

Projektarbeit
Studiengang : Informatik

Projektarbeit AR-Schnitzeljagd

von

Lukas Steckbauer, Rosario Aranzulla

85836, 85816

Betreuender Mitarbeiter : Dr. Marc Hermann

Einreichungsdatum : 13.08.2024

Eidesstattliche Erklärung

Hiermit erkläre ich, **Lukas Steckbauer**, 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)

Hiermit erkläre ich, **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 umfassend das Projekt *Web-basierte Schnitzeljagd*. Das Projekt umfasst den Entwurf, die Implementierung und Inbetriebnahme eines Schnitzeljagd-Editors, der zur Erstellung und Bearbeitung bestehender Schnitzeljagden dient. Zudem wurde eine Web-Anwendung entwickelt, die es den Teilnehmern ermöglicht, an Schnitzeljagden teilzunehmen und diese zu absolvieren.

Ursprünglich war geplant, eine mobile Anwendung mit AR-Unterstützung auf Basis von Unity zu entwickeln. Im Laufe des Projekts wurde jedoch entschieden, stattdessen auf eine Web-basierte Lösung umzusteigen. Die Gründe für diese Entscheidung und die damit verbundenen Überlegungen werden ausführlich in der Arbeit dokumentiert. Trotz dieses Wechsels wurde intensiv mit AR-Technologien, insbesondere WebAR, gearbeitet, um die Möglichkeiten der Integration von AR-Elementen in eine Webumgebung zu erforschen.

Die Arbeit bietet einen umfassenden Einblick in die technischen Details der Projektrealisierung, beginnend mit einer Einführung in relevante Frameworks, Technologien und Bibliotheken, über die Konzeption und den Entwurf, bis hin zur konkreten Implementierung und den Herausforderungen, die während des Projektdurchlaufs auftraten.

Im Rahmen der Projektarbeit wurden vielfältige Aspekte der Software-Entwicklung behandelt, wobei der Schwerpunkt auf der Entwicklung einer skalierbaren und flexiblen Weblösung lag. Besondere Aufmerksamkeit wurde der Verwendung von Microservices, der Interprozesskommunikation über REST-Apis und der Integration von WebAR-Technologien gewidmet.

Inhaltsverzeichnis

Eidesstattliche Erklärung	i
Kurzfassung	ii
Quelltextverzeichnis	vii
Abkürzungsverzeichnis	viii
1. Einleitung	1
1.1. Motivation	1
1.2. Zielsetzung	1
1.3. Vorgehen	2
2. Grundlagen	3
2.1. Software Design	3
2.1.1. Einführung	3
2.1.2. Architektonische Patterns und Ansätze	3
2.1.3. Kollaboration mehrerer Dienste	5
2.2. Technologien	7
2.2.1. ASP.NET	7
2.2.2. Unity und Mobile-Apps	8
2.2.3. Svelte und Web-Apps	9
2.2.4. Docker	11
3. Problemanalyse	12
3.1. Anforderungen und Problemabgrenzung	12
3.2. Architektonische Überlegungen	14
4. Software-Entwurf	15
4.1. Blockansicht	15
4.2. Hunt-API Backend	16
4.2.1. Übersicht	16
4.2.2. Hunts-Service	17
4.2.3. Participants-Service	20
4.3. Hunt-Editor Web-App	22
4.3.1. Übersicht	22
4.3.2. Wireframing	22

4.4. Participation Web-App	27
4.4.1. Übersicht	27
4.4.2. Wireframing	28
4.5. Hunt-Game Web-App	29
4.5.1. Übersicht	29
4.5.2. Spielablauf	30
5. Implementierung	32
5.1. Entwickeln des Frontends	32
5.1.1. Aufbau der Web-Anwendungen	32
5.1.2. UI-Design	33
5.1.3. Dynamic Routing	33
5.1.4. Walkthrough der Web-Anwendungen	33
5.2. Entwickeln einer AR-Anwendung mit Unity und AR-Foundation . . .	33
5.2.1. Anlegen einer Reference-Image-Library	34
5.2.2. Anlegen eines AR-Tracked-Image-Managers	35
5.2.3. QR-Code-Daten aus einem Referenz-Bild lesen	35
5.2.4. Erfahrung mit Unity und AR-Foundation	36
5.2.5. Alternativen	38
5.3. Algorithmen zur Hinweisbestimmung für Nutzerlösungen	39
5.3.1. Haversine-Algorithmus zur Bestimmung geografischer Distanzen	39
5.3.2. Levenshtein-Algorithmus zur Bestimmung textueller Unter- schiede	43
6. Inbetriebnahme	47
6.1. Einführung	47
6.2. Backend Containerisierung	47
6.3. Frontend Containerisierung	48
7. Zusammenfassung und Ausblick	49
7.1. Erreichte Ergebnisse	49
7.2. Ausblick	49
Literatur	51
A. Anhang: Entscheidungen bei der Implementierung	53
A.1. Architektonische Entscheidungen	53
A.1.1. Wahl eines Message-Bus für das Hunt-API Backend	53
A.1.2. Wahl eines API-Gateways für das Hunt-API Backend	54
A.1.3. Hinweise und Lösungen ohne Abstrakte Klassen	55
A.2. Entscheidungen im Frontend	56
A.2.1. Wahl von Flowbite statt DaisyUI	56
B. Anhang: Implementierung und Inbetriebnahme	59
B.1. Containerisierung des Backends	59

B.2. Dynamic Routing in Svelte	59
B.3. Walkthrough der Web-Apps	59

Abbildungsverzeichnis

2.1. Bild Onion Architecture, nach [3]	4
3.1. Darstellung der Projekt-Ziele aufgeteilt in Vier Felder	12
4.1. Bild Systemkontext als Blackbox-Diagramm	15
4.2. Bild Hunt-API Subsystem	16
4.3. Bild Hunts Microservice	17
4.4. Skizze der Hunts Ressourcenansicht	18
4.5. Skizze ER-Modell der Hunts	19
4.6. Bild Participants Microservice	20
4.7. Skizze der Participants Schnittstelle	21
4.8. Skizze ER-Modell zwischen Hunts und Participants	22
4.9. Skizze Dashboard des Hunt-Editors	23
4.10. Skizze für Eingabe von Basisdaten im Hunt-Editor	24
4.11. Skizze für Anlegen von Aufgaben im Hunt-Editor	25
4.12. Skizze zur Übersicht einer Schnitzeljagd im Hunt-Editor	26
4.13. Skizze zur Auflistung von Aufgaben im Hunt-Editor	26
4.14. Skizze zur Anzeige von Statistiken in der Participation Web-App	28
4.15. Skizze zur Benachrichtigung eines Organisators in der Participation Web-App	29
4.16. Skizze Spielablauf als UML-Programmablaufplan	30
5.1. UML-Diagramm Whitebox Frontend	32
5.2. Bildschirmabschnitt für das Erstellen einer Reference-Image-Library	34
5.3. Bildschirmabschnitt Reference-Image-Library Objekt im Editor	34
5.4. Bildschirmabschnitt für das Erstellen eines AR-Tracked-Image-Managers	35
5.5. Bildschirmabschnitt für das Lesen eines QR-Codes	36
5.6. Bildschirmabschnitt Abstand Teilnehmer und Lösung	41
5.7. Bildschirmabschnitt Abstands-Hinweis an Teilnehmer	42
5.8. Bildschirmabschnitte Text Unterschied an Teilnehmer	45
6.1. UML-Diagramm für das Deployment	47

Listings

5.1. Code Ausschnitt zum Haversine Algorithmus in C#	40
5.2. Code Ausschnitt zum Levenshtein Distanz Algorithmus in C#	44
A.1. Code Ausschnitt DaisyUI Beispiel	56
A.2. Code Ausschnitt Flowbite-Svelte Beispiel	57

Abkürzungsverzeichnis

DDD	Domain-driven Design	3
REST	Representational State Transfer	7
EF Core	Entity Framework Core	7
ORM	Object-Relational Mapping	7
VR	Virtual Reality	8
LINQ	Language Integrated Query	8
SSR	Server-Side Rendering	9
CLI	Command Line Interface	10

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. 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

Nach der Verfeinerung der für das Projekt relevanten technologischen Grundkenntnisse wurde ein erster Entwurf der grundlegenden Architektur vorgelegt. Nachdem dieser ausreichend ausgereift war, konnten die ersten Benutzer-Workflows erstellt werden. Nachdem die Baseline für die Implementierung festgelegt war, konnten die ersten 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.

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 Microservices versteht man den Ansatz, ein Softwaresystem als Orchestrierung mehrerer Dienste zu unterteilen. Jeder Dienst ist für eine Aufgabe im Gesamtkontext des Systems verantwortlich und kann unabhängig von anderen Diensten als eigener Prozess ausgeführt werden. [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:

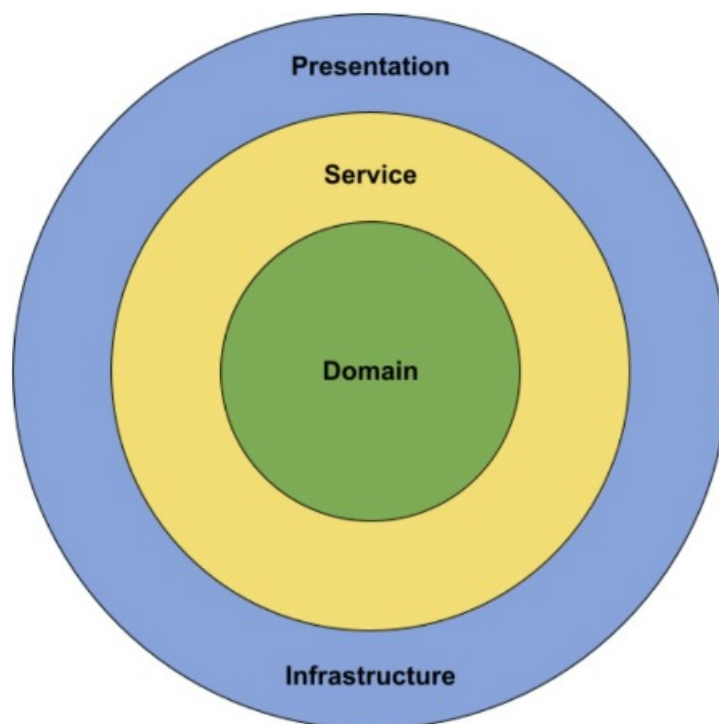


Abbildung 2.1.: Bild Onion Architecture, nach [3]

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 [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 *subscribe*. 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 lediglich 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. [6]

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. [7]

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. [8]

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. [9]

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. [10]

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 platziert werden. Wenn die Anwendung an der realen Stelle des angegebenen GPS-Standorts geöffnet wird, wird das in Unity angelegte Objekt am richtigen Ort angezeigt. [11]

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 erledigen, verlagert Svelte diese Arbeit in die Kompilierungsphase. Das bedeutet, dass der Code während des Build-Prozesses in effizientes, optimiertes JavaScript umgewandelt wird, das direkt im Browser ausgeführt wird. Dadurch wird die Laufzeitlast erheblich reduziert, was zu einer besseren Performance und kürzeren Ladezeiten führt. [12]

Ein wesentlicher Vorteil von Svelte ist seine einfache Syntax, die es Entwicklern ermöglicht, reaktiven Code zu schreiben, ohne auf komplexe Zustandsverwaltungslösungen zurückgreifen zu müssen. Die Komponenten von Svelte bestehen aus HTML, CSS und JavaScript, was die Lernkurve für neue Benutzer verkürzt und die Entwicklungserfahrung vereinfacht. [12]

SvelteKit ist ein Framework zur Erstellung von Svelte-Anwendungen. Es erweitert die Möglichkeiten von Svelte, indem es zusätzliche Werkzeuge und Funktionen bereitstellt, die speziell für die Entwicklung komplexer, leistungsstarker Webanwendungen benötigt werden. SvelteKit vereinfacht die Einrichtung und Strukturierung von Projekten und bietet Funktionen wie Routing, Server-Side Rendering (SSR), statische Seitengenerierung und eine integrierte Entwicklungsumgebung. [12]

Vorteile

Svelte integriert CSS-Styles direkt in die Komponenten mittels `<style>-Tags`. Dadurch entfällt die Notwendigkeit einer zusätzlichen Konfiguration. Alternativ kann auf bestehende CSS-Bibliotheken wie TailwindCSS zurückgegriffen werden. Standardmäßig werden Styles scoped erstellt, um Konflikte zwischen Komponenten zu vermeiden. Dies sorgt für eine sichere und wartbare Codebasis der einzelnen Komponenten durch reduzierte Abhängigkeiten. Svelte verfügt über eingebaute Barrierefreiheitsprüfungen (*Accessibility-Linting*), die auf offensichtliche und auch komplexere Probleme hinweisen. Dies hilft Entwicklern, von Anfang an barrierefreie

Anwendungen zu erstellen. [13]

Svelte-Pages verfügen standardmäßig über ein einfaches Prerendering, das durch Setzen eines Attributs gesteuert werden kann. SvelteKit bietet auch einfache Einstellungen, um verschiedene Startpfade für das Prerendering zu definieren, was die Flexibilität bei der Konfiguration erhöht. [13]

Ein weiteres zentrales Feature von Svelte ist die Reaktivität. Durch die Definition sogenannter *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 ermöglicht reaktive und dynamische Benutzerschnittstellen. [13]

Die deklarative Natur von Svelte reduziert die Menge des benötigten Codes erheblich. Dies führt zu einer übersichtlicheren Codebasis, welche die Entwicklung und Wartung von Anwendungen vereinfacht. [13]

Nachteile

Im Vergleich zu bekannteren JavaScript-Frameworks wie React oder Angular ist die Community von Svelte aufgrund der Neuheit des Frameworks noch relativ klein. Dies kann die Fehlersuche erschweren, da einige Bugs möglicherweise nicht so gut dokumentiert oder gelöst sind und somit weniger Ressourcen in Foren wie Stack Overflow zur Verfügung stehen. [14]

Ein weiterer Nachteil ist das Fehlen von Entwicklertools oder einer Command Line Interface (CLI). Bei Angular können beispielsweise Komponenten oder Services schnell und effizient mit dem Angular CLI erstellt werden. Diese Art von Tooling fehlt bei Svelte, was den Entwicklungsprozess etwas weniger komfortabel machen kann. [14]

Darüber hinaus ist die Verfügbarkeit und Nutzbarkeit der vorhandenen Bibliotheken eingeschränkt. Zwar können diese Bibliotheken problemlos mit einem Paketmanager (z.B. npm von Node.js) installiert werden, jedoch ist es oft mit zusätzlichem Aufwand verbunden, diese in Svelte-Komponenten zu integrieren. [14]

Pages & Routes

Eine Svelte-Anwendung ist verzeichnisbasiert. Jedes Verzeichnis entspricht einer Route und kann ein oder mehrere Unterverzeichnisse haben, die die Route erweitern. Jede Route kann eine Seite haben, die immer `+page.svelte` heißt. Svelte verwendet diese Seiten als Hauptanzeige, wenn zu der entsprechenden Route navigiert wird.

UI Components

Für die Darstellung, Wiederverwendbarkeit und Konsistenz zwischen verschiedenen Seiten werden Svelte-Komponenten verwendet. Diese bilden die Bausteine für die Darstellung unterschiedlicher Daten. Zusätzlich können UI-Komponentenbibliotheken wie z.B. *flowbite-svelte* eingebunden werden, um standardisierte und ansprechende UI-Elemente wie Buttons, Inputs oder Tabellen zu implementieren. Die Verwendung einer vorgefertigten Bibliothek vereinfacht zudem die Entwicklung und trägt zu einer konsistenten und intuitiven Benutzererfahrung bei.

Jede Svelte-Komponente besteht aus einem Skriptteil und einem Design-/Styleteil. Eine Komponente kann auch andere Komponenten einbinden. In diesem Fall fungiert die übergeordnete Komponente als Elternteil (Parent), während die eingebetteten 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.

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 Webserverns ermöglicht *Node.js* auch die serverseitige Verarbeitung von Api-Anfragen und die Anbindung an Datenbanken.

2.2.4. Docker

Docker ist eine Technologie zum Deployment von Anwendungen (z.B. Dienste oder Datenbankverbindungen), die zu Testzwecken in der Entwicklungsphase eingesetzt werden können. Docker-Compose und Dockerfiles können ebenfalls zum Deployment von Anwendungen und Systemen verwendet werden.

3. Problemanalyse

3.1. Anforderungen und Problemabgrenzung

Die Schnitzeljagden sollen in erster Linie den Studienanfängern 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.

Must have	Should have
<ul style="list-style-type: none">- Editor (Web-Oberfläche)<ol style="list-style-type: none">1. Nutzer hat Übersicht aller Schnitzeljagden.2. Schnitzeljagd kann erstellt, aktualisiert, gelöscht werden.3. Schnitzeljagd besteht aus Hinweis (Text/Frage) und Antwort.- Schnitzeljagd API<ol style="list-style-type: none">1. Editor App kann mit dieser API kommunizieren. ⇒ Basis-Endpunkte für Schnitzeljagd.2. AR App kann später mit dieser API kommunizieren ⇒ Get Next Question, ...	<ul style="list-style-type: none">- "AR" App (Unity?)<ol style="list-style-type: none">Stufe 1: App kann eine Aufgabe (Frage) aus dem Backend anzeigen (in AR?), Nutzer kann Frage durch Click auf Button beantworten ⇒ Nächste Frage.Stufe 2: App kann einen physischen QR-Code einscannen und durch diesen die nächste Aufgabe anzeigen.Stufe 3: Nutzer kann eine Frage aus dem Backend anzeigen und durch Antwortoptionen (Multiple Choice) beantworten.
Could have	Nice to have
<ul style="list-style-type: none">- "AR" App (Unity?)<p>Nutzer kann durch Invite-Link (Code?) einer Schnitzeljagd beitreten.</p>	<ul style="list-style-type: none">- "AR" App (Unity?)<p>In Editor App kann ausgewählt werden, an welcher Location die nächste Aufgabe ist. AR App zeigt Pfeile in die Richtung an (oder beschreibt textuell weg), ist User an richtiger Stelle Aufgabe erfolgreich abgeschlossen.</p>

Abbildung 3.1.: Darstellung der Projekt-Ziele aufgeteilt in Vier Felder

Für die erfolgreiche Projektumsetzung sind folgende Eigenschaften zu berücksichtigen.

Flexibilität

Ein wichtiger Aspekt des Projektes ist, dass eine Schnitzeljagd so flexibel wie möglich umgesetzt werden soll. Aufgabenstellung und Lösung sollen voneinander entkoppelt und ohne großen Implementierungsaufwand dynamisch erweiterbar sein.

Skalierbarkeit

Das System muss in der Lage sein, mehrere Schnitzeljagden gleichzeitig ohne Probleme durchzuführen. Es sollte möglich sein, die Ressourcen des Systems dynamisch an die Anzahl der gerade aktiven Benutzer anzupassen, ohne dass es zu Leistungseinbußen oder ähnlichen Beeinträchtigungen kommt.

Benutzerfreundlichkeit

Der Anmelde- und der Durchführungsprozess sollten unabhängig voneinander klar strukturiert sein, um den Benutzern eine reibungslose und intuitive Erfahrung zu bieten. Während der Schnitzeljagd sollten keine Schwierigkeiten auftreten. Die Schnitzeljagd sollte den Benutzern ein einzigartiges Erlebnis bieten, ohne sie durch unnötige oder ablenkende Elemente zu stören. Idealerweise fungiert die Anwendung als Schnittstelle zur Durchführung der Schnitzeljagd, wobei die Lösung 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 Zusammenhang mit sensiblen Benutzerdaten wie Passwörtern sehr wichtig. Durch die im Kapitel 4 beschriebenen Architekturüberlegungen wäre es möglich, Benutzerdaten durch kryptographisch sichere Hashfunktionen zu speichern. Um den Projektumfang auf das Wesentliche zu reduzieren, wurde dies im Projekt nicht berücksichtigt.

3.2. Architektonische Überlegungen

Bei der Wahl einer angemessenen Architektur für das System sind viele Aspekte zu berücksichtigen, unter anderem die in Kapitel 3.1 genannten Eigenschaften.

In Kapitel 4 wird daher der service-orientierte Ansatz für den Entwurf der Software allumfassend beschrieben.

4. Software-Entwurf

4.1. Blockansicht

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

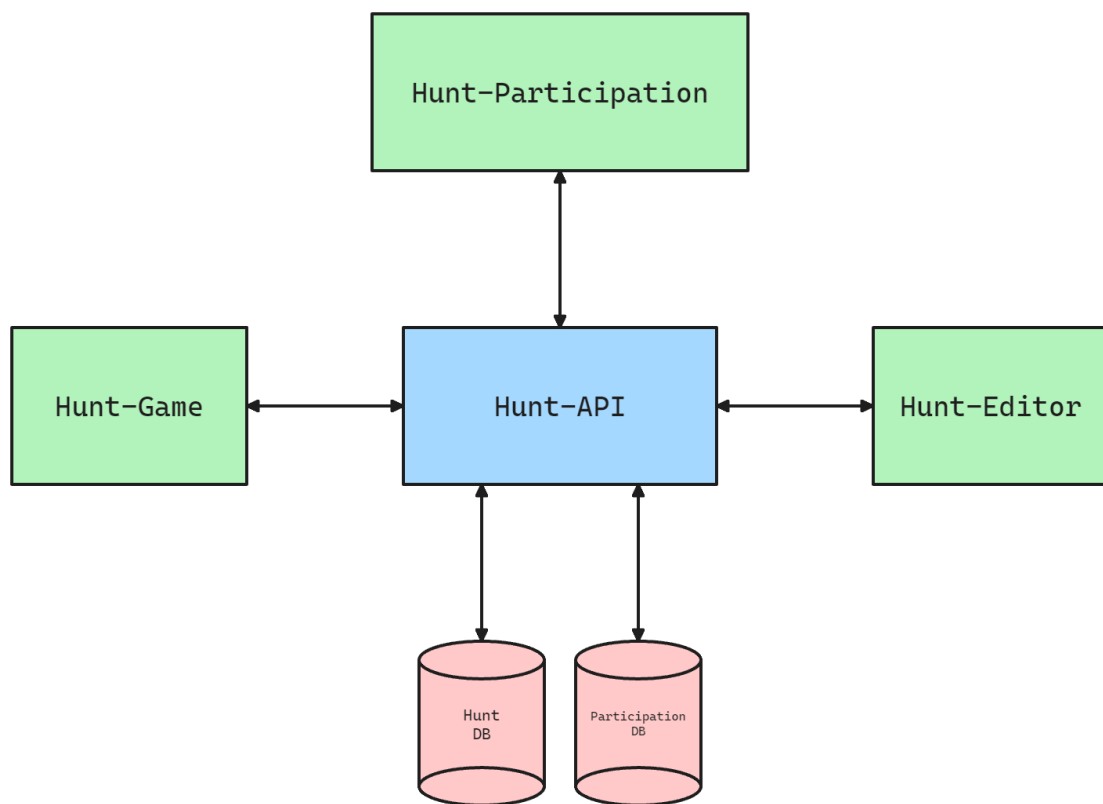


Abbildung 4.1.: Bild Systemkontext als Blackbox-Diagramm

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. Hunt-API Backend

4.2.1. Übersicht

Anhand einer zentral zur Verfügung stehenden Schnittstelle *Hunt-API* können die verschiedenen Anwendungen zur Gesamt-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.

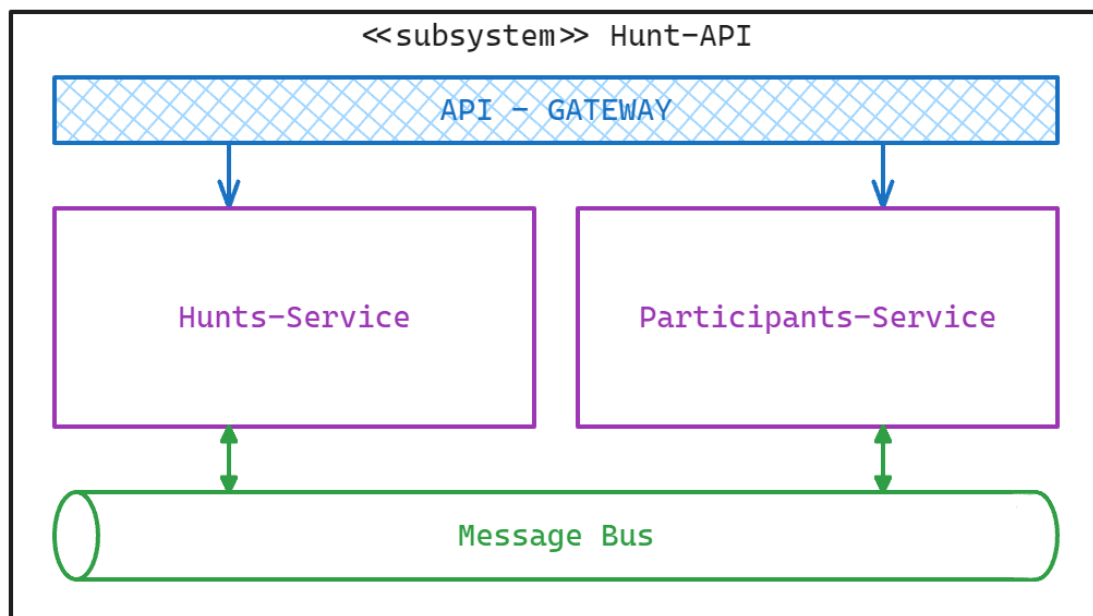


Abbildung 4.2.: Bild Hunt-API Subsystem

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 und repräsentieren den jeweiligen Dienst als Microservice (vgl. Kapitel 2.1.2).

Für die Kommunikation unterschiedlicher Dienste ist ein Message-Bus vorhergesehen (vgl. Kapitel 2.1.3), worüber eventgesteuert Nachrichten ausgetauscht werden. Dieser ist in Abbildung 4.2 grün hervorgehoben. Wieso ein Message-Bus nötig ist, wird im Anhang A.1.1 näher beschrieben.

Um eine einheitliche Schnittstelle bereitzustellen, die anwendungsübergreifend genutzt werden kann, wurde ein Api-Gateway vorhergesehen. Hierrüber werden Anfragen an an den entsprechenden Dienst weitergeleitet. Das Api-Gateway ist in Abbildung 4.2 blau gekennzeichnet. Im Anhang A.1.2 wird näher beschrieben, weshalb ein Api-Gateway gewählt wurde.

4.2.2. Hunts-Service

Übersicht

Für die Verwaltung der erstellten Schnitzeljagden eines Organisations, ist der Hunts-Service vorhergesehen. Abbildung 4.3 zeigt die Struktur des Dienstes.

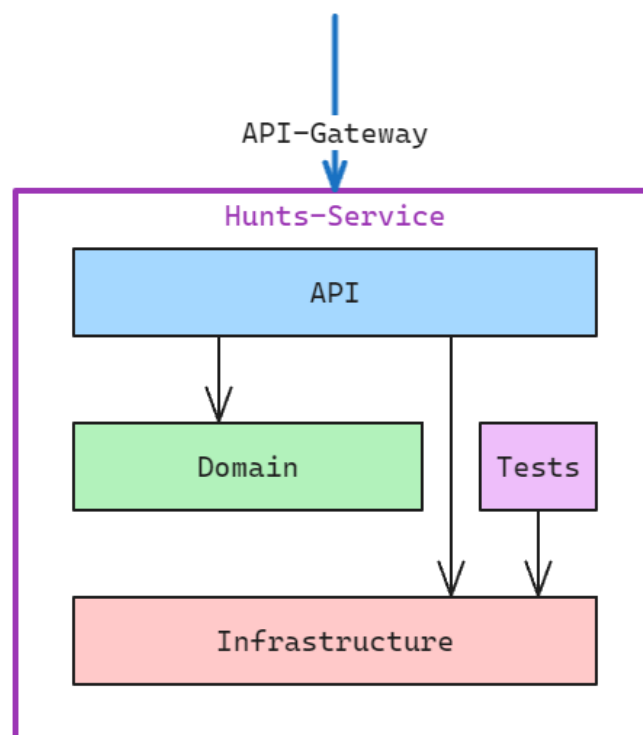


Abbildung 4.3.: Bild Hunts Microservice

Innerhalb des Hunts-Service wurde ein schichtenorientierter Ansatz gewählt, welcher 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 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.

Schnittstellendefinition

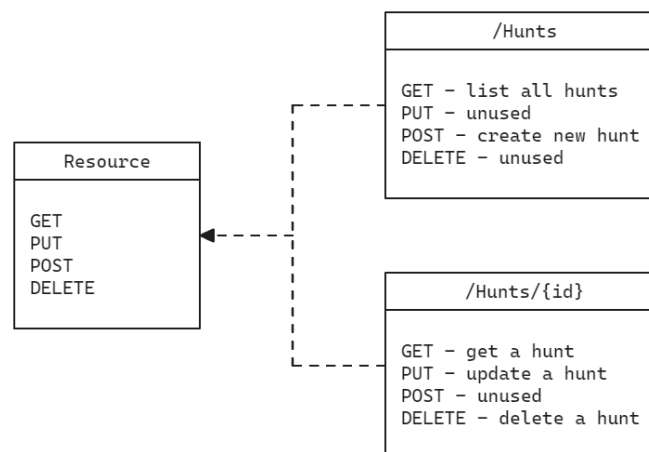


Abbildung 4.4.: Skizze der Hunts Ressourcenansicht

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

entspricht der tatsächlichen Schnittstelle, die für den Hunts-Service vorhergesehen wurde.

Datenbank-Modell

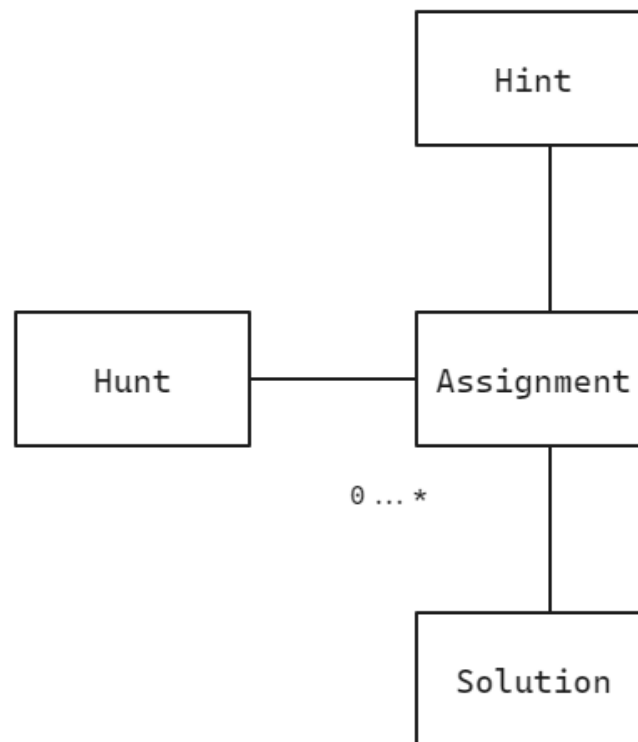


Abbildung 4.5.: Skizze ER-Modell der Hunts

Abbildung 4.5 beschreibt das gewählte Entity-Relationship Diagram für das Datenbank-Modell des Hunts-Service.

- **Hunt:** Die Schnitzeljagd. Enthält einen Titel, eine Beschreibung und eine Liste von 0 bis n Aufgaben (*Assignments*).
- **Assignment:** Eine konkrete Aufgabe einer Schnitzeljagd. Besteht aus einem Hinweis (*Hint*) und eine Lösung (*Solution*).
- **Hint:** Ein Hinweis. Besteht aus einem Hinweis-Typ (als Enumeration) und den Hinweis-Daten als String.
- **Solution:** Eine Lösung zu einem gegebenen Hinweis. Besteht aus einem Lösungs-Typ (als Enumeration) und den Lösungs-Daten als String.

Wieso auf die Verwendung eines abstrakten Datenmodells verzichtet wurde, wird im Anhang A.1.3 näher beschrieben.

4.2.3. Participants-Service

Übersicht

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 4.6 zeigt die Struktur des Dienstes.

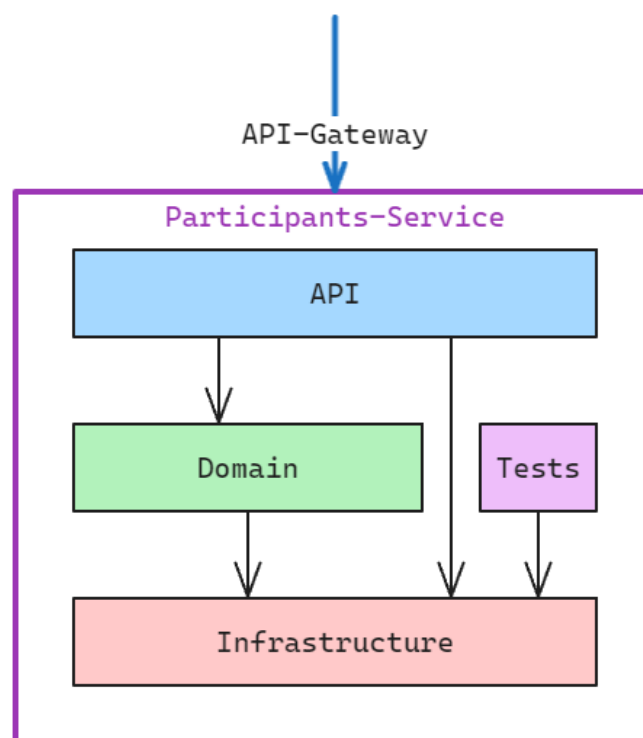


Abbildung 4.6.: Bild Participants Microservice

Der Participants-Service besitzt eine ähnliche Struktur wie der Hunts-Service aus Kapitel 4.2.2 und erfüllt die Aufgabe der Teilnahmen-Verwaltung.

Schnittstellendefinition

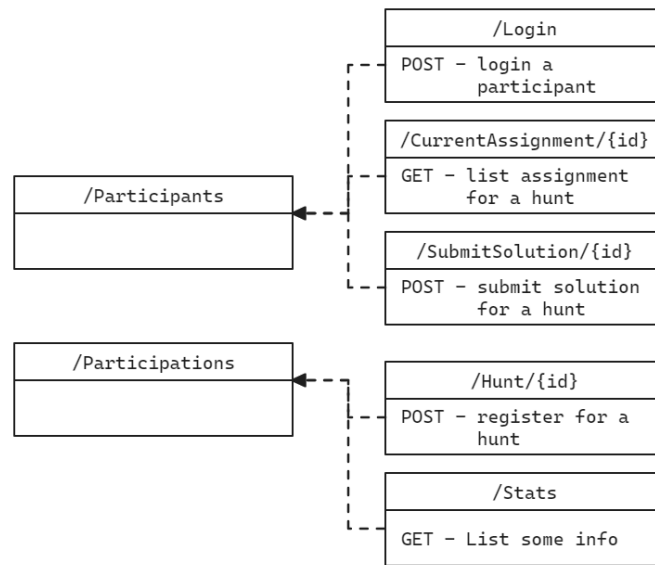


Abbildung 4.7.: Skizze der Participants Schnittstelle

Abbildung 4.7 definiert die vorhergesehene Schnittstelle für den Participants-Service.

Unter dem Endpunkt *Teilnehmer* wird dem Teilnehmer die Funktionalität zum *Login* angeboten. Hierüber erhält er zusätzlich Informationen über die aktuell angemeldeten Schnitzeljagden. Möchte er eine entsprechende Schnitzeljagd starten oder fortsetzen, kann er dazu über *Current-Assignment* den aktuellen Hinweis anfordern. Wenn der Teilnehmer dem Hinweis gefolgt ist und die Lösung gefunden hat, kann er diese dem System mit *Submit-Solution* mitteilen. Das System stellt dann fest, ob die Lösung korrekt ist.

Der genaue Ablauf und das Zusammenspiel der einzelnen Endpunkte wird in Kapitel 4.5.2 erläutert.

Damit sich Teilnehmer für eine Schnitzeljagd anmelden können, ist unter dem Endpunkt *Teilnahmen* mit *Schnitzeljagd* und entsprechender *Schnitzeljagd-ID* eine Registrierungsmöglichkeit vorgesehen. Zusätzlich können allgemeine Informationen wie z.B. die Anzahl der Teilnehmer oder die Anzahl aller Teilnahmen an Schnitzeljagden über den Endpunkt *Stats* aufgelistet werden.

Datenbank-Modell

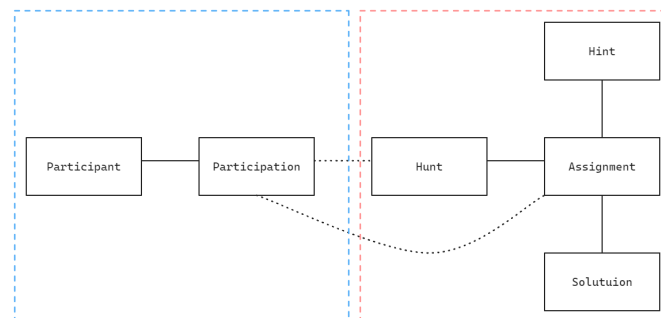


Abbildung 4.8.: Skizze ER-Modell zwischen Hunts und Participants

Abbildung 4.8 beschreibt das gewählte Entity-Relationship-Diagramm für das Datenbankmodell des Participants-Service. Die Teilnahmen (Participants und Participations) sollen relational gespeichert werden, während die Schnitzeljagden in einem nicht-relationalen Modell (z.B. Redis-Cache) gespeichert werden sollen, da diese bereits korrekt im Hunts-Service gespeichert werden (siehe Kapitel 4.2.2).

4.3. Hunt-Editor Web-App

4.3.1. Übersicht

Mit dem Hunt-Editor kann der Organisator die Schnitzeljagden verwalten. Dazu startet er die Anwendung und gelangt auf das Dashboard. Hier erhält er einen Überblick über die erstellten Schnitzeljagden. Ihm wird die Funktion zum Erstellen (*Create*) angeboten. Er kann Titel und Beschreibung sowie die jeweiligen Aufgaben mit Hinweis und Lösung anlegen und speichern. Die Reihenfolge der Aufgaben kann verändert werden.

4.3.2. Wireframing

Für die Entwicklung des Editors zur Erstellung und Verwaltung von Schnitzeljagden wurde zunächst der Wireframing-Ansatz verwendet. Wireframing ist eine wichtige Phase im Designprozess, die es ermöglicht, die Struktur und Funktionalität einer Anwendung visuell darzustellen, bevor mit dem detaillierten Design und der Entwicklung begonnen wird. In dieser frühen Planungsphase wird ein einfaches, oft schematisches Layout der Benutzeroberfläche erstellt, das die Anordnung der

verschiedenen Elemente wie Schaltflächen, Menüs und interaktive Komponenten zeigt.

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

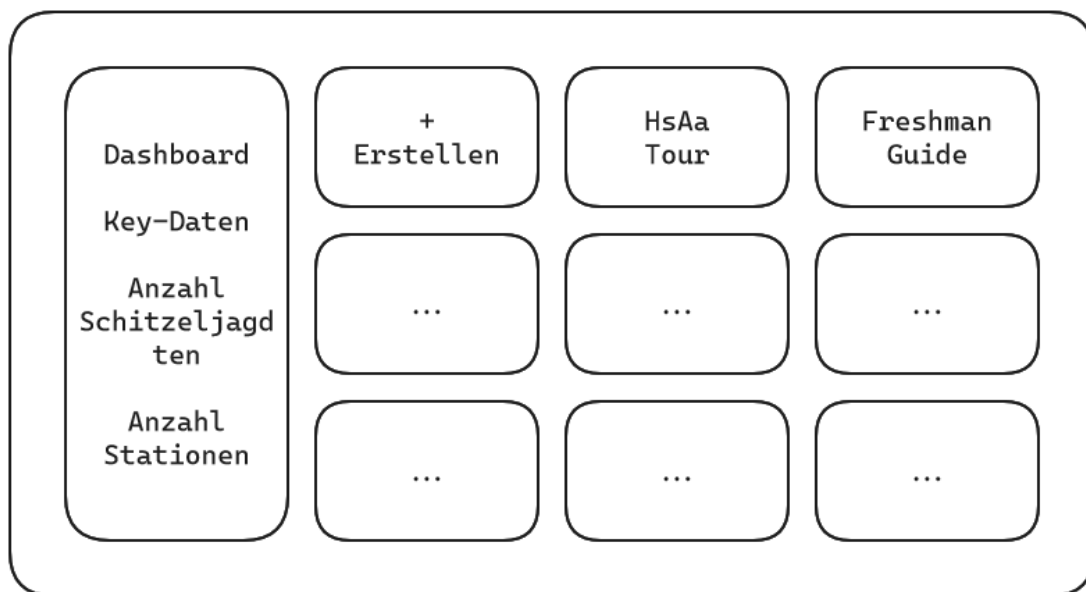


Abbildung 4.9.: Skizze Dashboard des Hunt-Editors

Dies ist ein Wireframe, das relativ am Anfang der Entwicklungsphase erstellt wurde. Hier wurde ein Grid-Layout verwendet, um die Schnitzeljagd darzustellen. An der Seite befindet sich eine Sidebar, in der verschiedene Schlüsseldaten abgelesen werden können. Da es aber aufgrund der Architekturänderungen keine Stationen mehr gibt, fällt diese Information weg. Das Grid-Layout soll jedoch beibehalten werden.

Titel der Schnitzeljagd

Beschreibung der Schnitzeljagd

Weiter

Abbildung 4.10.: Skizze für Eingabe von Basisdaten im Hunt-Editor

Hier wird nun der Prozess der Erstellung einer Schnitzeljagd beschrieben, wobei uns die Verwendung eines Fortschrittsbalkens gefallen hat, der den aktuellen Stand der Erstellung anzeigt. Auch die Eingabe von Titel und Beschreibung sollte später so übernommen werden.

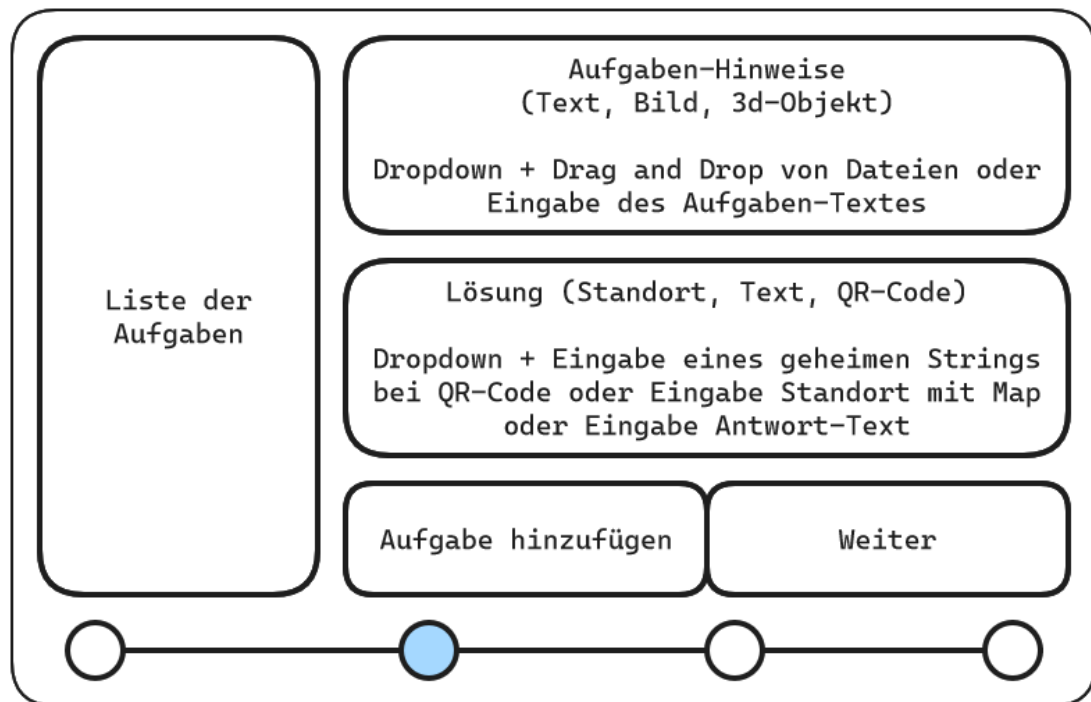


Abbildung 4.11.: Skizze für Anlegen von Aufgaben im Hunt-Editor

Nun kommt der schwierigste Teil, die Erstellung und Bearbeitung der Aufgaben in einer Schnitzeljagd. Hier war zunächst der Ansatz, auf der linken Seite eine Tabelle mit den Aufgaben zu führen. Durch Anklicken der entsprechenden Zeile wird dann auf der rechten Seite die Aufgabe und die Lösung angezeigt. Hier gibt es auch noch den Aufgabentyp "3D-Objekt", der am Ende nicht übernommen wurde.

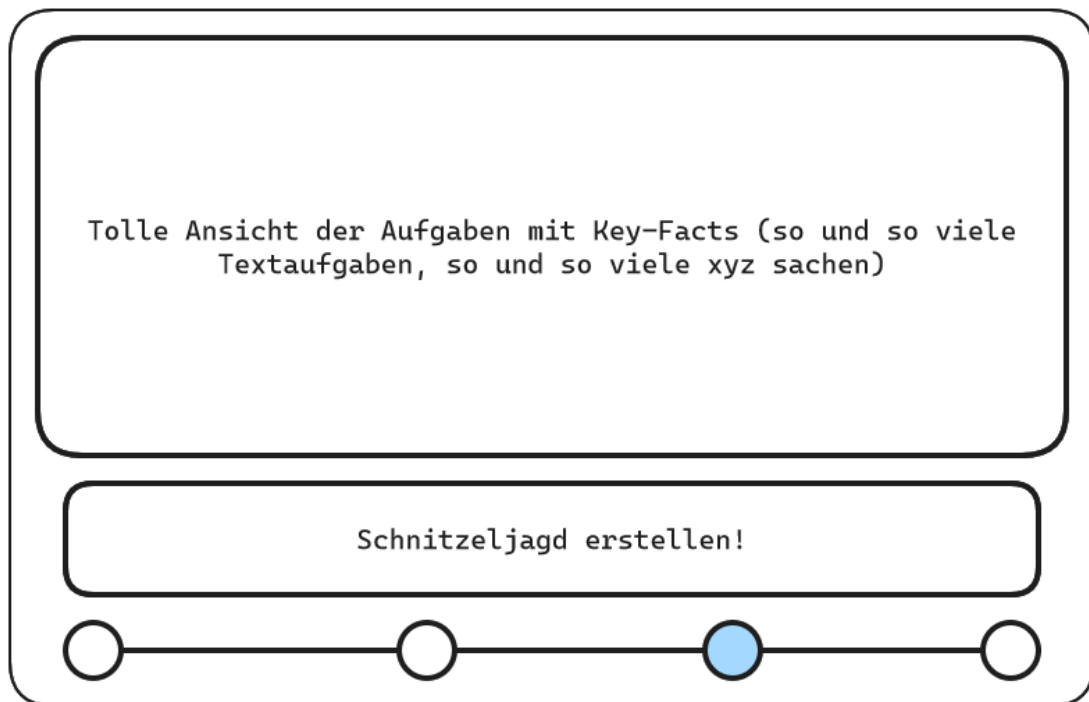


Abbildung 4.12.: Skizze zur Übersicht einer Schnitzeljagd im Hunt-Editor

Dieses Wireframe zeigt am Ende noch einmal alle wichtigen Informationen der Schnitzeljagd im Überblick. Dieses Konzept hat uns gut gefallen.

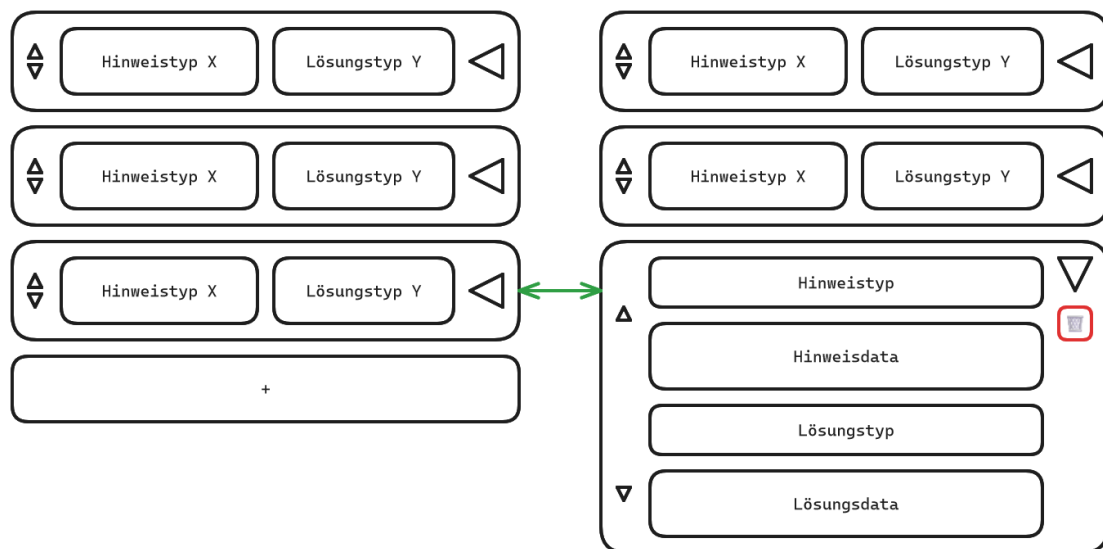


Abbildung 4.13.: Skizze zur Auflistung von Aufgaben im Hunt-Editor

Da die seitliche Liste in Abbildung 4.11 für die Darstellung der Aufgaben auf mobilen Endgeräten ungeeignet ist, haben wir uns für den Ansatz in Abbildung 4.13 entschieden. Hier werden die Aufgaben nacheinander angezeigt und ihre Reihenfolge kann über die Pfeiltasten verändert 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 sowie Eingabefelder / FileUpload oder eine Map-Integration für den Standort als Lösung.

4.4. Participation Web-App

4.4.1. Übersicht

Über die Participation Web App kann sich ein Teilnehmer für eine Schnitzeljagd anmelden. Dazu startet er die Anwendung und gelangt auf eine Willkommenseite. Hier kann er sehen, wie viele Teilnehmer und wie viele Teilnahmen es insgesamt gibt. Um an einer Schnitzeljagd teilzunehmen, kann der Teilnehmer eine E-Mail an den Organisator schreiben. Wenn er daraufhin einen Einladungslink für eine Schnitzeljagd erhält, muss er dort nur noch einen Benutzernamen und ein Passwort auswählen. Mit diesem Benutzernamen und Passwort loggt er sich in das im Kapitel 4.5 beschriebene Hunt-Game ein.

4.4.2. Wireframing

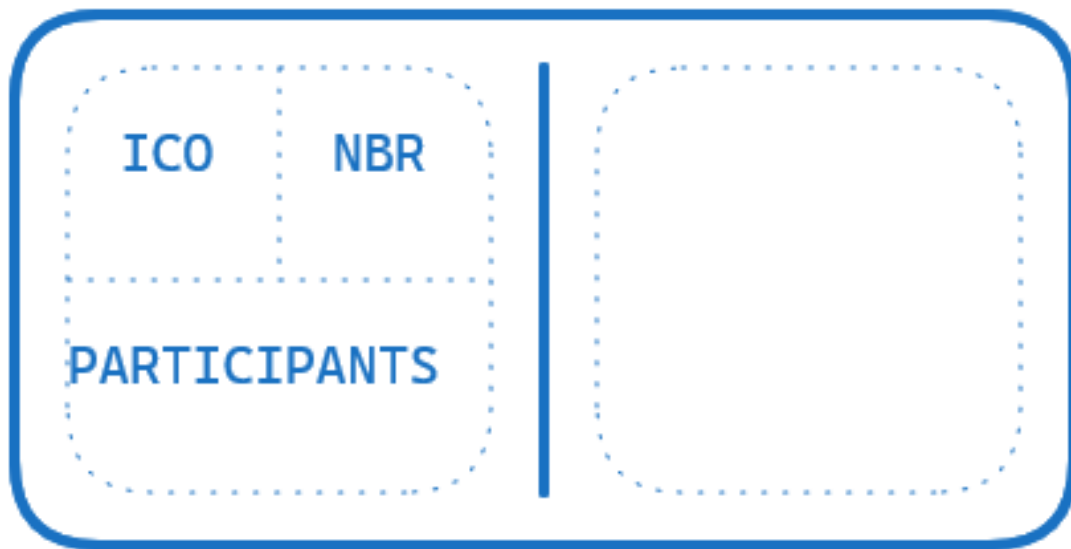


Abbildung 4.14.: Skizze zur Anzeige von Statistiken in der Participation Web-App

Um auf der Startseite der Participation Web-App interessante Statistiken anzeigen zu können, musste ein Konzept gefunden werden, wie diese ansprechend dargestellt werden können. Auf der linken Seite sollte die Anzahl aller Teilnehmer und auf der rechten Seite die Anzahl aller Beteiligungen dargestellt werden. Diese Informationen werden durch eine vertikale Trennlinie getrennt, um die verschiedenen Statistiken klar voneinander abzugrenzen.

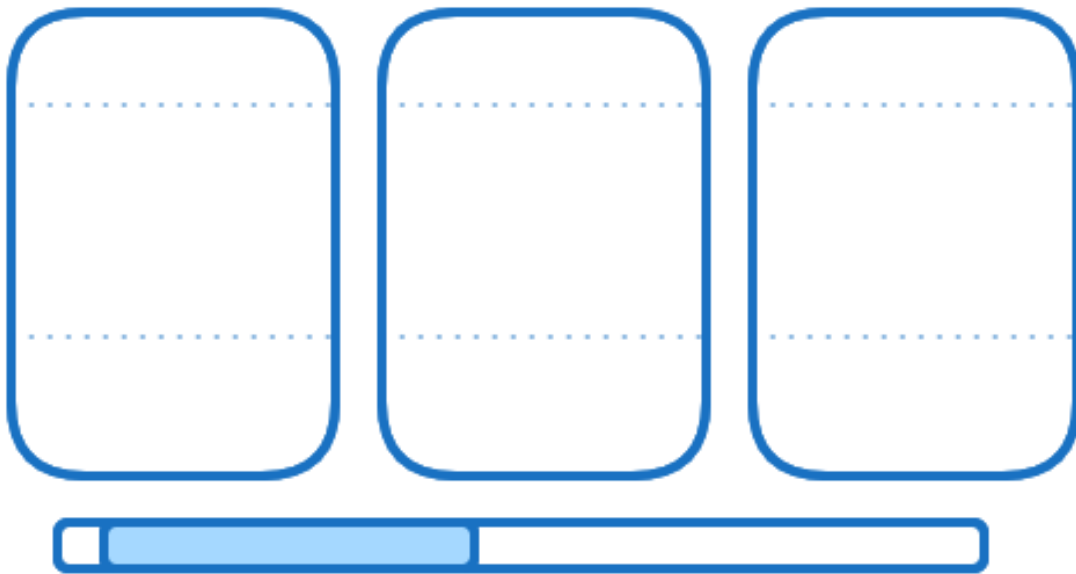


Abbildung 4.15.: Skizze zur Benachrichtigung eines Organisers in der Participation Web-App

Der Teilnehmer soll auch die Möglichkeit haben, einen Organisator um einen Einladungslink zu bitten. Dazu wurde das Wireframe erstellt, in dem die Organisatoren als Karten dargestellt werden. Hier kann auf mobilen Geräten durch horizontales Wischen zwischen den Organisatoren gewählt werden.

4.5. Hunt-Game Web-App

4.5.1. Übersicht

Beim Start der Anwendung sieht der Teilnehmer eine Sammlung von Bildern der Hochschule Aalen, darunter eine kleine Zeitleiste, die den Entwicklungsprozess der Projektarbeit beschreibt. Um an einer Schnitzeljagd teilnehmen zu können, muss sich der Teilnehmer mit dem gewählten Benutzernamen und Passwort der Participation Web-App anmelden. Anschließend wird ihm eine Übersicht aller Schnitzeljagden angezeigt, für die er sich registriert hat. Hierbei wird zwischen *Ongoing*, *Complete* und *Expired* unterschieden, was durch den aktuellen Teilnahme-Status bestimmt wird. Wählt er eine gültige Schnitzeljagd aus, startet der Spielablauf.

4.5.2. Spielablauf

Abbildung 4.16 stellt den Spielablauf als UML-Programmablaufplan dar.

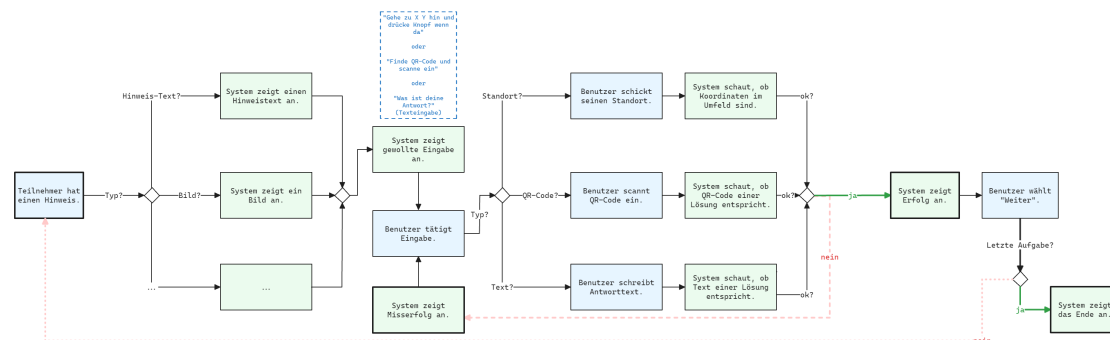


Abbildung 4.16.: Skizze Spielablauf als UML-Programmablaufplan

Nachdem der Teilnehmer eine gültige Schnitzeljagd ausgewählt hat, beginnt der eigentliche Spielablauf. Zu Beginn wird dem Teilnehmer der aktuelle Hinweis präsentiert, der ihn zur nächsten Station oder Aufgabe führen soll. Die Hinweise sind in zwei Arten unterteilt: Text und Bild. Bei einem Texthinweis wird dem Teilnehmer ein beschreibender Text angezeigt, der Informationen oder Anweisungen zur nächsten Station enthält. Bei einem Bildhinweis wird stattdessen ein Bild angezeigt, das visuelle Hinweise oder Details enthält, die zur Lösung der Aufgabe beitragen.

Unter dem Hinweis befindet sich ein Button, über den der Teilnehmer seine Lösung einreichen kann. Es gibt drei verschiedene Lösungstypen, die je nach Aufgabe variieren:

- **Textlösung:** Der Teilnehmer gibt seine Lösung in ein Textfeld ein. Dies kann z.B. ein gesuchtes Wort, ein Satz oder eine Zahlenkombination sein.
- **QR-Code:** Bei Aufgaben, die einen QR-Code erfordern, wird die Kamera des Gerätes aktiviert, um den QR-Code einzuscannen. Dieser QR-Code kann an verschiedenen Stellen versteckt sein und enthält die notwendigen Informationen oder Anweisungen, um zum nächsten Hinweis zu gelangen.
- **Location:** Wenn die Lösung in Form eines geographischen Ortes zur Verfügung gestellt werden soll, wird der Teilnehmer aufgefordert, den Zugriff auf den Ort zu erlauben. Der Browser ermittelt dann die aktuellen GPS-Koordinaten des Geräts und prüft, ob diese mit der erwarteten Lösung übereinstimmen.

Der Spielablauf wiederholt sich, bis alle Aufgaben gelöst sind. Bei den Lösungstypen Text und Ort erhält der Teilnehmer zusätzlich Hinweise, wenn seine Lösung nicht

korrekt ist. Diese Hinweise besagen, dass die Lösung fast richtig ist, aber noch angepasst werden muss. Dies soll den Teilnehmer ermutigen und ihm die Möglichkeit geben, seine Antwort zu verbessern.

5. Implementierung

5.1. Entwickeln des Frontends

5.1.1. Aufbau der Web-Anwendungen

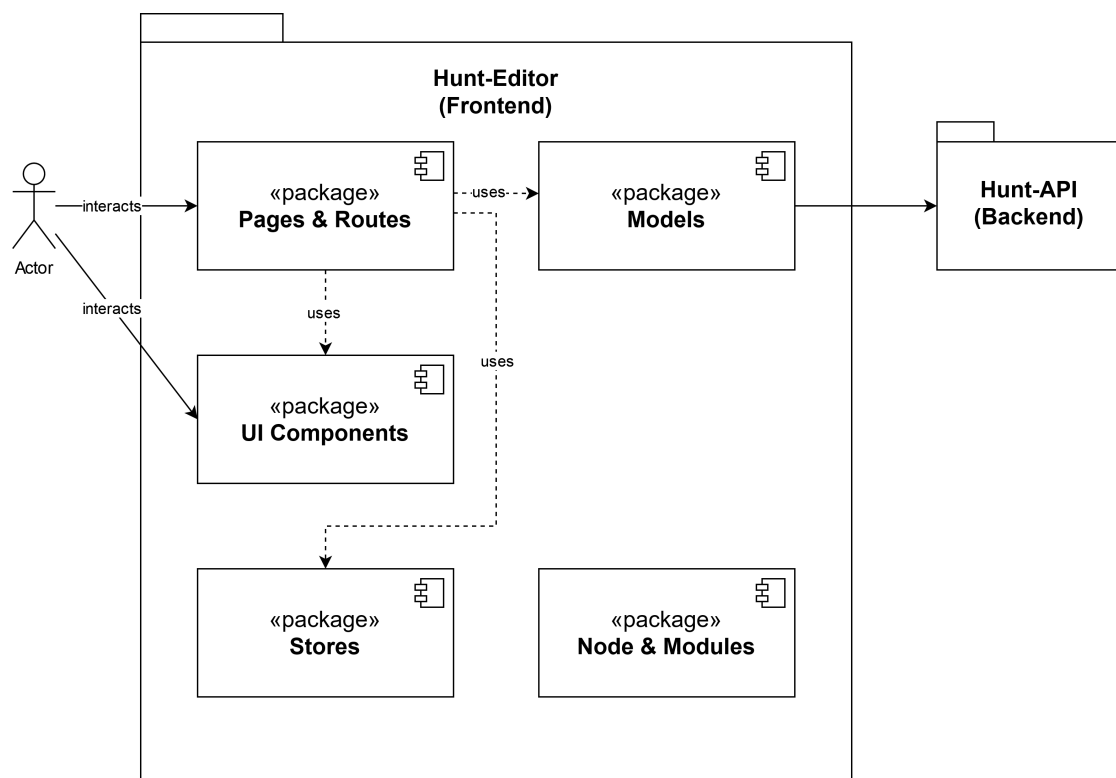


Abbildung 5.1.: UML-Diagramm Whitebox Frontend

Abbildung 5.1 veranschaulicht die grundlegenden Elemente, aus denen die Anwendungen bestehen. Für das Zwischenspeichern von anwendungsspezifischen Daten, wie beispielsweise Schnitzeljagd-Attribute (Titel, Beschreibung, Aufgaben) im Hunt-Editor oder der Teilnehmer-Token im Hunt-Game, werden Svelte Stores verwendet (vgl. Kapitel 2.2.3).

5.1.2. UI-Design

Um die Anwendungen zu erstellen, wurde zunächst versucht, mit DaisyUI zu arbeiten. Da dies nicht das gewünschte Ergebnis geliefert hat, wurde zu Flowbite-Svelte gewechselt. Kapitel A.2.1 begründet diese Entscheidung.

5.1.3. Dynamic Routing

In der Hunt-Editor Web-App wird das sogenannte *Dynamic Routing* für das Editieren von Schnitzeljagden verwendet. Hierbei wird über die Route `/edit/[huntId]` die *id* der Schnitzeljagd dem Platzhalter `[huntId]` in der Route eingefügt. Hierdurch kann Svelte über *Load-Functions* dynamisch die Informationen einer Schnitzeljagd abrufen und anwendungsspezifisch speichern.

Das Dynamic Routing findet auch in der Participation Web-App Verwendung. Über die Route `/participation/[huntId]` wird in einer Svelte *Load-Function* ebenso Titel und Beschreibung einer Schnitzeljagd geladen.

Im Anhang B.2 wird die Implementierung des Dynamic Routings näher verdeutlicht.

5.1.4. Walkthrough der Web-Anwendungen

Im Anhang B.3 wird zu den jeweiligen Web-Anwendungen ein Walkthrough geliefert.

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

Im Folgenden wird der Entwicklungsprozess für das Implementieren eines AR-Image-Trackers mithilfe des AR-Foundation Frameworks von Unity beschrieben. Im Anschluss erfolgt eine Erläuterung der Gründe, welche dazu führten, dass dieser Ansatz für das Projekt als nicht länger geeignet erachtet wurde.

Für die Entwicklung mit Unity wird die Installation des Unity-Hubs sowie einer unterstützten Version des Unity-Editors vorausgesetzt. Die im Projekt entwickelte Anwendung wurde mit der Editor-Version 2022.3.20f1 erstellt.

5.2.1. Anlegen einer Reference-Image-Library

Nach der Erstellung eines AR-Unity-Projekts gemäß [15] angelegt wurde und der erfolgreichen Integration des erforderlichen Frameworks, stehen im Editor verschiedene Funktionen des AR-Foundation-Frameworks zur Verfügung.

Zunächst soll eine *Reference-Image-Library* erstellt werden. Abbildung 5.2 markiert die grundlegenden Schritte hierfür.

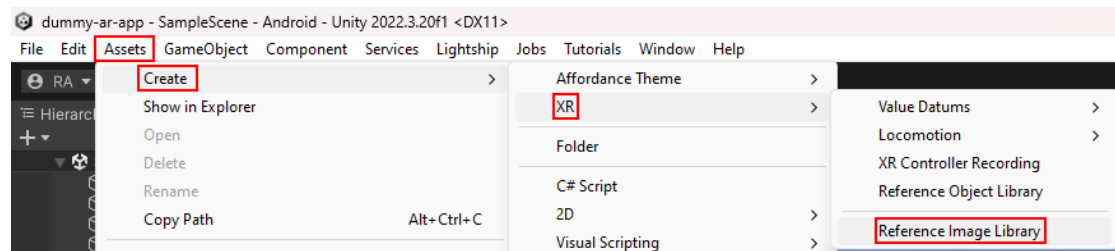


Abbildung 5.2.: Bildschirmabschnitt für das Erstellen einer Reference-Image-Library

In der geöffneten Szene in Abbildung 5.3 sollte nun ein leeres Reference-Image-Library Objekt existieren.

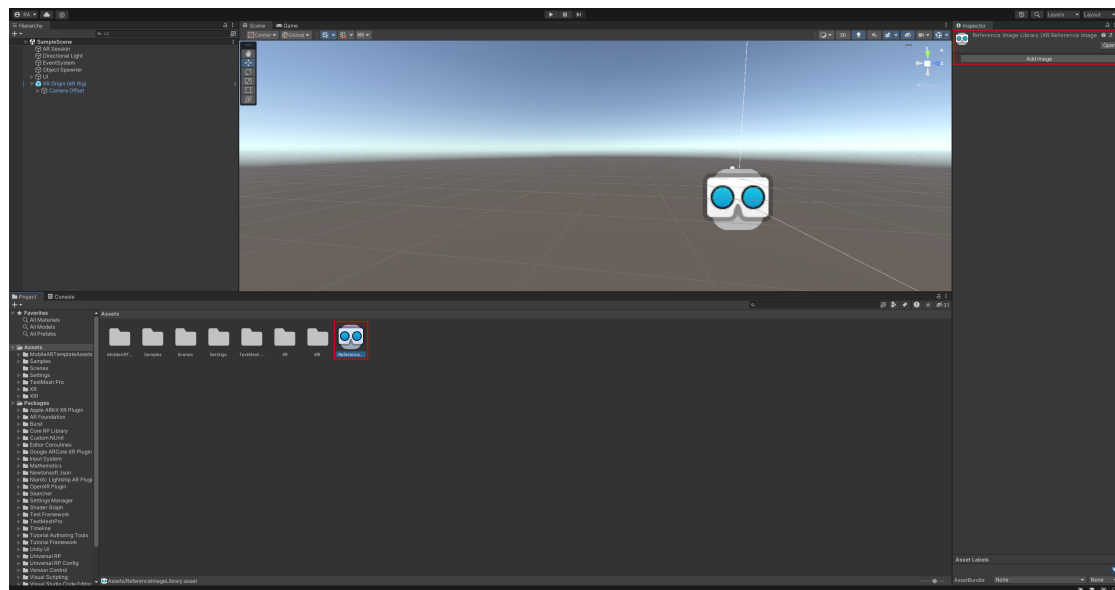


Abbildung 5.3.: Bildschirmabschnitt Reference-Image-Library Objekt im Editor

Die Reference-Image-Library wird benötigt, um bereits vor der Laufzeit bereits

Referenzbilder (*Reference-Images*) oder Objekte zu sammeln, die von einem AR-Tracker wie etwa der Komponente *XRIImageTrackingSubsystem* erkannt werden können.

5.2.2. Anlegen eines AR-Tracked-Image-Managers

Die Komponente *AR Tracked Image Manager* generiert *Game-Objects* für jedes gefundene Bild innerhalb des Viewports des Benutzers. Die Fähigkeit, Bilder zu finden, setzt voraus, dass die Komponente *Reference-Images* kennt, die in der *Reference-Image-Library* gespeichert sind.

Das Einbinden erfolgt durch die Anbindung eines Skripts an die *XR-Origin* Komponente und die entsprechende Einbindung der *Reference-Image-Library* per Drag-and-Drop. Dies aktiviert das Tracken von Bildern im AR-Raum.

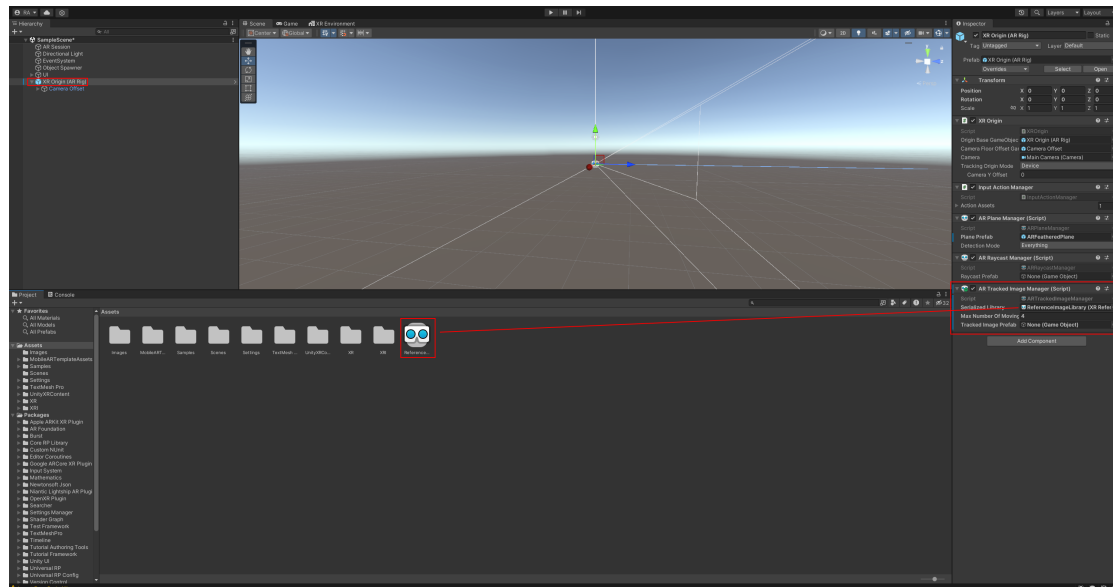


Abbildung 5.4.: Bildschirmabschnitt für das Erstellen eines AR-Tracked-Image-Managers

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

Die Erkennung eines QR-Codes setzt voraus, dass dieser zuvor in der *Reference-Image-Library* angelegt wurde. Der QR-Code kann dadurch als Textur eingelesen und über einen QR-Code-Konverter wie *ZXing* als Text konvertiert werden. Abbildung 5.5 zeigt das erfolgreiche Umkonvertieren eines QR-Codes im AR-Raum.

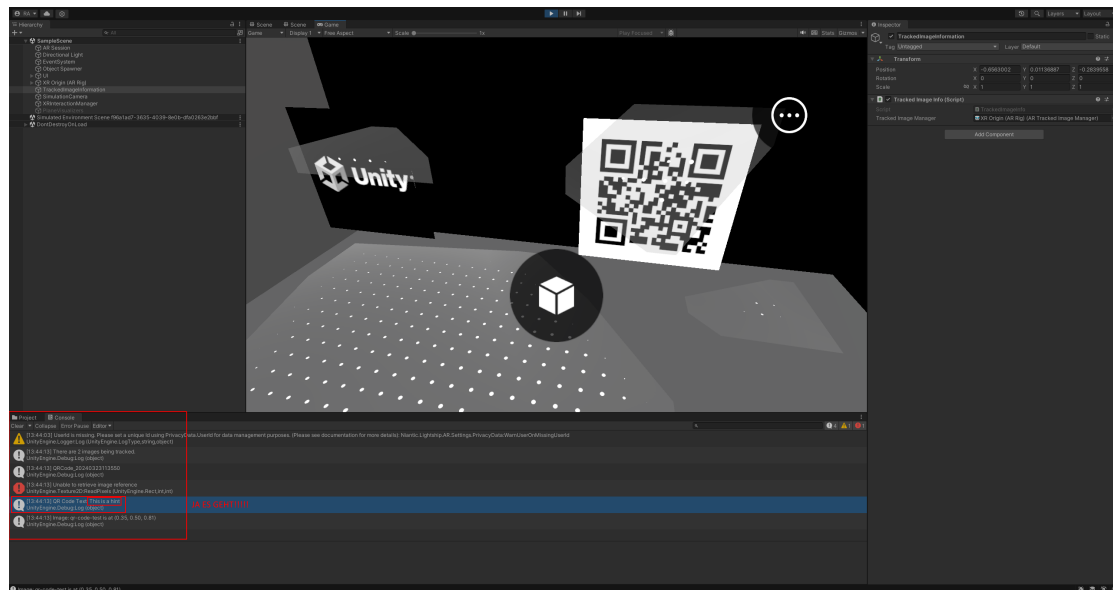


Abbildung 5.5.: Bildschirmabschnitt für das Lesen eines QR-Codes

5.2.4. Erfahrung mit Unity und AR-Foundation

Leider führte das in Kapitel 5.2 beschriebene Vorgehen nicht zum gewünschten Resultat.

Ein Hinweistext oder -bild an der Position des QR-Codes zu platzieren, könnte theoretisch funktionieren, wie in Abbildung 5.5 bereits dargestellt wird. Allerdings stellt die mangelnde Flexibilität der Reference-Image-Library ein erhebliches Hindernis für eine funktionierende Implementierung dar.

Um das beschriebene Problem zu vermeiden, wäre es erforderlich, dass 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.

Des Weiteren werden die Lösungen vor dem Erhalt eines ersten Hinweises auf dem Gerät des Teilnehmers geladen. Dies könnte zur Konsequenz haben, dass manche Lösungen aus den geladenen Bildern extrahiert werden können, wodurch das Konzept der Schnitzeljagd umgangen wird.

Die genannten Herausforderungen verdeutlichen, dass Unity und das AR Foundation Framework in diesem Kontext keine optimale Lösung darstellen. Die genannten

Einschränkungen erschweren die Entwicklung einer flexiblen, effizienten und sicheren Anwendung, die den Anforderungen einer Schnitzeljagd entspricht. Daher ist es empfehlenswert, alternative Technologien oder Ansätze zu berücksichtigen, die eine bessere Anpassungsfähigkeit und Sicherheit bieten.

Die Entscheidung, Unity als ganzes nicht für das Schnitzeljagd-Spiel zu verwenden, wird zudem in folgenden Aspekten begründet.

Entwicklungsumgebung und Tooling

Die Entwicklungsumgebung und der Entwicklungsprozess, die für die Erstellung einer Anwendung in Unity erforderlich sind, weisen erhebliche Nachteile auf. Das Tooling von Unity ist oft unhandlich und ineffizient. Häufig auftretende Probleme müssen durch Neustarts des Unity-Hubs oder des Unity-Editors gelöst werden. Diese häufigen Unterbrechungen führen zu einem erheblichen Zeitverlust, der den Entwicklungsprozess stark behindert.

Ein weiteres Problem ist die resultierende Größe der Anwendung nach dem Build. Die Unity-Engine erzeugt sehr große Dateien, selbst wenn nur grundlegende Funktionen verwendet werden. Eine Optimierung auf das Nötigste ist kaum möglich, was die Anwendung unnötig aufbläht und somit unpraktisch für mobile Geräte macht.

Anpassung auf Projektanforderungen

AR Foundation, die AR-Entwicklungsplattform von Unity, bietet für unsere Schnitzeljagd-Anwendung keine entscheidenden Vorteile. Unsere Anwendung konzentriert sich hauptsächlich auf die Darstellung von Texten und Bildern sowie das Scannen von QR-Codes, was keine umfassende AR-Funktionalität erfordert. Eine einfache und ressourcenschonende Lösung wäre hier weitaus effizienter.

Lizensierung und Business

In den vergangenen Monaten hat das Unternehmen *Unity Technologies*, welches hinter Unity steht, erheblich an Reputation verloren. Dies ist auf umstrittene Änderungen der Lizenzbedingungen und Preismodelle zurückzuführen. Diese Änderungen haben bei Entwicklern für Unmut gesorgt und das Vertrauen in das Unternehmen erschüttert. [16]

Die negativen Schlagzeilen und die Unsicherheit über zukünftige Geschäftsstrategien machen Unity zu einer risikobehafteten Wahl für langfristige Projekte.

5.2.5. Alternativen

Vuforia

Vuforia ist eine AR Plattform, die, ähnlich wie AR Foundation, ermöglicht, Augmented Reality in mobilen Anwendungen zu integrieren. Vuforia bietet verschiedene Funktionen wie Bilderkennung, Objektverfolgung und die Möglichkeit, 3D-Modelle in reale Umgebungen einzubetten. Diese Plattform ist besonders bekannt für ihre zuverlässige Erkennung und Nachverfolgung von Bildern und Objekten, was sie zu einer beliebten Wahl für AR-Entwickler macht [17].

Trotz dieser Fähigkeiten gibt es mehrere Gründe, warum Vuforia für das aktuelle Projekt nicht die optimale Wahl ist:

- **Abhängigkeit vom Unity-Editor:** Vuforia ist stark in den Unity-Editor integriert. Dies bedeutet, dass Entwickler, die Vuforia nutzen, auch die Unity-Plattform verwenden müssen, was bereits in Kapitel 5.2.4 als problematisch identifiziert wurde. Die Nachteile des Unity-Editors würden somit auch bei der Verwendung von Vuforia bestehen bleiben.
- **Alternativer Entwicklungsaufwand:** Eine Alternative zur Verwendung von Unity wäre die Integration von Vuforia in eine Android-App über Android Studio. Dies würde jedoch zusätzliche Lernkurven und Einarbeitungszeit erfordern, da ein neues Toolset gemeistert werden müsste. Für ein Projekt, das hauptsächlich auf Webtechnologien basiert, würde dies einen erheblichen zusätzlichen Aufwand bedeuten.
- **Kosten:** Vuforia hat eine Lizenzgebührstruktur, die je nach Nutzung und Funktionalitäten variieren kann. Die Preise können für kleinere Projekte oder akademische Arbeiten unerschwinglich sein [18].

Insgesamt bieten die Nachteile von Vuforia, insbesondere in Verbindung mit den bereits erwähnten Problemen von Unity, eine unzureichende Grundlage für die Umsetzung des Projekts. Die Kombination aus technischen Herausforderungen, zusätzlichem Entwicklungsaufwand und potenziellen Kosten macht Vuforia zu einer weniger idealen Wahl im Vergleich zu alternativen Ansätzen wie WebAR, die für die spezifischen Anforderungen des Projekts besser geeignet sind.

WebAR

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.

Schlussfolgerung

Falls die Schnitzeljagd Anwendung in Zukunft AR-Funktionen beinhalten soll, stellt WebAR die effizienteste und kostengünstigste Lösung dar. Es bietet eine einfache und zugängliche Möglichkeit, grundlegende AR-Funktionen bereitzustellen, ohne die Notwendigkeit einer komplexen Entwicklungsumgebung. Vuforia und Unity bieten erweiterte Funktionen, die für diese Anwendung nicht erforderlich sind und könnten daher überdimensioniert sein.

5.3. Algorithmen zur Hinweisbestimmung für Nutzerlösungen

Zwei Algorithmen wurden implementiert, um dem Benutzer Hinweise zu geben, dass eine von ihm eingegebene Lösung nahezu richtig ist. Diese Algorithmen helfen, den Grad der Übereinstimmung zwischen der vom Benutzer angegebenen Lösung und der tatsächlich gesuchten Antwort zu bestimmen.

5.3.1. Haversine-Algorithmus zur Bestimmung geografischer Distanzen

Einer der Algorithmen kommt zum Einsatz, wenn die Lösung einen geografischen Ort umfasst, der durch Breiten- und Längengrade (Latitude und Longitude) definiert ist. Hier wird die Haversine-Formel verwendet, um die Entfernung zwischen den beiden Punkten zu berechnen.

Die Haversine-Formel erweist sich als überaus nützlich bei der Ermittlung der kürzesten Entfernung über die Erdoberfläche, da sie die Krümmung der Erde adäquat berücksichtigt. Diese Berechnung ist von essenzieller Bedeutung, um zu

bestimmen, wie nahe ein Nutzer mit seiner Antwort an der korrekten Position liegt.

Der folgende C#-Code zeigt die Implementierung des Haversine-Algorithmus:

```
1 public static double Haversine(double lat1, double lon1,
2   double lat2, double lon2)
3 {
4     // Convert degrees to radians
5     double dLat = ToRadians(lat2 - lat1);
6     double dLon = ToRadians(lon2 - lon1);
7
8     lat1 = ToRadians(lat1);
9     lat2 = ToRadians(lat2);
10
11    // Haversine formula
12    double a = Math.Sin(dLat / 2) * Math.Sin(dLat / 2) +
13              Math.Sin(dLon / 2) * Math.Sin(dLon / 2) *
14    Math.Cos(lat1) * Math.Cos(lat2);
15    double c = 2 * Math.Atan2(Math.Sqrt(a), Math.Sqrt(1 -
16    a));
17
18    // Distance in meters
19    return RADIUS_OF_EARTH_METERS * c;
20 }
21
22 private static double ToRadians(double angleInDegrees)
23 {
24     return angleInDegrees * Math.PI / 180.0;
25 }
```

Quelltext 5.1: Code Ausschnitt zum Haversine Algorithmus in C#

In diesem Algorithmus wird zunächst die Differenz zwischen den Längen- und Breitengraden der beiden Punkte berechnet und in Radianen umgewandelt. Anschließend wird die Haversine-Formel angewendet, um den Anteil des großen Kreises (die kürzeste Strecke zwischen zwei Punkten auf der Oberfläche einer Kugel) zu berechnen, den der Abstand zwischen den Punkten darstellt. Dieser Anteil wird dann mit dem Radius der Erde multipliziert, um die tatsächliche Entfernung in Metern zu erhalten.

Die Berechnung ermöglicht dem System die Bestimmung der Distanz zwischen dem vom Nutzer angegebenen Ort und der korrekten Position. Bei einem geringen Abstand liefert das System ein Feedback, das darauf hinweist, dass die Lösung nahe dran ist.

Anwendungsbeispiel des Haversine-Algorithmus

Im folgenden wird der Haversine-Algorithmus zum Bestimmen der Distanz des Benutzers zur Lösung verwendet.

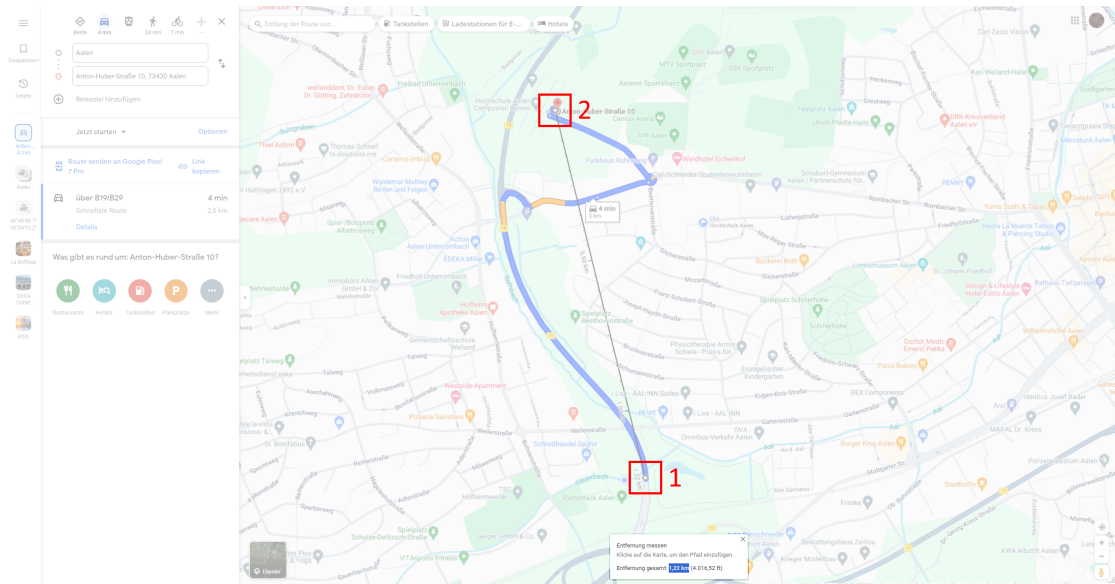


Abbildung 5.6.: Bildschirmabschnitt Abstand Teilnehmer und Lösung

Abbildung 5.6 stellt den Standort des Teilnehmers (Abbildung 5.6 - 1) zu einem gesuchten Standort (Abbildung 5.6 - 2) dar. Die tatsächliche Distanz entspricht in diesem Beispiel 1213.14 Meter.

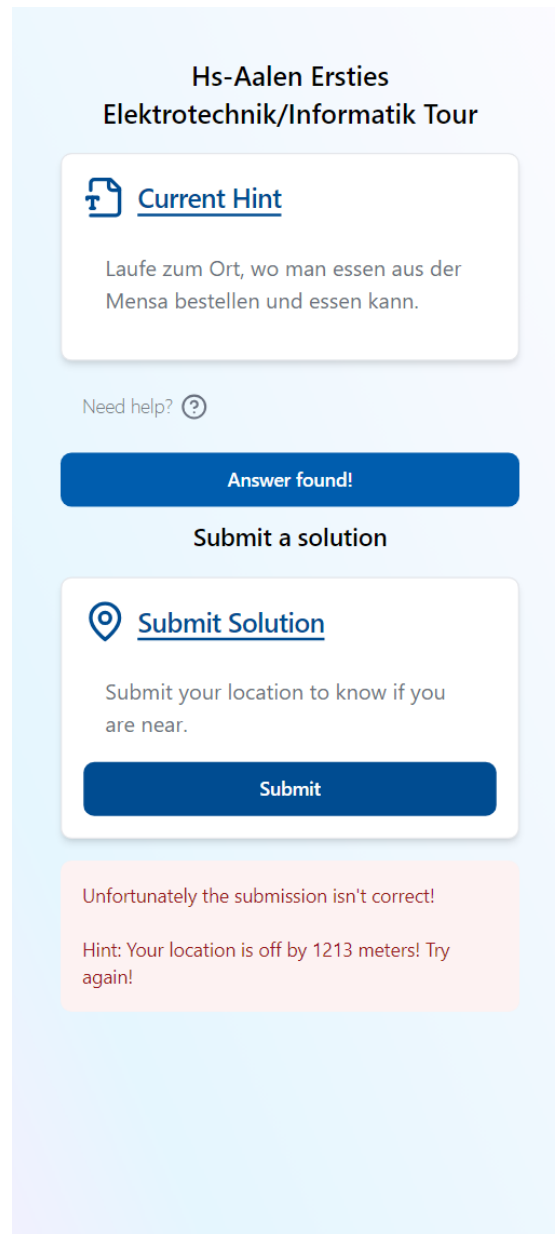


Abbildung 5.7.: Bildschirmabschnitt Abstands-Hinweis an Teilnehmer

Abbildung 5.7 zeigt den Zustand der Benutzeroberfläche an, nachdem ein Teilnehmer seine Position als Lösung an das System schickt. Es wurde korrekterweise erkannt, dass der Teilnehmer 1213.14 Meter von der gewollten Lösung entfernt ist.

5.3.2. Levenshtein-Algorithmus zur Bestimmung textueller Unterschiede

Um den Nutzern Feedback darüber zu geben, wie nah ihre eingesandte Antwort an der korrekten Lösung liegt, wird neben der Haversine-Formel auch ein Algorithmus zur Berechnung der Levenshtein-Distanz eingesetzt. Diese Methode wird verwendet, wenn die Lösung aus Textdaten besteht, wie etwa bei der Eingabe von Textantworten auf Rätsel oder Aufgabenstellungen.

Die Levenshtein-Distanz misst den Unterschied zwischen zwei Zeichenketten in Bezug auf die minimale Anzahl von Einfüge-, Lösch- und Ersetzungsoperationen, die erforderlich sind, um die eine Zeichenkette in die andere zu transformieren. Dies ist besonders nützlich, um Tippfehler oder leichte Abweichungen in den Antworten der Nutzer zu identifizieren und entsprechende Rückmeldungen zu geben, die darauf hinweisen, dass die Antwort fast richtig ist, wenn nur geringe Abweichungen vorliegen.

Der folgende C#-Code zeigt die Implementierung der Levenshtein-Distanz:

```
1 public static int LevenshteinDistance(string givenText,  
2     string expectedText)  
3     {  
4         int n = givenText.Length;  
5         int m = expectedText.Length;  
6         int[,] d = new int[n + 1, m + 1];  
7  
8         // Initialize the distance matrix  
9         for (int i = 0; i <= n; i++)  
10            {  
11                d[i, 0] = i;  
12            }  
13         for (int j = 0; j <= m; j++)  
14            {  
15                d[0, j] = j;  
16            }  
17  
18         // Compute the distances  
19         for (int i = 1; i <= n; i++)  
20            {  
21                for (int j = 1; j <= m; j++)  
22                {  
23                    int cost = (givenText[i - 1] ==  
24                        expectedText[j - 1]) ? 0 : 1;  
25  
26                    d[i, j] = Math.Min(  
27                        Math.Min(d[i - 1, j] + 1, d[i, j - 1] +  
28                        1),  
29                        d[i - 1, j - 1] + cost);  
30                }  
31            }  
32  
33         return d[n, m];  
34     }
```

Quelltext 5.2: Code Ausschnitt zum Levenshtein Distanz Algorithmus in C#

Die Matrix *d* wird so initialisiert, dass die ersten Zeilen und Spalten mit aufsteigenden Werten gefüllt sind. Diese Werte entsprechen den Kosten für das Einfügen oder Löschen von Zeichen, um die Länge der anderen Zeichenfolge zu erreichen. Die Hauptlogik des Algorithmus durchläuft die Matrix und berechnet die minimalen Kosten für die Transformation der Teilstrings. Dies geschieht durch Vergleich der aktuellen Zeichen der beiden Zeichenketten und Anwendung der entsprechenden

Operationen (Einfügen, Löschen, Ersetzen). Der letzte Wert der Matrix $d[n, m]$ gibt die minimale Anzahl von Transformationen an, die erforderlich sind, um die beiden Zeichenketten einander anzugleichen.

Die Levenshtein-Distanz ermöglicht es dem System, eine präzise Rückmeldung darüber zu geben, wie nahe der Benutzer an der richtigen Antwort ist, und unterstützt so eine positive Benutzererfahrung durch konstruktives Feedback.

Anwendungsbeispiel des Levenshtein-Algorithmus

Im Folgenden wird der Levenshtein-Algorithmus zum Bestimmen der Anzahl an abweichenden Ziffern zu einem gesuchten Lösungstext verwendet.

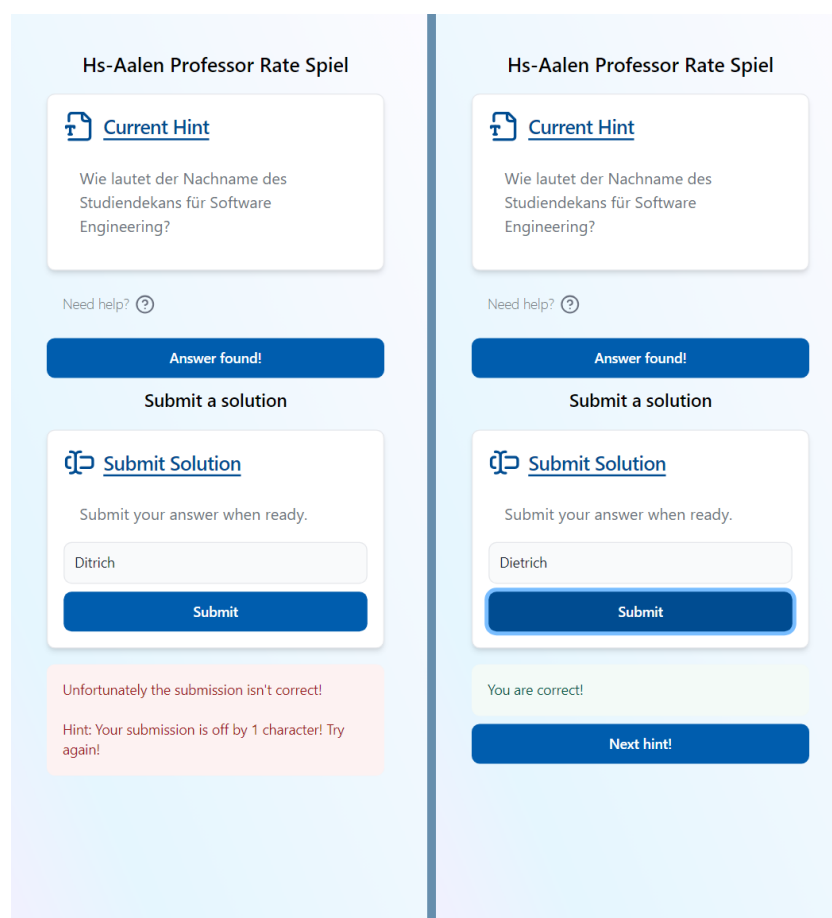


Abbildung 5.8.: Bildschirmabschnitte Text Unterschied an Teilnehmer

Der linke Bildschirmabschnitt in Abbildung 5.8 zeigt den Hinweis, wenn sich ein Teil-

nehmer in seiner Lösung um eine Ziffer vertan hat. Der rechte Bildschirmabschnitt in Abbildung 5.8 wird angezeigt, wenn der Benutzer eine korrekte (Text-)Lösung eingegeben hat.

6. Inbetriebnahme

6.1. Einführung

In diesem Kapitel werden die Maßnahmen für das Deployment der Anwendung näher beschrieben. Anhand einer Übersicht wird gezeigt, wie das System im laufenden Zustand aussieht und wie die unterschiedlichen Anwendungen mit dem Backend interagieren.

6.2. Backend Containerisierung

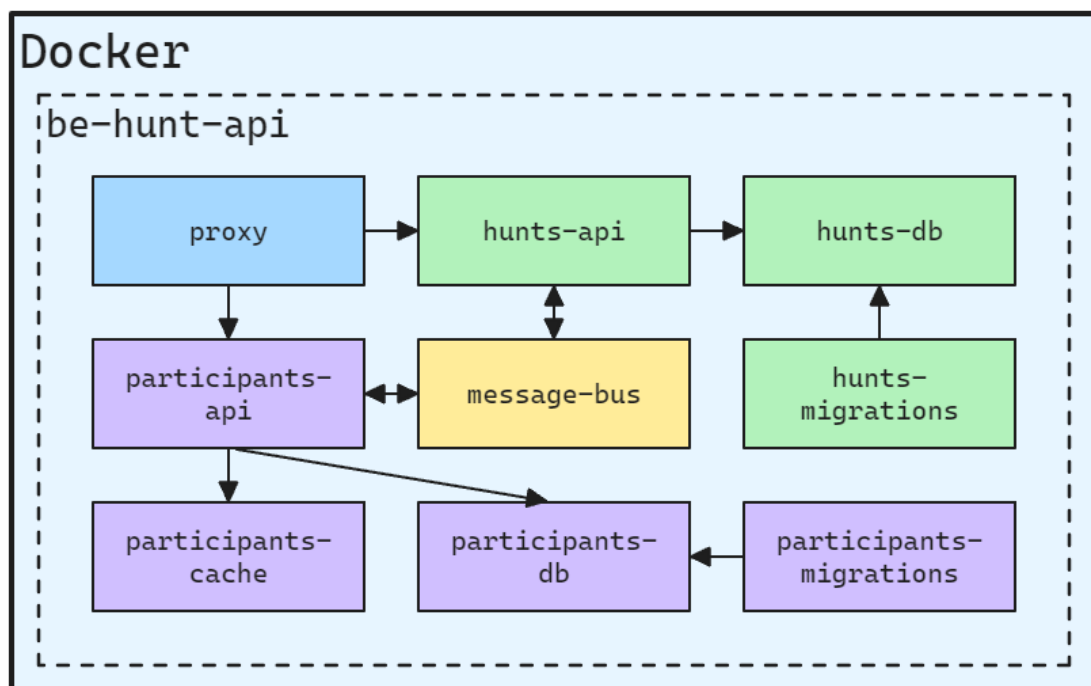


Abbildung 6.1.: UML-Diagramm für das Deployment

Abbildung 6.1 stellt die Deployment-Ansicht des Backends dar. Das Deployment wird durch die Docker Containerisierung ermöglicht (vgl. Kapitel 2.2.4). Die Konfiguration der verschiedenen Dienste erfolgt über *Docker-Compose* definiert.

Der Proxy (vgl. Abbildung 6.1 - blau) fungiert als zentraler Einstiegspunkt für die Anwendungen von außen. Er leitet alle Anfragen an den jeweils zuständigen Dienst weiter. Im Falle von falschen oder ungültigen Anfragen wird die Verbindungsanfrage beendet. Die Implementierung und Konfiguration des Proxies ist im Anhang B.1 beschrieben.

Die beiden Container *Participants-API* und *Hunts-API* (vgl. Abbildung 6.1 - grün und violett) repräsentieren die deployten Release-Builds der entworfenen Services aus Kapitel 4.2. Diese verfügen jeweils über einen eigenen Datenbank-Container zur persistenten Datenhaltung.

Für die initiale Erstellung und Aktualisierung bestehender Datenbank-Container sind zusätzlich die beiden Migration-Container *Participants-Migrations* und *Hunts-Migrations* vorgesehen (vgl. Abbildung 6.1 - grün und violett). Beide öffnen eine Verbindung zur jeweiligen Datenbank und erstellen die initial benötigten Datenbanktabellen, damit diese für die Verwendung im jeweiligen Service mit den aktuellen Datenbanktabellen existieren.

Die jeweiligen Dockerfiles sind im Anhang näher beschrieben (vgl. Anhang B.1).

6.3. Frontend Containerisierung

Da das Frontend mit Web-Technologien (vgl. Kapitel 2.2.3) implementiert wurde, kann es als plattformunabhängige Browseranwendung auf einem Webserver deployt werden. Dies ist auch über Docker Containerisierung möglich, wurde aber im Rahmen des Projektes nicht entworfen und getestet.

Für das Deployment der Hunt-Editor Web-App für Desktops wäre es auch möglich, eine echte Anwendung daraus zu machen. Hierfür stehen bereits implementierte Technologien wie *Tauri* zur Verfügung (siehe [19] und [20]).

7. Zusammenfassung und Ausblick

7.1. Erreichte Ergebnisse

Im Rahmen der Projektarbeit wurde eine interaktive Schnitzeljagd-Anwendung entwickelt, die speziell auf die Bedürfnisse von Studienanfängern zugeschnitten ist. Das Ziel, den neuen Studierenden den Campus und dessen Einrichtungen auf spielerische Weise näherzubringen, wurde erfolgreich erreicht. Die entwickelte Lösung umfasst ein robustes Backend auf Basis von .NET, welches nicht nur skalierbar ist, sondern auch einfach erweiterbar, um neue Hinweis- und Lösungstypen zu integrieren.

Ein zentraler Aspekt der Implementierung war die Flexibilität des Systems. Die Trennung von Aufgabenstellung und Lösung ermöglicht eine dynamische Erweiterbarkeit, ohne dass erhebliche Änderungen an der bestehenden Infrastruktur erforderlich sind. Sowohl der Web-Editor als auch das Web Game wurden mithilfe von SvelteKit und Komponenten-basiertem Design entwickelt. Dies erlaubt eine leichte Erweiterbarkeit und Anpassung der Benutzeroberflächen und Funktionalitäten. Der benutzerfreundliche Web-Editor ermöglicht die Verwaltung von Schnitzeljagden durch das Anlegen und Editieren von Aufgaben, während das Web Game alle bereits implementierten Hinweis- und Lösungstypen unterstützt, darunter Texte, Bilder, QR-Codes und geolocation-basierte Aufgaben. Dies gewährleistet eine umfassende und interaktive Spielerfahrung.

Insgesamt wurden die gesteckten Ziele erreicht und das System konnte in einem Testlauf erfolgreich eingesetzt werden.

7.2. Ausblick

In Zukunft bietet die entwickelte Lösung zahlreiche Möglichkeiten zur Weiterentwicklung und Anpassung. Eine naheliegende Erweiterung ist die Integration weiterer Hinweis- und Lösungstypen, um die Schnitzeljagden noch abwechslungsreicher zu gestalten. Auch die Einführung von WebAR-Technologien könnte in Betracht gezogen werden, um Augmented Reality-Elemente einzubinden und die Spielerfahrung weiter zu verbessern. Dies wäre insbesondere interessant für Aufgaben, die

eine visuelle Interaktion mit der realen Umgebung erfordern.

Ein weiterer Entwicklungsstrang könnte die Implementierung von Gamification-Elementen sein, wie etwa Ranglisten oder Belohnungssysteme, um die Motivation der Teilnehmer zu steigern. Darüber hinaus könnten Sicherheitsaspekte, wie die Verwendung kryptographischer Hashfunktionen zur Speicherung von Benutzerdaten, weiter ausgearbeitet werden, um die Datenintegrität und den Datenschutz zu verbessern.

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] Microsoft. *Introduction to ASP.NET Core*. 2024. URL: <https://learn.microsoft.com/en-us/aspnet/core/introduction-to-aspnet-core?view=aspnetcore-8.0> (besucht am 04. 07. 2024).
- [7] Microsoft. *Entity Framework Core Documentation*. 2021. URL: <https://learn.microsoft.com/en-us/ef/core/> (besucht am 04. 07. 2024).
- [8] Unity Technologies. *Unity Documentation*. 2024. URL: <https://docs.unity.com/> (besucht am 04. 07. 2024).
- [9] Unity. *AR Foundation*. 2024. URL: <https://unity.com/de/unity/features/arfoundation> (besucht am 20. 04. 2024).
- [10] Salad Lab. *Json.Net for Unity3D*. 2016. URL: <https://github.com/SaladLab/Json.Net.Unity3D/#upm> (besucht am 23. 04. 2024).
- [11] Daniel Fortes. *AR + GPS Location*. 2023. URL: <https://assetstore.unity.com/packages/tools/integration/ar-gps-location-134882> (besucht am 23. 04. 2024).
- [12] Svelte contributors. *Svelte - Cybernetically enhanced web apps*. 2024. URL: <https://svelte.dev/> (besucht am 20. 03. 2024).
- [13] 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).

- [14] 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).
- [15] Unity. *Project Setup*. 2024. URL: <https://docs.unity3d.com/Packages/com.unity.xr.arfoundation@5.1/manual/project-setup/project-setup.html> (besucht am 20. 04. 2024).
- [16] Paul Tassi. *The Entire Gaming Industry Unites Against Unity's Baffling Pricing Change*. 2023. URL: <https://www.forbes.com/sites/paultassi/2023/09/13/the-entire-gaming-industry-unites-against-unitys-baffling-pricing-change/> (besucht am 26. 07. 2024).
- [17] PTC, Inc. *Vuforia Features*. 2024. URL: <https://developer.vuforia.com/library/getting-started/vuforia-features> (besucht am 27. 07. 2024).
- [18] PTC, Inc. *Vuforia Engine Pricing*. 2024. URL: <https://www.ptc.com/en/products/vuforia/vuforia-engine/pricing> (besucht am 27. 07. 2024).
- [19] *GitHub.com - Tauri*. 2024. URL: <https://github.com/tauri-apps/tauri> (besucht am 30. 07. 2024).
- [20] *Tauri*. 2024. URL: <https://tauri.app/> (besucht am 30. 07. 2024).

A. Anhang: Entscheidungen bei der Implementierung

A.1. Architektonische Entscheidungen

A.1.1. Wahl eines Message-Bus für das Hunt-API Backend

Kontext

Im Rahmen der Konzeption der Backend Architektur ist zu berücksichtigen, dass das Erstellen von Schnitzeljagden und die Teilnahme an diesen zwei unterschiedlichen Domänen zuzuordnen sind. Allerdings spielen die Erstellung neuer Schnitzeljagden sowie das Bearbeiten und Entfernen bestehender Schnitzeljagden im Rahmen der Teilnahme eine wesentliche Rolle.

Entscheidung

Für die (unidirektionale) Kommunikation von dem Hunts-Service zu dem Participants-Service soll ein Message-Bus verwendet werden.

Begründung

- **Validierung und Entkopplung:** Einkommende Events können mühelos von einem *Event-Processor* abgefangen und gekoppelt verarbeitet werden. Durch die Verwendung eines Message-Bus wird die direkte Abhängigkeit zwischen Hunts-Service und Participants-Service minimiert. Beide Services können unabhängig voneinander weiterentwickelt und gewartet werden.
- **Erweiterbarkeit:** Ein Message-Bus ermöglicht es, weitere Services problemlos hinzuzufügen, die ebenfalls auf die Nachrichten des Hunts-Service reagieren können. Dies erleichtert zukünftige Erweiterungen des Systems.
- **Skalierbarkeit:** Der Message-Bus kann die Last verteilen und Nachrichten in einer Weise verarbeiten, die es ermöglicht, die Anzahl der Teilnehmer oder

die Komplexität der Schnitzeljagden ohne Leistungseinbußen zu erhöhen.

Konsequenzen

- **Operationaler Overhead:** Die Einführung und Wartung eines Message-Bus erfordert zusätzliche Infrastruktur und Überwachung, was den operativen Aufwand erhöhen kann.
- **Latenz:** Da die Kommunikation über einen Message-Bus asynchron erfolgt, kann es zu einer zusätzlichen Latenz kommen, bis Nachrichten verarbeitet werden.
- **Komplexität des Debuggings:** Das Debugging von Problemen kann komplexer werden, da die Nachrichtenverarbeitung verteilt ist und möglicherweise in mehreren Services parallel abläuft.

A.1.2. Wahl eines Api-Gateways für das Hunt-API Backend

Kontext

Im Rahmen der Konzeption der Backend Architektur ist zu berücksichtigen, dass mehrere Dienste (Microservices) vorhanden sind.

Entscheidung

Über ein Api-Gateway soll die Kommunikation des Backends einheitlich erfolgen.

Begründung

Ein API-Gateway bietet eine zentrale Anlaufstelle für Anfragen an die verschiedenen Microservices. Dies vereinfacht die Verwaltung und Überwachung der Kommunikation zwischen den Diensten und den externen Clients. Durch die einheitliche Schnittstelle des API-Gateways wird die Notwendigkeit eliminiert, dass Clients direkt mit den einzelnen Microservices interagieren müssen. Dies reduziert die Komplexität auf Client-Seite und ermöglicht eine bessere Kontrolle und Sicherheit der gesamten Systemarchitektur. Außerdem kann das API-Gateway zusätzliche Funktionen wie Authentifizierung, Autorisierung und Lastverteilung bereitstellen, was die Gesamtarchitektur robuster und skalierbarer macht.

Konsequenzen

Durch die Einführung eines API-Gateways wird die Interaktion mit den Microservices zentralisiert, was die Verwaltung und Wartung des Systems erleichtert. Clients müssen nicht mehr direkt mit jedem einzelnen Dienst kommunizieren, sondern nur noch mit dem API-Gateway, was die Schnittstellenlandschaft vereinheitlicht. Darüber hinaus bietet das API-Gateway die Möglichkeit, sicherheitsspezifische Maßnahmen wie Authentifizierung und Autorisierung über spezielle Header zu implementieren. Langfristig führt dies zu einer robusteren und besser skalierbaren Systemarchitektur, die flexibler auf zukünftige Anforderungen reagieren kann.

A.1.3. Hinweise und Lösungen ohne Abstrakte Klassen

Kontext

Für das Speichern der Aufgaben (*Assignments*) einer Schnitzeljagd soll es möglich sein, aus unterschiedlichen Arten von Hinweisen (*Hints*) und Lösungen (*Solutions*) zu wählen.

Entscheidung

Statt für jeden Hinweis-Typ und Lösungs-Typ unterschiedliche abstrakte Datenklassen zu definieren, werden diese lediglich als Daten-String gespeichert.

Begründung

Es wurde im Rahmen der Projektdurchführung versucht, mit abstrakten Datenklassen zu arbeiten. Jedoch erhöht dies die Komplexität enorm.

Zudem ist diese Lösung nicht skalierbar, denn es wird mehr Aufwand erfordert, einen neuen Hinweis-Typ oder Lösungs-Typ in das System einzubauen. Zunächst muss eine neue Datenklasse definiert werden, Dann muss diese in das Datenbank-Modell als konkrete neue Tabelle eingefügt werden.

In den meisten Fällen unterscheiden sich die Hinweis-Typen oder Lösungs-Typen kaum am Datenformat. Texte oder Bilder werden im System als String gespeichert. Daher entspringt die zentrale Frage: Wieso überhaupt mit abstrakten Klassen alles verkomplizieren?

Konsequenzen

- **Keine Type-Safety:** Geo-Locations im Format *lat;lon* werden als verketteter String gespeichert und müssen beim Ausleseprozess jeweils umgewandelt werden, was sehr fehleranfällig ist.

A.2. Entscheidungen im Frontend

A.2.1. Wahl von Flowbite statt DaisyUI

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 A.1: Code Ausschnitt 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 A.2: Code Ausschnitt 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.

B. Anhang: Implementierung und Inbetriebnahme

Hinweis: Der gesamte im Rahmen der Projektarbeit entstandene Source-Code steht im folgenden Github-Repository zur Verfügung: [xLolalilo/hsaa-projektarbeit](#).

B.1. Containerisierung des Backends

Im Projekt-Repository befinden sich folgende Dateien für das Realisieren der Containerisierung des Backends:

- **Hunts-** und **Participants** Api Dockerfiles: Siehe [hier](#) und [hier](#).
- **Migrations:** Siehe [hier](#) und [hier](#).
- **Proxy** Konfiguration und Dockerfile: Siehe [hier](#).
- **Docker-Compose:** Siehe [hier](#).

B.2. Dynamic Routing in Svelte

Das dynamic Routing findet beispielsweise in der Participation Web App Verwendung. Siehe [hier](#).

B.3. Walkthrough der Web-Apps

Walkthroughs zu den unterschiedlichen Web Anwendungen können [hier](#) heruntergeladen werden.