Projektbericht
Studiengang : Informatik

Projektarbeit AR-Schnitzeljagd

von

Lukas Steckbauer, Rosario Aranzulla

85836, 85816

Betreuender Mitarbeiter : Dr. Marc Hermann

Einreichungsdatum : TODO

Eidesstattliche Erklärung

Hiermit erkläre ich, **Lukas Steckbauer, Rosario Aranzulla**, dass ich die vorliegenden Angaben in dieser Arbeit wahrheitsgetreu und selbständig verfasst habe.

Weiterhin versichere ich, keine anderen als die angegebenen Quellen und Hilfsmittel benutzt zu haben, dass alle Ausführungen, die anderen Schriften wörtlich oder sinngemäß entnommen wurden, kenntlich gemacht sind und dass die Arbeit in gleicher oder ähnlicher Fassung noch nicht Bestandteil einer Studien- oder Prüfungsleistung war.

Ort, Datum Unterschrift (Student)

Kurzfassung

Die vorliegende Projektarbeit dokumentiert eingehend das bearbeitete Projekt *AR-Schnitzeljagd*. Das Projekt umfasst den Entwurf, die Implementierung und Inbetriebnahme eines Schnitzeljagd-Editors, welcher zur Erstellung und zum Bearbeiten bestehender Schnitzeljagdten Verwendung findet. Zudem wurde eine Smartphone-Anwendung für die Durchführung einer Schnitzeljagd mit Augmented-Reality-Unterstützung entwickelt.

Im Folgenden wird ein umfassender Einblick in die technischen Details der Projektrealisierung dargelegt, angefangen mit einer Einführung in projektrelevante Frameworks, Technologien und Libraries, der Konzeption und Entwurf, bis hin zur konkreten Implementierung und im Projektdurchlauf entstandene Hindernisse.

Im Rahmen der Projektarbeit wurden vielfältige Aspekte der Software-Entwicklung berührt, wobei der Schwerpunkt auf der Entwicklung mit Micro-Services, Interprozesskommunikation über REST-API und dem Umgang mit Augmented-Reality-Bibliotheken lag.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Listings

Abkürzungsverzeichnis

1. Einleitung

1.1. Motivation

Die ersten Tage und Wochen an der Hochschule können für neue Studierende eine herausfordernde und überwältigende Zeit sein. Sie müssen sich in einer neuen Umgebung zurechtfinden, viele neue Menschen kennenlernen und gleichzeitig den akademischen Anforderungen gerecht werden. Eine effektive Methode, um den Einstieg zu erleichtern, ist eine Einführungstour, die jedoch oft nur begrenzt interaktiv und ansprechend ist.

Traditionelle Einführungstouren sind meist sehr statisch und linear. Studierende folgen einer festgelegten Route und hören passiv zu, während Informationen vermittelt werden. Dies führt oft dazu, dass wichtige Details nicht in Erinnerung bleiben, da die Interaktion und das eigenständige Entdecken fehlt.

Zudem bieten solche Touren selten die Möglichkeit zur aktiven Teilnahme und Mitgestaltung. Die Studierenden sind Zuschauer statt Akteure, was die Aufnahmefähigkeit und das Engagement reduziert. Ohne die Möglichkeit, selbst Entscheidungen zu treffen oder Aufgaben zu lösen, bleibt der Lerneffekt gering und die Tour wird schnell langweilig.

Schließlich ist die Interaktion zwischen den neuen Studierenden bei herkömmlichen Einführungstouren meist eingeschränkt. Da der Fokus auf der Vermittlung von Informationen liegt, bleibt wenig Raum für soziale Interaktionen und Teamarbeit. Somit fällt es den Studierenden schwer, frühzeitig Kontakte zu knüpfen, um eine Lerngruppe zu finden und ein Gemeinschaftsgefühl entwickeln zu können.

Diese Herausforderungen verdeutlichen die Notwendigkeit für innovativere und interaktivere Ansätze, wie sie durch eine Schnitzeljagd geboten werden können.

1.2. Zielsetzung

Das Ziel dieses Projekts ist die Entwicklung eines Systems zur Erstellung, Anmeldung und Durchführung von Schnitzeljagden an der Hochschule Aalen. Durch eine Schnitzeljagd werden die Studierenden nicht nur spielerisch mit den verschiedenen

Gebäuden, Räumen und Einrichtungen vertraut gemacht, sondern auch zur aktiven Teilnahme und Zusammenarbeit angeregt. Dies fördert nicht nur das Verständnis der Campusstruktur, sondern auch das Gemeinschaftsgefühl und den Zusammenhalt unter den neuen Studierenden.

1.3. Anforderungen und Problemabgrenzung

Die Schnitzeljagden sollen in erster Linie den Studienanfängern (Ersties) dienen, um ihnen auf interaktive Weise den Campus nahe zu bringen und sie mit den wichtigsten Orten und Einrichtungen vertraut zu machen.

Im Rahmen des Projekts wurden folgende Ziele in einer Vier-Felder-Matrix aufgeteilt.

Für die erfolgreiche Projektumsetzung sind folgende Eigenschaften zu berücksichtigen.

Flexibilität

Ein wichtiger Aspekt des Projekts ist, dass eine Schnitzeljagd so flexibel wie möglich durchgeführt werden soll. Eine Aufgabenstellung und die dazu gehörende Lösung sollten hierbei entkoppelt und dynamisch erweiterbar sein, ohne einen größeren Aufwand in der Implementierung zu benötigen.

Skalierbarkeit

Das System muss in der Lage sein, mehrere Schnitzeljagden gleichzeitig durchzuführen, ohne dass es zu Problemen kommt. Es sollte möglich sein, die Ressourcen des Systems dynamisch entsprechend der Anzahl der aktuell aktiven Benutzer zu skalieren, ohne dass dabei Performanceeinbußen oder ähnliche Beeinträchtigungen auftreten.

Benutzerfreundlichkeit

Der Anmelde- und Durchführungsprozess sollte unabhängig voneinander klar strukturiert sein, um den Benutzern eine reibungslose und intuitive Erfahrung zu bieten. Während der Durchführung der Schnitzeljagd sollten keine Schwierigkeiten auftreten. Die Schnitzeljagd soll den Benutzern eine einzigartige Erfahrung bieten, ohne durch unnötige oder ablenkende Elemente zu stören. Idealerweise fungiert die

Anwendung als Schnittstelle zur Durchführung der Schnitzeljagd, wobei das Lösen der Aufgaben direkte Interaktionen im realen Leben erfordert.

Die digitale Plattform ermöglicht eine einfache Anmeldung und Durchführung der Schnitzeljagd, wodurch der organisatorische Aufwand minimiert und die Zugänglichkeit maximiert wird.

Sicherheit

Das Thema Sicherheit ist im Bezug auf sensible Benutzerdaten wie Passwörter sehr wichtig. Durch die architektonischen Überlegungen die in Kapitel 4 beschrieben werden ist wäre es möglich, im Anschluss die Benutzerdaten sicher zu speichern durch beispeisweise verschlüsselung. Um den Projektumfang auf das Wesentliche zu reduzieren, wurde dies im Projekt nicht berücksichtigt.

1.4. Vorgehen

Um das beschriebene Projekt zu realisieren, wurden folgende Projekt-Phasen durchlaufen:

Technologische Forschungsphase

Die Durchführung einer Schnitzeljagd sollte über das Smartphone erfolgen. Im Verlauf der Jahre haben sich einige Möglichkeiten, Software für mobile Geräte zu entwickeln, durchgesetzt, die jeweils ihre Vor- und Nachteile besitzen. In dieser Projektphase war es wichtig, die vielen unterschiedlichen Technologien zu erforschen und eine für das Projekt passende Plattform zu wählen, in welcher die Durchführung der Schnitzeljagden erfolgen kann.

Entwurfsphase

Nachdem das grundlegende, projektrelevante technologische Wissen verfeinert worden war, wurde ein erster Entwurf der fundamentalen Architektur vorgestellt. Nachdem dieser ausreichend ausgereift war, konnten erste Workflows der Benutzer erstellt werden. Sobald die Baseline für die Implementierung festgelegt worden war, konnten erste Aufgaben verteilt werden.

Implementierungsphase

Die einzelnen Funktionalitäten wurden daraufhin implementiert und getestet. Hierbei war es entscheidend, die geplanten Features schrittweise zu realisieren und kontinuierlich mit den Anforderungen zu überprüfen.

Reviewphase

2. Grundlagen

2.1. Software Design

2.1.1. Einführung

Software-Design ist ein zentraler Aspekt der Softwareentwicklung, der maßgeblich den Erfolg und die Wartbarkeit eines Projekts beeinflusst. Verschiedene architektonische Patterns bieten Lösungen für wiederkehrende Probleme und helfen Entwicklern, robuste und skalierbare Anwendungen zu erstellen. Robert C. Martin, ein Pionier in der Software-Architektur, betont in seinem Buch *Clean Architecture* die Bedeutung von guten Architekturen:

> The goal of software architecture is to minimize the human resources required to build and maintain the required system. [1]

Dieses Zitat unterstreicht, dass eine durchdachte Architektur nicht nur die Entwicklungszeit verkürzt, sondern auch die langfristige Wartung vereinfacht. In diesem Kontext werden im Folgenden zwei bedeutende architektonische Patterns vorgestellt, die im Rahmen der Projektdurchführung relevant sind: Domain-driven Design (DDD) und Microservices.

2.1.2. Architektonische Patterns und Ansätze

Microservices

Unter dem Begriff *Microservices* versteht sich der Ansatz, ein Software-System als Orchestration mehrerer Dienste zu unterteilen. Jeder Dienst ist für eine Aufgabe des gesamten System-Kontexts verantwortlich und kann unabhängig von anderen Diensten als ein eigener Prozess laufen. [2]

Domain Driven Design

Das *DDD* ist eine Methodologie, die hauptsächlich bei komplexen Softwareprojekten zum Einsatz kommt. Es gehört weder zur Kategorie der Frameworks noch zu den architektonischen Patterns. Vielmehr ist es eng mit der Denkweise der Microservices, wie in Kapitel 2.1.2 beschrieben, verbunden. Domain-driven Design konzentriert sich darauf, Aktivitäten und Prozesse aus dem realen Leben in die Softwareentwicklung zu abstrahieren. [2]

Onion-Architecture

Die *Onion-Architecture*, auch bekannt als *Clean Architecture*, wurde von Jeffrey Palermo entwickelt und wird insbesondere für C#-Projekte von Microsoft empfohlen. Sie zielt darauf ab, die typischen Probleme monolithischer Anwendungen, wie hohe Kopplung und geringe Wartbarkeit, zu vermeiden. Die Architektur visualisiert die Software als konzentrische Schichten, vergleichbar mit den Schichten einer Zwiebel. Folgende Abbildung stellt diesen Aufbau nochmals dar:

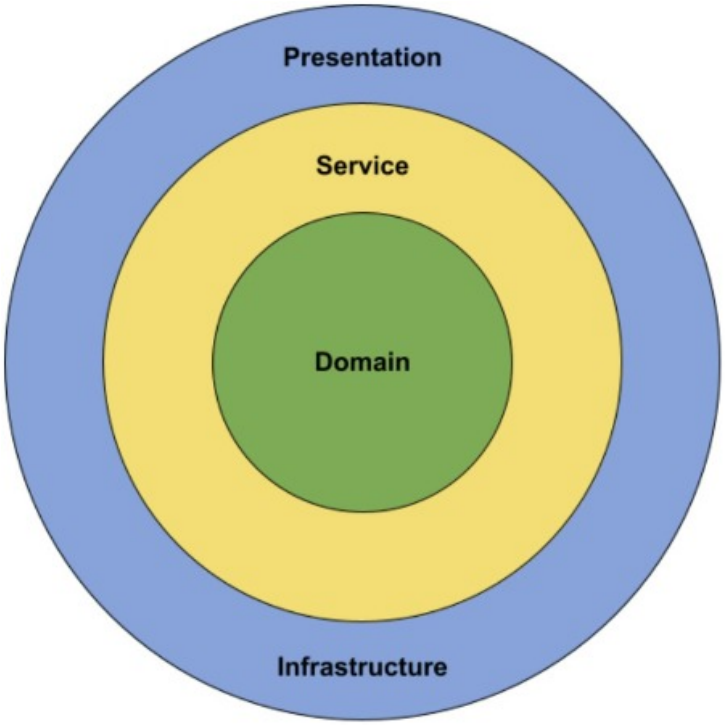

Abbildung 2.1.: Bild Onion Architecture, nach [3]

Die Onion-Architecture gliedert sich in vier Hauptschichten:

- **Domain Layer:** Im Zentrum steht die Domäne, die Geschäftslogik und Geschäftsregeln beinhaltet und keinerlei Abhängigkeiten zu äußeren Schichten hat.
- **Service Layer:** Diese Schicht enthält die Anwendungsspezifische Logik und nutzt die in der Domain Layer definierten Schnittstellen.
- **Infrastructure Layer:** Hier befinden sich Implementierungen für Datenzugriff, Netzwerkkommunikation und andere externe Dienste.
- **Presentation Layer:** Diese Schicht ist für die Benutzeroberfläche und die Präsentation der Daten verantwortlich.

Konzepte der Onion-Architecture fördern die Abhängigkeit von Abstraktionen (Interfaces) anstatt von konkreten Implementierungen. Diese Abhängigkeitsinversion erlaubt es, Implementierungen zur Laufzeit auszutauschen, was die Flexibilität und Erweiterbarkeit der Software erhöht.

Die Vorteile der Onion-Architecture sind:

- **Hohe Testbarkeit:** Da alle Schichten über definierte Schnittstellen kommunizieren, können einzelne Komponenten leicht isoliert und getestet werden.
- **Klare Abhängigkeiten:** Abhängigkeiten fließen strikt in Richtung der zentralen Domänenschicht, wodurch höhere Schichten die Implementierungen der unteren Schichten verwenden können, aber nicht umgekehrt.
- **Trennung von Geschäftslogik und Implementierungsdetails:** Die Geschäftslogik kann unabhängig von technischen Implementierungsdetails entwickelt werden. Notwendige Schnittstellen zu externen Systemen werden definiert, aber deren konkrete Implementierung wird in den äußeren Schichten gekapselt.

Diese Struktur ermöglicht es, komplexe Anwendungen modular zu entwickeln und erleichtert die Wartung und Weiterentwicklung der Software [3].

2.1.3. Kollaboration mehrerer Dienste

Eventbasierte Architektur: Message Broker und Message Bus

Eine eventbasierte Architektur ermöglicht es, Nachrichten schnell und flexibel zwischen mehreren Diensten auszutauschen. Ein verbreiteter Ansatz hierfür ist

die Nutzung eines *Message-Brokers*. Im Folgenden werden die vier grundlegenden Aspekte dieser Architektur beschrieben (vgl. [4]):

1. **Initiierendes Event**: Ein *Ereignis*, das den Event-Fluss anstößt und an einen Event-Kanal des Event-Brokers gesendet wird.
2. **Event-Broker**: Ein *Orchestrator*, der mehrere Kanäle verwaltet, auf denen Event-Prozessoren lauschen können.
3. **Event-Prozessor**: Eine Einheit, die auf einem Kanal lauscht und ein eingehendes Event verarbeitet. Dabei kann ein neues (verarbeitetes) Event erzeugt und an einen anderen Kanal gesendet werden.
4. **Das zu verarbeitende Event**: Ein Ereignis, das durch die beschriebenen Mechanismen verarbeitet oder erzeugt wird.

Vorteile

1. **Architektonische Erweiterbarkeit**: Events können vorläufig erzeugt und an den Message-Broker gesendet werden, ohne dass der Event-Prozessor bereits implementiert sein muss. Dies ermöglicht die spätere Implementierung des Prozessors.
2. **Abkapselung**: Jedes Modul ist für seine eigene Funktionalität verantwortlich, während alles außerhalb Bestandteil anderer Module ist.
3. **Skalierbarkeit**: Events können von mehreren (gleichen) Event-Prozessoren verarbeitet werden. Der Kanal fungiert als FIFO-Warteschlange und ermöglicht eine notwendige Synchronisation.
4. **Asynchronität**: Die Kommunikation erfolgt asynchron, was die Entkopplung der Dienste und die Verbesserung der Systemleistung ermöglicht.

Nachteile

1. **(De-)Marshalling**: Nachrichten müssen in ein anderes Format (z.B. JSON, String, Byte-Array) konvertiert und wieder zurückkonvertiert werden. Dies kann die Performance beeinträchtigen.
2. **Policies**: In einer traditionellen Message-Broking-Architektur kann theoretisch jeder auf einen Kanal *subscriben*. Es gibt keine klaren Schichten zur Trennung der Zugriffsrechte.
3. **Komplexität**: Das Management mehrerer Module und die Nachverfolgung von *Event-Publishern* und *Event-Subscriber* können schwierig sein.

RESTful APIs

Für die Kommunikation und den Datenaustausch zwischen mehreren Diensten stehen verschiedene Kommunikations-Protokolle zur Verfügung. HTTP ist ein standardisiertes Kommunikationsprotokoll und ist in Bereichen der Web-Anwendungen weit verbreitet.

Representational State Transfer (REST) definiert kein neues Kommunikationsprotokoll. Es ist ledigliche eine Sammlung von Architekturbeschränkungen und definiert Regeln, die der Entwickler befolgen muss, um eine Zustandslosigkeit in der Kommunikation zu erfüllen [5].

2.2. Technologien

Für die Implementierung wurden zahlreiche Technologien der Software-Entwicklung in Betracht gezogen. In diesem Abschnitt werden die für das Projekt relevanten Frameworks und Plattformen beschrieben.

2.2.1. ASP.NET

ASP.NET ist ein Open-Source-Web-Framework, das von Microsoft entwickelt wurde und die Erstellung moderner, skalierbarer und leistungsfähiger Webanwendungen ermöglicht. Zudem werden in ASP.NET moderne Webtechnologien und -standards unterstützt, wodurch sich die Entwicklung von interaktiven und responsiven Webanwendungen erleichtert. Dies schließt auch APIs und Echtzeit-Kommunikation ein, die für die Anwendung relevant sein könnten. Diese Merkmale ermöglichen es auch, die Anwendung modular zu ergänzen, indem externe APIs wie OpenStreetMaps oder andere Dienste einfach angebunden werden können.

Entity Framework Core

Entity Framework Core (EF Core) ist ein leichtgewichtiges, erweiterbares und Open-Source-Object-Relational Mapping (ORM) Framework für .NET, das entwickelt wurde, um den Datenzugriff und die Datenmanipulation in .NET-Anwendungen zu vereinfachen. EF Core bietet Datenbankunabhängigkeit, da es verschiedene Datenbanksysteme wie SQL Server, MySQL, PostgreSQL und SQLite unterstützt.

Ein weiterer Vorteil von EF Core ist Möglichkeit, das Datenbankschema aus konkreten Models über Source-Code zu generieren, ohne *Create-Table* SQL-Statements schreiben zu müssen. EF Core bietet hierbei eine einfach zu verstehende *Fluent-API*

an. Zudem ermöglicht EF Core die einfache Verwaltung von Datenbankmigrationen, was es erlaubt, Änderungen am Datenmodell nachzuverfolgen und auf die Datenbank anzuwenden. Dies erleichtert die Wartung und Weiterentwicklung erheblich.

Ein zusätzliches Merkmal von EF Core ist die Integration von Language Integrated Query (LINQ), die es ermöglicht, komplexe Abfragen auf eine intuitive und typsichere Weise zu schreiben. Dies verbessert die Lesbarkeit und Wartbarkeit des Codes erheblich. Schließlich bietet EF Core verschiedene Mechanismen zur Performanceoptimierung, wie zum Beispiel asynchrone Abfragen und Caching-Strategien.

2.2.2. Unity und Mobile-Apps

Unity ist eine Plattform zur Entwicklung und Darstellung interaktiver 3D-Inhalte, die in vielen Bereichen wie Spieleentwicklung, Filmproduktion, Architekturvisualisierung und Virtual Reality (VR) eingesetzt wird. Sie ist eine der am weitesten verbreiteten Spiel-Engines weltweit und bietet eine Vielzahl an Werkzeugen und Funktionen, für das implementieren immersiver und realistischer Umgebungen.

Unity basiert auf einer Engine, die eine umfassende Sammlung von Software-Bibliotheken bereitstellt. Diese Bibliotheken ermöglichen die Verarbeitung von Grafiken, Physik, Sound und Benutzereingaben. Der Kern der Engine ist für die Rendern von 2D- und 3D-Grafiken verantwortlich, die in Echtzeit berechnet und auf dem Bildschirm angezeigt werden. Unity unterstützt die Programmiersprachen C# und UnityScript (eine Art JavaScript, welche allerdings nicht mehr weiterentwickelt wird), wobei C# die primäre Sprache für die Entwicklung in Unity ist.

AR-Foundation

Für das Erstellen von Augmented-Reality Anwendungen bietet Unity unter der *AR Foundation* grundlegende Bausteine an. Über vorgefertigte Projekt-Templates lässt sich eine funktionierende und ausführbare Demo-Anwendung für Android-Geräte generieren. Das AR Foundation Framework ermöglicht das Einbinden von Features wie beispielsweise Oberflächen-Erkennung, Objekt-, Bild- und Gesicht-Verfolgung sowie Sitzungs- und Geräte-Management [6].

Nützliche Bibliotheken

Newtonsoft.Json.NET ist eine in der Praxis verwendete Bibliothek in C# bzw. .NET Anwendungen für das Serialisieren und Deserialisieren von Objekten im JSON-Format. Für die Verwendung in Unity-Anwendungen existiert ein GitHub-Fork

(*Json.Net.Unity3D*) und ermöglicht das Einbinden über eine `.unitypackage` Datei. [7]

AR+GPS Location ist eine Utility aus dem Unity-Asset-Store und ermöglicht eine Übersetzung von GPS-Koordinaten im Unity-Raum. Hierdurch können Objekte über die Angabe eines realen GPS-Standorts im Unity-Raum plaziert werden. Wenn die Anwendung an der realen Stelle des angegebenen GPS-Standorts geöffnet wird, wird das in Unity angelegte Objekt am richtigen Ort angezeigt. [8]

2.2.3. Svelte und Web-Apps

Svelte ist ein modernes JavaScript-Framework zur Erstellung von Benutzeroberflächen. Im Gegensatz zu herkömmlichen Frameworks wie React oder Vue, die den Großteil ihrer Arbeit im Browser ausführen, verschiebt Svelte diese Arbeit in die Kompilierungsphase. Dies bedeutet, dass der Code während des Build-Prozesses in effizientes, optimiertes JavaScript umgewandelt wird, das direkt im Browser ausgeführt wird. Dadurch wird die Laufzeitbelastung erheblich reduziert, was zu einer besseren Performance und geringeren Ladezeiten führt [9].

Ein wesentlicher Vorteil von Svelte ist seine einfache Syntax, die es Entwicklern ermöglicht, reaktiven Code zu schreiben, ohne auf komplexe State-Management-Lösungen zurückzugreifen. Die Komponenten in Svelte bestehen aus HTML, CSS und JavaScript, was die Lernkurve für neue Benutzer verringert und die Entwicklungserfahrung vereinfacht [9].

SvelteKit ist ein Framework für den Aufbau von Svelte-Anwendungen. Es erweitert die Fähigkeiten von Svelte, indem es zusätzliche Werkzeuge und Funktionen bereitstellt, die speziell für die Entwicklung komplexer, leistungsstarker Webanwendungen notwendig sind. SvelteKit vereinfacht die Einrichtung und Strukturierung von Projekten und bietet Funktionen wie Routing, Server-Side Rendering (SSR), statische Seitengenerierung und eine integrierte Entwicklungsumgebung [9].

Vorteile

Svelte integriert CSS-Styles direkt in die Komponenten mittels <*style*>-*Tags*. Dies eliminiert die Notwendigkeit zusätzlicher Setups.

Alternativ kann auf bereits existierende CSS-Bibliotheken wie TailwindCSS zurückgegriffen werden. Standardmäßig werden Styles scoped angelegt, sodass keine Konflikte zwischen den Komponenten entstehen. Dies sorgt für eine sichere und wartbare Codebase einzelner Komponenten aufgrund der reduzierten Abhängigkeiten.

Svelte-Pages sind standardmäßig mit einem einfachen Prerendering ausgestattet, das sich über das Setzen eines Attributes steuern lässt. SvelteKit bietet zudem einfache Einstellungen, um verschiedene Startpfade für das Prerendering zu definieren, was die Flexibilität bei der Konfiguration erhöht.

Ein weiteres zentrales Feature von Svelte ist die Reaktivität. Durch das Definieren von sogenannten *Reactive Statements* kann ein Ablauf beschrieben werden, der ausgeführt wird, wenn eine Abhängigkeit innerhalb dieses Blocks aktualisiert wird. Dies erleichtert das Management von Zustandsänderungen und sorgt für reaktive und dynamische Benutzeroberflächen.

Svelte besitzt eingebaute Prüfungen für Barrierefreiheit (*Accessibility-Linting*), die auf offensichtliche und auch auf komplexere Probleme hinweisen. Dies unterstützt Entwickler dabei, von Anfang an barrierefreie Anwendungen zu erstellen.

Letztendlich reduziert sich die Menge des notwendigen Codes erheblich durch die deklarative Natur von Svelte. Diese führt zu einer klareren und übersichtlicheren Codebasis, was die Entwicklung und Wartung von Anwendungen vereinfacht [10].

Nachteile

Die Community-Größe von Svelte ist im Vergleich zu bekannteren JavaScript-Frameworks wie React oder Angular aufgrund der Neuheit des Frameworks noch relativ klein. Dies kann die Problemlösung erschweren, da einige Fehler möglicherweise nicht so gut dokumentiert oder gelöst sind und somit weniger Ressourcen auf Foren wie Stack Overflow zur Verfügung stehen.

Ein weiterer Nachteil ist das fehlende Entwickler-Tooling oder eine Command Line Interface (CLI). Bei Angular können Komponenten oder Dienste beispielsweise über die Angular CLI schnell und effizient erstellt werden. Diese Art von Tooling fehlt bei Svelte, was den Entwicklungsprozess etwas weniger komfortabel machen kann.

Zudem ist die Verfügbarkeit und Nutzbarkeit von bereits vorhandenen Libraries eingeschränkt. Obwohl diese Libraries nahtlos mit einem Paketmanager (z.B. npm von Node.js) installiert werden können, erfordert es oft zusätzlichen Aufwand, um sie in Svelte-Komponenten zu integrieren [11].

3. Problemanalyse

Die Analyse des zu lösenden Problems ist Grundlage für jedes ingenieurmäßige Vorgehen. Daher soll in diesem Kapitel das zu lösenden Problem auf Basis des im Grundlagenkapitel aufbereiteten Wissens analysiert werden. Hierzu ist insbesondere notwendig zu klären, wie sich das Gesamtproblem in Teilprobleme zerlegen lässt und welche Abhängigkeiten zwischen diesen bestehen.

4. Software-Entwurf

4.1. Blockansicht

Im folgenden ist die Struktur (*High-Level-Ansicht*) des Software-Systems in Form eines Blackbox-Diagrams dargestellt.

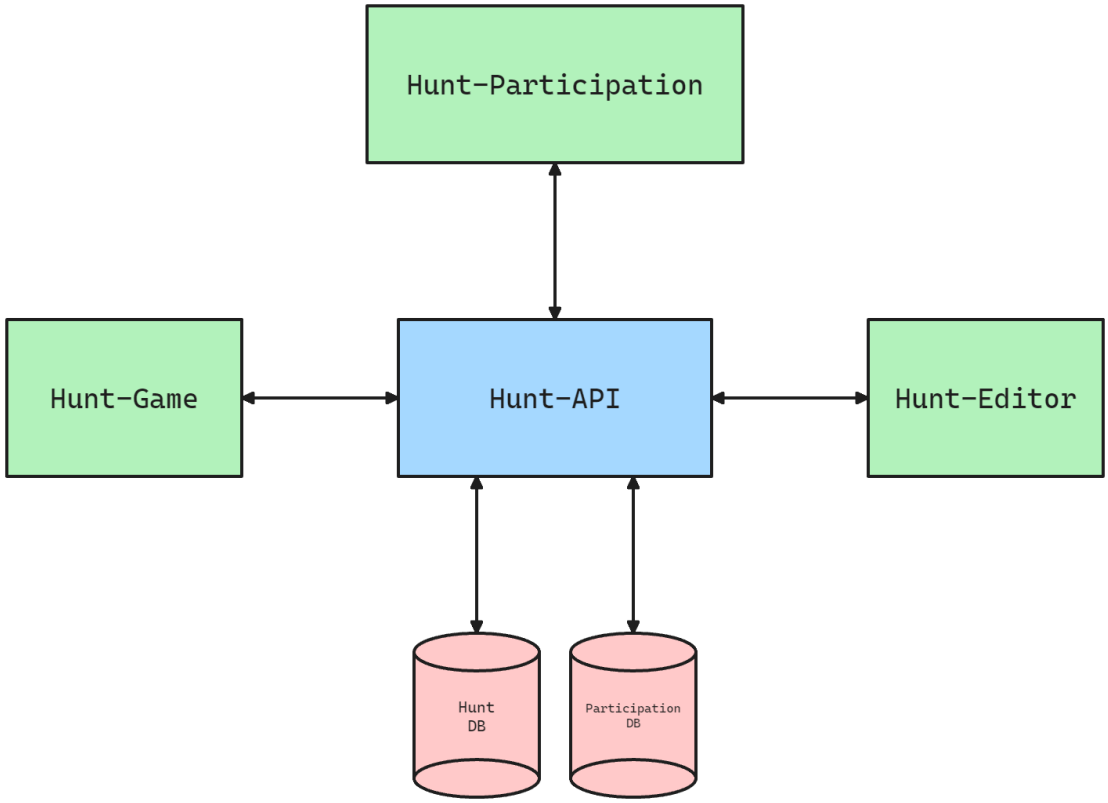

Abbildung 4.1.: Bild Systemkontext als Blackbox-Diagram

Das in Abbildung 4.1 dargestellte Blackbox-Diagramm beschreibt den Zusammenhang der unterschiedlichen Subsysteme zueinander. Die verschiedenen Aufgaben, die das System zu leisten hat, wurden als einzelne Anwendungen vorhergesehen. Diese sind in Abbildung 4.1 grün markiert. Dies ermöglicht eine klare Trennung

der verschiedenen Domänen (Erstellung, Anmeldung, Durchführung). Das Backend steht als zentrale Schnittstelle für die Bereitstellung Systemspezifischer Funktionalitäten zur Verfügung. Dieses sind in Abbildung 4.1 blau markiert. Für die Datenpersistierung besitzt das Backend zwei Datenbank-Verbindungen. Dieses sind in Abbildung 4.1 rot markiert.

4.2. Subsysteme

4.2.1. Hunt-Api

Übersicht

Anhand einer zentral zur Verfügung stehenden Schnittstelle *Hunt-Api* können die verschiedenen Anwendungen zur Funktionalität des Systems beitragen. Die Schnittstelle bietet verschiedene Endpunkte für die Verwaltung von Schnitzeljagden, Teilnahmen und die bewältigten Aufgaben der Teilnehmer. Eine Darstellung der Hunt-API Architektur ist in Abbildung 4.2 näher beschrieben.

Anhand der zentral zur Verfügung stehenden Schnittstelle *Hunt-Api* können die verschiedenen Anwendungen zur Funktionalität des Systems beitragen. Diese Schnittstelle bietet diverse Endpunkte für die Verwaltung von Schnitzeljagden, Teilnahmen und die bearbeiteten Aufgaben der Teilnehmer. Eine allgemeine Übersicht der Hunt-Api-Architektur ist in Abbildung 4.2 beschrieben.

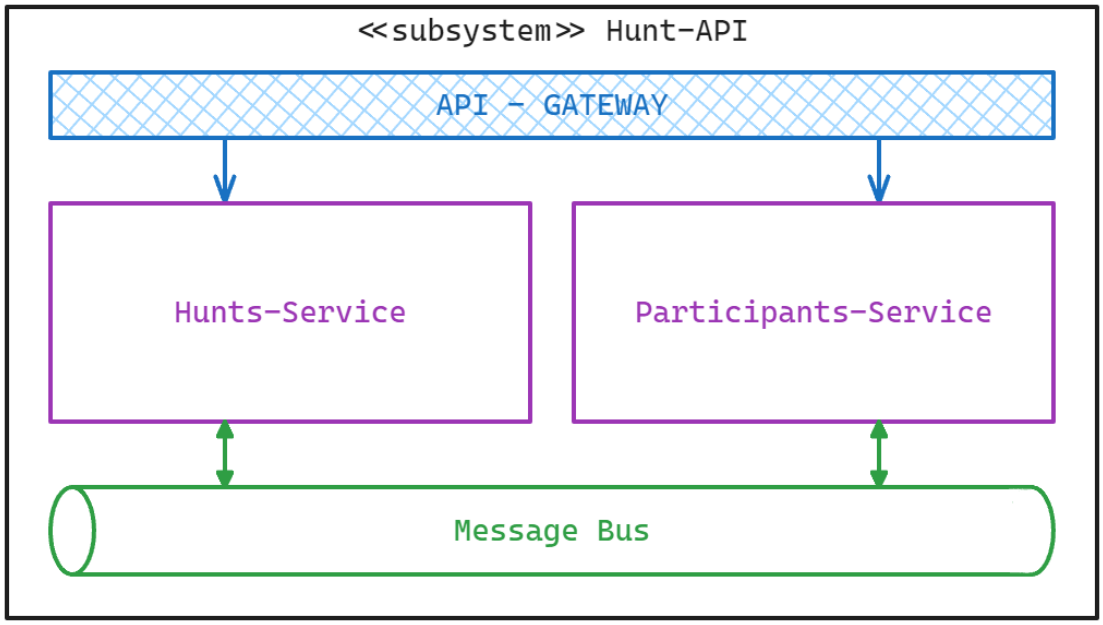

Abbildung 4.2.: Bild Hunt-Api Subsystem

Als Architekturmuster wurde ein domänenorientierter Ansatz gewählt, der in Kapitel 2.1.2 näher erläutert wird. Jede Domäne repräsentiert dabei einen spezifischen Teilaspekt des Gesamtsystems. Die ausgewählten Domänen, *Hunts* und *Participants*, sind in Abbildung 4.2 lila hervorgehoben unr repräsentieren den jeweiligen Dienst (Microservice).

Für die Kommunikation unterschiedlicher Dienste ist ein Message-Bus vorhergesehen, worüber eventgesteuert Nachrichten ausgestauscht werden. Dieser ist in Abbildung 4.2 grün hervorgehoben.

Um eine einheitliche Schnittstelle bereitzustellen, die anwendungsübergreifend genutzt werden kann, wurde ein Api-Gateway vorhergesehen. Dieses Gateway ermöglicht es, Anfragen an das Backend zu stellen und sie an die entsprechenden Dienste weiterzuleiten. Das API-Gateway ist in Abbildung 4.2 blau gekennzeichnet.

Hunts-Service

Für die Verwaltung der erstellten Schnitzeljagden eines Organisators, stellt der Hunts-Service eine Schnittstelle hierfür bereit. Abbildung 4.3 zeigt die Struktur des Dienstes.

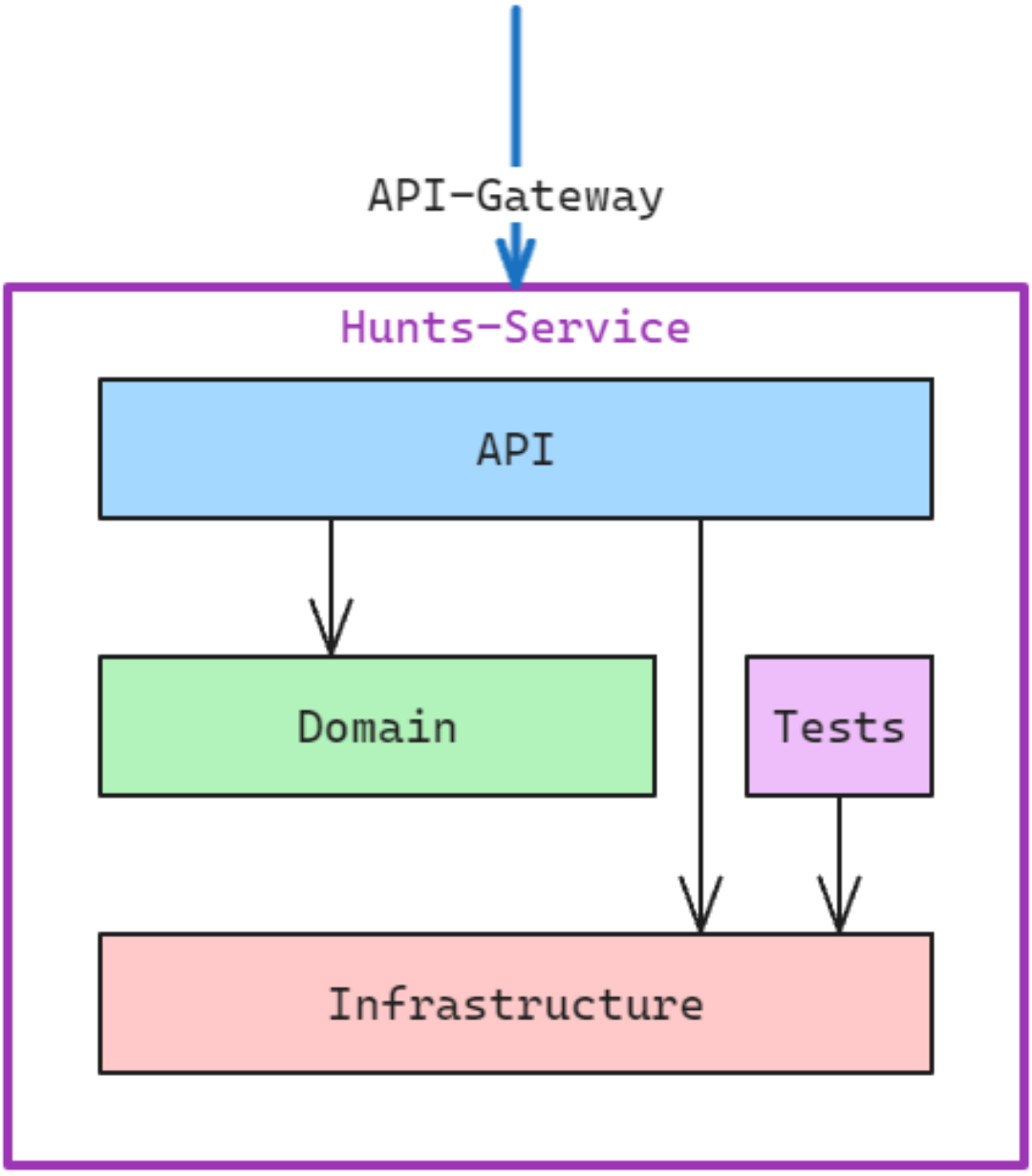

Abbildung 4.3.: Bild Hunts Microservice

Innerhalb des Hunts-Service wurde ein schichtenorientierter Ansatz gewählt, welche Ähnlichkeiten mit der in Kapitel 2.1.2 beschriebenen *Onion-Architecture* teilt. Jede Schicht entspricht einer horizontalen Teilung der unterschiedlichen Anwendungs-Aspekte. Für die Persistierung der Schnitzeljagden (Hunts) wurde das Repository-Pattern vorhergesehen.

Die grundlegende Funktionalität der Domäne (*Hunts*, *Assignments*, etc.) wird in der *Domain-Schicht* zur Verfügung gestellt, welche in Abbildung 4.3 grün gekennzeichnet wird. Hier sind Modelle und Entities, sowie Datentypen, Enumerations und Repository-Interfaces definiert, die im System durchweg Verwendung finden. Die Domain-Schicht ist abgekapselt von Anwendungs- und Infrastrukturlogik und besitzt daher keine Abhängigkeiten zu externen Modulen.

Eine Implementierung der Repository-Interfaces wird in der *Infrastruktur-Schicht* (Infrastructure) zur Verfügung gestellt, die in Abbildung 4.3 rot hervorgehoben wird. Diese kann zudem unabhängig von der Domänen-Logik isoliert getestet werden, wie in in Abbildung 4.3 lila dargestellt wird.

Die *Anwendungs-Schicht* ist in Abbildung 4.3 blau hervorgehoben und bündelt die Funktionalität der Domain-Schicht und Infrastruktur-Schicht gemeinsam. Über eine einheitliche Schnittstelle (*api/Hunt*) können Schnitzeljagden erstellt, bearbeitet, gelöscht und aufgelistet werden.

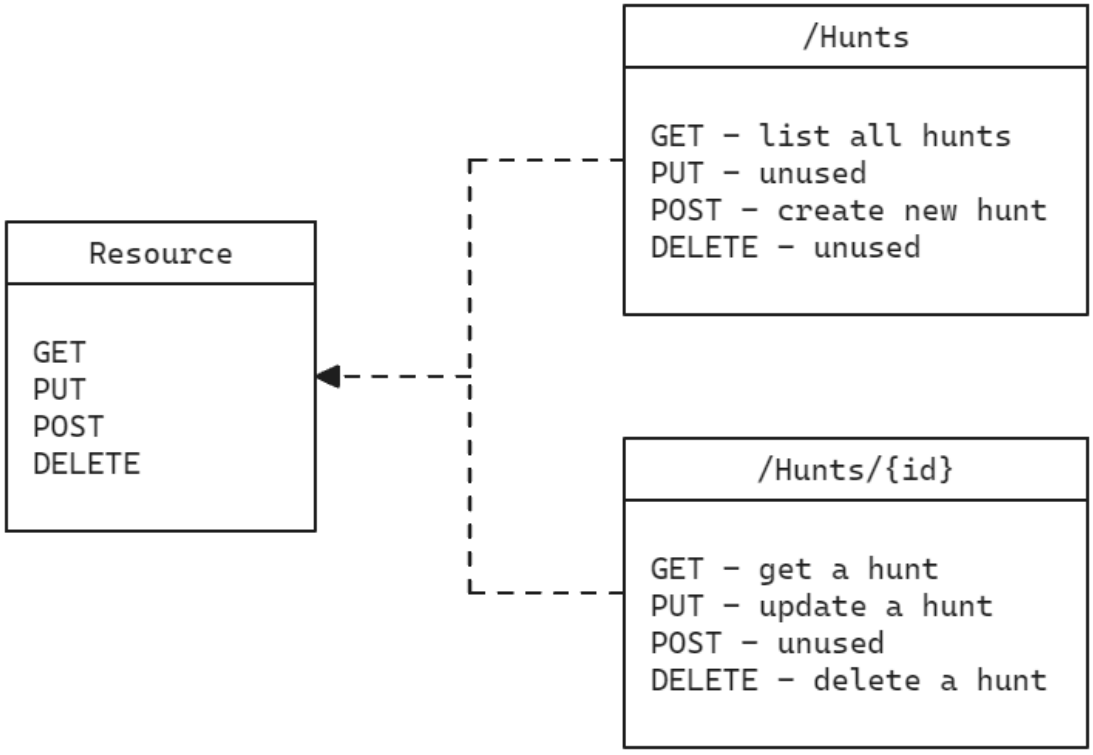

Abbildung 4.4.: Skizze der Hunts Ressourcenansicht

In Abbildung 4.4 sind die unterschiedlichen Operationen auf Schnitzeljagden (*Hunts*) anhand des ressourcen-orientierten Ansatz aus Kapitel 2.1.3 dargestellt.

Participants-Service

Damit ein Teilnehmer an einer Schnitzeljagd teilnehmen und diese auch durchführen kann, soll ihm die Anmelde- und Spielfunktionalität zur Verfügung gestellt werden. Dies wird über den Participants-Service ermöglicht. Abbildung 4.5 zeigt die Struktur des Dienstes.

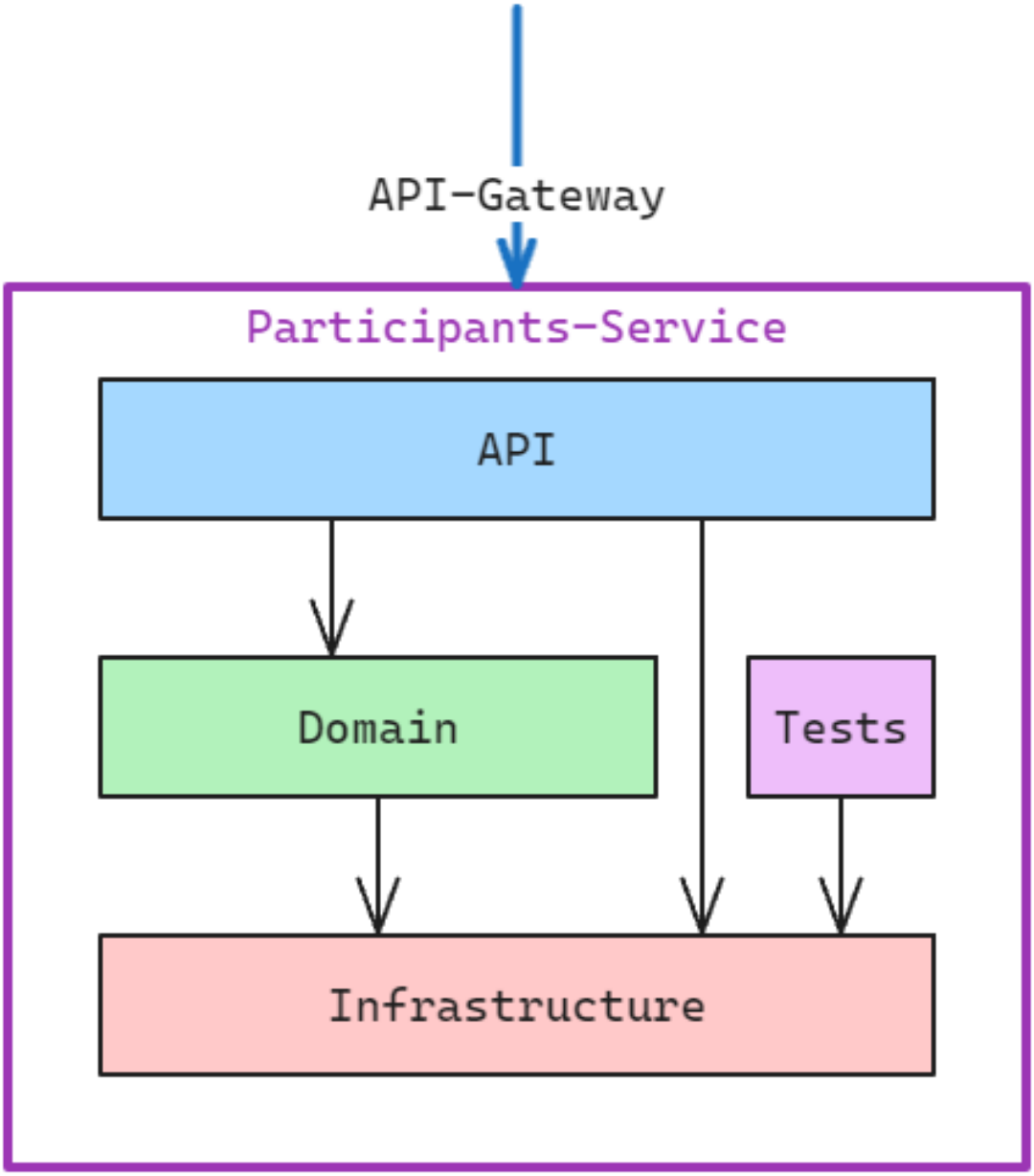

Abbildung 4.5.: Bild Participants Microservice

Der Participants-Service besitzt eine ähnliche Struktur wie der Hunts-Service aus Kapitel 4.2.1 und erfüllt die Aufgabe der Teilnahmen-Verwaltung.

4.2.2. Datenbanken-Modell

Übersicht

Hunts-Datenbank

Participants-Datenbank

4.2.3. Hunt-Game

Entscheidung Web-App anstatt Unity

Vorteile der Web-App gegenüber Unity

Unity ist eine leistungsfähige Engine, die für die Entwicklung von 3D-Spielen und AR-Anwendungen bekannt ist. Die Implementierung von AR-Funktionen in Unity kann jedoch erheblich komplexer sein als die Entwicklung einer Web-App. Dies liegt an mehreren Faktoren:

- **Entwicklungsaufwand**: Die AR-Funktionalität in Unity erfordert umfangreiche Kenntnisse in der Programmierung von AR-Erfahrungen, was die Entwicklungszeit verlängern und den Aufwand erhöhen kann. Die Handhabung von AR-spezifischen Aspekten wie der Positionierung virtueller Objekte im Raum und der Verarbeitung von Sensordaten kann zusätzliche Komplexität und Entwicklungszeit bedeuten.
- **Coroutines und yield**: Unity verwendet Coroutines, um asynchrone Operationen durchzuführen. Diese Mechanismen ermöglichen es, lange Operationen, wie das Warten auf eine Antwort von einem HTTP-Request, ohne das Einfrieren der gesamten Anwendung durchzuführen. Das Management von Coroutines erfordert jedoch ein spezifisches Verständnis der Unity-eigenen Programmierschnittstellen und kann zu komplexen und schwer nachvollziehbaren Code-Strukturen führen.

Das Styling und die Gestaltung von Benutzeroberflächen in Unity unterscheidet sich erheblich von der Webentwicklung. Unity verwendet ein eigenes System für die Gestaltung von UI-Elementen, das auf dem Unity Editor basiert und spezielle Komponenten wie Canvas, Panels und UI-Elemente umfasst. Für spezifische visuelle Effekte sind oft eigene Shader und Materialien erforderlich, was die Komplexität

weiter erhöht. Diese benötigen tiefere Kenntnisse in der grafischen Programmierung und können den Entwicklungsprozess verlangsamen.

Preisstruktur von Unity
Die im September 2023 von Unity angekündigten Preisänderungen, die eine neue Laufzeitgebühr für kommerziell erfolgreiche Projekte einführten, haben die Kostenstruktur erheblich beeinflusst. Diese Gebühren sind an die Anzahl der Installationen und den jährlichen Umsatz gebunden, was insbesondere für erfolgreichere Entwickler zu einer zusätzlichen finanziellen Belastung führen kann. Die Einführung dieser Gebühren, die auch nachträglich angewendet werden können, stieß auf breite Kritik und Unsicherheit innerhalb der Entwicklergemeinschaft.

Für kleinere Projekte oder Entwickler, die nicht die Umsatz- oder Installationsschwellen erreichen, mag die neue Preisstruktur von Unity zunächst weniger relevant erscheinen. Dennoch wirft die Ungewissheit über zukünftige Preisanpassungen ernste Bedenken auf. Es ist gut möglich, dass Unity seine Preispolitik weiter verschärft, um zusätzliche Einnahmen zu erzielen. Solche Änderungen könnten die Kosten für Entwickler erheblich steigern und zusätzliche Unsicherheiten mit sich bringen. Dies könnte für das Projekt riskant sein, da dieses in der Hochschule möglichst lang eingesetzt werden sollte, ohne dafür große Lizenzkosten tragen zu müssen.

WebAR als Alternative
Eine vielversprechende Alternative zur nativen AR-Entwicklung mit Unity ist WebAR. WebAR ermöglicht AR-Erlebnisse direkt im Webbrowser ohne die Notwendigkeit einer separaten App-Installation. Dies bietet mehrere Vorteile:

- **Zugänglichkeit**: WebAR ist auf jedem modernen Webbrowser verfügbar, was die Zugänglichkeit für Benutzer erleichtert und die Notwendigkeit, eine spezielle App herunterzuladen, beseitigt. Dies kann die Benutzerfreundlichkeit und die Reichweite der Anwendung erhöhen.
- **Einfachere Entwicklung**: WebAR nutzt Web-Technologien wie JavaScript und WebGL, die für viele Entwickler vertrauter und zugänglicher sind als die speziellen Tools und Skripte von Unity. Die Entwicklung kann daher schneller und einfacher sein, besonders wenn bereits Erfahrung mit Web-Technologien besteht.
- **Kosteneffizienz**: Da WebAR keine zusätzlichen Lizenzgebühren oder Laufzeitgebühren erfordert, ist es eine kostengünstigere Lösung.

Übersicht

4.2.4. Hunt-Editor

Übersicht

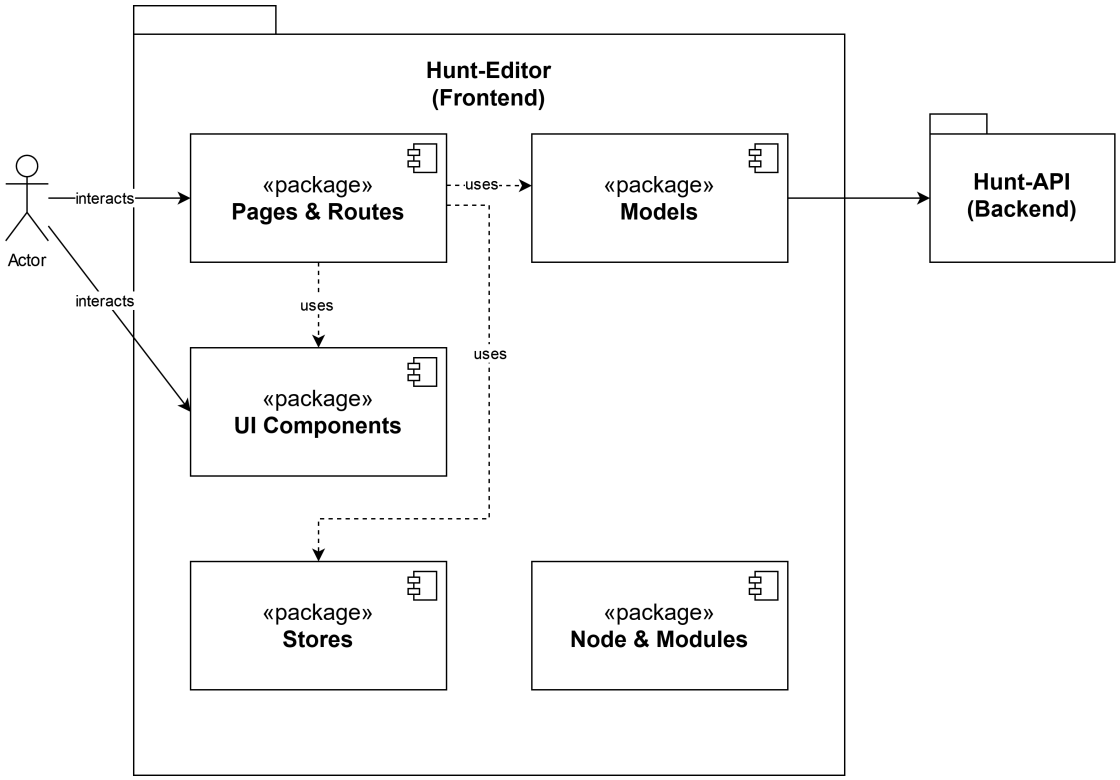

Abbildung 4.6.: UML-Diagramm Whitebox Frontend Hunt-Editor

Pages & Routes

Eine Svelte-Anwendung ist verzeichnisbasiert aufgebaut. Jedes Verzeichnis entspricht einer Route und kann ein oder mehrere Unterverzeichnisse haben, die die Route erweitern. Jede Route kann eine Seite haben, die immer den Namen *+page.svelte* trägt. Svelte verwendet diese Seiten als Hauptanzeige, wenn zur zugehörigen Route navigiert wird.

Im Folgenden ist die Struktur des Routings als Baumdiagramm dargestellt.

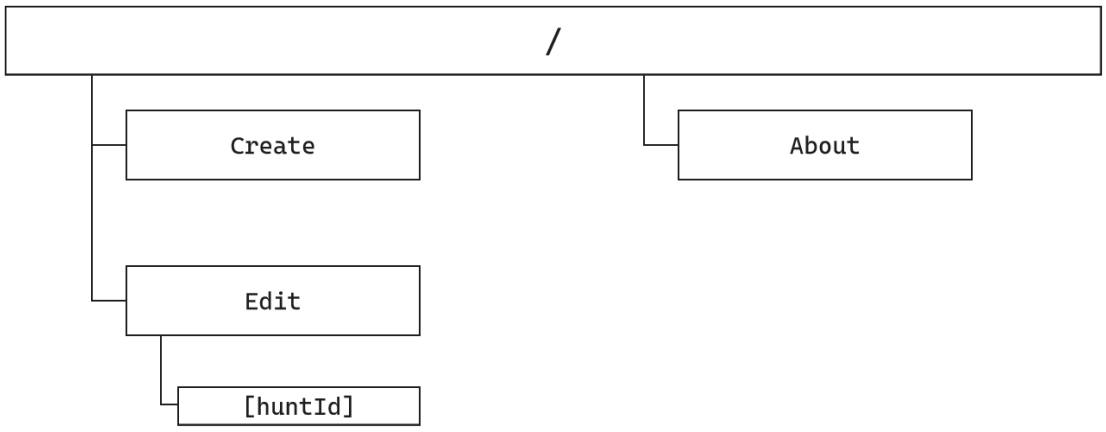

Abbildung 4.7.: Skizze Frontend Hunt-Editor Routes

Für die Anzeige und das Entfernen von Schnitzeljagden ist die Home Route / vorgesehen. Auf der Route */about* werden Informationen zur Anwendung und zum Projekt bereitgestellt.

Der Workflow zum Erstellen einer Schnitzeljagd ist in der Route */create* implementiert.

Zum Editieren einer Schnitzeljagd werden die Komponenten aus der */create*-Route wiederverwendet, hier wird jedoch über die */edit/[huntId]*-Route gearbeitet, wobei *[huntId]* ein Platzhalter für die *Id* der jeweiligen Schnitzeljagd ist. Dadurch können dynamisch über die URL Informationen zur Schnitzeljagd abgerufen werden. Diese Informationen (wie bspw. Titel und Beschreibung) werden dann in die Eingabefelder eingetragen.

UI Components

Für die Darstellung, Wiederverwendbarkeit und Konsistenz über verschiedene Seiten hinweg werden Svelte-Komponenten verwendet. Diese Komponenten bilden die Bausteine zur Anzeige unterschiedlicher Daten. Zusätzlich wurde die UI Component Library *flowbite-svelte* verwendet, um standardisierte und ansprechende Benutzeroberflächenelemente wie Buttons, Inputs, Tabellen und andere UI-Komponenten effizient zu implementieren. Diese Bibliothek erleichtert die Entwicklung und trägt dazu bei, eine einheitliche und intuitive Benutzererfahrung sicherzustellen.

Jede Svelte-Komponente besteht aus einem Skript-Teil und einem Design-/Style-Teil. Eine Komponente kann auch weitere Komponenten einbinden. In diesem Fall fungiert die übergeordnete Komponente als Elternteil (Parent), während die eingebundenen Komponenten als Kinder (Children) bezeichnet werden.

Stores

Um während des Erstellungs- und Bearbeitungsprozesses Daten zu speichern, wird ein Svelte Store verwendet. Dies ermöglicht es verschiedenen Komponenten, auf diese Daten zuzugreifen. Beispielsweise kann die Komponente um Basisdaten einzugeben auf Titel und Beschreibung zugreifen, während die Komponente zum Anlegen/Bearbeiten der Aufgaben auf die Aufgaben zugreift.

Node & Node Modules

Node.js ist eine JavaScript-Laufzeitumgebung, die serverseitige Anwendungen ermöglicht. Sie stellt über den Paketmanager "*npm*"Werkzeuge zur Verwaltung und Bereitstellung von externen Paket-Abhängigkeiten sowie Entwickler-Tools bereit. Neben dem Betrieb eines Webservers ermöglicht *Node.js* auch die serverseitige Verarbeitung von API-Anfragen und die Anbindung an Datenbanken.

Eine Übersicht der einzelnen Paketabhängigkeiten wird in Kapitel 4.2.4 gegeben.

Wireframing

Um den Editor zu entwickeln, der das Anlegen und Verwalten von Schnitzeljagden ermöglicht, wurde zunächst der Ansatz des Wireframings verwendet. Wireframing ist eine wesentliche Phase im Designprozess, die es ermöglicht, die Struktur und Funktionalität einer Anwendung visuell darzustellen, bevor detaillierte Design- und Entwicklungsarbeiten beginnen. In dieser frühen Planungsphase wird ein einfaches, oft schematisches Layout der Benutzeroberfläche erstellt, das die Anordnung der verschiedenen Elemente wie Buttons, Menüs, und interaktiven Komponenten zeigt.

Im Folgenden werden die verschiedenen Wireframes aufgezeigt und erläutert, wieso sie eine solide Grundlage für die weitere Entwicklung des Editors boten, welche Stärken sie in Bezug auf Benutzerfreundlichkeit und Funktionalität aufwiesen, und welche Schwächen oder Herausforderungen während der Umsetzung erkannt wurden, die in späteren Phasen berücksichtigt werden mussten.

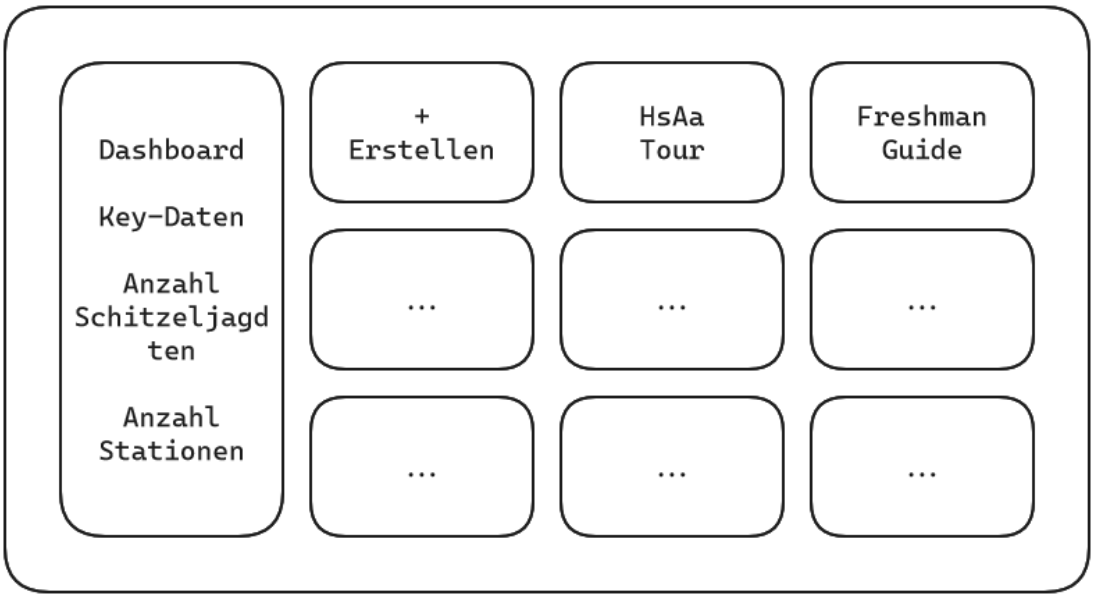

Abbildung 4.8.: Wireframing Frontend Hunt-Editor 3

Dies war ein Wireframe, was relativ am Anfang der Entwicklungsphase entstanden ist. Hier wurde auf ein Grid-Layout gesetzt, um die Schnitzeljagden anzuzeigen. An der Seite befindet sich eine Sidebar, in der verschiedene Key-Daten abgelesen werden können. Da es aber durch Architekturänderungen keine Stationen mehr gibt, fällt diese Information weg. Das Grid-Layout sollte aber bestehen bleiben.

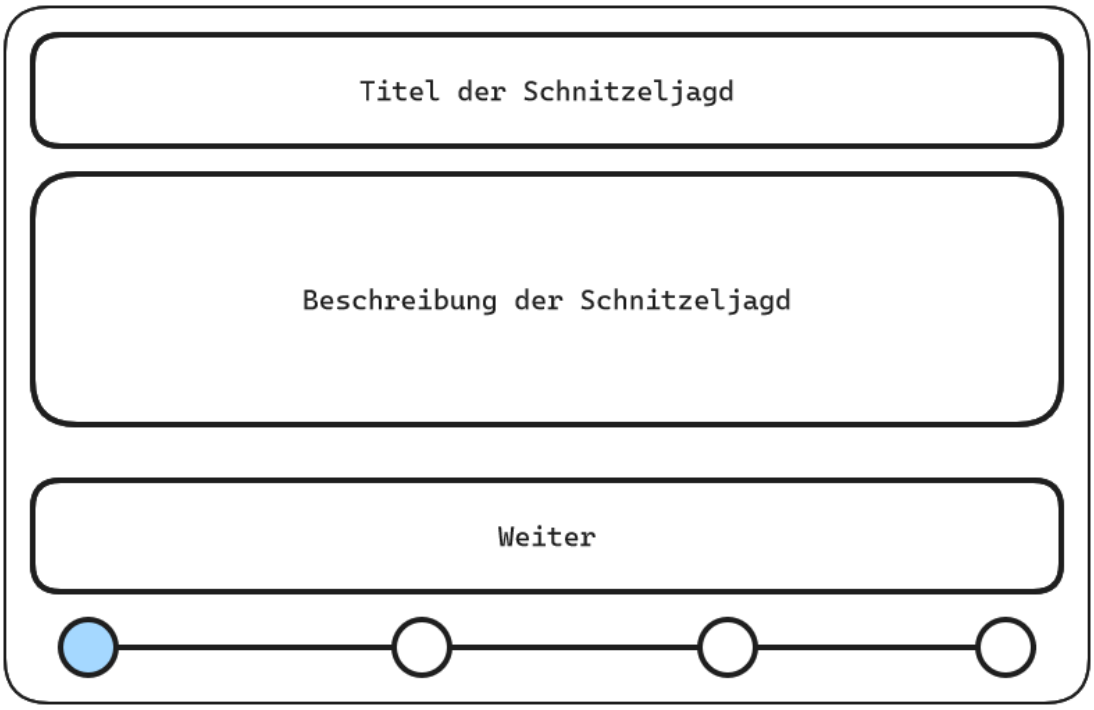

Abbildung 4.9.: Wireframing Frontend Hunt-Editor 4

Hier wird nun der Prozess des Erstellen einer Schnitzeljagd beschrieben, gefallen hat uns hier das Verwenden einer Progressbar, um den aktuellen Fortschritt beim Erstellen anzuzeigen. Auch die Eingabe von Titel und Beschreibung sollte so später übernommen werden.

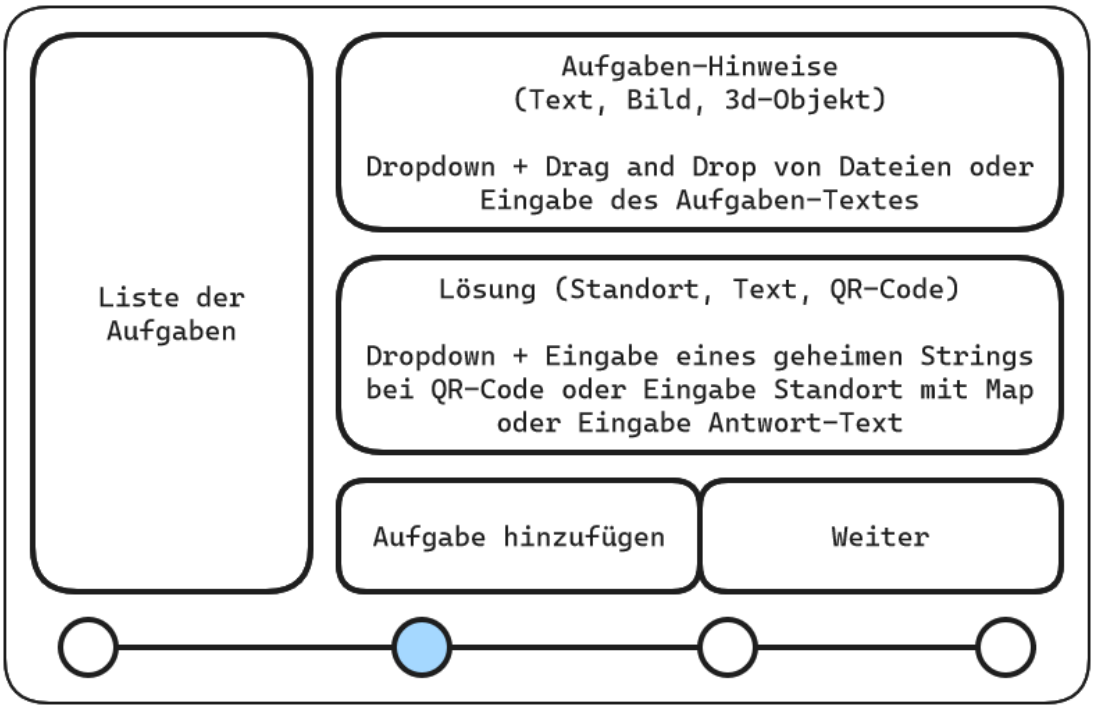

Abbildung 4.10.: Wireframing Frontend Hunt-Editor 5

Nun zum schwersten Teil, das Anlegen und Bearbeiten von Aufgaben in einer Schnitzeljagd. Hier war zunächst der Ansatz, auf der linken Seite eine Tabelle der Aufgaben zu führen. Durch Klick auf die entsprechende Zeile werden dann auf der rechten Seite die Aufgabe und Lösung angezeigt. Hier ist auch noch der Aufgabentyp "3d-Objekt"vorhanden, welcher am Schluss nicht übernommen wurde.

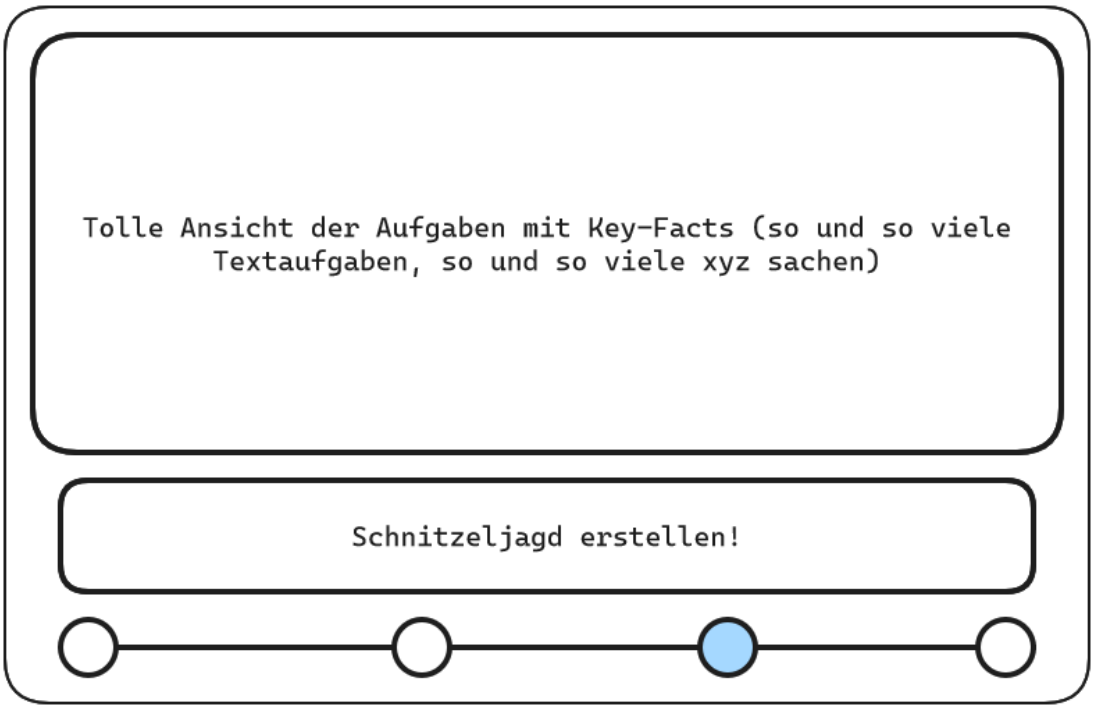

Abbildung 4.11.: Wireframing Frontend Hunt-Editor 6

Dieses Wireframe zeigt zum Schluss nochmal eine Übersicht aller wichtigen Infos der Schnitzeljagd an. Dieses Konzept hat uns gut gefallen.

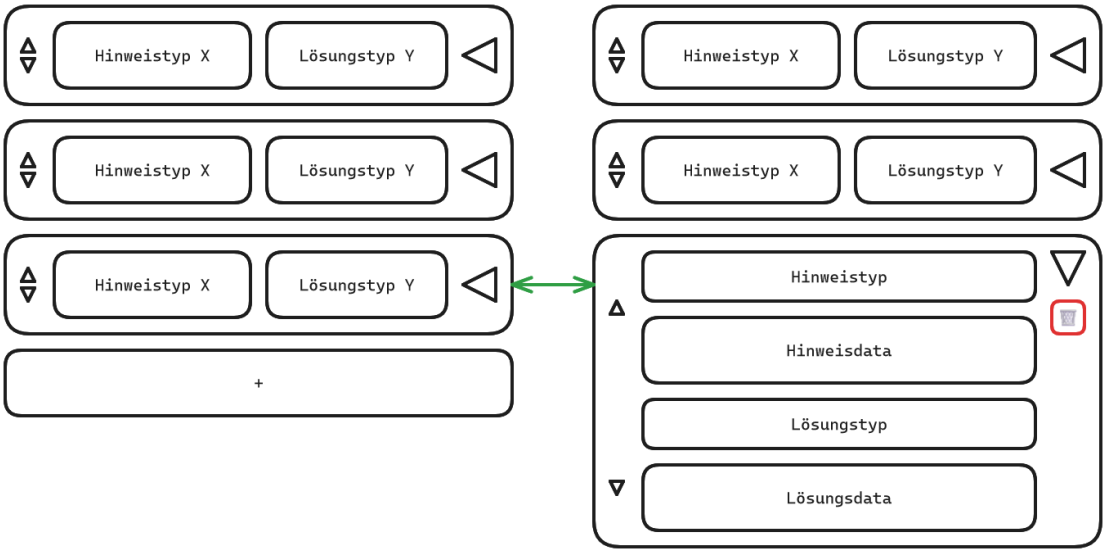

Abbildung 4.12.: Wireframing Frontend Hunt-Editor 7

Da die seitliche Liste aus Abbildung 4.10 zum Anzeigen der Aufgaben für mobile Geräte ungeeignet ist, haben wir uns für den Ansatz aus Abbildung 4.12 entschieden. Hier werden die Aufgaben nacheinander angezeigt und deren Reihenfolgen kann über das Bedienen der Pfeiltasten angepasst werden. Um die Aufgaben zu bearbeiten, können diese über den Button auf der rechten Seite aufgeklappt und wieder zugeklappt werden. Im aufgeklappten Zustand befindet sich jeweils ein Dropdown für Aufgabe und Lösung als auch Eingabefelder / FileUpload oder eine Map Integration für Standort als Lösung.

Entscheidung Flowbite

Um den Editor für das Verwalten von Schnitzeljagden zu erstellen, wurde zunächst versucht, mit DaisyUI zu arbeiten. Da dies nicht das gewünschte Ergebnis geliefert hat, wurde zu Flowbite gewechselt.

Unterschied zwischen Flowbite-Svelte und DaisyUI

DaisyUI ist eine Komponentenbibliothek für Tailwind CSS, die eine Vielzahl von vorgefertigten UI-Komponenten bereitstellt. Es ist darauf ausgelegt, die Nutzung von Tailwind CSS zu vereinfachen, indem es gebrauchsfertige Klassen und Designs bietet. DaisyUI ist besonders bekannt für seine einfache Integration und die Möglichkeit, schnell und effizient ästhetisch ansprechende Benutzeroberflächen zu erstellen, ohne viel eigene CSS-Regeln schreiben zu müssen.

Im Folgenden ein Beispiel für die Verwendung von DaisyUI, um ein Modal zu erstellen:

```
1  <button class="btn" onclick="my_modal_1.showModal()">open
       modal</button>
2  <dialog id="my_modal_1" class="modal">
3    <div class="modal-box">
4      <h3 class="text-lg font-bold">Hello!</h3>
5      <p class="py-4">Press ESC key or click the button below
       to close</p>
6      <div class="modal-action">
7        <form method="dialog">
8          <!-- if there is a button in form, it will close
       the modal -->
9          <button class="btn">Close</button>
10       </form>
11     </div>
12   </div>
13 </dialog>
```

Quelltext 4.1: DaisyUI Beispiel

Flowbite-Svelte ist eine auf Tailwind CSS basierende UI-Komponentenbibliothek, die speziell für die Integration mit Svelte entwickelt wurde. Sie bietet eine umfassende Sammlung von UI-Komponenten wie Buttons, Modale, Formulare und vieles mehr. Flowbite-Svelte kombiniert die Leistungsfähigkeit von Tailwind CSS mit der reaktiven Natur von Svelte, um die Entwicklung dynamischer und interaktiver Webanwendungen zu erleichtern. Zudem bietet Flowbite-Svelte eine nahtlose Svelte-Integration, was bedeutet, dass die Komponenten als echte Svelte-Komponenten bereitgestellt werden, die sich perfekt in das Svelte-Ökosystem einfügen.

Im Folgenden ein Beispiel für die Verwendung von Flowbite-Svelte, das das gleiche Modal erstellt wie im DaisyUI-Beispiel:

```
1  <script>
2    import { Button, Modal } from 'flowbite-svelte';
3    let defaultModal = false;
4  </script>
5
6  <Button on:click={() => (defaultModal = true)}>open
       modal</Button>
7  <Modal title="Hello!" bind:open={defaultModal} autoclose>
8    <p class="text-base leading-relaxed text-gray-500
       dark:text-gray-400">Press ESC key or click the button
       below to close</p>
9    <svelte:fragment slot="footer">
10     <Button on:click={() => (defaultModal =
       false)}>Close</Button>
11   </svelte:fragment>
12 </Modal>
```

Quelltext 4.2: Flowbite-Svelte Beispiel

Begründung für den Wechsel zu Flowbite-Svelte

Der Wechsel von DaisyUI zu Flowbite-Svelte ist aus mehreren Gründen gerechtfertigt. Erstens bietet Flowbite-Svelte eine speziell auf Svelte abgestimmte Komponentenbibliothek, die eine engere und effizientere Integration ermöglicht. Dies bedeutet, dass die UI-Komponenten in Flowbite-Svelte als native Svelte-Komponenten verfügbar sind, was die Entwicklungsarbeit vereinfacht und die Nutzung von Svelte-spezifischen Features erleichtert.

Zweitens ist die Anpassungsfähigkeit und Erweiterbarkeit von Flowbite-Svelte ein wesentlicher Vorteil. Während DaisyUI zwar eine einfache Möglichkeit bietet,

Tailwind CSS zu nutzen, ist Flowbite-Svelte besser darauf ausgelegt, komplexere und dynamischere Benutzeroberflächen zu unterstützen. Dies ist besonders wichtig für eine Anwendung wie den Schnitzeljagd-Editor, die eine hohe Interaktivität und eine benutzerfreundliche Oberfläche erfordert.

Darüber hinaus bietet Flowbite-Svelte eine breitere Palette an vorgefertigten Komponenten und Layout-Optionen, was die Entwicklung beschleunigt und gleichzeitig die Konsistenz und Qualität der Benutzeroberfläche verbessert. Dies trägt dazu bei, dass die Anwendung nicht nur funktional, sondern auch optisch ansprechend und intuitiv zu bedienen ist.

Paketabhängigkeiten

Dependency	Beschreibung
@sveltejs/adapter-auto	Adapter für SvelteKit, der automatisch den besten Adapter für die Zielumgebung auswählt
@sveltejs/kit	Das offizielle Framework zur Erstellung von Svelte-Anwendungen
@sveltejs/vite-plugin-svelte	Vite-Plugin zur Integration von Svelte
@types/eslint	TypeScript-Typdefinitionen für ESLint
@types/qrcode	TypeScript-Typdefinitionen für das qrcode-Paket
@types/uuid	TypeScript-Typdefinitionen für das uuid-Paket
autoprefixer	PostCSS-Plugin zur automatischen Ergänzung von Vendor-Präfixen in CSS
eslint	Ein pluggable Linter-Tool für JavaScript und JSX
eslint-config-prettier	ESLint-Konfiguration, die Konflikte zwischen ESLint und Prettier deaktiviert
eslint-plugin-svelte	ESLint-Plugin zur Unterstützung von Svelte
globals	Bibliothek, die globale Variablen und Umgebungen bereitstellt
postcss	Ein Werkzeug zur Transformation von CSS mit Plugins
prettier	Ein Code-Formatter, der konsistente Stilregeln erzwingt
prettier-plugin-svelte	Prettier-Plugin zur Formatierung von Svelte-Dateien
svelte	Das Svelte-Framework für reaktive Benutzeroberflächen
svelte-check	Ein Werkzeug zur statischen Analyse von Svelte-Projekten
tailwindcss	Ein Utility-First-CSS-Framework für die Erstellung maßgeschneiderter Designs
tslib	TypeScript-Hilfsbibliothek
typescript	Ein typisiertes Superset von JavaScript, das zu reinem JavaScript kompiliert
typescript-eslint	Eine Kombination von ESLint-Plugins und -Konfigurationen zur Unterstützung von TypeScript
vite	Ein schneller Entwicklungs-Server und Build-Tool, optimiert für moderne Webprojekte

Tabelle 4.1.: Liste der verwendeten Dependencies und deren Beschreibung 1

Dependency	Beschreibung
@popperjs/core	Ein Tool zur Verwaltung von Popper-Elementen, wie Dropdowns und Tooltips
flowbite	Eine UI-Komponentenbibliothek basierend auf Tailwind CSS
flowbite-svelte	Eine Integration von Flowbite-Komponenten in Svelte
lucide-svelte	Eine Sammlung von Icons für Svelte
qrcode	Ein JavaScript-Toolkit zur Generierung von QR-Codes
tailwind-merge	Ein Utility zur Kombination und Zusammenführung von Tailwind CSS-Klassen
uuid	Ein Werkzeug zur Generierung von UUIDs (Universally Unique Identifiers)

Tabelle 4.2.: Liste der verwendeten Dependencies und deren Beschreibung 2

4.2.5. Participant Web-App

Übersicht

4.3. Architektonische Entscheidungen

4.3.1. Wahl eines Message-Bus für das Hunt-Api Backend

4.3.2. Wahl eines Api-Gateways für das Hunt-Api Backend

5. Implementierung

5.1. Entwickeln einer AR-Anwendung mit Unity und AR-Foundation

Im Folgenden ist der Entwicklungsprozess für das Implementieren eines AR-Image-Trackers mithilfe des AR-Foundation Frameworks von Unity beschrieben. Im Anschluss wird erläutert, weshalb dieser Ansatz für das Projekt obsolet wurde.

5.1.1. Anlegen einer Reference-Image-Library

Nachdem ein AR-Unity-Projekt nach *QUELLE* angelegt wurde und das nötige Framework eingebunden wurde, stehen verschiedene Funktionen des AR-Foundation-Frameworks im Editor zur Verfügung.

Zunächst soll eine *Reference-Image-Library* erstellt werden. Abbildung 5.1 markiert die grundlegenden Schritte hierfür.

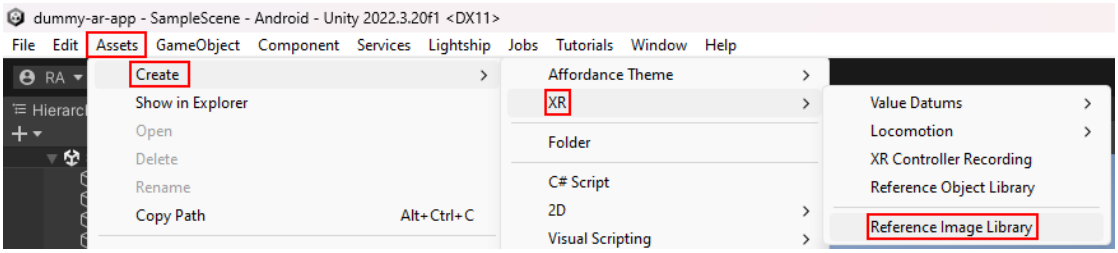

Abbildung 5.1.: Bildschirmabschnitt für das Erstellen einer Reference-Image-Library

In der geöffneten Szene in Abbildung 5.2 sollte nun ein leeres Reference-Image-Library Objekt existieren.

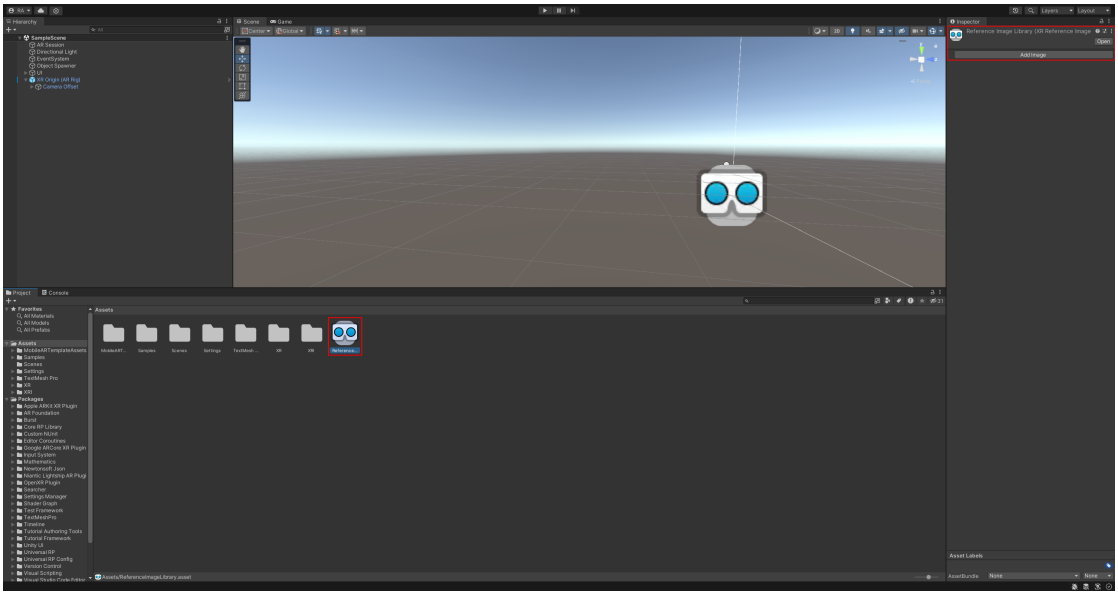

Abbildung 5.2.: Bildschirmabschnitt Reference-Image-Library Objekt im Editor

Die Reference-Image-Library wird benötigt, um vor der Laufzeit bereits Referenzbilder (*Reference-Images*) oder Objekte zu sammeln, die von einem AR-Tracker wie etwa die Komponente *XRImageTrackingSubsystem* erkannt werden können.

5.1.2. Anlegen eines AR-Tracked-Image-Managers

Die *AR Tracked Image Manager* Komponente erstellt *Game-Objects* für jedes gefundene Bild innerhalb des Viewports des Benutzers. Bevor ein Bild gefunden werden kann, muss die Komponente Referenz-Bilder kennen, die in der *Reference-Image-Library* gespeichert sind.

Das Einbinden erfolgt, indem ein Skript an die XR-Origin Komponente angebunden wird und die dementsprechende Reference-Image-Library per Drag-and-Drop eingebunden wird. Dies aktiviert das Tracken von Bildern im AR-Raum.

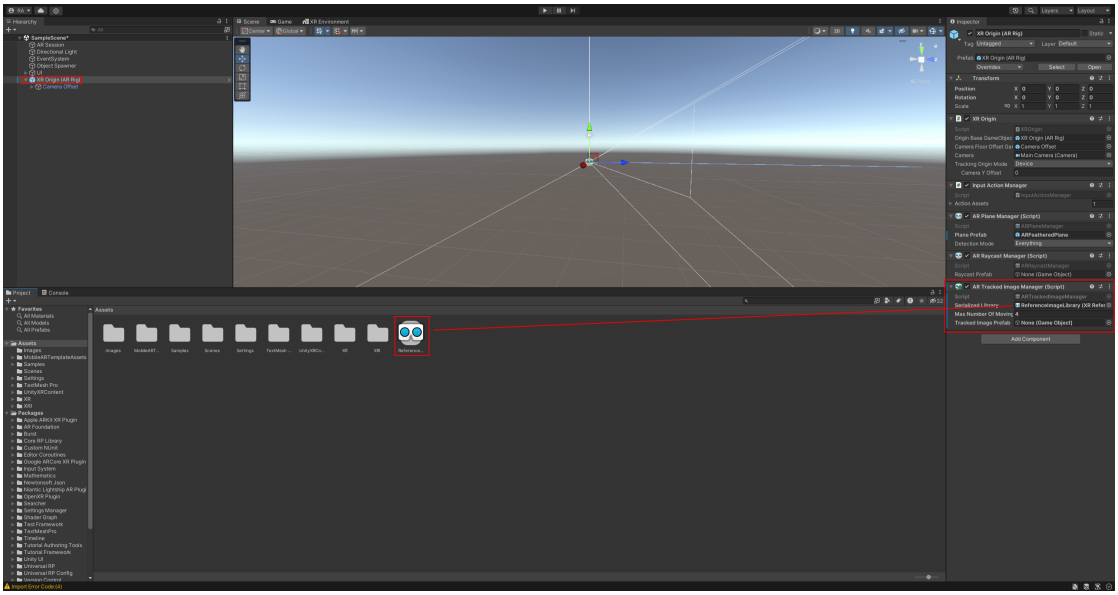

Abbildung 5.3.: Bildschirmabschnitt für das Erstellen eines AR-Tracked-Image-Managers

5.1.3. QR-Code-Daten aus einem Referenz-Bild lesen

Das erkennen eines QR-Codes erfordert, dass dieser in der Reference-Image-Library bereits angelegt wurde. Nachdem dies erfolgt ist, kann der QR-Code als Textur gelesen und über einen QR-Code-Konverter wie *ZXing* als Text konvertiert werden. Abbildung 5.4 zeigt das erfolgreiche Umkonvertieren eines QR-Codes im AR-Raum.

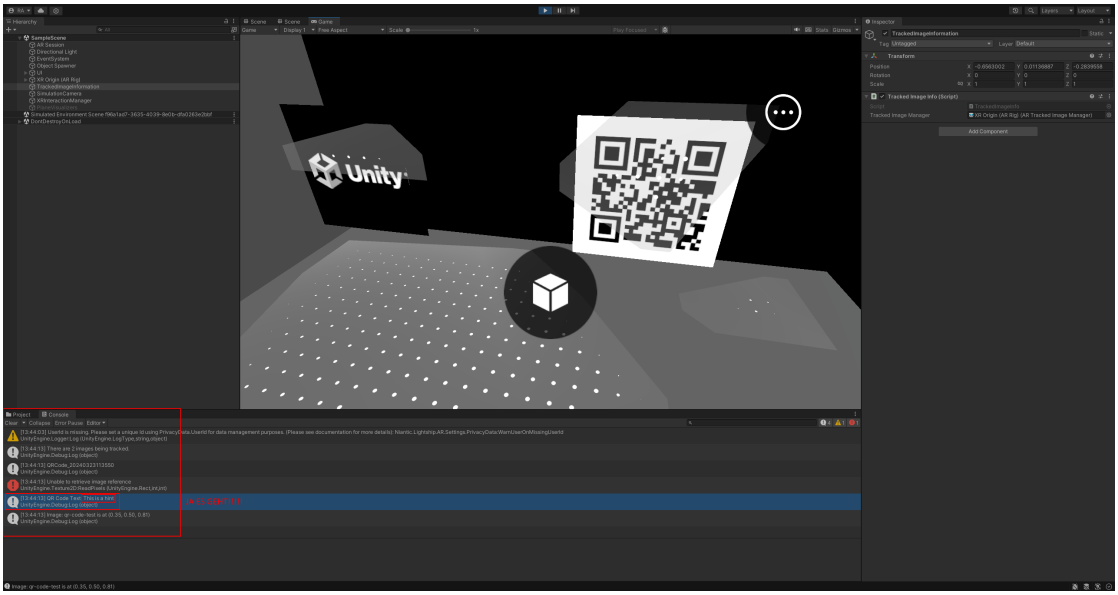

Abbildung 5.4.: Bildschirmabschnitt für das Lesen eines QR-Codes

5.1.4. Schwierigkeiten

Leider kann über den oben beschriebenen Ansatz kein gewünschtes Ergebnis erzielt werden.

Ein Hinweistext oder -bild an der Position des QR-Codes zu platzieren, könnte theoretisch funktionieren, wie in Abbildung 5.4 bereits dargestellt wird. Allerdings stellt die mangelnde Flexibilität der Reference-Image-Library ein erhebliches Hindernis für eine funktionierende Implementierung dar.

Um dieses Problem zu umgehen, müsste die Anwendung beim Start alle im System vorhandenen QR-Codes und ihre entsprechenden Lösungen über das Backend abrufen. Dies würde jedoch bei einer großen Anzahl von QR-Codes zu einer suboptimalen Benutzererfahrung führen, da die Anwendung erheblich verzögert wäre. Zusätzlich verursacht das Laden mehrerer QR-Codes einen hohen Speicheraufwand. Ein weiteres Problem besteht darin, dass die Lösungen bereits im Anwendungsspeicher geladen würden, bevor der Benutzer einen Hinweis erhalten hat. Dies könnte dazu führen, dass die Benutzer die Lösungen direkt aus dem Anwendungsspeicher extrahieren und so das Konzept der Schnitzeljagd umgehen könnten.

Diese Herausforderungen zeigen deutlich, dass Unity und AR Foundation in diesem Kontext nicht die optimale Lösung darstellen. Die genannten Einschränkungen machen es schwierig, eine flexible, effiziente und sichere Anwendung zu entwickeln, die den Anforderungen einer Schnitzeljagd entspricht. Daher ist es ratsam, alternative

Technologien oder Ansätze zu berücksichtigen, die eine bessere Anpassungsfähigkeit und Sicherheit bieten.

6. Inbetriebnahme

Aufgabe des Kapitels Inbetriebnahme ist es, die Überführung der in Kapitel 5 entwickelte Lösung in das betriebliche Umfeld aufzuzeigen. Dabei wird beispielsweise die Inbetriebnahme eines Programms beschrieben oder die Integration eines erstellten Programmodules dargestellt.

7. Zusammenfassung und Ausblick

7.1. Erreichte Ergebnisse

Die Zusammenfassung dient dazu, die wesentlichen Ergebnisse des Praktikums und vor allem die entwickelte Problemlösung und den erreichten Fortschritt darzustellen. (Sie haben Ihr Ziel erreicht und dies nachgewiesen).

7.2. Ausblick

Im Ausblick werden Ideen für die Weiterentwicklung der erstellten Lösung aufgezeigt. Der Ausblick sollte daher zeigen, dass die Ergebnisse der Arbeit nicht nur für die in der Arbeit identifizierten Problemstellungen verwendbar sind, sondern darüber hinaus erweitert sowie auf andere Probleme übertragen werden können.

7.2.1. Erweiterbarkeit der Ergebnisse

Hier kann man was über die Erweiterbarkeit der Ergebnisse sagen.

7.2.2. Übertragbarkeit der Ergebnisse

Und hier etwas über deren Übertragbarkeit.

Literatur

[1] Robert C. Martin. *Clean Architecture: A Craftsman's Guide to Software Structure and Design*. Part 1, Chapter 1: The Goal. Prentice Hall, 2018, S. 33.

[2] James Lewis und Martin Fowler. *Microservices: a definition of this new architectural term*. 20114. URL: https://martinfowler.com/articles/microservices.html (besucht am 20. 04. 2024).

[3] Code Maze. *Onion Architecture in ASP.NET Core*. 2024. URL: https://code-maze.com/onion-architecture-in-aspnetcore/ (besucht am 13. 07. 2024).

[4] Übersetzung von Jørgen W. Lang Mark Richards Neal Ford. *Handbuch moderner Softwarearchitektur*. O'REILLY, 2021, S. 184–189.

[5] Red Hat. *What are application programming interfaces (APIs)?* 2023. URL: https://www.redhat.com/de/topics/api/what-are-application-programming-interfaces (besucht am 20. 04. 2024).

[6] Unity. *AR Foundation*. 2024. URL: https://unity.com/de/unity/features/arfoundation (besucht am 20. 04. 2024).

[7] Salad Lab. *Json.Net for Unity3D*. 2016. URL: https://github.com/SaladLab/Json.Net.Unity3D/#upm (besucht am 23. 04. 2024).

[8] Daniel Fortes. *AR + GPS Location*. 2023. URL: https://assetstore.unity.com/packages/tools/integration/ar-gps-location-134882 (besucht am 23. 04. 2024).

[9] Svelte contributors. *Svelte - Cybernetically enhanced web apps*. 2024. URL: https://svelte.dev/ (besucht am 20. 03. 2024).

[10] Alex Bespoyasov. *My Experience with Svelte and SvelteKit*. 2022. URL: https://dev.to/bespoyasov/my-experience-with-svelte-and-sveltekit-342e (besucht am 06. 06. 2024).

[11] Tien Nguyen. *Svelte vs SvelteKit: Understanding Key Differences and Similarities*. 2023. URL: https://www.frontendmag.com/insights/svelte-vs-sveltekit/#Pros_and_Cons (besucht am 20. 03. 2024).

A. Anhang A

B. Anhang B