.

---

**Risiko 2** wurde als relativ unwahrscheinlich bewertet, da der Scope des Projektes klar definiert und eingeschränkt wurde. Die Auswirkungen wären auch mild, da es keine externe Abhängigkeit oder Deadline gibt.

**Mitigation:** Bei der Zeitplanung wird etwas Puffer berücksichtigt und eingebaut für unerwartete Fehler und Verzögerungen.

---

**Risiko 3** wurde als relativ unwahrscheinlich bewertet, da die Implementation des Logins über externe Anbieter wie Google-Auth gelöst werden können. Falls es zu Verzögerungen kommt, wäre der Schaden ebenfalls mild, da es keine externe Abhängigkeit gibt.

**Mitigation:** Bei der Implementierung des Logins wird auf externe Anbieter wie Google-Auth zurückgegriffen, um die Komplexität zu reduzieren und die Implementierung zu beschleunigen.

## Zeitplanung

Bei der Zeitplanung wurde für jedes Arbeitspaket eine Dauer in Tagen geschätzt. Anschließend wurde ein Gantt-Diagramm erstellt, welches diese Zeitplanung visualisiert.

Arbeitspakete	Dauer in Tagen
Initiales App Set Up	7
Integration der Manga Api	4
Landing-Page	4
Such Funktion	5
Datenbank-Anbindung	4
Login	5
Favoriten	4
Warenkorb	5
Bezahlvorgang	5
Bestellungsübersicht	4

Arbeitspakete	Dauer in Tagen
Testing	5
Puffer für Bug Fixes	7
Puffer für Design und UX	7
Verbesserungen	
App Deployment	5

### Gantt-Diagramm

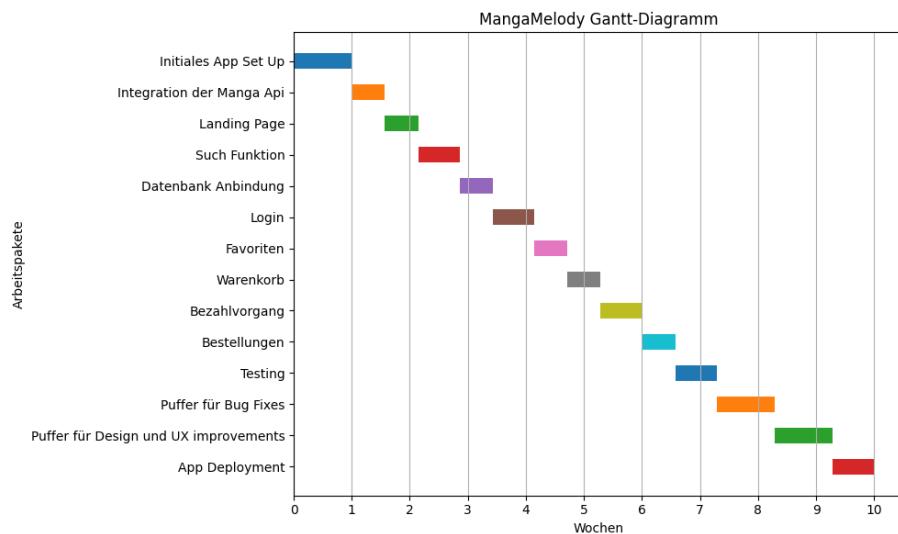


Figure 2: Gantt-Diagramm erstellt mit Python und Matplotlib.

### Benutzeranleitung

Der Quellcode des Projektes kann unter dem folgenden [GitHub](#) Link aufgerufen werden. Da das Projekt privat ist, müssen Sie hierbei zusätzlich die Einladung als Mitwirkende Person annehmen.

Die Anwendung kann entweder lokal ausgeführt werden, oder über die öffentliche Domäne [Manga-Melody.web.app](http://Manga-Melody.web.app) aufgerufen werden.

### Lokale Ausführung

Für die optionale lokale Ausführung der Angular-Anwendung sind einige Vorbereidungen nötig:

- installation von [Git](#) für das Clonen des Projektes
- installation von [Node](#)

- installation der Angular CLI mithilfe von `npm install -g @angular/cli`

Für mehr Infos bietet die offizielle [Angular Dokumentation](#) weitere Hinweise.

Mit diesen Voraussetzungen kann das Projekt mit `git clone geklont` werden. Anschließend kann in das Projekt über `cd manga-melody` navigiert werden. Von hier aus muss nur noch der Befehl `ng serve` ausgeführt werden, um das Projekt lokal zu starten.

## Nutzung der Anwendung

Sobald die Anwendung auf einem der erwähnten Wege aufgerufen wurde, ist die Home-Page zu sehen. Als nicht-eingeloggter Nutzer sind einige Features jedoch nicht verfügbar. Um sich einzuloggen, kann über die Navigationsleiste zur Log-in Seite navigiert werden. Hier kann entweder ein neuer Account erstellt werden, das Google-Login genutzt werden, oder ein von mir bereitgestellter Test-Account verwendet werden. Dieser enthält einen Manga im Warenkorb und zwei Beispiel-Bestellungen.

Test-Account Daten:

- E-Mail: `max@mangamann.com`
- Passwort: `manga123456`

Nach dem Login stehen dem Nutzer sämtliche Features von MangaMelody zur Verfügung. Die Homepage dient hierbei hauptsächlich dazu, den Nutzer zu begrüßen, und diesen zur Nutzung der Seite einzuladen. Auf der Such-Seite stehen dem Nutzer mehr Funktionen zur Verfügung.

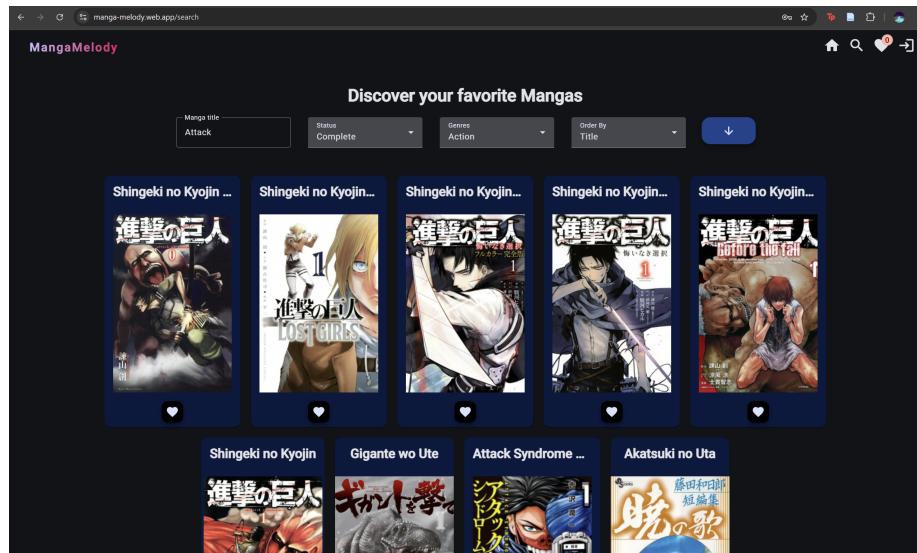


Figure 3: Suchfunktion in Aktion.

Die Suche erlaubt folgende Filtermöglichkeiten:

- Titel: Filtern nach Titel des Manga
- Status: Filtern nach Publizierungsstatus
  - Publishing
  - Complete
  - Hiatus
  - Discontinued
  - Upcoming
- Genres: Filtern nach Genres
- Order-By: Sortierung nach einem bestimmten Attribut
- Pfeil-Button: Sortierungsreihenfolge (Standardmäßig absteigend)

Zudem kann jeder Manga Favorisiert bzw. De-favorisiert werden über den Herz-Button. Standardmäßig wird der Originaltitel angezeigt. Beim Fahren mit der Maus über den Manga wird die englische Übersetzung des Titels angezeigt. Durch das Clicken auf einen Manga wird man zur Manga-Details-Seite weitergeleitet.

Auf dieser kann der Nutzer sich mehr Infos zum Manga anschauen und diesen auch in den Warenkorb legen (sofern der Nutzer eingeloggt ist). Hierbei muss immer der jeweilige Manga Band (Volume) und die Anzahl (maximal 50) angegeben werden.

Sobald der Warenkorb gefüllt ist, kann der Nutzer auf die Warenkorb-Seite navigieren, die Artikel überprüfen und anschließend zum Checkout.

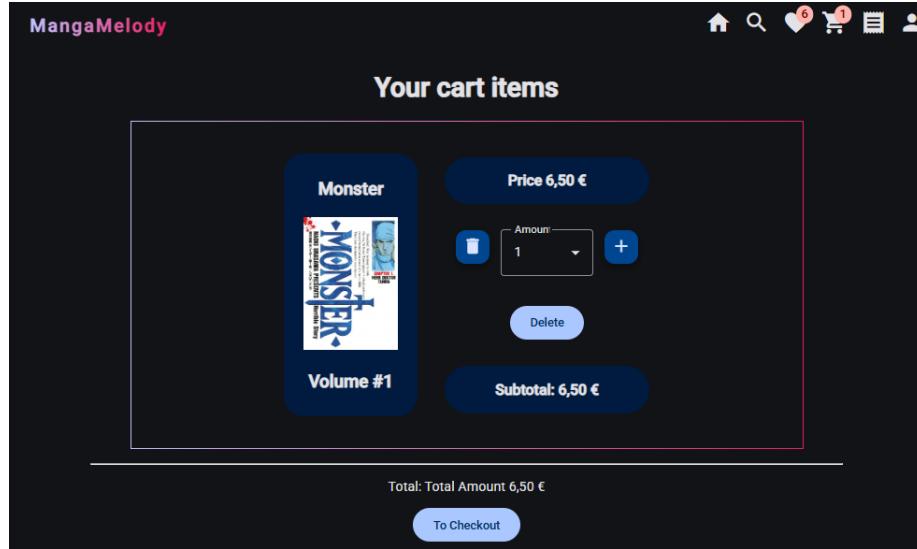


Figure 4: Warenkorb-Seite. Von hier aus kann zum Checkout navigiert werden.

Der Checkout-Prozess besteht aus drei Schritten.

1. Eintragung der Versanddetails
2. Eintragung der Zahlungsinformationen
3. Finale Zusammenfassung und Bestellungsbestätigung

Für die IBAN kann folgende Mock-IBAN genutzt werden

DE 23 10000000 1234567890

Nach dem Abschluss der Bestellung wird der Nutzer auf die Bestellungsseite weitergeleitet, wo alle vorherigen Bestellungen eingesehen können.

### Hinweis zu Browser-Support

Aufgrund der Nutzung des Angular Frameworks ist der Support von Browsern und Versionen eingeschränkt. In der folgenden Tabelle sind die unterstützten Browser-Versionen aufgelistet. Diese Informationen sind von der [offiziellen Angular Dokumentation](#).

Browser	Unterstützte Versionen
Chrome	2 neueste Versionen
Firefox	neueste und erweiterte Support-Version (ESR)
Edge	2 neueste Hauptversionen
Safari	2 neueste Hauptversionen
iOS	2 neueste Hauptversionen
Android	2 neueste Hauptversionen

# Anforderungsdokument

Im Folgenden werden die Anforderungen näher spezifiziert anhand von

- Einer Stakeholder Analyse
- Einer Dokumentation der funktionalen Anforderungen
- Einer Dokumentation der nichtfunktionalen Anforderungen

## Stakeholder-Analyse

### Ziel- und Benutzergruppen

Die geplante Applikation richtet sich an mehrere Stakeholder, die in verschiedene Kategorien eingeteilt werden können. Im Folgenden werden die wichtigsten Zielgruppen und ihre jeweiligen Bedürfnisse sowie Interessen beschrieben:

**Manga-Fans** Die primäre Zielgruppe sind Manga-Enthusiasten, die regelmäßig Mangas konsumieren und nach einer Plattform suchen, die das Auffinden und den Erwerb von Titeln vereinfacht. Diese Gruppe ist breit gefächert und umfasst:

- **Gelegenheitsleser:** Personen, die Manga als gelegentliche Unterhaltung genießen.
- **Hardcore-Fans:** Leidenschaftliche Sammler und regelmäßige Leser, die über Neuerscheinungen, Raritäten und spezial Editionen informiert bleiben möchten.

### Demografische Merkmale

Laut Media Guide, 2019 sind 90 % der Manga-Leser männlich und 10 % weiblich. Die Altersverteilung der Manga-Leser ist gezeigt in der folgenden Grafik.

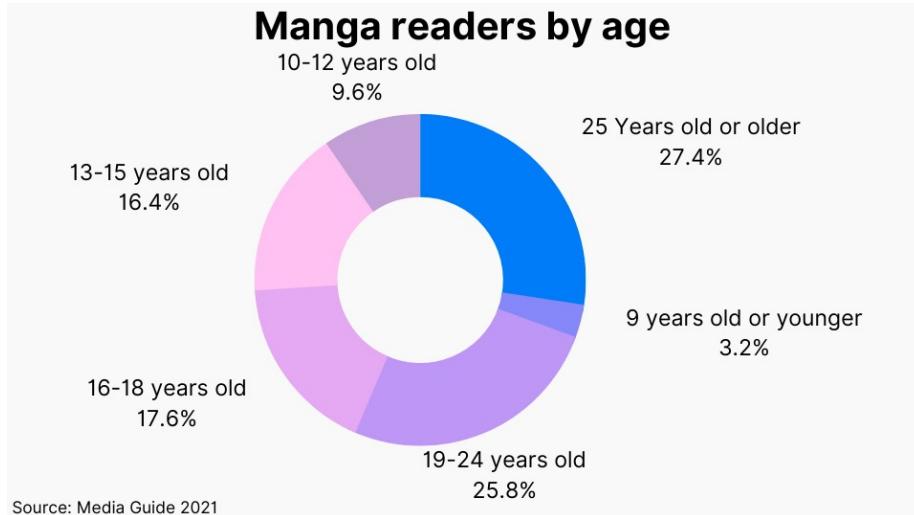


Figure 5: Manga-Leser nach Altersgruppe ([Ferjan, 2024](#)).

Im Folgenden ebenfalls dargestellt als Tabelle ([Ferjan, 2024](#)).

Altersgruppe	Anteil
25 years old or older	27.4 %
19-24 years old	25.8 %
16-18 years old	17.6 %
13-15 years old	16.4 %
10-12 years old	9.6 %
9 years old or younger	3.2 %

Somit ist die Hauptzielgruppe der Applikation männlich und zwischen 19 und 24 Jahren alt.

#### Bedürfnisse und Interessen:

- Schnelle und einfache Such- und Filteroptionen (z. B. nach Genre, Autor, Erscheinungsdatum).
- Bequeme Kaufmöglichkeiten.
- Modernes und zeitgemäßes Design.

**Künstler und Autoren** Manga-Schaffende, darunter Mangaka und Illustratoren, sind indirekte Stakeholder, die von der Verbreitung ihrer Werke profitieren.

#### Bedürfnisse und Interessen:

- Sichtbarkeit ihrer Werke bei einem breiten Publikum.

- Feedback und Interaktion mit den Fans.
- Faire Monetarisierung.

### Zusammenfassung der Hauptinteressen

Stakeholder	Interessen/Bedürfnisse
Manga-Fans	Bequemer Zugang, Personalisierung, Kauf- und Lesemöglichkeiten
Künstler und Autoren	Sichtbarkeit, Fan-Feedback, faire Monetarisierung

Diese Analyse hilft, die Anforderungen aller Stakeholder besser zu verstehen und die Applikation optimal an die Erwartungen anzupassen.

### Funktionale Anforderungen

Im folgenden werden die in der Projektdokumentation genannten Features erläutert und beschrieben in Form von User-Stories mit der Struktur

„Als [Rolle]  
möchte ich [Aktion/Wunsch],  
damit [Nutzen/Ziel]“.

#### Manga-Suche

- Als **Manga-Fan möchte ich** effizient nach Mangas suchen können, **damit** ich schnell die gewünschten Titel finde.
- Als **Manga-Fan möchte ich** Filteroptionen nutzen können, **damit** ich die Suchergebnisse nach meinen Vorlieben eingrenzen kann.
- Als **Manga-Fan möchte ich** eine Pagination-Funktion haben, **damit** ich bequem durch die Suchergebnisse blättern kann.

#### Favoriten

- Als **Manga-Fan möchte ich** meine Lieblingsmangas als Favoriten speichern können, **damit** ich sie schnell wiederfinden kann.

#### Sign Up/Log in

- Als **neuer Benutzer möchte ich** mich mit meiner E-Mail oder meinem Google-Account anmelden können, **damit** ich ein Nutzerprofil erstellen kann.
- Als **Benutzer möchte ich** ein Nutzerprofil erstellen können, **damit** meine persönlichen Daten und Präferenzen gespeichert werden.

- Als **Benutzer möchte ich**, dass meine Nutzerdaten persistent gespeichert werden, **damit** ich bei zukünftigen Anmeldungen darauf zugreifen kann.
- Als **Benutzer möchte ich** meine Nutzerdaten ändern können, **damit** ich meine Informationen aktuell halten kann.

### Kauffunktion

- Als **Manga-Fan möchte ich** einen (simulierten) Kaufprozess durchlaufen können, **damit** Mangas kaufen kann.
- Als **Manga-Fan möchte ich** eine Warenkorb-Funktion nutzen können, **damit** ich mehrere Mangas gleichzeitig kaufen kann.
- Als **Manga-Fan möchte ich** eine Rechnung für meine Käufe erhalten, **damit** ich eine Übersicht über meine Ausgaben habe.
- Als **Manga-Fan möchte ich**, dass die Rechnungen in meinem Nutzerprofil gespeichert werden, **damit** ich sie jederzeit einsehen kann.

### Use-Case Diagram

Die nachfolgende Grafik zeigt die Funktionalitäten der Applikation in einem Use-Case Diagramm:

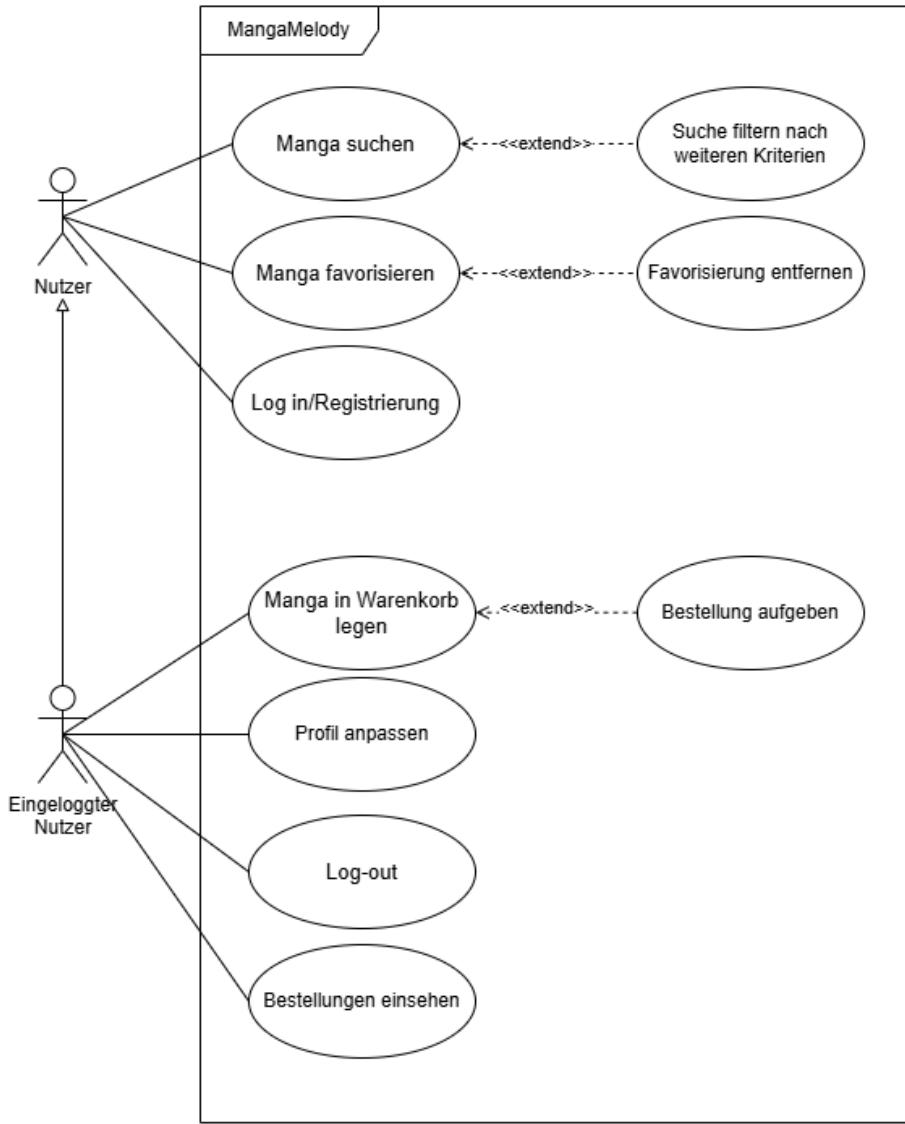


Figure 6: Use-Case Diagram der Funktionalitäten.

Bei diesem wird ein Nutzer wird unterteilt in allgemeine Nutzer, und Nutzer, die eingeloggt sind

## Nichtfunktionale Anforderungen

### Benutzerfreundlichkeit und Design

- Die Anwendung sollte eine intuitive Benutzeroberfläche haben, die es den Benutzern ermöglicht, die Funktionen einfach zu finden und zu nutzen.
- Die Anwendung wird sich hierbei nach dem Material Design von Google richten, um eine konsistente und moderne Benutzererfahrung zu bieten.

### Performance und Robustheit

- Die Anwendung sollte schnell und zuverlässig sein, um eine reibungslose Benutzererfahrung zu gewährleisten.
- Die Ladezeiten sollten bei stabiler Verbindung unter 3 Sekunden sein, um die Benutzer nicht zu verärgern.
- Im Falle von einer Unterbrechung der Verbindung sollte die Anwendung eine Fehlermeldung anzeigen und den Benutzer über die Situation informieren. Zusätzlich sollte die Anwendung dem Benutzer eine eingeschränkte Anzahl an Manga-Titeln anzeigen, die lokal gespeichert sind.

### Responsiveness

- Die Anwendung sollte auch auf mobilen Geräten mit kleineren Bildschirmgrößen gut funktionieren und eine reibungslose Benutzererfahrung bieten.
- Die hierbei minimal unterstützte Bildschirmbreite sollte 360 Pixel betragen.
- Alle relevanten Informationen und Features sollten auch auf kleineren Bildschirmen gut lesbar und bedienbar sein.

### Sicherheit

- Die Anwendung sollte die Nutzerdaten sicher speichern und übertragen, um die Privatsphäre der Benutzer zu schützen.
- Nutzer sollten nur in der Lage sein, die eigenen Profilinformationen und Bestellungen einzusehen.

## Glossar

Begriff	Beschreibung
API	Application Programming Interface, eine Schnittstelle zur Kommunikation zwischen Softwareanwendungen.
Jikan-API	Eine RESTful API, die Daten von MyAnimeList bereitstellt, um Informationen über Manga und Anime abzurufen.
Manga	Japanische Comics oder Graphic Novels, die in einem spezifischen Stil gezeichnet sind.
REST	Representational State Transfer, ein Architekturstil für verteilte Systeme, insbesondere für Webservices.

Begriff	Beschreibung
UI	User Interface, die Schnittstelle, über die Benutzer mit einer Anwendung interagieren.
UX	User Experience, das Gesamterlebnis eines Benutzers bei der Interaktion mit einer Anwendung.

# Spezifikationsdokument

## Datenmodell

Bei der Modellierung der Daten konzentrieren wir uns zunächst auf die Hauptentitäten. Hierzu gehören

- **Kunde:** Der Kunde ist der Hauptakteur der Anwendung. Er kann sich einloggen, Bestellungen tätigen, Favoriten speichern und sein Profil verwalten.
- **Manga:** Die Mangas sind die Hauptprodukte der Anwendung. Sie enthalten Informationen wie Titel, Autor, Genre, Preis etc.
- **Bestellung:** Die Bestellungen enthalten Informationen über die gekauften Mangas, den Gesamtpreis, das Bestelldatum und die Lieferadresse.
- **Warenkorb:** Der Warenkorb enthält die ausgewählten Mangas, die der Kunde kaufen möchte.
- **Favoriten:** Die Favoriten sind eine Liste von Mangas, die der Kunde als besonders interessant markiert hat.

Das folgende UML-Klassendiagramm zeigt die Beziehungen zwischen den Entitäten und ihre Attribute.

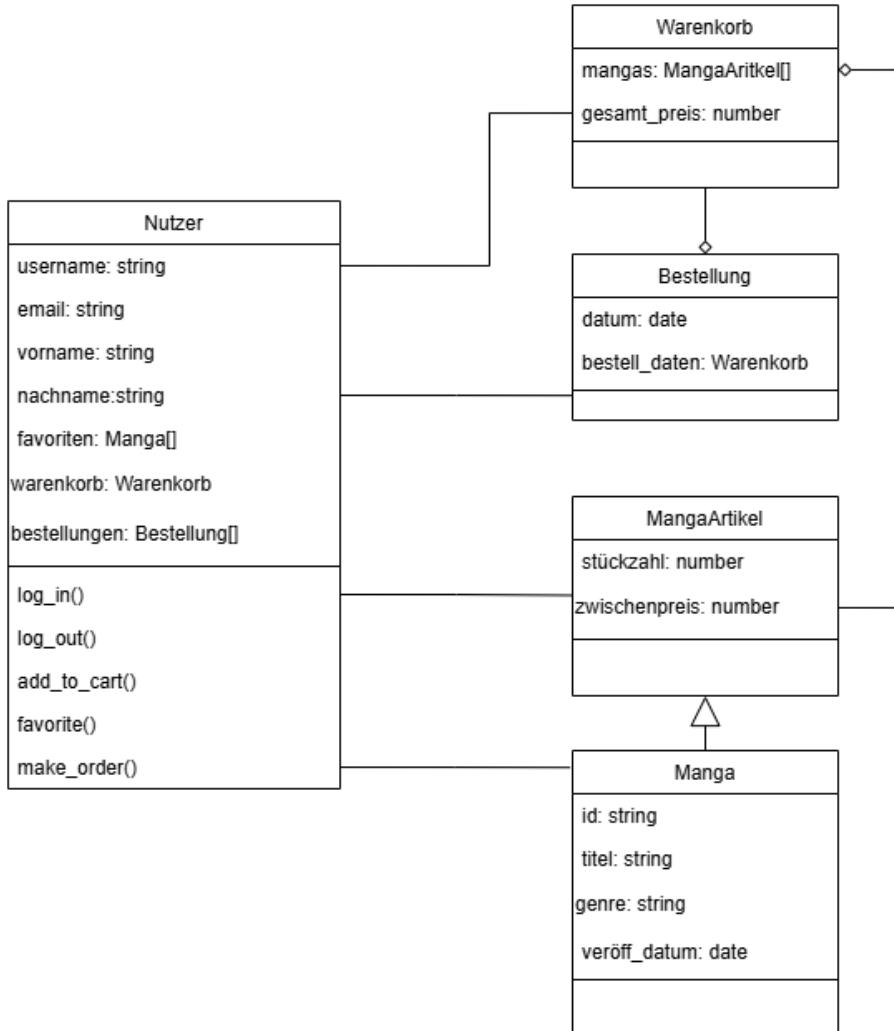


Figure 7: UML-Klassendiagramm der wichtigsten Entitäten.

Zu sehen ist eine Modellierung, die die Hauptentitäten und ihre Beziehungen zueinander darstellt. Einige zusätzliche Entitäten wurden zur Modellierung der Funktionalitäten erstellt. Im Folgenden wird die Modellierung der Hauptentitäten genauer erläutert. Hierbei wird vor allem auf die Beziehungen zwischen den Entitäten eingegangen, die Attribute und Methoden sind weniger relevant.

Ein Nutzer kann Lieblingsmangas haben. Diese werden als Liste gespeichert. Somit besteht eine Aggregation zwischen Nutzer und Mangas. Das Gleiche gilt für Bestellungen. Zudem hat der Nutzer einen Warenkorb.

Eine Bestellung besteht aus einem Datum, und den Bestelldaten, die hier als

eine Aggregation zu Warenkorb modelliert wurden.

Ein Warenkorb besteht aus einer Liste der Manga-Items und einem Gesamtpreis. Hierfür wurde ebenfalls eine Aggregation gewählt. In einigen Modellierungen hätte hierfür auch eine Komposition gewählt werden können, was ausdrücken würde, dass der Warenkorb ohne Artikel nicht existieren kann und somit eine Existenzabhängigkeit besteht. In diesem Fall wurde jedoch eine Aggregation gewählt, da ein leerer Warenkorb als valide angesehen wird. Hierbei wäre die Liste leer und der Gesamtpreis 0.

Ein Manga-Artikel ist eine Generalisierung von Manga. Die **MangaArtikel** Entität ergänzt die Manga-Klasse um die zusätzlichen Informationen der Stückzahl und des Zwischenpreises. Diese Entität wird benötigt, um den Gesamtpreis eines Artikels im Warenkorb zu berechnen.

## Geschäftsprozesse

Im Folgenden werden die wesentlichen Geschäftsprozesse der Anwendung erläutert.

### Favorisierung

Der Nutzer kann auch, ohne eingeloggt zu sein, bereits Favoriten einspeichern. Ist der Nutzer nicht eingeloggt, werden die Favoriten in dem Speicher des Browsers (**localStorage**) persistiert. Das hat den Vorteil, dass die Favoriten auch nach dem Schließen und Öffnen des Fensters oder nach einem Refresh weiterhin erhalten bleiben. Die Einschränkung hierbei ist jedoch, dass die Daten nicht über mehrere Geräte hinweg gespeichert werden, sondern nur in dem spezifischen Browser.

Ist der User jedoch eingeloggt, werden diese Daten in der Datenbank gespeichert, sodass auf die Favoriten von überall zugegriffen werden können.

Als Pseudocode lässt sich dieser Prozess wie folgt beschreiben:

```
IF User.isLoggedIn  
THEN Database.save(User.favorites)  
ELSE LocalStorage.save(User.favorites)
```

### Login-Prozess

Um alle Funktionen der Applikation zu nutzen, ist eine Anmeldung des Nutzers notwendig. Hierfür soll die Applikation die Möglichkeit bereitstellen, sich entweder klassisch über E-Mail und Passwort anzumelden, oder ein Google-Login zu nutzen.

## **Bestellungsprozess**

Um einen Bestellungsprozess erfolgreich abzuschließen, muss der Nutzer die folgenden Schritte absolvieren. Dieser Prozess wird in dem folgenden UML-Aktivitätsdiagramm visuell dargestellt. Eine Besonderheit hierbei ist, dass der Login Prozess nicht explizit dargestellt wird, da dieser bereits zuvor beschrieben wurde. Innerhalb des Diagramms wird dieser Prozess also als eine Art Blackbox betrachtet.

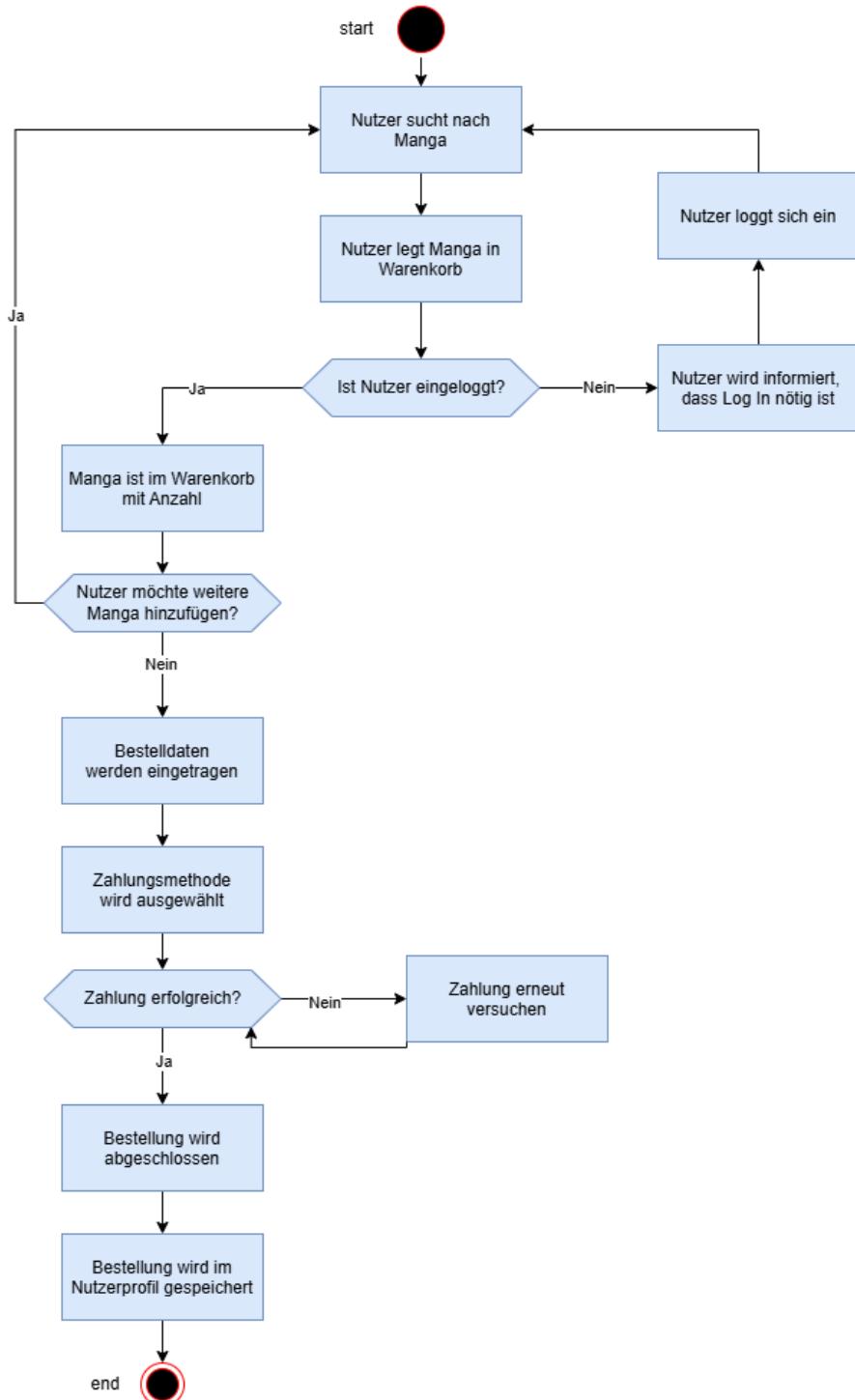


Figure 8: UML-Aktivitätsdiagramm des Bestellungsprozesses.  
19

## Systemschnittstellen

### Manga-Schnittstelle

Die Hauptschnittstelle wird die [Jikan API](#) sein. Mit dieser wird die Applikation direkt über HTTP Requests kommunizieren. Das Datenformat für die Kommunikation ist JSON.

### Firebase Datenbank

Für die Persistenz bestimmter Daten wie Nutzernamen, Bestellungen etc. und auch für Funktionalitäten für das Login verwendet die Applikation die Echtzeit-Datenbank [Firebase](#) von Google. Das Kommunikationsprotokoll hierfür ist WebSocket Communication.

## Geschäftsregeln

### Nutzer

- Nutzer können nur ihre eigenen Daten und Bestellungen einsehen
- Nur eingeloggte User können Artikel in den Warenkorb legen
- Nur eingeloggte User können Bestellungen tätigen

### Bestellungen

- Die maximale Bestellmenge pro Manga ist 50 Stück
- Das Bestelldatum darf nicht in der Zukunft liegen, sondern muss dem aktuellen Tag entsprechen
- Die Anzahl der Waren im Warenkorb darf nicht unter 0 sein
- Jede getätigte Bestellung muss gespeichert werden und für den Nutzer im Nachhinein verfügbar sein

### Preise

- Der Preis (sowohl pro Stück als auch gesamt) darf nie negativ sein
- Der angezeigte darf nur maximal zwei Nachkommastellen haben (keine Darstellungsfehler durch Rundungsfehler)

## Benutzerschnittstellen

Im Folgenden werden Skizzen der Benutzerschnittstellen gezeigt. Diese sind nur grobe Entwürfe und dienen dazu, die Struktur der GUI zu visualisieren.

Bei dem Betreten der Seite wird der Nutzer auf die Landing-Page geleitet. Als "Hero" Komponente bezeichnet man eine große, auffällige Komponente, die bei dem Nutzer Interesse wecken soll. Von hier aus kann der Nutzer über eine Navigationsleiste auf die verschiedenen Seiten der Applikation navigieren.

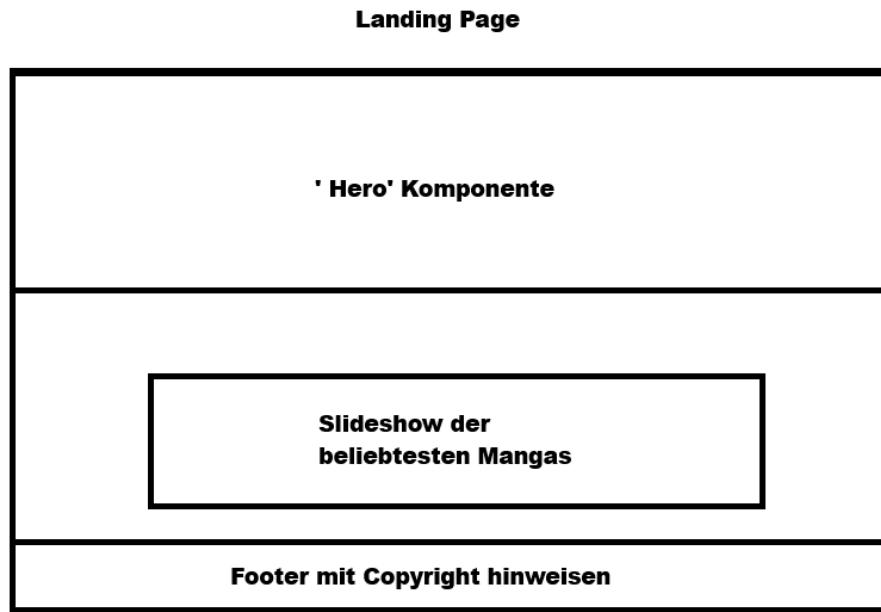


Figure 9: Skizze der GUI der Landing-Page.

Auf der Login/ Sign-up Seite kann der Nutzer sich entweder einloggen oder ein neues Konto erstellen. Hierbei wird unterschieden, ob der Nutzer bereits ein Konto hat oder nicht. Die GUI ist in zwei Teile aufgeteilt, die sich durch Tabs wechseln lassen.

Hierbei soll es eine Eingabevervalidierung geben, die sicherstellt, dass die E-Mail-Adresse gültig ist und das Passwort bestimmten Sicherheitsanforderungen entspricht. Zudem muss bei der Registrierung das wiederholte Passwort mit dem ersten Passwort übereinstimmen. Auch die Adressdaten müssen validiert werden.

**Log in/Sign up Page im Log in Tab**

This sketch shows a user interface for logging in. At the top, there are two tabs: "Log in" (underlined in blue) and "Sign up". Below the tabs are two input fields: "E-Mail" and "Passwort". Underneath these fields is a link "password vergessen? oder". At the bottom is a button labeled "G Login with Google".

**Log in/ Sign up page im Sign up Tab**

This sketch shows a user interface for signing up. At the top, there are two tabs: "Log in" and "Sign up" (underlined in blue). Below the tabs are five input fields: "Nutzername", "E-Mail", "Passwort", "Passwort wiederholen", and "Adresse PLZ etc.". Underneath these fields is a link "oder". At the bottom is a button labeled "G Sign up with Google".

Figure 10: Skizze der GUI der Login/ Sign-up Ansicht.

Die Profil-Ansicht zeigt dem Nutzer seine persönlichen Daten, wie z.B. Name, E-Mail, Adresse etc.

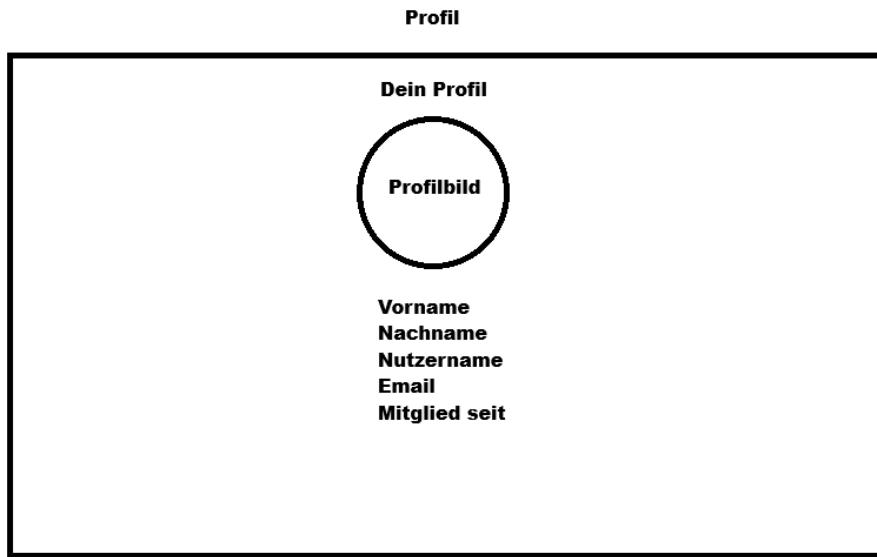


Figure 11: Skizze der GUI der Profil-Ansicht.

Die Manga-Suche ermöglicht es dem Nutzer, nach Mangas zu suchen. Hierbei kann der Nutzer nach verschiedenen Kriterien filtern, wie z.B. Genre, Autor, Preis etc. Die Manga Suche ist das Hauptfeature der Applikation und sollte daher einfach und intuitiv zu bedienen sein. Durch das anklicken eines Mangas wird der Nutzer auf die Detailansicht des Mangas weitergeleitet. Es soll dem Nutzer auch möglich sein, Mangas zu favorisieren und in den Warenkorb zu legen.

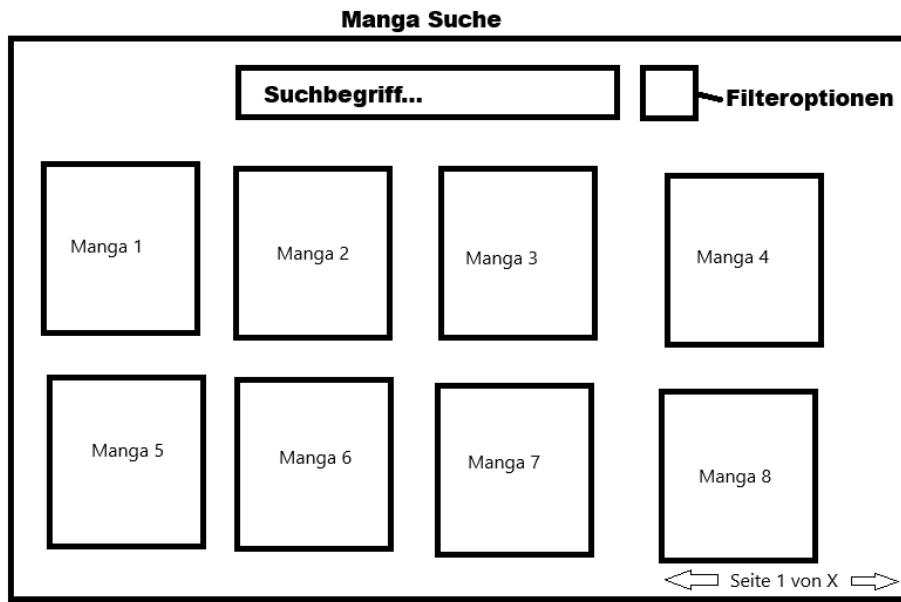


Figure 12: Skizze der GUI der Manga Suche.

Die Favoriten-Ansicht zeigt dem Nutzer eine Liste seiner favorisierten Mangas. Hierbei kann der Nutzer die Mangas auch aus der Liste entfernen.

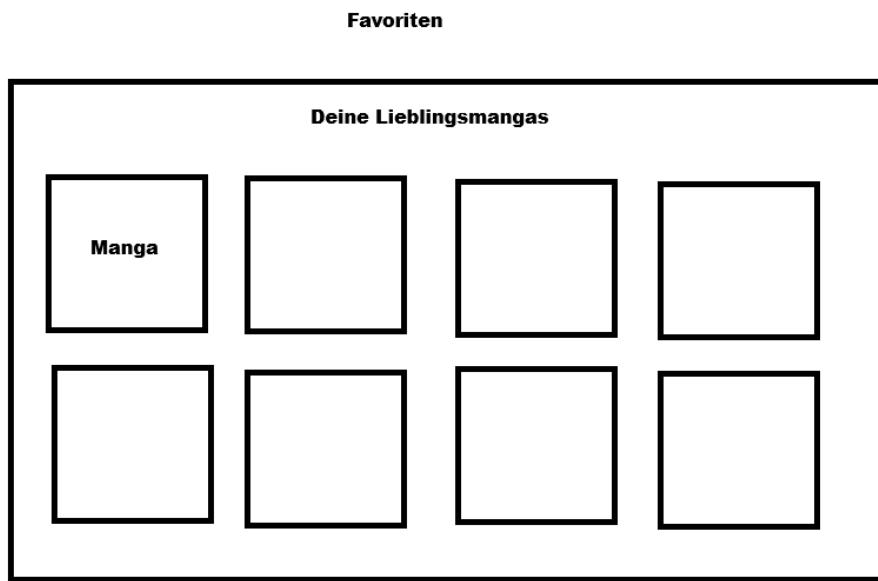


Figure 13: Skizze der GUI der Favoriten.

Die Warenkorb-Ansicht zeigt dem Nutzer eine Liste der Mangas, die er kaufen möchte. Hierbei kann der Nutzer die Anzahl der Mangas ändern oder den Manga aus dem Warenkorb entfernen. Der Gesamtpreis wird automatisch berechnet.

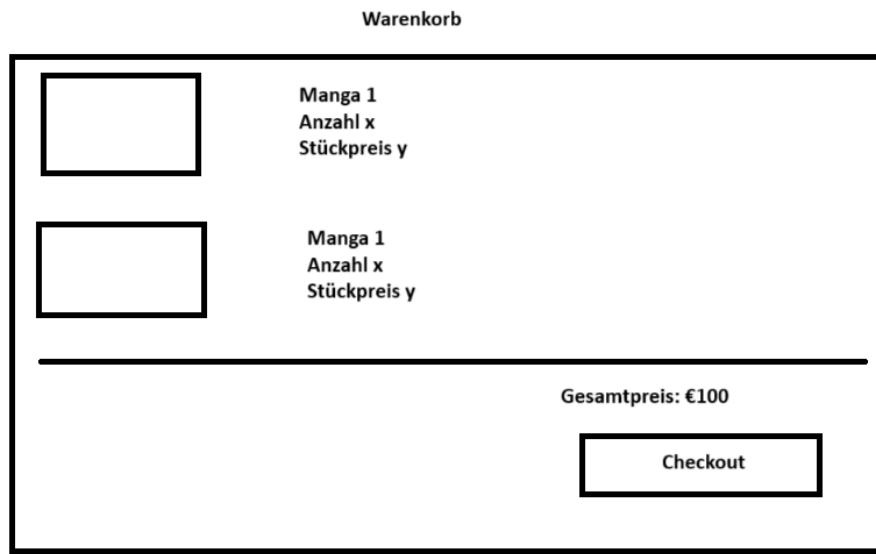


Figure 14: Skizze der GUI des Warenkorbs.

Die Bestellungen-Ansicht zeigt dem Nutzer eine Übersicht seiner bereits getätigten Bestellungen.

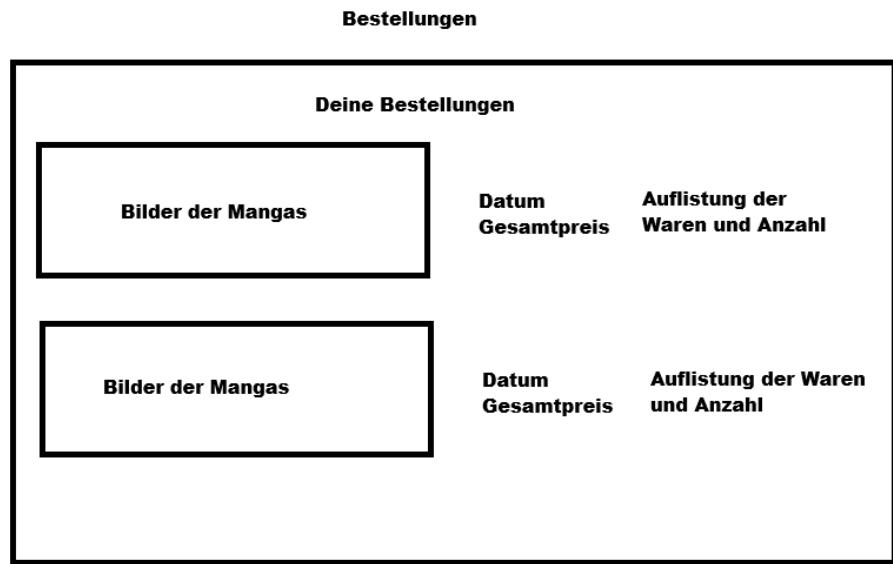


Figure 15: Skizze der GUI der Bestellungen-Ansicht.

## Architekturdokument

In dieser Projektphase werden die Architektur und technische Details dokumentiert sowie die Anwendung durch Schreiben des Quellcodes implementiert. Zunächst soll hierzu ein Architekturdokument erstellt werden.

### Technologieübersicht



Figure 16: Offizielles Angular Logo.

**Angular** ist ein Open-Source-Framework zur Erstellung von Webapplikationen. Es wurde von Google in Zusammenarbeit mit der Open-Source-Community entwickelt und gewartet. Wie viele moderne UI-Bibliotheken setzt Angular auf eine *komponentenbasierte Architektur*. Diese erlaubt es, komplexe Anwendungen nach dem Baukastenprinzip zu entwickeln, indem einzelne, modular aufgebaute Komponenten gezielt zusammengesetzt und in die Anwendung integriert werden. Dadurch erhöht sich die Wiederverwendbarkeit und Wartbarkeit des Quellcodes erheblich. Zudem erleichtert diese Architektur die Skalierung eines Projekts. Neue Features können durch das Erstellen neuer oder das Erweitern bestehender Komponenten hinzugefügt werden, ohne dass umfangreiche Änderungen an anderen Teilen des Systems notwendig sind.

Eine Angular-Komponente besteht typischerweise aus einem Template (.html Datei), einem Stylesheet (.scss Datei), und der Logik, die in einer TypeScript (.ts) Datei verwaltet wird. Dadurch gibt es eine saubere Trennung zwischen der Logik und der Darstellung.

Somit bietet Angular eine robuste Grundlage für die Entwicklung skalierbarer Anwendungen, da es eine klare Struktur, Wiederverwendbarkeit von Komponenten, und eine effiziente Zusammenarbeit zwischen mehreren Entwicklungsteams durch die Modularität und Isolation der Komponenten ermöglicht.



# Angular Material

Figure 17: Offizielles Angular Material Logo.

**Angular Material** ist eine offizielle UI-Komponentenbibliothek für Angular, die auf den Designprinzipien von Googles Material-Design basiert. Sie bietet eine Sammlung vorgefertigter und anpassbarer UI-Komponenten wie Buttons, Formulare, Tabellen, Dialoge, Toolbars und mehr. Angular Material verfolgt das Ziel, eine konsistente, benutzerfreundliche und visuell ansprechende Benutzeroberfläche zu ermöglichen. Für das Projekt verbessert es die Entwicklungsproduktivität, fördert ein einheitliches Design und integriert sich nahtlos in das Angular-Framework ein.

## Backend

**Firebase** ist eine umfassende Backend-Plattform von Google, die Funktionen wie Authentifizierung, Datenbankmanagement (Firestore), Cloud-Funktionen, Hosting und mehr bietet. Firebase beschleunigt die Entwicklung, da es viele der üblichen Backend-Aufgaben übernimmt und skalierbare, serverlose Architekturen unterstützt.

**AngularFire** ist eine Bibliothek für die Integration von Firebase-Diensten in Angular-Anwendungen. Sie bietet eine einfache Möglichkeit, Firebase-Funktionen wie Authentifizierung, Datenbanksynchronisierung und Dateispeicherung in Angular-Projekte einzubinden.

## Programmiersprachen

**TypeScript** ist ein Supersatz von JavaScript, welcher statische Typisierung hinzufügt. Es ermöglicht eine bessere Fehlerprüfung während der Entwicklungszeit und erleichtert das Schreiben und Verstehen von komplexem Code. Angular selbst ist in TypeScript geschrieben und bietet umfangreiche Unterstützung für diese Sprache.

**SCSS** (Sassy CSS) ist eine präprozessorische Skriptsprache, die zu CSS (Cascading Style Sheets) kompiliert wird. SCSS erweitert die Möglichkeiten von CSS mit Funktionen wie Variablen, verschachtelten Regeln und Mixins, was die Verwaltung und Wiederverwendung von Stilvorlagen deutlich vereinfacht.

## Sonstige Tools, Pattern und Paradigmen

**REST-API:** Um an die Manga-Daten zu kommen, verwendet das Projekt die Open Source Schnittstelle der Jikan API verwendet, die auf dem REST

Paradigma basiert. Dabei steht REST für

- REpresentational
- State
- Transfer

Es basiert auf einer zustandslosen Kommunikation und verwendet HTTP-Methoden wie GET, POST, PUT und DELETE, um Ressourcen zu manipulieren. RESTful APIs sind darauf ausgelegt, einfach und skalierbar zu sein, indem sie standardisierte URLs und Datenformate wie JSON oder XML verwenden. In diesem Fall wird für die Kommunikation JSON verwendet, mit welchem sich leicht in modernen Programmiersprachen arbeiten lässt.

**Reaktive Programmierung** Reaktive Programmierung ist ein Programmierparadigma, das sich auf die asynchrone Verarbeitung und den Datenfluss konzentriert. Dabei werden Daten als Streams betrachtet, die kontinuierlich verarbeitet und transformiert werden können. Neue Werte in der Datenquelle propagieren sich automatisch durch das System, ohne dass explizite Events oder manuelle Abfragen nötig sind. Dies unterscheidet sich zu anderen Ansätzen, wie der imperativen Programmierung, bei der die einzelnen Schritte nacheinander definiert werden. Dies hat den Nachteil, dass mehr Code nötig ist und dieser schwieriger zu warten ist.

**Signals:** sind eine von verschiedenen konkreten Kommunikationsmethoden, für die Implementierung einer reaktiven Anwendung in Angular. Sie ermöglichen eine effiziente und leistungsstarke Zustandsverwaltung, indem sie Änderungen an Daten automatisch verfolgen und die betroffenen Komponenten bei Bedarf neu rendern.

Das folgende Listing zeigt ein Beispiel für eine einfache Zähler-Komponente, die Signals nutzt.



```

Einfaches Beispiel für Signals
— □ ×

import { Component, signal } from '@angular/core'

@Component({
  selector: 'app-counter',
  template: `
    <h1>Counter: {{ count() }}</h1>
    <button (click)="increment()">+</button>
    <button (click)="decrement()">-</button>
  `,
})
export class CounterComponent {
  count = signal(0) // Signal mit initialm Wert 0

  increment() {
    this.count.set(this.count() + 1) // Wert setzen
  }

  decrement() {
    this.count.set(this.count() - 1)
  }
}

```

Figure 18: Code-Beispiel für Signals in Angular.

Mit dem `signal(0)` Befehl wird ein neues Signal `count` deklariert. Dieses kann dann mit `count.set()` einem neuen Wert zugeschrieben werden. In dem Template kann dieser reaktive Wert dann sehr leicht über den `count()` Syntax genutzt werden. Der Vorteil ist hier, dass das User-Interface sehr präzise und performant erkennt, wenn sich der `count` Wert verändert hat, wodurch nicht die gesamte Komponente, sondern nur der relevante Bereich der Seite neu gerendert werden muss.

Durch die Nutzung von Signals in Angular kann man also sehr einfach das reaktive Programmierparadigma umsetzen und bekommt dazu noch bessere Performanz und Wartbarkeit.

## Architekturübersicht

Insgesamt verwendet die Applikation eine **serverlose Architektur**. Das Merkmal dieser Architektur besteht nicht darin, dass es überhaupt keine Server gibt, sondern, dass die Verwaltung, die Bereitstellung der Infrastruktur, und die Skalierung der Server an einen Drittanbieter (z. B. Google, Amazon oder Microsoft) ausgelagert wird. Bei der Nutzung einer solchen Architektur muss sich also nur noch um den eigenen Quellcode gekümmert werden. Dies bietet

den Vorteil, dass enorm an Aufwand, Zeit und Ressourcen gespart werden kann, da das Aufsetzen und Verwalten von Servern sehr aufwendig ist.

In diesem Fall ist Google der Anbieter, durch das Firebase Produkt, welches von der Anwendung als Backend-as-a-service (Cloud Firestore) und zur Authentifizierung (Firebase Auth) verwendet wird. Somit besteht die Applikation aus einem Frontend-Client, einem Backend-as-a-service (Cl) und der Jikan REST-API, die die Manga-Daten liefert.

Das Folgende Diagramm zeigt eine abstrakte Darstellung der beschriebenen Architektur.

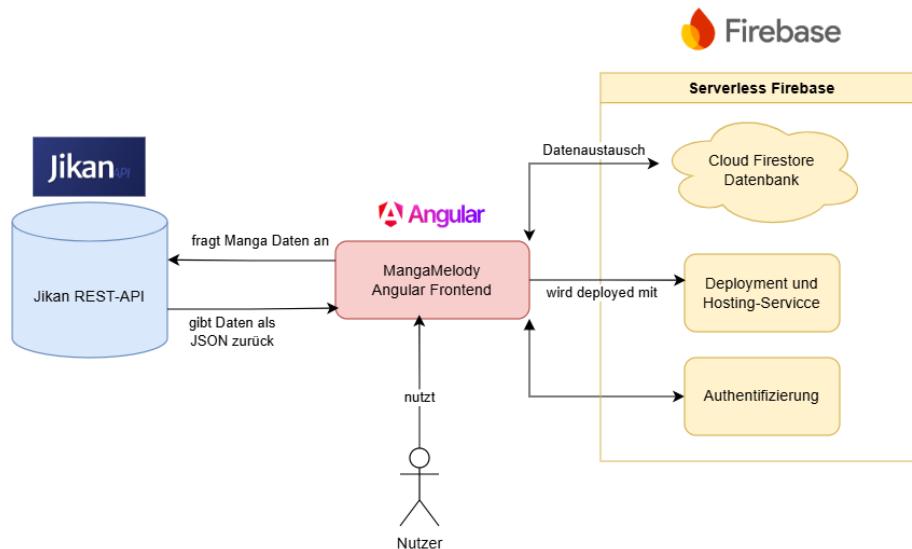


Figure 19: Externes Architekturdiagramm der Anwendung und ihrer Komponenten.

## Struktur

Das Folgende UML-Klassendiagramm zeigt die Hauptkomponenten der Anwendung sowie deren Beziehungen zueinander.

Die Klassen Service, Page, und Component sind dabei abstrakte Klassen, was durch den kursiven Klassennamen gekennzeichnet ist. Sie bilden die Hauptbausteine der Anwendung. Aus diesen Klassen werden Unterklassen Abgeleitet, aus denen die Applikation zusammengebaut wird.

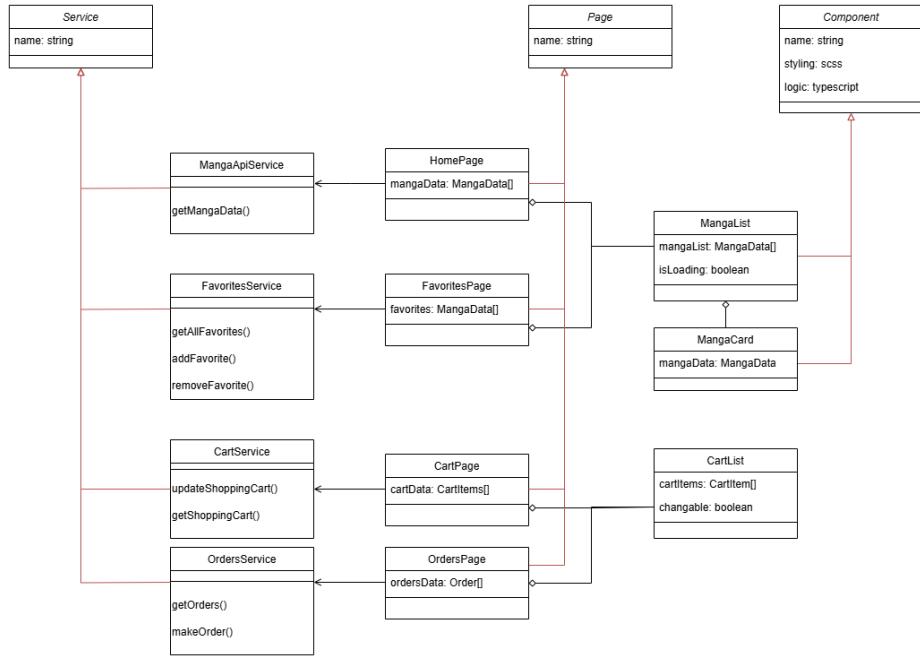


Figure 20: UML-Klassendiagramm der Hauptkomponenten der Anwendung.

## Verhalten

Im folgenden soll das Verhalten der Anwendung beschrieben werden, wenn ein Nutzer auf der Manga-Details Seite, einen Manga in den Warenkorb legt. Das folgende Bild soll zur Nachvollziehbarkeit den Kontext aufzeigen, in dem diese Sequenz auftritt.

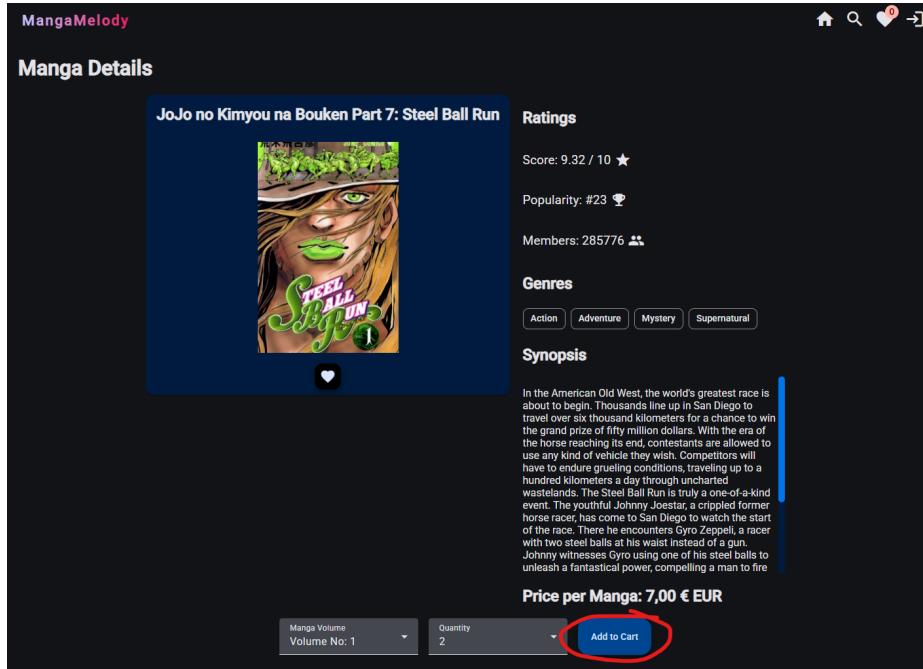


Figure 21: Auf dieser Seite kann der Nutzer einen Manga zum Warenkorb hinzufügen.

### Add Manga to Cart Process

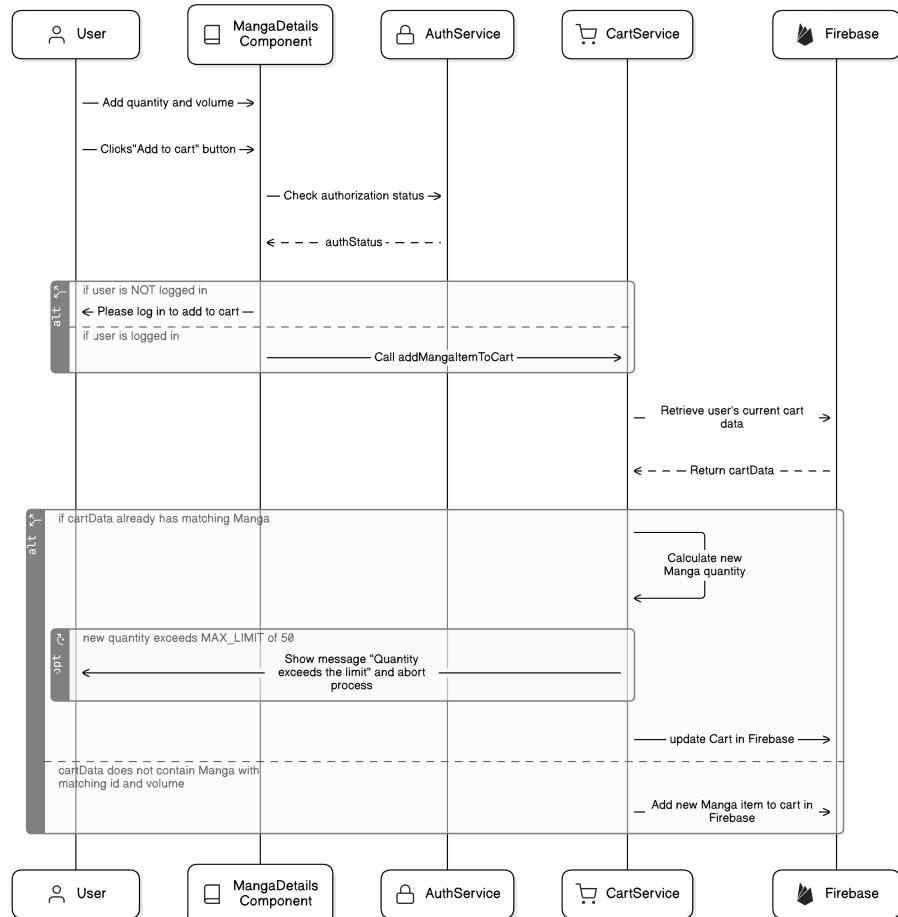


Figure 22: UML-Sequenzdiagramm, welches den Prozess zeigt, mit dem ein Nutzer einen Manga in den Warenkorb legt.

## Testdokument

Dieses Testdokument dient der Erläuterung der durchgeführten Qualitätssicherungsmaßnahmen im Rahmen des Softwareentwicklungsprozesses. Qualitätssicherung ist ein essenzieller Bestandteil, um sicherzustellen, dass die entwickelte Software den festgelegten Anforderungen entspricht und fehlerfrei ist. Durch systematische Tests und Überprüfungen wird die Zuverlässigkeit, Funktionalität und Benutzerfreundlichkeit der Software gewährleistet. Es beschreibt die angewendeten Teststrategien, die durchgeführten Testfälle und deren Ergebnisse, um die Qualitätssicherung der Software transparent und nachvollziehbar zu machen.

## Teststrategie

Die Teststrategie konzentriert sich auf Unit-Tests und manuelle End-to-End-Tests (E2E), während Integrationstests bewusst vernachlässigt wurden. Dieser Ansatz verfolgt das Ziel, mit Unit-Tests die interne Logik der Applikation sowie den Programmfluss detailliert zu überprüfen, während E2E-Tests ein umfassendes, ganzheitliches Testen der gesamten Anwendung ermöglichen – einschließlich aller Schnittstellen und externen Abhängigkeiten. Durch diese gezielte Kombination wird sowohl die korrekte Funktionalität einzelner Komponenten als auch das reibungslose Zusammenspiel im Gesamtsystem effizient abgedeckt. Auf Integrationstests wurde vor allem wegen drei Gründen verzichtet.

1. **Aufwand:** Die Implementierung von Integrationstests für die Anwendung wäre mit einem erheblichen Mehraufwand verbunden. Da Integrationstests darauf abzielen, die korrekte Zusammenarbeit mehrerer Komponenten oder Module einer Applikation sicherzustellen, müssten umfangreiche Testfälle für verschiedene Interaktionen zwischen Frontend, Backend und externen Schnittstellen (Firebase, Firebase Authentication, Manga-REST-API) erstellt werden. Aufgrund der Vielzahl an Abhängigkeiten und möglichen Szenarien wäre die Entwicklung und Wartung solcher Tests besonders zeitintensiv. Es wurde daher entschieden, diesen Aspekt der Qualitätssicherung effizienter durch manuelle End-to-End-Tests abzudecken, da diese einen realistischen Nutzungskontext abbilden und schneller umgesetzt werden können.
2. **Systemabhängigkeiten:** Die Applikation ist stark von externen Diensten abhängig, insbesondere von Firebase und einer externen Manga REST API. Integrationstests erfordern in der Regel stabile, kontrollierbare Testumgebungen für alle beteiligten Systeme. Da die externen APIs jedoch nicht immer eine zuverlässige Testumgebung oder Mocking-Möglichkeiten bereitstellen, könnten unerwartete Änderungen oder Downtimes der externen Dienste zu fehlschlagenden Tests führen. Dies würde den Mehrwert der Integrationstests einschränken und könnte zu einer hohen Wartungslast führen, um ständig auf Änderungen der Schnittstellen zu reagieren.
3. **Technische Limitationen:** Firebase bietet kein optimales Tooling für die

Durchführung von Integrationstests, speziell wenn es um das Testen mit Authentifizierung und Datenbankoperationen geht. Eine der größten Herausforderungen ist, dass Firebase standardmäßig mit einer Live-Datenbank arbeitet, wodurch es schwierig ist, eine dedizierte Testumgebung aufzusetzen, ohne produktive Daten zu beeinflussen. Alternativen wie das lokale Firebase Emulator Suite bieten zwar teilweise Unterstützung, sind jedoch nicht für alle Anwendungsfälle gleichwertig zur Produktionsumgebung. Zudem ist es aufwendig, Testdaten zu generieren und zu verwalten, ohne dabei unerwünschte Seiteneffekte auf die tatsächliche Datenbank zu haben.

### Unit-Tests

Durch das Testen von isolierten Einheiten der Software können Fehler frühzeitig erkannt und behoben werden, was die Kosten und den Aufwand für spätere Korrekturen reduziert. Unit-Tests fördern zudem eine modulare und wartbare Codebasis, da sie Entwickler dazu anregen, kleine, gut definierte Funktionen zu schreiben.

Innerhalb dieser Strategie ist es das Ziel, durch die Unit-Tests die wichtigsten Klassen und Services auf ihre Korrektheit zu prüfen. Hierbei soll keine vollständige Testabdeckung erreicht werden, sondern eine solide Absicherung der wichtigsten Business-Logik der Anwendung. Hierbei wurden vor allem für die *Services* der Applikation umfassende Unit-Tests entwickelt. Das Unit-Testing der Pages und Komponenten, die hauptsächlich zur Darstellung der Daten dienen, wurde hierbei weniger in den Fokus genommen.

Eine weitere Teststrategie, die in Verbindung mit Unit-Testing verwendet wurde, ist das *Black-Box-Testing*. Beim Black-Box Testing werden die internen Strukturen oder das Verhalten der zu testenden Komponente nicht berücksichtigt. Stattdessen konzentriert sich dieser Ansatz auf die Eingaben und die erwarteten Ausgaben. Dies ist besonders nützlich für Unit-Tests, da es ermöglicht, die Funktionalität einzelner Module unabhängig von ihrer internen Implementierung zu überprüfen. Wenn es bei dem Testen von Service A hierbei eine Abhängigkeit zu Service B oder C gibt, so werden die Ausgaben von B und C simuliert oder *gemockt*. Dadurch können die Tests isoliert durchgeführt werden, ohne dass externe Abhängigkeiten das Testergebnis beeinflussen. Dies verbessert nicht nur die Stabilität der Tests, sondern reduziert auch deren Laufzeit, da keine echten API-Aufrufe oder Datenbankzugriffe erforderlich sind. Hier kommt ebenfalls die Abgrenzung zum Integrationstesting auf. Während ein Integrationstest echte API-Abrufe nutzt, werden diese in einem Unit-Test nur simuliert, was die Komplexität und den Aufwand enorm verringert.

Die folgende Abbildung zeigt ein Beispiel eines solchen Unit-Tests.



```

Beispiel eines Unit-Tests mit Mocking
- □ ×

// Tests für den Cart Service
describe('CartService', () => {
  let service: CartService;
  let authServiceFixture: AuthServiceFixture;

  // Wird vor jedem Test ausgeführt
  beforeEach(() => {
    authServiceFixture = new AuthServiceFixture();
    TestBed.configureTestingModule({
      providers: [
        { provide: AuthService, useValue: authServiceFixture },
      ],
    });
    service = TestBed.inject(CartService);
  });

  // Unit Test mit Mocking
  it('should return empty array if user data is undefined', async () => {
    authServiceFixture.getUserData.and.returnValue(of(undefined));
    expect(await firstValueFrom(service.getShoppingCart())).toEqual([]);
  });
});

```

Figure 23: Beispiel eines Unit-Tests für den Cart-Service

Der Vorteil von Black-Box-Testing bei Unit-Tests liegt darin, dass es eine objektive Bewertung der Funktionalität ermöglicht. Entwickler können sicherstellen, dass die Komponente korrekt auf verschiedene Eingaben reagiert, ohne sich um die internen Details kümmern zu müssen. Dies fördert eine klare Trennung von Implementierung und Tests und erleichtert die Wartung und Erweiterung des Codes.

### End-to-End-Tests

Manuelle End-to-End-Tests simulieren reale Benutzerinteraktionen und testen das gesamte System von Anfang bis Ende. Diese Tests sind am aufwendigsten und zeitintensivsten, bieten jedoch die höchste Sicherheit, dass das System wie erwartet funktioniert. Manuelle End-to-End-Tests stellen sicher, dass alle Teile des Systems, einschließlich der Benutzeroberfläche, Backend-Services und Datenbanken, nahtlos zusammenarbeiten. Sie sind entscheidend für die Validierung der Benutzererfahrung und die Sicherstellung, dass die Software in einer realen Umgebung zuverlässig funktioniert. Hierzu wurde die Live-Applikation anhand von mehreren Testfällen getestet, inklusive einer Beobachtung der Datenbank und Authorization. Bei Bedarf wurden diese Tests auch Mithilfe von Screenshots dokumentiert.

## Testprotokoll

Im folgenden werden die durchgeführten Testfälle und ihre jeweilige Resultate protokolliert. Hierbei wird folgendes festgehalten:

- eine Testfall-ID
- die Vorbedingungen
- die Eingabedaten oder die ausgeführten Aktionen
- das erwartete Ergebnis
- das tatsächliche Ergebnis

Wir betrachten zunächst die implementierten Unit-Tests. Anschließend werden die manuellen E2E-Tests dokumentiert.

### Order Service

Testfall-ID: ut-order-service-1

Vorbedingungen:

- Der Nutzer ist nicht eingeloggt.
- Die `makeOrder` Funktion des Services wird mit Checkout-Daten aufgerufen.

Erwartetes Ergebnis:

- Die Funktion soll `null` zurückgeben, da nur eingeloggte Nutzer eine Bestellung aufgeben können.
- Keine Firebase-Methode soll aufgerufen werden.

Tatsächliches Ergebnis:

- Stimmt mit dem Erwarteten überein.

---

Testfall-ID: ut-order-service-2

Vorbedingungen:

- Der Nutzer ist eingeloggt.
- Die `makeOrder` Funktion des Services wird mit Checkout-Daten aufgerufen.

Erwartetes Ergebnis:

- Die `clearCart` Funktion des `CartService` wird aufgerufen, da nach der Bestellung der Warenkorb geleert wird.
- Die `addDoc` Funktion von Firebase wird aufgerufen.

Tatsächliches Ergebnis:

- Stimmt mit dem Erwarteten überein.

### **Manga-API Service**

Testfall-ID: ut-manga-api-service-1

Vorbedingungen:

- Die `getJikanMangaData` Funktion wird mit einem leeren Objekt aufgerufen.

Erwartetes Ergebnis:

- Eine HTTP-Anfrage wird an die `https://api.jikan.moe/v4/manga` URL geschickt mit folgenden Standardparametern:
  - `sfw=true`
  - `genres_exclude = 9,49,12`
  - `page = 1`
  - `limit = 24`
  - `order_by = score`
  - `sort = desc`
- Die HTTP-Methode ist vom Typ GET.
- Die Funktion gibt eine Response zurück.

Tatsächliches Ergebnis:

- Stimmt mit dem Erwarteten überein.

### **Favorites Service**

Testfall-ID: ut-favorites-service-1

Vorbedingungen:

- Ein Manga wird den Favoriten hinzugefügt.
- Derselbe Manga wird erneut hinzugefügt.

Erwartetes Ergebnis:

- Der Manga wird nur einmal in die Favoritenliste aufgenommen.

Tatsächliches Ergebnis:

- Stimmt mit dem Erwarteten überein.

---

Testfall-ID: ut-favorites-service-2

Vorbedingungen:

- Zwei verschiedene Manga werden den Favoriten hinzugefügt.

Erwartetes Ergebnis:

- Beide Manga sind in der Favoritenliste enthalten.

Tatsächliches Ergebnis:

- Stimmt mit dem Erwarteten überein.
- 

Testfall-ID: ut-favorites-service-3

Vorbedingungen:

- Ein Manga wird den Favoriten hinzugefügt.
- Die Favoritenliste wird geleert.

Erwartetes Ergebnis:

- Die Favoritenliste ist leer.

Tatsächliches Ergebnis:

- Stimmt mit dem Erwarteten überein.

## Cart Service

Testfall-ID: ut-cart-service-1

Vorbedingungen:

- Der Warenkorb enthält Elemente.
- Die `clearCart` Methode wird aufgerufen.

Erwartetes Ergebnis:

- Der Warenkorb wird geleert.

Tatsächliches Ergebnis:

- Stimmt mit dem Erwarteten überein.

## Snackbar Service

Testfall-ID: ut-snackbar-service-1

Vorbedingungen:

- Die Methode `openSnackBar` wird mit einer Standardnachricht aufgerufen.

Erwartetes Ergebnis:

- Die Snackbar wird mit der Nachricht “Test message”, Button “Okay”, Klasse `snackbar-success` und einer Dauer von 2000ms geöffnet.

Tatsächliches Ergebnis:

- Stimmt mit dem Erwarteten überein.
- 

Testfall-ID: ut-snackbar-service-2

Vorbedingungen:

- Die Methode `openSnackBar` wird mit einer benutzerdefinierten Nachricht, Klasse, Aktionsnachricht und Dauer aufgerufen.

Erwartetes Ergebnis:

- Die Snackbar wird mit der Nachricht “Custom message”, Button “Dismiss”, Klasse `snackbar-warning` und einer Dauer von 3000ms geöffnet.

Tatsächliches Ergebnis:

- Stimmt mit dem Erwarteten überein.
- 

Abschließend zeigt die folgende Abbildung die Visualisierung der Unit-Tests durch das Testing-Framework Karma. An dieser Stelle ist es wichtig zu erwähnen, dass Angular standardmäßig für jede erstellte Klasse (Service oder Komponente) eine Test-File erstellt. Diese enthält einen simplen, automatisch erstellten Test, der nur prüft, ob die Klasse erfolgreich instanziert werden kann (`should create`). Diese Tests wurden nicht selbst implementiert und wurden demnach nicht im Testprotokoll dokumentiert. Dennoch tauchen diese bei der Visualisierung auf.

**Karma v 6.4.4 - connected; test: complete;**

Chrome 133.0.0.0 (Windows 10) is idle

@@ Jasmine 4.6.1

.....

45 specs, 0 failures, randomized with seed 65842

```

CartService
  • should return empty array if user data is empty object
  • should modify the cart item quantity
  • should return empty array if user data is undefined
  • should add the cart item if the manga does not exist already
  • should update the cart item quantity if manga already exists
  • should add the manga item
  • should upsert manga item to cart
  • should clear the cart
  • should return the cart item count
  • should return empty array if cart data is undefined
  • should delete the cart item
  • should only delete the cart item with matching mal id and volume

OrderService
  • should return null if the user is not logged in
  • should be created
  • should create an order when the user is logged in and clear the cart

PrintComponent
  • should create

MangaAPIService
  • should overwrite the given parameters
  • should overwrite the parameters with custom ones
  • should be created
  • should create an empty query from empty form data
  • should call the API with the correct standard parameters

OrdersComponent
  • should create

FavoritesComponent
  • should create

FavoritesService
  • should return false if a given manga is not a favorite
  • should clear the favorite
  • should add the same manga to favorites twice
  • should add two manga if they are different
  • should be created
  • should return true if a given manga is already a favorite

AppComponent
  • should create the app

HomeComponent
  • should create

FirebaseWrapperService
  • should be created
  • should create

MangaDetailComponent
  • should create

CartListComponent
  • should create

LoginRegisterComponent
  • should create

MangaCardComponent
  • should create

MangaListComponent
  • should create

SearchFieldComponent
  • should create

AuthService
  • should be created

CartComponent
  • should create

SearchComponent
  • should create

CheckoutComponent
  • should create

SnackbarService
  • should be created
  • should open a Snackbar with default values
  • should open a Snackbar with custom values

```

Figure 24: Unit-Test Ergebnisse

Es folgt die Dokumentierung der manuellen E2E-Tests, gruppiert nach Seite.

---

## Homepage

Testfall-ID: e2e-home-manga-display

Vorbedingungen:

- Die Applikation wird aufgerufen.
- Der Nutzer ist nicht eingeloggt.

Erwartetes Ergebnis:

- Die Homepage wird geladen.
- Die Top 5 populärsten Mangas werden geladen.

- Die Navigationsleiste zeigt die folgenden Icon-Buttons:
  - Home
  - Suche
  - Favoriten
  - Login

Tatsächliches Ergebnis:

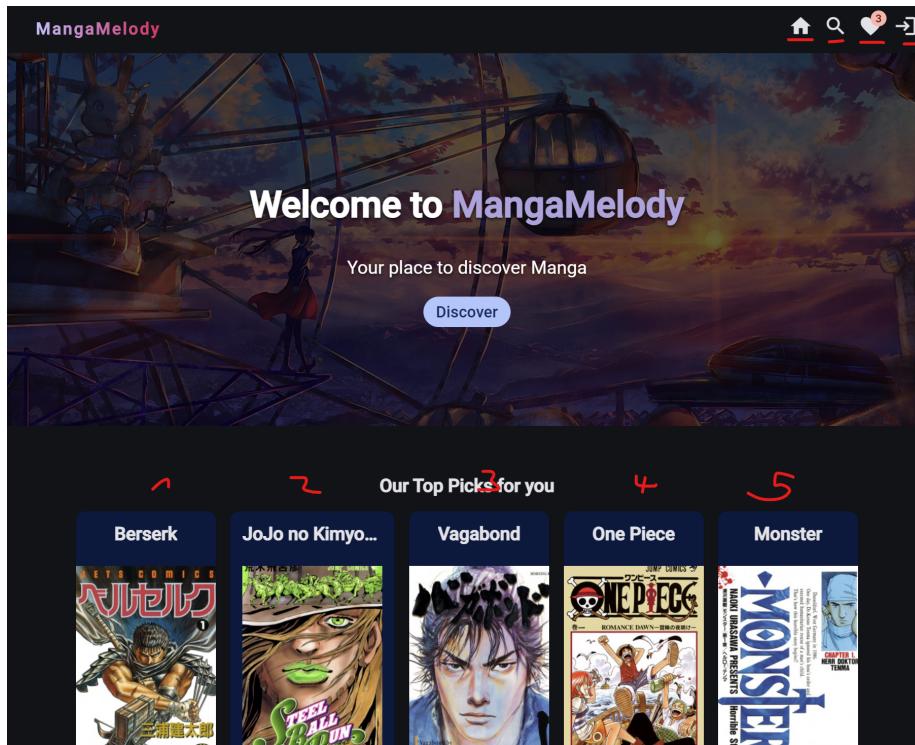


Figure 25: Ergebnis für Testfall e2e-home-manga-display.

- Stimmt mit dem Erwarteten überein.

---

Testfall-ID: e2e-home-manga-discover

Vorbedingungen:

- Der Nutzer befindet sich auf der Home Page.
- Der Nutzer clickt auf den Discover-Button.

Erwartetes Ergebnis:

- Der Nutzer wird auf die /search Search-Page umgeleitet

Tatsächliches Ergebnis:

- Stimmt mit dem Erwarteten überein.

### Search-Page

Testfall-ID: e2e-search-manga

Vorbedingungen:

- Der Nutzer befindet sich auf der Search-Page
- Der Nutzer nutzt die Suche nach Titel, die Filter nach Status, Genre, und die Ordnungs-und Sortier-Funktion

Erwartetes Ergebnis:

- Die Manga-Suche wird mit den vom Nutzer eingestellten Kriterien gefiltert, geordnet und sortiert.
- Die Filter sollten auch in der Kombination funktionieren.
- Um nicht zu viele Anfragen zu senden, soll es eine kleine Verzögerung (0,5s) zwischen dem Tippen neuer Zeichen in die Suche und dem Abschicken der Anfrage geben.

Tatsächliches Ergebnis:

- Stimmt mit dem Erwarteten überein.

---

Testfall-ID: e2e-search-manga-pagination

Vorbedingungen:

- Der Nutzer befindet sich auf der Search-Page
- Der Nutzer nutzt die Pagination-Funktion, um durch die Ergebnisse zu blättern
- Der Nutzer ändert die Items per page

Erwartetes Ergebnis:

- Die Seite soll die spezifizierte Anzahl von Items per Page anzeigen
- Durch das Blättern der Ergebnisse wird die API-Anfrage so angepasst, dass die aktuelle Seite und das Limit übergeben wird

Tatsächliches Ergebnis:

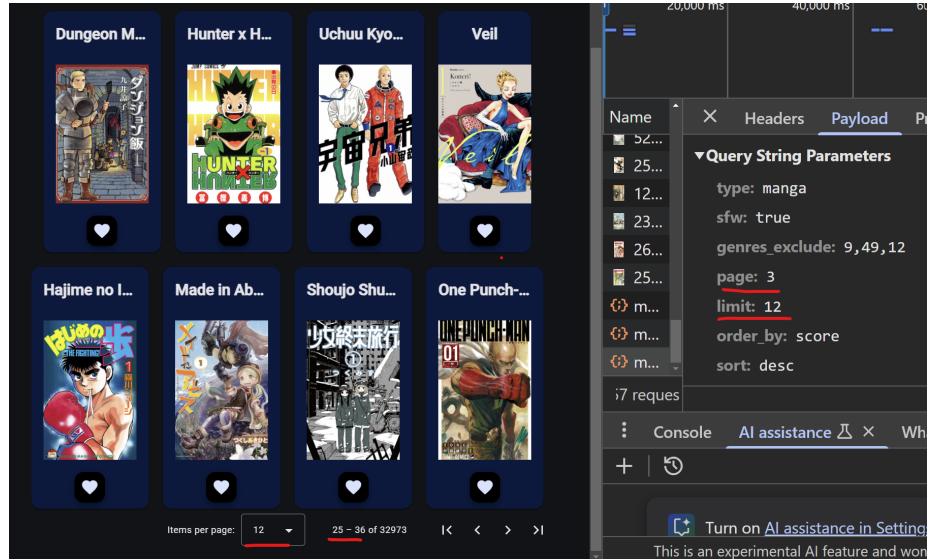


Figure 26: Screenshot von Ergebnis für Testfall e2e-search-manga-pagination.

- Stimmt mit dem Erwarteten überein.

### Favorisierung

Testfall-ID: e2e-favorite

Vorbedingungen:

- Der Nutzer befindet sich auf einer Seite, wo eine Manga-Card von einem noch nicht favorisierten Manga angezeigt wird (siehe Abbildung).
- Der Nutzer favorisiert den Manga über den Herz-Button.



Figure 27: Beispiel einer Manga-Card.

Erwartetes Ergebnis:

- Der Herz-Button soll farbig markiert werden.
- Dem Nutzer soll eine Nachricht angezeigt werden, die den Nutzer über die Favorisierung informiert.
- Der Manga soll auf der Favoriten-Seite angezeigt werden.

Tatsächliches Ergebnis:

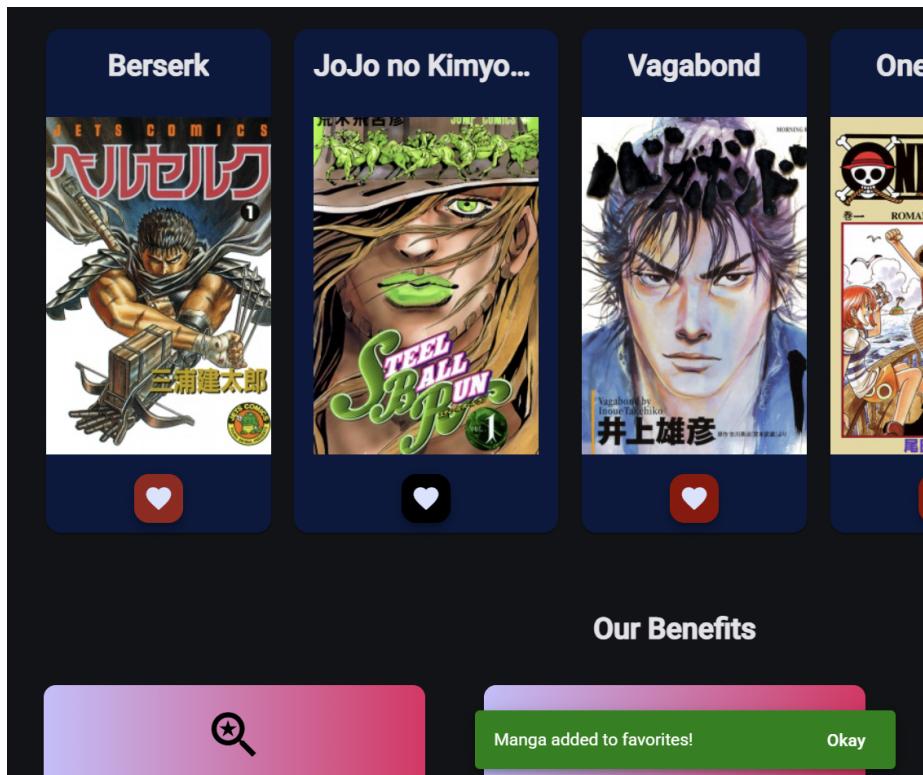


Figure 28: Screenshot von Ergebnis für Testfall e2e-favorite - Benutzernachricht.

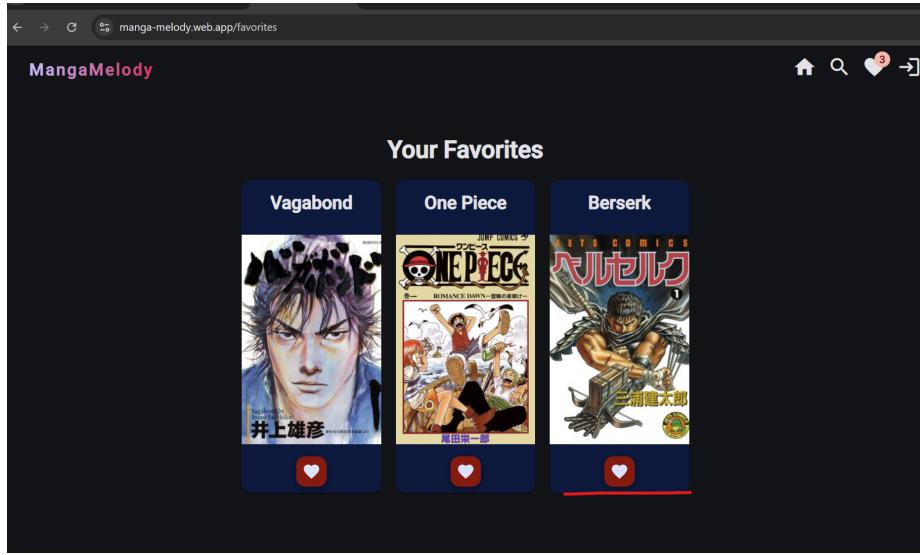


Figure 29: Screenshot von Ergebnis für Testfall e2e-favorite - Anzeige auf Favoriten-Seite.

- Stimmt mit dem Erwarteten überein.

Testfall-ID: e2e-favorite-details

Vorbedingungen:

- Der Nutzer befindet sich auf der Favoriten-Seite.
- Der Nutzer klickt auf die Manga-Card eines Favoriten.

Erwartetes Ergebnis:

- Der Nutzer wird auf die Manga-Details-Seite weitergeleitet
- Der Nutzer kann sämtliche Daten des Mangas einsehen können.

Tatsächliches Ergebnis:

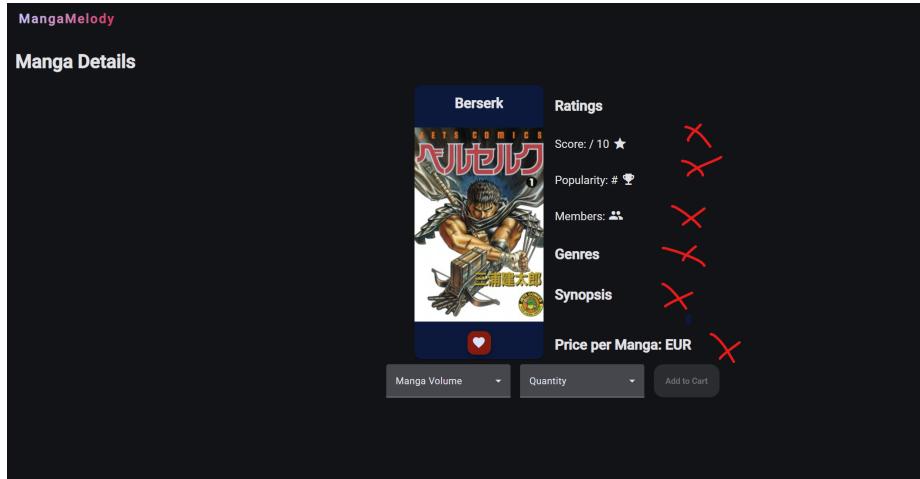


Figure 30: Screenshot von Ergebnis für Testfall e2e-favorite-details: Test wurde nicht bestanden.

- Der Nutzer wird auf die Manga-Details-Seite weitergeleitet.
- Die Daten des Mangas sind unvollständig.
- Ratings, Synopsis, Genres und der Preis fehlen.

Maßnahme:

- Fehler wurde behoben über fix.
- Commit message: removing minmangaitem.
- Commit-ID: a61b8a9.
- Nach erneutem Testen entspricht das Ergebnis den Anforderungen.

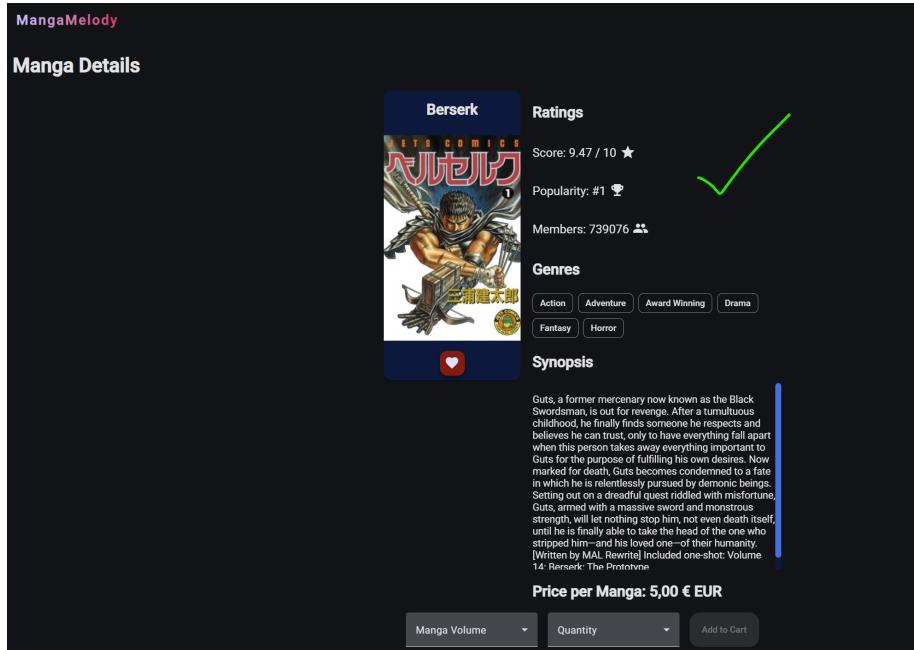


Figure 31: Ergebnis nach Fix - Resultat entspricht den Anforderungen.

Testfall-ID: e2e-unfavorite

Vorbedingungen:

- Der Nutzer befindet sich auf der Favoriten-Seite
- Der Nutzer clickt auf den Herz-Button

Erwartetes Ergebnis:

- Der Manga wird aus der Favoriten-Seite entfernt.
- Eine Nachricht wird dem Nutzer angezeigt, die ihn darüber informiert.

Tatsächliches Ergebnis:

- Stimmt mit dem Erwarteten überein.

## Login/Register

Testfall-ID: e2e-register

Vorbedingungen:

- Der Nutzer ist nicht eingeloggt, und hat noch keinen Account.
- Der Nutzer navigiert zur /login Seite.
- Der Nutzer geht auf den *Register* Tab.

- Der Nutzer füllt die Registrierungsmaske aus und clickt auf den *Register* Button

Erwartetes Ergebnis:

- Der Button ist gesperrt, wenn die Eingabe invalide ist, also
  - Eines der Felder ist nicht ausgefüllt.
  - Die E-Mail ist folgt keinem validen Pattern.
  - Die Passwörter stimmen nicht überein.
- Der Button ist verfügbar, wenn die Eingabe valide ist
- Nach dem clicken vom *Register* Button, wird der Nutzer auf die Profil-Seite weitergeleitet.
- Auf der Profil-Seite sieht der Nutzer seine Informationen wie Nutzernname und E-Mail.
- Der Nutzer wird in der Firebase-Datenbank angezeigt

Tatsächliches Ergebnis:

- Stimmt mit dem Erwarteten überein.
- 

Testfall-ID: e2e-login

Vorbedingungen:

- Der Nutzer ist nicht eingeloggt, hat jedoch schon einen Account.
- Der Nutzer navigiert zur /login Seite.
- Der Nutzer füllt die Login-Maske aus und clickt auf den *Login* Button.

Erwartetes Ergebnis:

- Der Button ist gesperrt, wenn die Eingabe invalide ist, also
  - Eines der Felder ist nicht ausgefüllt.
  - Die E-Mail folgt keinem validen Pattern.
- Der Nutzer wird nur mit bestehenden korrekten Anmeldedaten eingeloggt,
- Andernfalls wird dem Nutzer eine Fehlermeldung angezeigt.
- Nach dem Login wird der Nutzer zur Profil-Seite navigiert.

Tatsächliches Ergebnis:

- Stimmt mit dem Erwarteten überein.

## Cart

Testfall-ID: e2e-cart

Vorbedingungen:

- Der Nutzer ist eingeloggt und auf der Cart-Seite und hat mindestens einen Manga im Warenkorb.
- Der Nutzer passt die Anzahl über den Inkrement und Dekrement Button an.

- Der Nutzer passt die Anzahl über das Dropdown an.

Erwartetes Ergebnis:

- Die Anzahl darf nicht 50 übersteigen.
- Wenn die Anzahl auf 0 fällt, wird der Artikel aus dem Warenkorb entfernt.
- Die Anzahl darf nicht negativ werden
- Die Zwischensumme (Subtotal) sowie die Gesamtsumme (Total Amount) müssen den arithmetisch korrekten Wert haben.

Tatsächliches Ergebnis:

- Bei der Gesamtsumme steht Total: Total Amount, was redundant ist.
- Die restlichen Anforderungen entsprechen dem erwarteten Ergebnis.

Maßnahme:

- Der Text wird von “Total: Total Amount” zu “Total Amount” geändert.
- Commit message: fix cart total amount redundancy.
- Commit-ID: dca8a90.
- Nach erneutem Testen entspricht das Ergebnis vollständig den Anforderungen.

## Bestellungsprozess

Testfall-ID: e2e-checkout

Vorbedingungen:

- Der Nutzer ist eingeloggt und ist mit einem Warenkorb zur Checkout-Seite navigiert.
- Der Nutzer durchschreitet den Checkout-Prozess.
  - Shipping Information
  - Payment Information
  - Summary
- Der Nutzer schließt die Bestellung ab.

Erwartetes Ergebnis:

- Für alle Schritte kann der nächste nur dann erreicht werden, wenn alle notwendigen Informationen ausgefüllt sind.
- Die IBAN im zweiten Schritt muss eine valide IBAN sein.
- Nach dem klicken auf den *Confirm payment* Button wird dem Nutzer eine Nachricht angezeigt und er wird auf die Orders-Seite weitergeleitet.
- In der Firebase-Datenbank wird ein neuer Eintrag unter `orders` erstellt.
- Auf der Orders-Seite wird die getätigte Bestellung angezeigt mit den korrekten Daten.

Tatsächliches Ergebnis:

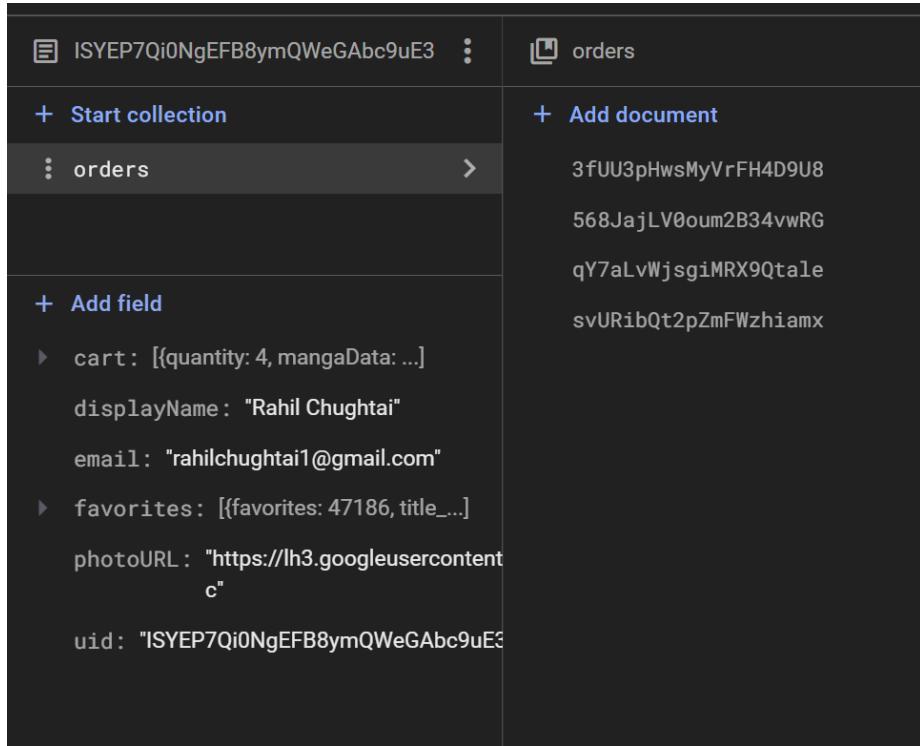


Figure 32: Orders Eintrag in der Firebase Datenbank wurde korrekt erstellt.

- Stimmt mit den erwarteten überein.

## **Abstract**

### **Einleitung**

Mangas sind eine besondere Form von Comics mit Ursprung in Japan. Sie zeichnen sich durch ihren einzigartigen künstlerischen Stil sowie innovative Erzähltechniken aus und haben weltweit an Popularität gewonnen. Neben ihrer Funktion als Unterhaltungsmedium behandeln Mangas oft tiefgründige kulturelle, soziale und philosophische Themen. Durch komplexe Charakterentwicklungen und gesellschaftsrelevante Fragestellungen sprechen sie ein breites Publikum an, von Kindern bis zu Erwachsenen.

Im Rahmen des Projekts "MangaMelody" wurde eine Webanwendung entwickelt, die eine intuitive und benutzerfreundliche Plattform zum Entdecken und Erwerben von Mangas bietet. Die Anwendung ermöglicht gezielte Suchen nach Manga-Titeln, das Speichern von Favoriten und eine simulierte Kaufabwicklung. Technologisch basiert die Anwendung auf Angular, Firebase und der Jikan-Manga-API.

Im Folgenden wird die Durchführung dieses Projekts zusammengefasst und kritisch reflektiert – insbesondere im Hinblick auf die Entwicklung, die gewonnenen Erkenntnisse, mögliche Verbesserungen, aufgetretene Probleme sowie die Einhaltung des Zeitplans.

### **Durchführung**

Die Entscheidung für die Entwicklung eines Manga-Webshops mit Angular wurde aus persönlicher Leidenschaft für Mangas sowie dem Wunsch getroffen, neue Features des im beruflichen Umfeld verwendeten Angular-Frameworks auszuprobieren. Nach der Auswahl geeigneter Technologien und der Erstellung der Dokumente für Phase Eins begann die Entwicklung der Anwendung.

Das Projekt wurde in mehrere Arbeitspakete unterteilt, die den verschiedenen Features oder Seiten entsprachen. Anfangs erfolgte die Entwicklung direkt im Main-Branch. Später wurde zur Gewährleistung einer sicheren Entwicklung für jedes Arbeitspaket ein eigener Git-Branch erstellt, der über Pull Requests in den Haupt-Branch integriert wurde. Die Teststrategie wurde erst nach Fertigstellung der Hauptfunktionalitäten entworfen und die Unit-Tests parallel zur Finalisierungsphase implementiert.

### **Kritische Reflexion**

Die Entwicklung von "MangaMelody" verlief insgesamt effizient und reibungslos. Dank bestehender Erfahrung mit Angular war die Planung gut umsetzbar. Auch die Verwendung der neuesten Angular-Version sowie neuer Features wie Signals stellten keine Herausforderungen dar, sondern verbesserte die Entwicklererfahrung. Die Anbindung externer Schnittstellen wie Firebase und der Jikan-API funktionierte problemlos, da die umfangreiche Dokumentation der Technologien eine wertvolle Unterstützung bot. Somit war dies eine positive Möglichkeit für mich, brandneue Features des Frameworks an einem Praxisbeispiel zu testen.

Einige Unsicherheiten traten bei der Erstellung der UML-Diagramme auf. Hierbei sollte für zukünftige Projekte auf eine genauere Einhaltung der UML-Spezifikationen geachtet werden, um Unklarheiten und Syntaxfehler bei der Modellierung zu vermeiden.

Ein Arbeitspaket, welches bei der Durchführung unterschätzt wurde, war die Implementation der Unit-Tests. Ein technisches Problem bestand darin, dass die verwendete Firebase-Bibliothek (AngularFire) nur eingeschränkt mit dem Angular Testing Framework kompatibel war. Dadurch konnten Funktionsaufrufe nicht wie geplant simuliert ("gemockt") werden. Das führte dazu, dass Firebase-Aufrufe direkt gegen die echte Datenbankinstanz liefen, was einem Integrationstest gleichkam und einen erheblichen Mehraufwand verursacht hätte.

Dieses Problem wurde gelöst, indem ein *Wrapper-Service* erstellt wurde, der als Abstraktion für sämtliche Firebase-Methoden diente. Dieser eigene Wrapper-Service kann dann sehr einfach durch die Verwendung von Fixtures gemockt werden, wodurch die Unit-Tests fertiggestellt werden konnten. Somit ergab sich ein höherer Aufwand für die Erstellung der Unit-Tests, jedoch blieb das Ausmaß hierbei noch unter Kontrolle. Künftig sollte frühzeitig die Best Practice etabliert werden, externe Bibliotheken durch Wrapper-Services zu kapseln, um die Testbarkeit von Beginn an sicherzustellen.

## Fazit

Zusammenfassend war das Projekt erfolgreich und effizient. Die Zeitplanung wurde mit ausreichend Puffer eingehalten, und Risiken wurden korrekt eingeschätzt, wodurch Verzögerungen vermieden wurden. Verbesserungspotenzial besteht in der genaueren Modellierung der UML-Diagramme sowie in einer frühzeitigeren Berücksichtigung des Testaufwands.

Für zukünftige Projekte empfiehlt es sich:

- UML-Spezifikationen konsequent einzuhalten
- Unit-Tests parallel zur Implementierung zu schreiben
- Hauptänderungen stets auf separaten Feature-Banches zu entwickeln

Die geplanten Features und Anforderungen wurden erfolgreich umgesetzt, sodass eine benutzerfreundliche Anwendung entstanden ist, mit der Mangas spielerisch entdeckt werden können.

Aus dem Projekt konnte ich wertvolle praxisrelevante Kenntnisse gewinnen, darunter die strukturierte Zeitplanung, die Aufteilung von Features in kleinere Arbeitspakete und die korrekte UML-Modellierung. Zudem bot es mir die Möglichkeit, neue Programmierparadigmen zu erproben und kreative Freiheit zu nutzen. Da die verwendeten Features noch nicht im beruflichen Umfeld eingesetzt werden, war es eine spannende und abwechslungsreiche Erfahrung. Besonders die Arbeit mit Angular Signals zeigte mir, wie reaktive Programmiermuster den Datenfluss vereinfachen, die Performance verbessern, den Code prägnanter machen und das mentale Modell erleichtern.