

Augmented Reality für Flutter Web: Entwicklung eines Plugins anhand eines Praxisbeispiels

BACHELORARBEIT

ausgearbeitet von

Felix Günthner

zur Erlangung des akademischen Grades

BACHELOR OF SCIENCE (B.Sc.)

vorgelegt an der

TECHNISCHEN HOCHSCHULE KÖLN

CAMPUS GUMMERSBACH

FAKULTÄT FÜR INFORMATIK UND

INGENIEURWISSENSCHAFTEN

im Studiengang

MEDIENINFORMATIK

Erster Prüfer:in:: Prof. Christian Noss
Technische Hochschule Köln

Zweiter Prüfer:in:: Dominik Deimel
Technische Hochschule Köln

Köln, Oktober 2023

Kontakt Details:

Felix Günthner
Kyffhäuserstr. 57
50674 Köln
felix-guenthner@web.de
Matrikelnummer: 11127239

Prof. Christian Noss
Technische Hochschule Köln
Fakultät für Informatik und Ingenieurwissenschaften
Steinmüllerallee 1
51643 Gummersbach
christian.noss@th-koeln.de

Dominik Deimel
Technische Hochschule Köln
Fakultät für Informatik und Ingenieurwissenschaften
Steinmüllerallee 1
51643 Gummersbach
dominik.deimel@th-koeln.de

Kurzfassung

Im Fokus dieser Arbeit steht die Erweiterung einer existierenden Dating-App, die mit dem Cross-Plattform Framework Flutter entwickelt wurde. Die App soll um das Feature „Virtuelle Geschenke“ mittels Augmented Reality (AR) erweitert werden. Die Arbeit adressiert das Fehlen eines AR-Plugins für Flutter Web, das AR-Erlebnisse für Flutter Web ermöglicht. Das angestrebte Ziel ist die Entwicklung eines AR-Plugins, das den spezifischen Anwendungsfall „Virtuelle Geschenke“ unterstützt und die bestehende Lücke im Flutter-Ökosystem schließt. Vor dem Hintergrund einer Prognose des mobilen AR-Markts, der bis 2026 einen Wert von über 36 Milliarden US-Dollar erreichen soll, wird das Potenzial der AR-Technologie und ihre Bedeutung für das Flutter-Ökosystem verdeutlicht.

Um das Ziel zu erreichen, wurde eine Anforderungserhebung durchgeführt und die Entwicklung in zwei Sprints aufgeteilt. Das Ziel konnte weitgehend erreicht werden. Es erfüllt vier von fünf funktionalen und sieben von acht nicht-funktionalen Anforderungen. Die Interaktion mit AR-Objekten und die universelle Kompatibilität mit Smartphones und Webbrowsern wurden jedoch als Einschränkungen identifiziert. Das Plugin ist [hier](#) verfügbar.

Die Ergebnisse dieser Arbeit erweitern die Möglichkeiten von Flutter für die Erstellung interaktiver, immersiver Web-Erlebnisse und positionieren Flutter als wichtigen Akteur im Bereich der AR-Webanwendungen. Die Arbeit unterstreicht das Potenzial von Flutter im Web-Bereich und zeigt, dass es mit etablierten Web-Frameworks konkurrieren kann. Trotz einiger Limitationen und Herausforderungen bietet die Forschungsarbeit eine solide Grundlage für zukünftige Entwicklungen und leistet einen wertvollen Beitrag zum Flutter-Ökosystem und zur Web-Entwicklung im Allgemeinen. Das entsprechende GitHub-Repository mit dem Quellcode und weiteren Informationen finden Sie [hier](#).

Keywords: Flutter Web, Plugin, Augmented Reality, WebXR, Three.js

Abstract

The focus of this thesis is on extending an existing dating app developed with the cross-platform framework Flutter. The app is to be enhanced with the feature „Virtual Gifts“ through Augmented Reality (AR). The work addresses the lack of an AR plugin for Flutter Web that enables AR experiences for Flutter Web. The desired goal is to develop an AR plugin that supports the specific use case of „Virtual Gifts“ and fills the existing gap in the Flutter ecosystem. Against the backdrop of a forecast for the mobile AR market, which is expected to reach a value of over 36 billion US dollars by 2026, the potential of AR technology and its significance for the Flutter ecosystem are highlighted.

To achieve the goal, a requirements gathering was conducted, and the development was divided into two sprints. The objective was largely achieved, fulfilling four out of five functional and seven out of eight non-functional requirements. However, interaction with AR objects and universal compatibility with smartphones and web browsers were identified as limitations. The plugin is available [here](#).

The results of this work expand the capabilities of Flutter for creating interactive, immersive web experiences and position Flutter as a key player in the field of AR web applications. The work underscores the potential of Flutter in the web domain and demonstrates that it can compete with established web frameworks. Despite some limitations and challenges, the research provides a solid foundation for future developments and makes a valuable contribution to the Flutter ecosystem and web development in general. The corresponding GitHub repository with the source code and additional information can be found [here](#).

Keywords: Flutter Web, Plugin, Augmented Reality, WebXR, Three.js

Inhaltsverzeichnis

1 Einleitung	5
1.1 Relevanz	5
1.2 Forschungsfragen und Ziele	7
1.3 Motivation	8
1.4 Aufbau der Arbeit	8
2 Hintergrund	9
2.1 Augmented Reality	9
2.1.1 Die dynamische Welt der Augmented Reality	9
2.1.2 Varianten von AR	10
2.1.3 AR, VR und Mixed Reality: Ein Überblick	10
2.2 Flutter Web	11
2.2.1 Möglichkeiten und Herausforderungen	11
2.2.2 Paketmanager Pub	12
3 Methodik	14
3.1 Methoden und Techniken	14
3.2 Technologien und Tools	14
3.3 Design und Architektur	16
3.3.1 Grundlegende Designprinzipien	16
3.3.2 Architekturüberblick	17
3.4 Implementierungsschritte	26
4 Anforderungserhebung	28
4.1 Funktionale Anforderungen	28
4.2 Nicht-funktionale Anforderungen	29
5 Implementierung	30
5.1 Sprint 1	30
5.1.1 Einführung in die Flutter-Plugin-Entwicklung	30
5.1.2 Zugriff auf Web-APIs mit der html-Bibliothek	31
5.1.3 Dart trifft JavaScript: Integration und Interaktion	31
5.1.4 Integration von Three.js	35
5.1.5 Fallback-Konzept	36
5.1.6 Implementierung von AR-Basismodulen	37
5.1.7 Implementierung von Three.js-Basismodulen	44
5.1.8 Integration der Basismodule	49
5.2 Sprint 2	54
5.2.1 Optimierung der Module und Architektur	54
5.2.2 Entwicklung von Widgets	56
5.2.3 3D-Modellformate und Three.js	60
5.2.4 Veröffentlichung des Plugins	63

6 Herausforderungen	66
7 Fazit und Ausblick	68
7.1 Fazit	68
7.1.1 Zusammenfassung der Hauptergebnisse	68
7.1.2 Bewertung der Ergebnisse	69
7.2 Ausblick	70
Abbildungsverzeichnis	72
Listingverzeichnis	73
Literaturverzeichnis	78

Einleitung

Der folgende Abschnitt beleuchtet das Forschungsthema und erläutert dessen Relevanz. Darüber hinaus werden die zugehörigen Forschungsfragen und -ziele präsentiert und erörtert. Ebenso wird die persönliche Motivation für dieses Forschungsthema dargelegt. Abschließend wird der Aufbau der vorliegenden Arbeit verdeutlicht.

1.1 Relevanz

Die Aufgabe besteht darin, die von mir bestehende Progressive Web App aus dem Bereich der Dating-Apps weiterzuentwickeln und um neue Features zu ergänzen. Die fortwährende Evolution von Dating-Apps verlangt aufgrund der Konkurrenzsituation nach innovativen Features, die das Nutzererlebnis verbessern und der Anwendung ein Alleinstellungsmerkmal verschaffen. Ein Feature, das die App von der Masse anderer Apps abheben könnte, ist „Augmented Reality“ (AR). Sie hat in letzter Zeit immer mehr an Bedeutung gewonnen. AR erweitert die physische Welt mit digitalen Elementen, die es ermöglichen, eine interaktive und immersive Erfahrung zu schaffen (Microsoft, 2023). Deshalb soll die Dating-App um das Feature „Virtuelle Geschenke“ erweitert werden. Dadurch können Benutzer ihren Matches virtuelle Geschenke senden und diese werden mithilfe von Augmented Reality im Umfeld der Benutzer dargestellt.

Die Dating-App wurde mit dem Cross Plattform Framework Flutter entwickelt, das in den letzten Jahren an Popularität gewonnen hat (JetBrains, 2021). Das Framework ermöglicht nativ kompilierte Anwendungen für Mobile, Web und Desktop. Flutter zeichnet sich durch seine Effizienz und plattformübergreifende Möglichkeiten aus. Seit Anfang März 2021 garantiert das Framework stabilen Web-Support, sodass die Entwicklung von Progressive Web Apps und Single Page Apps möglich ist und das Web als neue Plattform zugänglich wurde (Hasnany, 2021). Was allerdings im Flutter-Ökosystem fehlte, ist ein Plugin, das die Integration von AR für Web-Apps ermöglicht. Das Ziel dieser Arbeit besteht daher darin, ein solches AR-Plugin für die Web-Plattform zu entwickeln und damit die gewünschte Funktionalität in der App zu ermöglichen.

Das neu eingeführte Feature zielt darauf ab, durch den Einsatz von AR eine innovative Lösung im Bereich der Dating-Apps zu bieten. Ein weiterer Grund für ein Einsatz von Augmented Reality ist dessen vielversprechende Zukunft. Abbildung 1.1 illustriert das Wachstumspotenzial des mobilen AR-Marktes. Im Jahr 2021 wurde der Wert des mobilen Augmented-Reality-Marktes auf 12,45 Milliarden US-Dollar geschätzt. Es wird prognostiziert, dass der Wert bis 2026

auf über 36 Milliarden US-Dollar ansteigt. Vor diesem Hintergrund ist es meine Motivation, diese Lücke zu schließen und AR-Funktionalitäten für Flutter Web bereitzustellen. Die Entwicklung eines solchen Plugins würde die Grenzen des Möglichen erweitern und das Potenzial von Flutter als Werkzeug für die Erstellung interaktiver, immersiver Web-Erlebnisse verbessern.

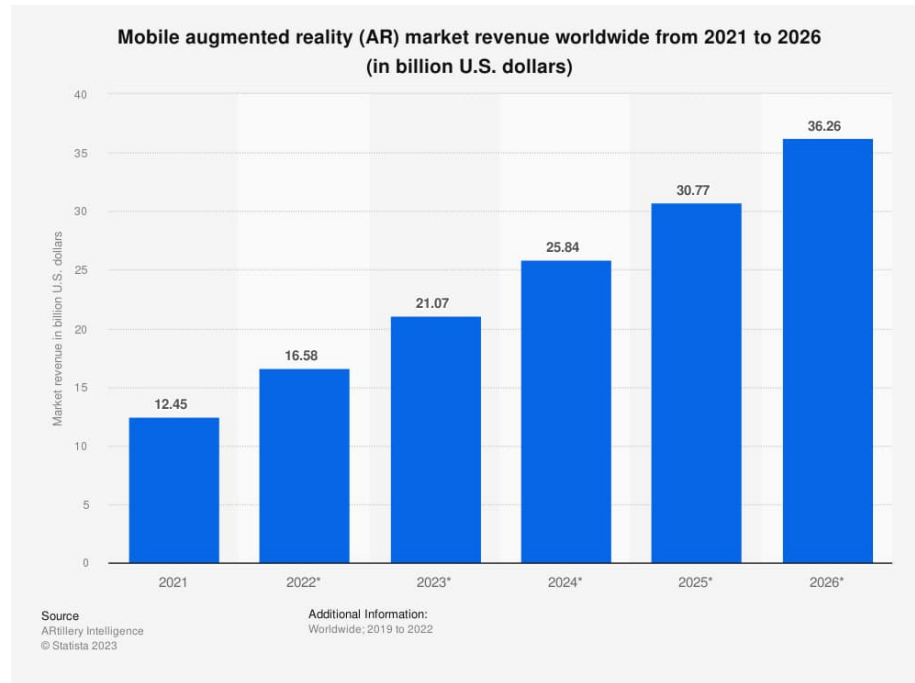


Abbildung 1.1: Marktumsatz der Mobile-AR weltweit von 2021 bis 2026. (ARtillery Intelligence, 2023)

Für iOS und Android sind kompatible Software-Entwicklungskits für AR verfügbar (Ochuko, 2019). Für die Web-Plattform gibt es die WebXR-API, die AR-Erfahrungen im Web bereitstellen soll. Diese befindet sich jedoch noch in der Entwicklung, verändert sich stetig und hat einige Einschränkungen in Bezug auf Geräte und Browser-Kompatibilität (MDN, 2023e). Beispielsweise gibt es keine Unterstützung für iOS-Geräte, insbesondere für den Browser Safari und Firefox (Can I Use, 2023). Das kann in der folgenden Abbildung 1.2 erkannt werden.

Chrome	Edge	Safari	Firefox	Opera	Chrome for Android	Safari on iOS	Samsung Internet	Opera Mini	UC Browser for Android
						15.6			
						16.0			
						16.1			
						16.2			
109						16.3			
112	112					16.4			
113	113	15.6	113	98		16.5			
114	114	16.3	114	99		16.6			
115	115	16.5	116	101	115	16.5	21	all	15.5
116		16.6	117			16.6			
117		17	118			17			
118		TP	119						

Abbildung 1.2: Can I use WebXR Device API (Can I Use, 2023)

Trotz seiner Stabilität weist Flutter Web noch einige Probleme auf, darunter fallen Leistungsschwankungen abhängig vom verwendeten Renderer, fehlende Web-APIs und Herausforderungen bei der Plugin-Entwicklung (Günthner, 2023). Vor diesem Hintergrund stellt sich nun die Frage, ob ein funktionales AR-Plugin für Flutter Web entwickelt und veröffentlicht werden kann, das gleichzeitig einen spezifischen Anwendungsfall unterstützt. Da das Risiko eines Misserfolgs besteht, wird in der Forschungsarbeit mit einem Fallback gearbeitet, der im Falle eines Scheiterns der Entwicklung zum Einsatz kommt.

1.2 Forschungsfragen und Ziele

Das Ziel dieser Forschungsarbeit besteht darin, die folgende Frage zu beantworten: Kann ein funktionales AR-Plugin für Flutter Web entwickelt und veröffentlicht werden, das gleichzeitig einen spezifischen Anwendungsfall unterstützt? Aus dieser Forschungsfrage ergeben sich zwei weitere Fragen:

1. Was sind die Herausforderungen und Hürden bei der Entwicklung eines Augmented Reality Plugins für Flutter Web und wie können diese gelöst werden?
2. Was sind die spezifischen Anforderungen des identifizierten Anwendungsfalls an ein Augmented Reality-Plugin und welche Lösungsansätze eignen sich zur Erfüllung dieser Anforderungen?

1.3 Motivation

Ich habe bereits umfangreiche Erfahrung mit dem Cross-Plattform Framework Flutter und möchte durch die Plugin-Entwicklung noch tiefer in das Flutter-Ökosystem eintauchen. Zudem kann ich diese Forschungsarbeit nutzen, um das von mir entwickelte Praxisprojekt mit neuen Funktionalitäten zu erweitern. Das Zukunftspotential von AR hat mich zusätzlich motiviert. Es reizt mich, experimentelle Technologien zu nutzen, gleichzeitig bin ich mir den Risiken bewusst, die während der Entwicklung auftreten können. Insgesamt möchte ich mit dieser Forschungsarbeit mein Wissen und meine Fähigkeiten in diese Richtung erweitern.

1.4 Aufbau der Arbeit

Die Arbeit ist in mehrere Kapitel unterteilt. Die Einleitung stellt das Thema vor, erläutert die Forschungsfrage und präsentiert die Motivation. Das Kapitel Hintergrund bietet einen Überblick über Augmented Reality und über Flutter Web. Im Kapitel Methodik werden die verwendeten Methoden, Techniken, Technologien und Tools beschrieben. Darüber hinaus beleuchtet es das Design und die Architektur des Plugins und erörtert die geplanten Implementierungsschritte. Das nächste Kapitel, Anforderungserhebung, beschreibt die Anforderungen, die für das Plugin entwickelt wurden, um den spezifischen Anwendungsfall umzusetzen. Das Kapitel Implementierung konzentriert sich auf die detaillierte Ausarbeitung des Entwicklungsprozesses und präsentiert Codebeispiele des Plugins. Im Kapitel Herausforderungen werden die Hürden diskutiert, die bei der Entwicklung eines Augmented Reality Plugins für Flutter Web aufgetreten sind. Abschließend fasst das Kapitel Fazit und Ausblick die erreichten Forschungsergebnisse zusammen und bewertet sie.

Hintergrund

Im folgenden Abschnitt wird die Technologie Augmented Reality näher erläutert, es wird auf Flutter Web eingegangen und der Paketmanager Pub analysiert.

2.1 Augmented Reality

Der folgende Abschnitt gibt einen Einblick in die dynamische Welt der Augmented Reality und ihre Wachstum- und Einsatzgebiete. Anschließend werden die verschiedenen Variante von AR vorgestellt. Abschließend wird ein Überblick über AR und ihre Verwandten - Virtual Reality (VR) und Mixed Reality (MR) - gegeben, die gemeinsam das Spektrum der erweiterten Realitäten bilden.

2.1.1 Die dynamische Welt der Augmented Reality

Augmented Reality (AR) bezieht sich auf eine erweiterte Form der Realität, bei der digitale Informationen in Echtzeit in die physische Umgebung integriert werden. Durch AR wird die digitale Welt mit der physischen Welt verbunden. Dabei erleben Nutzer ihre reale Umgebung, die durch Grafiken, Videos, Geräusche, haptisches Feedback oder 3D-Animationen ergänzt wird, als erweiterte Realität. Dieses Erlebnis wird mithilfe der Kamera eines mobilen Geräts oder über AR-Headsets und Brillen umgesetzt (Alsop, 2023).

Es wird erwartet, dass der globale AR-Markt in den kommenden Jahren erheblich wachsen wird. Ein wichtiger Teil des AR-Marktes ist der mobile Einsatz der Technologie. Mobiles AR nutzt die große Anzahl von Smartphones, Tablets und anderen mobilen Geräten. Für das Jahr 2023 wird prognostiziert, dass die Anzahl potentieller AR-Gerätebenutzer 1,4 Milliarden erreicht (Alsop, 2023). Im Jahr 2021 wurde der Wert des mobilen Augmented-Reality-Marktes auf 12,45 Milliarden US-Dollar geschätzt. Es wird geschätzt, dass der Wert bis 2026 auf über 36 Milliarden US-Dollar steigt (ARtillery Intelligence, 2023). Das Wachstum wird sowohl im Unternehmens- als auch im Verbrauchersegment erwartet. Insbesondere soziale Medien-Apps wie Snapchat, Instagram, Facebook und Tiktok haben die Anwendungsbereiche der AR-Technologie im Verbrauchersegment vorangetrieben. Augmented Reality (AR) wurde hauptsächlich von den genannten Unternehmen für Foto-Filter verwendet, was ihnen Aufmerksamkeit verschaffte und inzwischen unverzichtbar geworden ist. Die App Pokémon Go konnte im Bereich Gaming im Verbrauchersektor durch den Einsatz von AR Popularität gewinnen. Mit insgesamt mehr als 600 Millionen Downloads erreichte die Anwendung ihren Höhepunkt im Bereich AR-Gaming. Auch andere Gaming-Apps beeinflussen das rasante Wachstum des Markts durch ähnliche Konzepte.

Im E-Commerce-Sektor erzielen Möbel-Apps wie Ikea Place oder Make-up-Apps wie Sephora Erfolge (Statista Market Insights, 2023).

Im Gegensatz zur AR-Software weist die Hardware noch Schwächen auf. Die bis zum Jahr 2023 erhältlichen AR-Brillen können technisch noch nicht vollständig überzeugen. Neben Herstellern wie Microsoft und Meta stieß im Juni 2023 ein weiterer Konkurrent im AR-Hardware-Sektor hinzu. Im Rahmen seiner jährlichen Entwicklerkonferenz präsentierte Apple im Juni 2023 sein erstes immersives Headset. Dieses kann beispielsweise mittels eines Drehschalters von AR zu weiteren Virtual-Reality-Erlebnissen wechseln. Die Einführung der Apple-AR-Brille und weiterer Hersteller, unterstützt und fördert das Wachstum des Markts (Alsop, 2023).

2.1.2 Varianten von AR

Es gibt verschiedene technologische Varianten von Augmented Reality, die verwendet werden können, um die Umgebung zu erkennen: markerbasiert und markerunabhängig. Jede Variante zeigt Bilder und Informationen auf unterschiedliche Weise an. Bei der markerbasierten AR-Variante werden mithilfe von Bilderkennung Objekte erzeugt und als Marker verwendet. Diese Marker dienen als Referenzpunkte, um die Position und Ausrichtung der Kamera zu bestimmen. Dies wird erreicht, indem die Kamera auf Graustufen umschaltet und eine Markierung erkennt. Anschließend wird diese Markierung mit anderen Markierungen in der Informationsdatenbank verglichen. Wenn das AR-Gerät eine Übereinstimmung findet, verwendet es diese Daten, um die genaue Position des AR-Bildes zu berechnen und es an der richtigen Stelle anzuzeigen. Marker können Bilder, QR-Codes oder andere 2D-Objekte sein. Die zweite Variante von AR arbeitet mit einem Erkennungsalgorithmus, um ein Objekt in der Umgebung zu platzieren. Diese Variante ist jedoch aufwendiger, da das Gerät keinen festen Bezugspunkt hat, auf den es sich ausrichten kann. Daher muss das Gerät Objekte im Blickfeld erkennen und anhand eines Erkennungsalgorithmus nach Farben, Mustern und ähnlichen Merkmalen suchen, um festzustellen, um welches Objekt es sich handelt. Dabei werden Sensoren, GPS und eine Kamera verwendet, um das virtuelle Element in der Umgebung anzuzeigen (Microsoft, 2023).

2.1.3 AR, VR und Mixed Reality: Ein Überblick

Augmented Reality gehört zur Gruppe der Extended Reality-Technologien, die unsere Sinne erweitern. Zu dieser Gruppe gehört auch noch Virtual Reality und Mixed Reality. Alle diese Technologien verwischen die Grenzen zur Realität (Microsoft, 2023). Im Gegensatz zu AR schafft Virtual Reality (VR) ein immersives Erlebnis, bei dem der Benutzer von der realen Welt isoliert wird. Das wird mithilfe eines Headsets und Kopfhörer umgesetzt, die speziell dafür entwickelt wurden. Während AR die reale Umgebung erweitert und mit virtuellen Elementen ergänzt, ist VR in der Regel komplett von der realen Welt losgelöst.

Die Mixed Reality kombiniert AR- und VR-Elemente, sodass digitale Objekte mit der realen Welt interagieren können. Dadurch können Elemente entworfen werden, die in einer realen Umgebung verankert sind (Microsoft, [2023](#)).

2.2 Flutter Web

Das folgende Kapitel konzentriert sich auf das Framework Flutter, insbesondere auf die Web-Plattform. Hier werden kurz die Möglichkeiten und Herausforderungen des Frameworks erläutert. Anschließend wird auf den Paketmanager Pub eingegangen.

2.2.1 Möglichkeiten und Herausforderungen

Flutter ist ein von Google entwickeltes Open-Source-Framework zur Entwicklung von plattformübergreifenden mobilen Anwendungen für iOS, Android, Web und Desktop. Es ermöglicht Entwicklern, Anwendungen mit nur einer Codebasis zu erstellen, die auf verschiedenen Plattformen ausgeführt werden können, ohne dabei Kompromisse bei der Benutzererfahrung eingehen zu müssen (Google, [2023](#)). Seit Anfang März 2021 garantiert das Framework stabilen Web-Support und macht das Web als Plattform für die Entwicklung von Flutter-Anwendungen zugänglich (Hasnany, [2021](#)). Der Engine-Baustein der Flutter Architektur für Web-Apps unterscheidet sich von mobilen Apps wie iOS und Android. Während er für die Entwicklung von mobilen Apps ausschließlich aus C++ besteht, teilt sich die Engine bei Webanwendungen in die Komponenten HTML/CSS, Canvas, WebGL und WebAssembly auf. Darüber hinaus stehen zwei verschiedene Renderer für Flutter Web zur Verfügung, die für unterschiedliche Anwendungsszenarien genutzt werden können (Hasnany, [2021](#)).

Trotz der Stabilität von Flutter Web können einige Probleme auftreten. Ein Problem kann der Zugriff auf Web-APIs sein, da das `Navigator`-Objekt der `html`-Bibliothek von Flutter nicht alle aktuellen Web-APIs unterstützt. Wenn keine passende Bibliothek für die nicht verfügbare Web-API im Paketmanager angeboten wird, muss die Web-API mittels JavaScript-Code und der JS-Bibliothek in Dart integriert werden, um die gewünschte Funktionalität nutzen zu können. Weitere Einschränkungen können Performance-, Browser-Kompatibilitätsprobleme, Suchmaschinenoptimierung und möglicherweise längere Ladezeiten sein.

Abgesehen von den eventuell auftretenden Problemen bietet Flutter Web folgende Vorteile. Es unterstützt Progressive Web Apps und Single Page Apps, was den modernen Web-Entwicklungsstandards entspricht. Durch die verschiedenen verfügbaren Renderer können bestimmte Anwendungsfälle umgesetzt werden. Außerdem gibt es einen einheitlichen Entwicklungs-Workflow für Web- und mobile Anwendungen. Wenn bereits eine native mobile Anwendung mit dem Framework Flutter entwickelt wurde, ist eine einfache Konvertierung zu einer Webanwendung möglich, was Zeit und Kosten spart. Zusätzlich eignet es sich

für die schnelle Umsetzung und Validierung von Proof-of-Concepts (Günthner, 2023). Darüber hinaus wird kontinuierlich am Dart2Wasm Compiler gearbeitet, der die Umwandlung von Dart-Code in WebAssembly ermöglicht. Dadurch kann WebAssembly experimentell als Kompilierungsziel für Flutter festgelegt werden. Dies ermöglicht eine nahezu native Leistung für Flutter im Web (Flutter Dokumentation, 2023c). Ein weiterer Vorteil ist der Zugriff auf den Paketmanager Pub für die Implementierung verschiedener Webplattform-Funktionalitäten.

2.2.2 Paketmanager Pub

In der Softwareentwicklung sind Paketmanager ein unverzichtbares Werkzeug. Ein Paketmanager verwaltet Softwarebibliotheken und -pakete, die ein Projekt benötigt, und automatisiert den Prozess der Installation, Aktualisierung, Konfiguration und Entfernung von Softwarepaketen in einem Betriebssystem auf einer konsistenten Art und Weise (MDN, 2023b). Diese Tools ermöglichen es Entwicklern, Code effizient zu teilen und zu nutzen. Im Zusammenhang mit Flutter und Dart spielt der Paketmanager Pub eine zentrale Rolle. In diesem Abschnitt wird auf die Funktionen und Vorteile von Pub und dessen Plattform eingegangen. Der Paketmanager von Flutter bietet eine breite Palette an Paketen an, die unterschiedliche Funktionen bereitstellen - von Netzwerkanfragen bis hin zur Zustandsverwaltung und UI-Elementen. Diese Pakete werden von der Flutter- und Dart-Community entwickelt und geteilt. Viele der auf Pub.dev verfügbaren Pakete sind für mehrere Plattformen ausgelegt, einschließlich iOS, Android, Web und Desktop. Derzeit sind 37103 Pakete verfügbar, davon 15357 für die Web-Plattform (Pub, 2023c). Pub.dev ist nicht nur ein Ort zum Herunterladen von Paketen, sondern auch eine Community, um zusammenzuarbeiten und das Ökosystem von Flutter und Dart zu verbessern.

Die Pub.dev-Startseite bietet einen Überblick über beliebte und qualitativ hochwertige Flutter-Pakete. Es werden verschiedene Kategorien von Paketen präsentiert, einschließlich der "Flutter Favorite"-Kategorie, die vom Flutter Ecosystem Committee ausgewählt wurde (Pub, 2023c). Flutter-Favoriten-Pakete müssen bestimmte Qualitätsstandards erfüllen, wie Aktualität, Wartbarkeit, Dokumentation und GitHub-Sterne (Flutter Dokumentation, 2023b). Auf der Startseite sind auch die am häufigsten heruntergeladenen Pakete der letzten 60 Tage sowie Top Flutter-Pakete und Dart-Pakete zu finden (Pub, 2023c).

Um das richtige Paket für einen bestimmten Anwendungsfall zu finden, ist eine effektive Suche auf Pub.dev unerlässlich. Die Plattform bietet verschiedene Suchwerkzeuge und -funktionen, um die passenden Pakete für ein Projekt zu finden. Eine spezifische Suche kann mit Filtern verfeinert werden, z.B. nach Software Development Kit (Flutter oder Dart) oder Plattform. Das Ranking der Suchergebnisse basiert auf Relevanz, Popularität und Pub-Punkten. Diese Parameter werden gewichtet und kombiniert, um eine Gesamtpunktzahl für jedes Ergebnis zu ermitteln (Pub, 2023g).

Die Plattform Pub.dev bietet viele Pakete an, die ähnliche Funktionen erfüllen. Es gibt ein Bewertungssystem, um die Qualität und Zuverlässigkeit der Pakete zu bewerten. Es gibt drei Bewertungsdimensionen für jedes Paket auf Pub.dev: Likes, Pub Points und Popularity. Likes zeigen, wie beliebt ein Paket bei Entwicklern ist und wie gut es in der Community ankommt. Pub Points sind ein Qualitätsmaß, das Code-Stil, Plattformunterstützung und Wartbarkeit umfasst. Die Beliebtheit eines Pakets zeigt, wie viele Entwickler es verwenden. Eine höhere Punktzahl bedeutet in der Regel höhere Qualität und Zuverlässigkeit, was das Paket attraktiver für andere Entwickler macht (Pub, 2023d). Das Suchergebnis enthält die Paketbewertung und eine allgemeine Vorschau des Pakets. In der Abbildung 2.1 ist das Suchergebnis mit Paketbewertung und genereller Vorschau des Pakets aufgeführt.

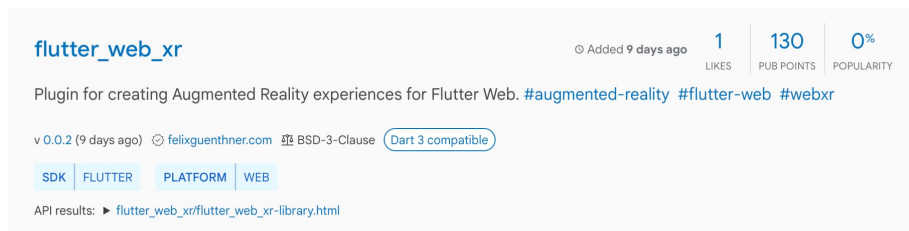


Abbildung 2.1: Suchfunktion der Plattform pub.dev (Pub, 2023f)

Methodik

Im nächsten Abschnitt wird die angewandte Forschungsmethodik erläutert, um das angestrebte Forschungsergebnis zu erreichen.

3.1 Methoden und Techniken

In diesem Abschnitt liegt der Fokus auf den Methoden und Techniken, die zur Erreichung des Forschungsziels verwendet wurden. Zunächst wird eine Literaturrecherche durchgeführt, um bestehende AR-Plugins für Flutter und andere Plattformen zu untersuchen. Dabei werden die Grenzen und Möglichkeiten dieser Plugins analysiert. Hierfür wird hauptsächlich eine systematische Suche in wissenschaftlichen Datenbanken und technischen Dokumentationen genutzt. Außerdem wird eine gründliche Suche in verschiedenen Paket-Managern durchgeführt, wie beispielsweise pub.dev und dem npm-Register. Des weiteren müssen die Anforderungen für die Entwicklung des AR-Plugins basierend auf dem spezifischen Anwendungsfall der in der Praxis entwickelten Dating-App festgelegt werden. Als nächster Schritt steht die prototypische Entwicklung an, um das Hauptziel der Forschungsarbeit zu erreichen. Dabei wird das Teilziel verfolgt, ein erstes funktionierendes AR-Plugin für Flutter Web zu erstellen. Die Entwicklung soll in Sprints erfolgen, um Teilerfolge festzuhalten und Probleme bewerten zu können. Nachdem ein Prototyp entwickelt wurde, ist es wichtig, sicherzustellen, dass das Plugin stabil läuft, mit dem Flutter-Ökosystem kompatibel ist und die Anforderungen des spezifischen Anwendungsfalls erfüllt. Hierfür können verschiedene Methoden wie kontinuierliche Integration, Unit-Tests, Funktionstests oder Performance-Tests eingesetzt werden, um das Plugin zu bewerten und zu optimieren. Sobald das Testen erfolgreich abgeschlossen ist, sollte eine klare und umfassende Dokumentation erstellt werden, um anderen Entwicklern zu helfen, das AR-Plugin effektiv zu nutzen. Diese Dokumentation sollte technische Spezifikationen, Anwendungsfälle, Tutorials und Beispielcode umfassen.

3.2 Technologien und Tools

Im folgenden Abschnitt wird erklärt, welche Technologien und Tools für die Forschungsarbeit und die Entwicklung des AR-Plugins für Flutter Web ausgewählt wurden. Während des Schreibprozesses dieser Arbeit wurde das Tool ChatGPT von OpenAI als Unterstützung verwendet. Es wurde eingesetzt, um bestimmte Formulierungen zu verbessern, Ideen klarer auszudrücken und auf Rechtschreibung und Grammatik zu achten. Da das Plugin für das Flutter-Ökosystem entwickelt wird, wurde Dart als Hauptprogrammiersprache gewählt. Dart bietet eine umfangreiche Sammlung von Bibliotheken und ist die primäre Sprache für

Flutter-Anwendungen. Für die Entwicklung dieses Plugins ist es notwendig, mit nativen JavaScript-APIs zu interagieren. Die JS-Bibliothek ermöglicht es, JavaScript-APIs per Dart-Code aufzurufen und bietet Annotationen und Funktionen, um festzulegen, wie der Dart-Code mit dem JavaScript-Code interopereiert (Pub, [2023b](#)). Daher ist diese Bibliothek ebenfalls Teil der ausgewählten Tools. Für die AR-Funktionalität wurden mehrere Technologien in Betracht gezogen. Dazu gehören etablierte AR-Lösungen wie ARCore (für Android) und ARKit (für iOS), die in der mobilen AR-Entwicklung führend sind. Für die Web-Plattform gibt es die WebXR Device API, welche den Kern des WebXR-Featuresatzes implementiert (MDN, [2023c](#)). Da diese API AR-Erlebnisse im Webbrowser unterstützt, wurde sie als geeignet erachtet. Für die Darstellung von dreidimensionalen Objekten im Web wird die Technologie WebGL (Web Graphics Library) genutzt. Die MDN Web Docs (MDN, [2023d](#)) beschreiben WebGL als "eine JavaScript-API zur Darstellung von interaktiven 3D- und 2D-Grafiken mit hoher Leistung in jedem kompatiblen Webbrowser ohne die Verwendung von Plug-Ins." Um mit einer vereinfachten Abstraktion von WebGL zu arbeiten, wird die 3D Bibliothek Three.js verwendet. Nach der Dokumentation von Three.js (Three.js Dokumentation, [2023a](#)) ist die Bibliothek als "eine 3D-Bibliothek, die versucht, es so einfach wie möglich zu machen, 3D-Inhalte auf einer Website zu integrieren" beschrieben.

Als Entwicklungsumgebung wurde Visual Studio Code (VS Code) verwendet, da dieser Editor mit allen verfügbaren Flutter- und Dart-Plugins kompatibel ist. VS Code bietet somit eine integrierte Entwicklungs-Umgebung, die den Entwicklungsprozess erleichtert, insbesondere durch das Flutter Plugin. Dadurch wird die Sprachunterstützung für die Programmiersprache Dart gewährleistet. Diese Plugins können über den Marketplace von Visual Studio Code installiert werden, um die Entwicklung und das Debugging effizienter zu gestalten. Das Debugging von Webanwendungen, die mit Flutter erstellt wurden, ist nur mit den Browsern Google Chrome und Microsoft Edge kompatibel (Flutter Dokumentation, [2023d](#)). Aus diesem Grund wurde Google Chrome als Browser für das Debugging während der Entwicklung ausgewählt. Für den Aspekt des Testings und der Qualitätssicherung wurden AR-Simulatoren verwendet, bevor mit realen Geräten getestet wurde. Hierfür wurde eine Chrome-Erweiterung namens "WebXR API Emulator" genutzt, die es Benutzern ermöglicht, WebXR-Inhalte in Desktop-Browsern zu starten und zu testen, ohne dass ein reales XR-Gerät benötigt wird. Die Erweiterung emuliert die WebXR API in Browsern, die sie noch nicht unterstützen, und stellt eine Liste von XR-Geräten mit ihren Controllern bereit, die emuliert werden können (Mozilla Mixed Reality, [2023](#)). Nachfolgend ist die Abbildung [3.1](#) dargestellt, die ein Anwendungsszenario der Chrome-Erweiterung zeigt, das unter den Entwickler-Tools im Browser zu finden ist.

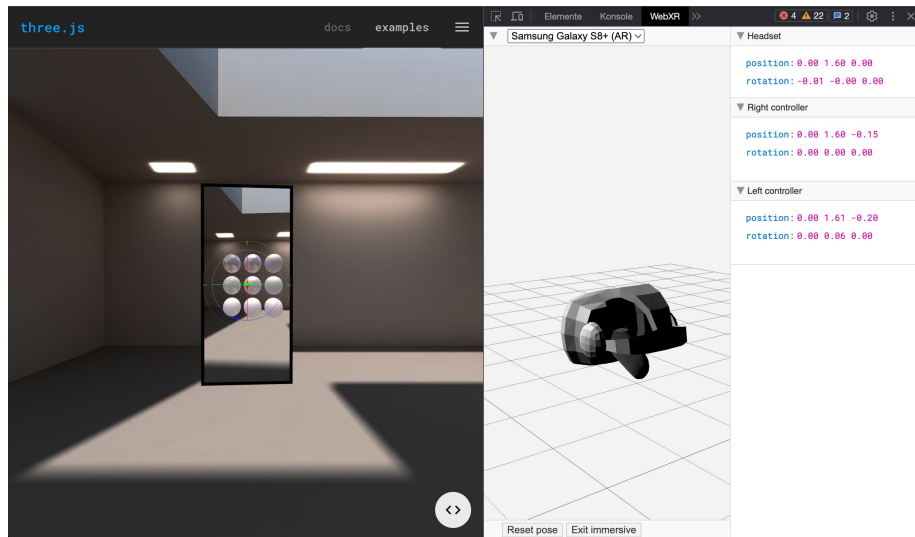


Abbildung 3.1: WebXR API Emulator in Google Chrome (Mozilla Mixed Reality, 2023)

Für die Zusammenarbeit und Versionierung wird Git und Github genutzt. Diese Tools dienen zur Verwaltung des Codes und Nachverfolgung von Änderungen. Für die Dokumentation wird das Tool DartDoc verwendet, welches Teil des Dart Software Development Kit ist (Dart Dokumentation, 2023b).

3.3 Design und Architektur

Da die Beschreibung des Designs und der Architektur ein zentraler Bestandteil ist, wenn es um die Entwicklung von Software oder Plugins geht, wird im folgenden Abschnitt ein kurzer Überblick über diese Themen gegeben.

3.3.1 Grundlegende Designprinzipien

Laut Singh und Hassan (Singh & Hassan, 2015) ist die Design-Phase eine der wichtigsten Etappen im Softwareentwicklungszyklus. Sie hat einen großen Einfluss auf den gesamten Lebenszyklus des Projekts. Basierend auf ihren Aussagen sollten die Programmier-, Wartungs- und Supportphasen des Softwareentwicklungszyklus ohne Stress oder Probleme verlaufen, wenn das Design die entsprechenden Regeln einhält. Singh und Hassan betonen, dass das Design eine bedeutende Rolle spielt und direkte Auswirkungen auf die Anforderungen an Qualität und Leistung hat. Die SOLID-Designprinzipien dienen als Standard in der Softwareentwicklung und sind eine gute Richtlinie für Design-Entscheidungen, um die Wiederverwendbarkeit, Wartbarkeit und Skalierbarkeit der Software zu

erhöhen. SOLID ist ein Akronym, das für fünf grundlegende Designprinzipien in der objektorientierten Programmierung und Softwareentwicklung steht. Diese Prinzipien helfen dabei, Software-Designs zu erstellen, die verständlich, flexibel und wartbar sind (Singh & Hassan, 2015). Durch das SOLID-Prinzip lassen sich einige Grundprinzipien ableiten, wie Modularität, Erweiterbarkeit und Performance. Das Single Responsibility Prinzip (SRP) fördert durch die Trennung von Verantwortlichkeiten in einzelne Module oder Klassen ein modulares Design. Dabei hat jede Einheit nur eine klar definierte Aufgabe, was die Modularität gewährleistet. Das Thema Erweiterbarkeit kann durch zwei SOLID-Prinzipien abgeleitet werden. Zum einen durch das Open/Closed Prinzip (OCP), das besagt, dass Software-Einheiten erweiterbar sein sollen, ohne bestehenden Code zu ändern. Das bedeutet, dass ein System mit Hilfe des OPC-Prinzips neue Funktionalitäten hinzufügen kann, ohne bestehenden Code zu modifizieren. Ein weiteres SOLID-Prinzip, das Liskov Substitution Prinzip (LSP), ist dafür verantwortlich, sicherzustellen, dass Subtypen die Eigenschaften ihrer Basistypen nicht verändern. Dadurch kann der Entwickler darauf vertrauen, dass abgeleitete Typen den vorhandenen Code nicht brechen und somit die Erweiterbarkeit erleichtert wird. Der dritte Aspekt, die Performance, kann zwar nicht direkt aus dem SOLID-Prinzip abgeleitet werden, spielt jedoch eine entscheidende Rolle in der Softwareentwicklung und kann von der Einhaltung dieser Prinzipien profitieren. Durch die Einhaltung der SOLID-Prinzipien kann ein System effizienter und einfacher optimiert werden, was sich positiv auf die Performance auswirkt (Singh & Hassan, 2015). Das Design des Plugins sollte modular gestaltet werden, um Flexibilität und Skalierbarkeit zu gewährleisten. Entwicklern soll es durch die Erweiterbarkeit ermöglicht werden, zusätzliche AR-Funktionen oder Anwendungsfälle hinzuzufügen. Performance spielt ebenfalls eine wichtige Rolle, weshalb das Design für schnelle Ladezeiten und flüssige AR-Erlebnisse optimiert werden sollte.

3.3.2 Architekturüberblick

Die Architektur für ein Plugin im Flutter Ökosystem verfolgt bestimmte Architektur-Entscheidungen, die bei Erstellung des Plugin-Projekts bereits vorhanden sind. Dazu gehört zum einen die Main Plugin Klasse `FlutterWebXr`, die als primären Einstiegspunkt des Plugins fungiert. Entwickler können über diese Klasse auf die Funktionen des Plugins zugreifen, indem sie eine Instanz des Plugins erstellen. Der folgende Code zeigt die `FlutterWebXr` Klasse, die zwei Beispielfunktionen enthält, um Informationen über die aktuell verwendete Plattform zu erhalten und eine AR-Session zu starten.

Listing 3.1: FlutterWebXr Klasse.

```
class FlutterWebXr {  
  /// Retrieves the platform version.  
  /// @return A [String] containing the version of the  
  /// platform.  
  String? getPlatformVersion() {  
    return FlutterWebXrPlatform.instance.getPlatformVersion  
      ();  
  }  
  
  /// Initiates a WebXR session.  
  /// @return A [Future] indicating the session start  
  /// process completion.  
  Future<void> startSession() {  
    return FlutterWebXrPlatform.instance.startSession();  
  }  
}
```

Diese Klasse ruft die Funktionen auf, die im Platform Interface definiert sind. Das Interface ermöglicht in Flutter eine eindeutige und konsistente Kommunikation zwischen Dart-Code und nativem Code. Insbesondere für die Entwicklung von Plugins, die plattformspezifische Funktionen bereitstellen, die in reinem Dart nicht verfügbar sind, ist das Interface von Bedeutung. Ein Vorteil durch den Einsatz des Platform Interface ist die Abstraktionsebene zwischen dem Dart-Code eines Plugins und dem nativen Code, der für verschiedene Plattformen wie Android, iOS oder Web geschrieben ist. Diese Abstraktion vereinfacht die Implementierung plattformspezifischer Logik, während der Dart-Code des Plugins konsistent bleibt. Durch die Verwendung des Platform Interface kann eine einheitliche Dart-API für ein Flutter-Plugin sichergestellt werden, ohne dass Abhängigkeiten von den nativen Implementierungen bestehen. Darüber hinaus kann ein Plugin durch das Platform Interface einfach um weitere Plattformen erweitert werden, beispielsweise für Desktop. Ein weiterer Aspekt ist die Sicherheit und Robustheit, die durch die Verwendung dieser Komponente gewährleistet wird (Pub, 2023e). Es gibt Mechanismen, die sicherstellen, dass Plugins korrekt implementiert werden. Zum Beispiel wird der Token geprüft, der für das Setzen einer Instanz der `FlutterWebXrPlatform` erforderlich ist. Hierfür steht eine Methode zur Verfügung, die den Token überprüft und das Setzen der Instanz durch unerwartete Codefragmente verhindert. Zusammenfassend dient das Platform Interface in Flutter als Brücke zwischen Dart und nativem Code. Es gewährleistet die konsistente und sichere Funktion des Plugins und unterstützt Erweiterbarkeit und Wartung. Es ist ein entscheidendes Werkzeug, um Flutter's Versprechen der plattformübergreifenden Entwicklung zu erfüllen.

Der folgende Code definiert eine abstrakte Klasse namens `FlutterWebXrPlatform`, die von der Klasse `PlatformInterface` erbt und als Schnittstelle innerhalb des Plugins fungiert. Die Klasse hat einen Konstruktor, der beim Erstellen eines Objekts aufgerufen wird. Hier wird jedoch lediglich der Konstruktor der übergeordneten Klasse `PlatformInterface` aufgerufen und ein spezieller Token übergeben. Dieser Token ist ein `private`, statisches Token-Objekt, das als eine Art Identifikator dient. Dieses Token stellt sicher, dass nur berechnete Implementierungen die Instanz setzen können. Danach wird eine Instanz vom Typ `MethodChannelFlutterWebXr` namens `_instance` initialisiert, die als Standard-Implementierung für die `FlutterWebXrPlatform` dient. Diese Instanz wird von einem öffentlichen Getter zurückgegeben. Ein öffentlicher Setter ermöglicht das Setzen der `_instance`-Variable. Dieser Setter prüft mithilfe der Funktion `verifyToken()`, die von der Klasse `PlatformInterface` bereitgestellt wird, ob die gegebene Instanz das korrekte Token hat. Wenn der Token korrekt ist, wird eine neue Plattform-Instanz gesetzt. Die Methode `getPlatformVersion()` soll die aktuelle Plattformversion zurückgeben und wird aufgrund der abstrakten Klasse `FlutterWebXrPlatform` nicht direkt implementiert. Hier wird standardmäßig ein Fehler ausgegeben, um Entwickler darauf hinzuweisen, dass diese Methode in einer Unterklasse bereitgestellt werden muss.

Listing 3.2: FlutterWebXrPlatform Klasse.

```
abstract class FlutterWebXrPlatform extends
    PlatformInterface {
    FlutterWebXrPlatform() : super(token: _token);

    static final Object _token = Object();

    static FlutterWebXrPlatform _instance =
        MethodChannelFlutterWebXr();

    /// The default instance of [FlutterWebXrPlatform] to use.
    /// Defaults to [MethodChannelFlutterWebXr].
    static FlutterWebXrPlatform get instance => _instance;

    /// Platform-specific implementations should set this with
    /// their own
    /// platform-specific class that extends [
    /// FlutterWebXrPlatform] when
    /// they register themselves.
    static set instance(FlutterWebXrPlatform instance) {
        PlatformInterface.verifyToken(instance, _token);
        _instance = instance;
    }

    /// Returns the platform version.
    Future<String?> getPlatformVersion() {
        throw UnimplementedError('platformVersion() has not been
            implemented.');
```

Um auf plattformspezifische Dienste wie die Geolocation oder den Batteriestatus zuzugreifen, nutzt Flutter ein flexibles System. Dabei greift es auf plattformspezifische APIs in einer Sprache zu, die direkt mit diesen APIs kommuniziert. Beispielsweise Kotlin oder Java auf Android und Swift oder Objective-C auf iOS. Die Kommunikation zwischen Dart und plattformspezifischen Code wird durch die Platform Channels ermöglicht. Ein Platform Channel ist ein benannter Kanal zum Kommunizieren mit der Flutter-Anwendung unter Verwendung asynchroner Methodenaufrufe (Flutter API, 2023a). Dadurch können

in Flutter Funktionen bereitgestellt werden, die das Framework selbst nicht nativ bietet. Ein Beispiel hierfür ist der Zugriff auf plattformspezifische Dienste wie die Geolocation. Laut der Flutter-Dokumentation (Flutter Dokumentation, 2023e) sendet der Flutter-Teil der App über einen Platform Channel Nachrichten an den Host, der den plattformspezifischen Teil der App ausmacht. Dieser Host hört auf den Platform Channel, empfängt die Nachrichten, ruft plattformspezifische APIs auf und sendet eine Antwort zurück an den Flutter-Teil, der als Client fungiert. Flutter sendet Nachrichten zwischen Dart und der Plattform asynchron. Beim Aufrufen einer Channel-Methode ist jedoch zu beachten, dass die ausgewählte Methode auf dem Hauptthread der Plattform aufgerufen werden muss (Flutter Dokumentation, 2023e). Auf der Client-Seite ermöglicht der `MethodChannel` das Senden von Nachrichten, die Methodenaufrufen entsprechen. Auf der Plattformseite kann der `MethodChannel` für Android (`MethodChannelAndroid`) und der `FlutterMethodChannel` für iOS (`MethodChannelIOS`) Funktionsaufrufe empfangen und Ergebnisse zurücksenden. Diese Klassen erleichtern die Entwicklung eines Plattform-Plugins durch den geringen Bedarf an Boilerplate-Code. Die folgenden Abbildung 3.2 veranschaulicht das System, das für die Kommunikation zwischen Dart und plattformspezifischen Code zuständig ist.

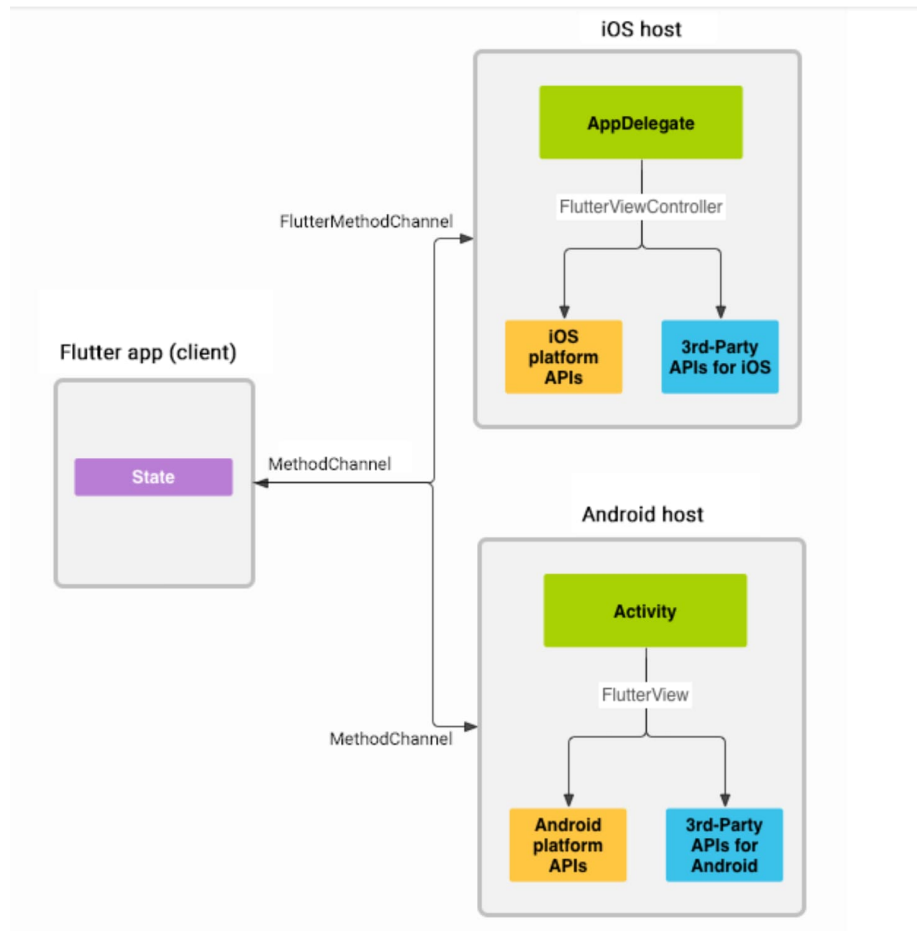


Abbildung 3.2: Architectural overview: platform channels. (Flutter Dokumentation, 2023e)

Im folgenden Code wird die Funktionsweise eines `MethodChannels` anhand eines praktischen Beispiels gezeigt. Das Ziel ist es, den Batteriestatus der Android-Plattform zurückzugeben. Zunächst wird eine Instanz des `MethodChannel` erstellt. Sobald die benötigte Funktion vom Plattform Interface eingebunden wurde, kann mithilfe der Instanz des `MethodChannel` eine Methode aufgerufen werden, die plattformspezifischen Code anspricht. Für diese Methode wird in diesem Anwendungsfall der konkrete Methodennamen „getBatteryLevel“ als Bezeichner angegeben, um den aktuellen Batteriestand zurückzugeben. Die Client- und Host-Seite eines Channels werden durch einen eindeutigen Channel Namen verbunden, der beim Erstellen der `MethodChannel`-Instanz als Parameter übergeben wird.

Listing 3.3: MethodChannelFlutterWebXr Klasse.

```
class MethodChannelFlutterWebXr extends FlutterWebXrPlatform
{
    /// The method channel used to interact with the native
    platform.
    @visibleForTesting
    final methodChannel = const MethodChannel('flutter_web_xr'
    );

    @override
    Future<int?> getPlatformVersion() async {
        final int result =
            await methodChannel.invokeMethod<String>('
            getBatteryLevel');
        return result;
    }
}
```

Dieser Code-Abschnitt zeigt die Host-Seite der Android Plattform für diesen Anwendungsfall. Der Code ist in Kotlin geschrieben und stellt eine typische Implementierung für die Kommunikation zwischen Flutter und nativem Android-Code über einen Method Channel dar. Die Klasse `MainActivity` erbt von `FlutterActivity` und initialisiert einen Method Channel, der den gleichen Namen wie der Method Channel der Client-Seite hat. Wenn der Flutter-Teil der App die Methode `getBatteryLevel` über diesen Channel aufruft, sendet der native Android-Code ein Ergebnis zurück.

Listing 3.4: MainActivity Klasse.

```
class MainActivity: FlutterActivity() {
    private val CHANNEL = "flutter_web_xr"

    override fun configureFlutterEngine(@NonNull flutterEngine
        : FlutterEngine) {
        super.configureFlutterEngine(flutterEngine)
        MethodChannel(flutterEngine.dartExecutor.binaryMessenger
            , CHANNEL).setMethodCallHandler {
            call, result ->
                if (call.method == "getBatteryLevel"
                    ) {
                    result.success("Hier sind
                        die Daten aus Android");
                } else {
                    result.notImplemented()
                }
            }
        }
    }
}
```

Wenn ein Plugin für Android oder IOS entwickelt wird, wird empfohlen, Method Channels zu verwenden. Wenn jedoch das Web als Zielplattform verfolgt wird, ist dieser Ansatz nicht geeignet. Für plattformspezifischen Code für das Web wird in der Regel JavaScript Interoperabilität oder die `dart:html` Bibliothek benutzt (Flutter Dokumentation, [2023e](#)). Wenn ein Plugin mit der Web-Plattform als Ziel entwickelt wird, ist eine weitere Klasse notwendig. Der folgende Code definiert eine Web-spezifische Implementierung der Klasse namens `FlutterWebXrWeb`, die von `FlutterWebXrPlatform` erbt. Die `FlutterWebXrWeb` Klasse enthält eine Funktion, die für die Registrierung dieser spezifischen Implementierung bei einer übergeordneten Klasse zuständig ist. Der Parameter `Registrar` wird genutzt, um zusätzliche Konfigurationen oder Anmeldungen durchzuführen, jedoch wird dieser im gezeigten Code nicht verwendet (Terkelsen, [2019](#)). Die statische Eigenschaft `instance` der `FlutterWebXrPlatform` wird auf eine neue Instanz von `FlutterWebXrWeb` gesetzt. Somit wird in diesem Anwendungsfall eine Instanz der `FlutterWebXrWeb` Klasse als Standard gesetzt. Eine weitere Funktion mit den Namen `getPlatformVersion`, die vom Platform Interface implementiert wird, soll den User-Agent des aktuellen Web-Browsers zurückgeben, um Informationen über den Browser und das Betriebssystem des Benutzers zu erhalten. Dabei greift die Funktion auf die `dart:html` Bibliothek zu, die HTML-Elemente und andere Ressourcen für webbasierte Anwendungen bereitstellt, die mit dem Browser und dem DOM (Document Object Model) interagieren müssen (Dart API, [2023a](#)). Wenn ein Plugin für mehrere Plattformen entwickelt wird, einschließlich der Web-Plattform, sollte erwägt werden,

die “Web”-Version des Plugins als separates Paket zu extrahieren, anstatt es im selben Paket wie der Kern des Plugins einzubetten.

Listing 3.5: FlutterWebXrWeb Klasse.

```
/// A web implementation of the FlutterWebXrPlatform of the
FlutterWebXr plugin.
class FlutterWebXrWeb extends FlutterWebXrPlatform {
  FlutterWebXrWeb();

  static void registerWith(Registrar registrar) {
    FlutterWebXrPlatform.instance = FlutterWebXrWeb();
  }

  //Returns a [String] containing the version of the
  platform.
  @override
  String? getPlatformVersion() => html.window.navigator.
    userAgent;

  /// Initiates a WebXR session.
  @override
  Future<void> startSession() async {
    try {
      /// logic will be added later
    } catch (e) {
      throw Exception('Failed to start xr session');
    }
  }
}
```

Neben den oben erläuterten Komponenten eines Flutter-Plugins gibt es weitere Komponenten, die zur Gesamtarchitektur gehören. Hierzu gehören Modelle und Entitäten, die Daten darstellen, mit denen das Plugin arbeitet. Enumeration (Aufzählungen) und Konstanten werden ebenfalls verwendet und dienen als fest definierte Werte oder Typen, die innerhalb des Plugins verwendet werden. Darüber hinaus gibt es Module, die als Schnittstelle zwischen Dart- und JavaScript-Code dienen. Das sind Klassen oder Funktionen in Dart, die JavaScript-Funktionen mithilfe der JS Bibliothek direkt aufrufen. So wird es möglich, JavaScript Objekte in Dart darzustellen und zugänglich zu machen. Auch UI/UX Komponenten, wie Widgets und Tools, die speziell für Flutter Web entwickelt wurden, um AR-Inhalte auf benutzerfreundliche Weise zu präsentieren, gehören zur Architektur. Jedes Flutter-Plugin enthält in der Regel eine Beispielanwendung, die demonstriert, wie das Plugin verwendet wird. Diese kann beim Testen und Entwickeln des Plugins nützlich sein und dient als einfaches

Tutorial für andere Entwickler. Zusätzlich ist eine API-Dokumentation Teil des Plugins, die die verfügbaren Funktionen und Eigenschaften beschreibt. Eine weitere standardisierte Komponente ist die README-Datei. Diese dient als ausführliche Anleitung zur Verwendung des Plugins, zu den unterstützten Plattformen und Funktionen sowie zu möglichen Einschränkungen.

3.4 Implementierungsschritte

In diesem Abschnitt werden die verschiedenen Implementierungsschritte erläutert, die für die Umsetzung des Plugins geplant sind. Es wird beschrieben, welche Schritte in welcher Reihenfolge durchgeführt werden, um das Ziel des Projekts zu erreichen.

1. Eine erste Einführung in das Flutter Plugin-Ökosystem und die Erstellung eines Test-Plugins für die Web-Plattform, um den Aufbau eines Plugin zu verstehen.
2. Erkundung der JS-Bibliothek.
3. Um das Funktionsprinzip der JS Bibliothek zu verstehen, sollte ein spezifischer Anwendungsfall mit dieser Bibliothek umgesetzt werden.
4. Sobald ein sicherer Umgang mit dem Flutter Ökosystem und der JS Bibliothek besteht, findet eine erste Auseinandersetzung mit der Web XR API statt.
5. Die Web XR API wird in einer nativen Webumgebung mit dem Framework Vue.js eingesetzt und getestet.
6. Three.js Flutter Bibliothek wird getestet, um 3D-Objekte in der AR-Umgebung darstellen zu können.
7. Fallback: Beschreibung der alternativen Umsetzungsmöglichkeiten des Anwendungsfalls, falls die geplante Entwicklung nicht erfolgreich sein sollte.
8. Implementierung von Core-Modulen: Erstellung der AR-Basisfunktionen des Plugins.
9. Implementierung von Interaktionsmodulen: Entwickeln von Klassen und Funktionen, die die Interaktion zwischen Dart und JavaScript ermöglichen.
10. Implementierung von UI/UX Elementen: Erstellung von Widgets und Tools für die Darstellung von AR-Inhalten in Flutter Web.
11. Erstellung der API-Dokumentation und README-Datei: In der Dokumentation werden die verfügbaren Funktionen und ihre Anwendung ausführlich beschrieben. Eine Anleitung zur Integration und Nutzung des Plugins wird ebenfalls bereitgestellt.

12. Veröffentlichung: Bereitstellung des Plugins für die Flutter-Community, z. B. auf pub.dev.

Anforderungserhebung

Dieser Abschnitt beschreibt die Anforderungen, die für das Plugin entwickelt wurden, um den spezifischen Anwendungsfall der in der Praxis entwickelten Dating-App umzusetzen. Hierfür wurden Anforderungsdokumente erstellt, die zwischen funktionalen und nicht-funktionalen Anforderungen unterscheiden. Klessascheck (Klessascheck, 2023) definiert die beiden Arten von Anforderungen folgendermaßen: “Funktionale Anforderungen sind Anforderungen mit Bezug zur Zweckbestimmung des Produkts. Zu den nicht-funktionale Anforderungen zählen Anforderungen wie die Zuverlässigkeit und das Zeitverhalten.” Das Erheben und Definieren von Anforderungen ist ein entscheidender Schritt in der Softwareentwicklung. Wenn die Anforderungen richtig formuliert werden, kann das Endprodukt den Bedürfnissen der Benutzer entsprechen und somit das Geschäftsziel erreichen. Für diese Forschungsarbeit müssen die Anforderungen also sowohl den spezifischen Anwendungsfall in der Dating-App als auch die generellen Bedürfnisse von Entwicklern berücksichtigen, die das Plugin in unterschiedlichen Kontexten nutzen möchten. Als Methode zur Anforderungserhebung wurde Prototyping gewählt, da ein einfacher Prototyp dabei helfen kann, die Anforderungen zu klären. Im Folgenden werden die funktionalen und nicht-funktionalen Anforderungen für das Plugin aufgelistet.

4.1 Funktionale Anforderungen

1. **FA1 - Geschenk-Auswahl:** Benutzer müssen in der Lage sein, aus einer Liste von virtuellen Geschenken auszuwählen.
2. **FA2 - 3D-Inhaltsverwaltung:**
 - Benutzer müssen die Möglichkeit haben, bestimmte einfache 3D-Inhalte (z.B. einfache Geometrien wie Würfel, Kugel, etc.) direkt innerhalb der Anwendung zu erstellen.
 - Das Plugin muss eine Importfunktion bieten, die das Integrieren von 3D-Modellen in gängigen Dateiformaten (z.B. .glTF oder .obj) unterstützt.
3. **FA3 - AR-Darstellung:** Benutzer müssen in der Lage sein, ein virtuelles Geschenk in ihrer realen Umgebung mithilfe der AR-Funktionalität zu platzieren und zu betrachten.
4. **FA4 - Interaktion mit AR-Geschenken:** Benutzer sollen mit den virtuellen Geschenken interagieren können, beispielsweise um sie zu öffnen, ihre Größe zu ändern oder sie in der AR-Umgebung zu verschieben.

5. **FA5 - Überprüfung der AR-Kompatibilität:** Das Plugin muss automatisch die Verfügbarkeit und Kompatibilität von AR-Funktionen auf dem jeweiligen Gerät erkennen. Basierend auf dieser Überprüfung muss die Benutzeroberfläche entsprechend angepasst werden, um entweder die AR-Funktionen zu aktivieren oder dem Benutzer mitzuteilen, dass die AR-Funktionalität auf seinem Gerät nicht verfügbar ist.

4.2 Nicht-funktionale Anforderungen

1. **NFA1 - Performance:** Die AR-Darstellung muss flüssig und ohne Verzögerung funktionieren.
2. **NFA2 - Kompatibilität:** Das AR-Feature sollte auf den meisten gängigen Smartphones und Webbrowsern funktionieren.
3. **NFA3 - Benutzerfreundlichkeit:** Das UI für das AR-Feature muss intuitiv und leicht verständlich sein.
4. **NFA4 - Skalierbarkeit:** Die Lösung sollte in der Lage sein, eine wachsende Anzahl von Benutzern und Geschenken zu unterstützen.
5. **NFA5- Erweiterbarkeit:** Das Plugin muss modular aufgebaut sein, um zukünftige Erweiterungen oder Integrationen mit anderen Systemen zu erleichtern.
6. **NFA6 - Ästhetik:** Die AR-Darstellung der Geschenke sollte von hoher Qualität und ansprechend sein.
7. **NFA7 - Einführung und Dokumentation:** Klares und umfassendes Dokumentationsmaterial und Tutorials muss vorhanden sein.
8. **NFA8 - Regelmäßige Updates und Support:** Das Plugin sollte regelmäßige Updates haben und einen klaren Support-Mechanismus für Entwickler bieten.

Implementierung

In diesem Abschnitt wird die Implementierung des Plugins ausführlich beschrieben. Zusätzlich werden Codebeispiele präsentiert, um die Funktionsweise des Plugins zu veranschaulichen. Der Entwicklungsprozess wird in Sprints durchgeführt, um Zwischenergebnisse festzuhalten und Herausforderungen zu bewerten. Es ist geplant, den Entwicklungsprozess in zwei Sprints aufzuteilen.

5.1 Sprint 1

In der ersten Entwicklungsphase soll ein erster Prototyp des Plugins erstellt werden, der grundlegende Funktionen aufweist. Hierfür müssen einige Vorbereitungen getroffen werden.

5.1.1 Einführung in die Flutter-Plugin-Entwicklung

Für die Implementierung wurde das Flutter SDK 3.13.1 verwendet. Zunächst wurde ein neues Flutter-Plugin erstellt. Dies kann über die Kommandozeile mithilfe verschiedener Parameter erfolgen, um erste grundlegende Rahmenbedingungen für das Plugin zu schaffen. In Flutter gibt es unterschiedliche Paket-Typen, die erstellt werden können. Es gibt das Dart-Paket, das Plugin-Paket und das FFI-Plugin-Paket, die sich hauptsächlich in der Art der bereitgestellten Funktionen unterscheiden. Dart Pakete sind generell in der Programmiersprache Dart geschrieben und enthalten modularisierte und wiederverwendbare Dart-Code-Elemente. Einige können auch eine Flutter-spezifische Funktionalität enthalten, wie z.B. String-Manipulation, Netzwerkanfragen und Zustandsverwaltung. Die Verwendung dieser Pakete ist auf Flutter beschränkt und erfordert eine Abhängigkeit vom Flutter-Framework. Laut der Flutter-Dokumentation (Flutter Dokumentation, [2023a](#)) ist das Plugin-Paket ein spezialisiertes Dart-Paket, das eine API in Dart enthält und plattform-spezifische Implementierungen bereitstellt. Diese sind besonders nützlich, wenn Entwickler auf Funktionen zugreifen müssen, die spezifisch für eine bestimmte Plattform (wie Android oder iOS) sind oder wenn sie APIs verwenden möchten, die nicht durch Dart-Code allein zugänglich sind. Beispiele hierfür sind Kamera-zugriff, Bluetooth-Kommunikation und Geolokalisierung. Plugin-Pakete können für Android (mit Kotlin oder Java), iOS (mit Swift oder Objective-C), Web, macOS, Windows oder Linux oder einer Kombination daraus geschrieben werden. Das FFI-Plugin-Paket ist ebenfalls ein spezialisiertes Dart-Paket, das eine Dart-API enthält und plattform-spezifische Implementierungen bereitstellt, die Dart FFI verwenden (Flutter Dokumentation, [2023a](#)). Dart FFI (Foreign Function Interface) ist ein Mechanismus, mit dem Dart-Code direkt mit C- und C++-Bibliotheken kommunizieren kann. Entwickler können somit native

Funktionen aus diesen Sprachen direkt aus Dart heraus aufrufen und von der Leistungsfähigkeit und den Fähigkeiten bestehender, nativ geschriebener Bibliotheken profitieren (Dart Dokumentation, [2023a](#)). Zusammenfassend kann gesagt werden, dass alle Plugins Pakete sind, aber nicht alle Pakete Plugins. Ein Paket wird zu einem Plugin, wenn es neben dem Dart-Code auch plattformspezifischen Code enthält. Für den Anwendungsfall der Forschungsarbeit werden plattformabhängige Implementierungen benötigt, weshalb ein Plugin-Paket entwickelt wird. Um ein Plugin-Paket zu erstellen, muss der Befehl `flutter create` mit dem Flag `--template=plugin` kombiniert werden. Mit der Option `-platforms=` können die Plattformen angegeben werden, die vom Plugin unterstützt werden sollen. Die verfügbaren Plattformen sind: `android`, `ios`, `web`, `linux`, `macos` und `windows`. Wenn keine Plattformen angegeben werden, unterstützt das resultierende Projekt keine Plattformen. Mithilfe der `--org` Option kann die Organisation des Entwicklers unter Verwendung der Notation des umgekehrten Domainnamens angegeben werden. Dieser Wert wird in verschiedenen Paket- und Bundle-Identifiern im generierten Plugin-Code verwendet (Flutter Dokumentation, [2023a](#)).

5.1.2 Zugriff auf Web-APIs mit der html-Bibliothek

Nachdem ein Plugin-Projekt erstellt wurde und die Funktionsweise eines Flutter-Plugins untersucht wurde, war eine direkte Schnittstelle zu den Web-APIs erforderlich. Diese Schnittstelle wird von der `html`-Bibliothek bereitgestellt, die Entwicklern den Zugriff auf niedrigere Web-Funktionalitäten ermöglicht. Die Bibliothek enthält Typen und Funktionen, die den Web-Plattform-APIs entsprechen, die im Browser verfügbar sind. So bietet die `html`-Bibliothek zum Beispiel direkten Zugriff auf das Document Object Model, um damit interagieren zu können. Darüber hinaus ermöglicht sie den Zugriff auf Teile des `navigator`-Objekts und spezifische Web-APIs, die in Flutter nicht direkt verfügbar sind. Das beinhaltet APIs wie den Zugriff auf den Local Storage, den Session Storage, die Fetch-API und viele andere (Dart API, [2023a](#)). Insgesamt versucht das `html`-Package, die Lücke zwischen der Flutter-Plattform und den spezifischen Web-APIs zu schließen.

5.1.3 Dart trifft JavaScript: Integration und Interaktion

Laut den MDN Web Docs (MDN, [2023a](#)) wird das `navigator`-Objekt wie folgt beschrieben: “Die Schnittstelle `Navigator` repräsentiert den Zustand und die Identität des Benutzeragents. Sie ermöglicht es Skripten, Abfragen an den Benutzeragenten zu stellen und sich für bestimmte Aktivitäten zu registrieren. Ein `Navigator` Objekt kann über die schreibgeschützte Eigenschaft `[window.navigator]` abgerufen werden.” Allerdings bietet die `html`-Bibliothek nicht den gleichen Zugriff auf das `navigator`-Objekt und seine Eigenschaften wie in reinen JavaScript-Browserumgebungen. Dies liegt daran, dass Flutter Web eine Abstraktionsebene über der tatsächlichen Browserumgebung bietet, um eine einheitliche Entwicklungserfahrung über verschiedene Plattformen hinweg zu gewährleisten.

Um Eigenschaften und Methoden, die in nativen Browserumgebungen über das `navigator`-Objekt verfügbar sind, direkt in Flutter Web zugänglich zu machen, bietet das Flutter-Ökosystem Plugins, die viele dieser Lücken schließen können. Im speziellen Anwendungsfall dieser Forschungsarbeit war der Zugriff auf die benötigte WebXR-API ebenfalls nicht zugänglich. Um spezifische APIs zugänglich für Flutter Web zu machen, kann die JS-Bibliothek verwendet werden. Mit dieser Bibliothek können JavaScript-APIs direkt per Dart-Code aufgerufen werden oder umgekehrt. Auch andere Web-Bibliotheken für Flutter wie beispielsweise die `sqlite3` Bibliothek nutzen die JS-Bibliothek, um Zugriff auf Web-APIs zu erhalten (Wangoo, 2023). Der Hauptteil der JS-Bibliothek bietet Annotationen und Funktionen, mit denen der Entwickler spezifizieren kann, wie der Dart-Code mit JavaScript-Code interoperiert. Die Dart-zu-JavaScript-Compiler - `dartdevc` und `dart2js` - erkennen diese Annotationen und verwenden sie, um den Dart-Code mit JavaScript zu verbinden (Pub, 2023b). Es werden zwei Arten der Interoperabilität (Interops) unterstützt, die als Brücke zwischen Dart-Code und JavaScript-Code fungiert. Allgemein wird die Interoperabilität für folgende Fälle angewendet:

1. Aufruf von JavaScript-Funktionen aus Dart: Hiermit können Flutter-Entwickler JavaScript-Funktionen direkt aus ihrem Dart-Code heraus aufrufen.
2. Aufruf von Dart-Funktionen aus JavaScript: Es erlaubt JavaScript, Dart-Funktionen als Callbacks oder direkt aufzurufen.

Zum einen gibt es die dynamische Interoperabilität, die lange Zeit als Standard in der JS-Bibliothek galt. Diese ermöglicht eine flexible und dynamische Interaktion zwischen JavaScript und Dart. Die Interoperabilität wird bei dieser Art zur Laufzeit erstellt, was mehr Flexibilität beim Zugriff auf Objekte von Dart und JavaScript gewährleistet. Es ist generell eine größere Anpassungsfähigkeit möglich, jedoch kann dies zu Typsicherheitsrisiken und potenziellen Laufzeitfehlern führen. Zusammenfassend kann gesagt werden, dass die dynamische Interoperabilität flexibel und nicht so streng ist wie die statische Interoperabilität, die vom Dart-Team bevorzugt wird (Wangoo, 2023). Das Dart-Team entfernt sich von der dynamischen Interoperabilität, da sie nur wenige Leistungsvorteile bietet. Auf der Pub-Seite der Bibliothek (Pub, 2023b) heißt es: „Obwohl wir Benutzern auch in absehbarer Zukunft weiterhin ermöglichen werden, diese Funktionalität zu nutzen, wechselt Dart zu einer statischeren, inline-klassenbasierten Interop.“ Diese statische Interoperabilität soll die bisherige dynamische Interoperabilität ersetzen. Der Grund dafür sind Vorteile wie Idiomatik, Leistung, Typsicherheit, die Fähigkeit zur Interoperabilität mit DOM-Typen und die Kompatibilität mit WebAssembly (Pub, 2023b). Um die statische Interoperabilität zu nutzen, muss das Dart Software Development Kit die minimale Version 2.19 beziehen und die JS-Bibliothek muss die Bedingung erfüllen, die minimale Version 0.6.6 zu haben. Bei dieser statische Art gibt es eine vordefinierte und statische Schnittstelle zwischen JavaScript und Dart. Diese

ermöglicht zuvor nicht verfügbare Funktionen, wie das einfache Verpacken und Transformieren von APIs, die Umbenennung von Mitgliedern und die statische Überprüfung (Dart Dokumentation, [2023c]). Zudem werden bei dieser Art die Typen, Methoden und Eigenschaften für den Zugriff von Dart und JavaScript aus spezifiziert. Die Interoperabilität wird zur Kompilierzeit hergestellt und gewährleistet somit die Typsicherheit. Die statische Interoperabilität ist strenger als die dynamische Interoperabilität (Wangoo, [2023]). Sie geht einen Schritt weiter, um Interoperabilität in Dart idiomatischer zu machen und die Grenze zwischen den beiden Sprachen sichtbarer zu machen. Beispielsweise erzwingt sie, dass JS-Klassen nicht mit Dart gemischt werden sollen (dynamische Aufrufe sind nicht erlaubt und JS-Interop-Typen können nicht als Dart-Klasse implementiert werden). Es gibt Wege und Regeln, wie etwas gemacht werden soll (Dart Dokumentation, [2023c]). Die statische Interoperabilität gilt als nächste Generation für diesen Anwendungsfall und wird auch vom Dart-Team aufgrund ihrer Leistungsvorteile empfohlen. Derzeit ist die statische Interoperabilität experimentell und in Entwicklung. Es wird zudem empfohlen, sich mit den neuen Mustern vertraut zu machen und mit diesen zu experimentieren, um die zukünftige Migration zu erleichtern (Pub, [2023b]). Die Bibliothek enthält auch einen Zusatz `names js_util`, der Low-Level-Utilities bietet und von den JS- und `dart2wasm` Backends unterstützt wird. Die Low-Level-Utilities können verwendet werden, wenn das Umschließen von JavaScript mit einer statischen, annotierten Dart-API nicht möglich ist. Das bietet mehr Flexibilität, insbesondere in seltenen Fällen, in denen die Interoperabilität nicht ausdrucksstark genug ist (Dart Dokumentation, [2023c]). Allerdings weist die Dart-Dokumentation (Dart Dokumentation, [2023c]) darauf hin: “Es ist jedoch nicht so ergonomisch und wir planen nicht, es auf die gleiche Weise wie die statische Interop zu optimieren. Daher empfehlen wir dringend, immer dann, wenn es möglich ist, die statische Interop gegenüber `dart:js_util` zu verwenden.”

Um das Prinzip der JS-Bibliothek zu verstehen, wurde ein spezifischer Anwendungsfall mit dieser Bibliothek umgesetzt. Das Ziel war es, den Ladestand des Geräte-Akkus über die Browser-API zu erhalten. Dafür wird die `getBattery()`-Methode benötigt, die Information über den Akku des Systems liefert. Die Methode gibt ein `Promise` zurück, das in einem `BatteryManager`-Objekt aufgelöst wird. Dieses Objekt implementiert die Battery Status API, die letztendlich Informationen zum Ladestand des Akkus des System bereitstellt (MDN, [2023e]). Zunächst wurde eine Methode für diese Funktionalität im Plattform Interface implementiert, damit alle Klassen, die dieses Interface erweitern, die Methode überschreiben und implementieren können. Als nächstes wurde diese Methode auch in der Hauptklasse des Plugins implementiert, die für den Endbenutzer zugänglich ist. Hier wird die `getBatteryLevel()`-Methode mithilfe einer Instanz des Plattform Interface aufgerufen.

Listing 5.1: `getBatteryLevel()`-Methode.

```
Future<double> getBatteryLevel() {  
    return FlutterWebXrPlatform.instance.getBatteryLevel();  
}
```

Anschließend wird die JS-Bibliothek erstmalig eingesetzt, indem die `getBatteryLevel()`-Methode in JavaScript aufgerufen wird. Im anschließenden Code wird dargestellt, wie das `BatteryManager`-Objekt in JavaScript über Dart-Code abgebildet wird. Hierfür ist die `@JS`-Annotation notwendig, die in diesem Fall durch die experimentelle `@staticInterop`-Annotation ergänzt wird. Aufgrund der statischen Ergänzung müssen Eigenschaften, Methoden, Getter und -Setter in eine `Extension` ausgelagert werden. In diesem Fall gibt es in der `BatteryManager`-Klasse zwei Getter, die jeweils eine Eigenschaft der JavaScript-Klasse abbilden. Um eine JavaScript-Funktion in Dart abzubilden, wird erneut die `@JS`-Annotation verwendet, die als Parameter den Namen der JavaScript-Funktion enthält. Der komplette Funktionsaufruf mit dem Namen `navigator.getBattery` wird in diesem Fall übergeben. Anschließend wird die Funktion in Dart mit dem `external`-Schlüsselwort und dem Rückgabotyp `BatteryManager` definiert. Das Dart-Team (Dart Team, [2023](#)) beschreibt Funktionen mit dem `external`-Schlüsselwort folgendermaßen: “Eine externe Funktion in Dart ist eine Funktion, deren Körper separat von ihrer Deklaration bereitgestellt wird.” Das bedeutet, dass die Implementierung an anderer Stelle existiert und zur Laufzeit verfügbar sein wird. In diesem Fall liegt die Implementierung des Funktionskörpers in externem JavaScript.

Listing 5.2: Veranschaulichung der JS Bibliothek anhand eines Beispiels.

```
@JS('BatteryManager')  
@staticInterop  
class BatteryManager {}  
  
extension on BatteryManager {  
    external bool get charging;  
    external double get level;  
}  
  
@JS('navigator.getBattery')  
@staticInterop  
external BatteryManager getBattery();
```

In der web-spezifischen Plugin-Klasse wird die Methode `getBatteryLevel`

definiert, die die Anwendungslogik enthält und somit die Methode aus dem Plattform Interface überschreibt. Diese asynchrone Methode gibt ein `Future<double>` zurück, welches den Batteriestand in Prozent als Gleitkommazahl repräsentiert. Der Try-Catch Block wird verwendet, um mögliche Fehler während der Codeausführung abzufangen und entsprechend zu behandeln. Hierbei wird davon ausgegangen, dass `getBattery()` eine JavaScript-Funktion ist, die ein Promise zurückgibt. Wenn das Promise aufgelöst wird, gibt es ein `BatteryManager`-Objekt zurück. Die `promiseToFuture`-Funktion konvertiert das JavaScript-Promise in ein Dart Future (Flutter API, 2023b). Anschließend wird auf das Future gewartet, das ein `BatteryManager`-Objekt zurückgibt und in einer Variable gespeichert wird. Im finalen Schritt wird mithilfe der Hilfsfunktion `getProperty` der JS-Bibliothek auf eine Eigenschaft des Objekts `batteryManager` zugegriffen. Dieser Wert gibt den aktuellen Batteriestand an und fungiert als Rückgabewert dieser Funktion. Wenn während eines der obigen Schritte ein Fehler auftritt, wird der Code im Catch-Block ausgeführt. In der Frontend-Anwendung kann die Funktionalität über die Hauptklasse des Plugins aufgerufen werden.

Listing 5.3: Darstellung der `getBatteryLevel`-Funktion.

```
@override
Future<double> getBatteryLevel() async {
  try {
    // converts a javascript promise to a dart future
    final Future<BatteryManager> batteryFuture =
      promiseToFuture<BatteryManager>(getBattery());

    final BatteryManager batteryManager = await
      batteryFuture;

    double level = getProperty(batteryManager, 'level');
    return level;
  } catch (e) {
    throw Exception('Failed to fetch battery level');
  }
}
```

5.1.4 Integration von Three.js

Dieser Abschnitt beschäftigt sich mit der 3D-Bibliothek Three.js und untersucht, ob sie für den Anwendungsfall der Forschungsarbeit geeignet ist. Für die Darstellung von dreidimensionalen Objekten im Web wird die Technologie WebGL eingesetzt. WebGL ist ein offener Webstandard für eine Low-Level-3D-Grafik-API, der plattformübergreifend und lizenzgebührenfrei ist. Er ist über das HTML5-

Canvas-Element in ECMAScript zugänglich (MDN, [2023d](#)). Three.js vereinfacht die Nutzung von WebGL und ermöglicht somit das Einbetten von 3D-Objekten im Web (Three.js Dokumentation, [2023a](#)). Da für den Einsatz von Augmented Reality 3D-Objekte benötigt werden, wurde diese Bibliothek für die Entwicklung des Plugins gewählt. Über den Paketmanager Pub von Flutter wurde eine Implementierung der Three.js Bibliothek in Flutter gefunden, die auch für die Web-Plattform von Flutter verfügbar ist. Diese Implementierung, `three_dart` genannt, basiert auf der `flutter_gl` Bibliothek (Pub, 2023i). `flutter_gl` greift plattformübergreifend auf die OpenGL API mithilfe von `dart:ffi` zu und stellt somit diese Technologie für Flutter bereit (Wasabia, [2023](#)).

Die Technologie OpenGL ist eine grafikbasierte Schnittstelle, die auf Spezifikationen basiert, um 2D- und 3D-Grafikanwendungen auf verschiedenen Plattformen zu ermöglichen. Sie bildet das Fundament von WebGL, da WebGL auf OpenGL ES aufbaut. Durch diese Konformität kann die WebGL API die Hardware-Grafikbeschleunigung nutzen, die vom Benutzergerät bereitgestellt wird. Entwickler, die mit OpenGL ES 2.0 vertraut sind, werden WebGL als shaderbasierte API erkennen, die die OpenGL Shading Language und Konstrukte verwendet, die semantisch ähnlich sind wie die der zugrunde liegenden OpenGL ES API. Die wichtigsten Browseranbieter wie Apple (Safari), Google (Chrome), Microsoft (Edge) und Mozilla (Firefox) sind Mitglieder der WebGL-Arbeitsgruppe, was bedeutet, dass alle modernen Browser unterstützt werden (Khronos Group, [2023](#)).

Die Integration der `three_dart` Bibliothek gestaltete sich als unkompliziert. In der Dokumentation des Plugins stehen zahlreiche Beispiele zur Verfügung, die Entwicklern die Implementierung erleichtern. Darüber hinaus bietet die Bibliothek den vollen Funktionsumfang der ursprünglichen JavaScript Three.js Bibliothek, wodurch vielseitige Szenarien realisiert werden können. In diesem Anwendungsfall der Forschungsarbeit wurde erfolgreich ein einfaches 3D-Objekt dargestellt. Auf dieser Grundlage kann die Entwicklung fortgesetzt werden, um die Bibliothek später zur Darstellung von 3D-Objekten zu nutzen.

5.1.5 Fallback-Konzept

Der folgende Abschnitt beschreibt alternative Umsetzungsmöglichkeiten des Anwendungsfalls, falls die geplante Entwicklung nicht erfolgreich sein sollte. Da die WebXR-API zurzeit eine experimentelle Technologie darstellt (MDN, [2023e](#)) und die Entwicklung von Flutter Web-Plugins Herausforderungen mit sich bringen kann, besteht für das Ziel dieser Forschungsarbeit das Risiko eines Misserfolgs. Daher spielt das Risikomanagement eine wichtige Rolle in dieser Arbeit. Es wird ein Fallback-Konzept verwendet, das als Alternativlösung im Kontext der Softwareentwicklung dienen sollte, falls der geplante Ansatz nicht funktioniert. Ein gutes Fallback-Konzept sollte flexibel und mit minimalem Aufwand integrierbar sein. Zudem können Strategien entwickelt werden, um die Risiken

bei der Entwicklung des Plugins zu minimieren. Das Ziel des Fallback-Konzept besteht darin, den Anwendungsfall erfolgreich umzusetzen. Falls die Entwicklung fehlschlägt und somit kein 3D-Objekt per Augmented Reality in der Flutter Web App dargestellt werden kann, kann auf andere Ansätze zurückgegriffen werden.

Eine Möglichkeit besteht darin, einen hybriden Ansatz zu wählen. Das bietet den Vorteil, das Beste aus beiden Welten zu nutzen, indem eine Kombination aus verschiedenen Technologien verwendet wird. Das bedeutet, dass Teile der Anwendung auf eine andere Plattform ausgelagert werden. Für den konkreten Anwendungsfall könnte auf eine native Web-App (z.B. Vue.js) verwiesen werden, die die Web XR API nativ implementiert. Auf diese Weise kann zwar kein Plugin für die Community für den spezifischen Anwendungsfall veröffentlicht werden, jedoch kann die Funktionalität des Anwendungsfalls umgesetzt werden. Eine weitere hybride Lösung besteht darin, die **flutter unity 3D widget** Bibliothek zu verwenden. Diese Bibliothek ermöglicht das Einbetten von Unity-Projekten in Flutter-Anwendungen. Unity ist eine vielseitige Spieleentwicklungsplattform, die es Entwicklern ermöglicht, interaktive 2D-, 3D-, VR- und AR-Erlebnisse für eine Vielzahl von Plattformen zu erstellen und zu veröffentlichen. Insbesondere die AR Foundation ermöglicht die AR-Entwicklung in Unity (Unity Technologies, [2023](#)). Die **flutter unity 3D widget** Bibliothek ist auch für die Web-Plattform von Flutter verfügbar und somit eine Option für das Fallback-Konzept (Pub, [2023a](#)). Der Anwendungsfall der Forschungsarbeit kann damit umgesetzt werden, indem die AR-Funktionalität mit der Plattform Unity umgesetzt wird und mithilfe der Bibliothek in die Flutter-Anwendung integriert wird.

Ein weiterer Ansatz für das Fallback-Konzept ist das Progressive Enhancement. Diese Design-Philosophie bietet eine Grundlage an essentiellen Inhalt und Funktionalität für so viele Nutzer wie möglich. Dabei bietet sie die bestmögliche Erfahrung nur für Nutzer der modernsten Browser, die den erforderlichen Code ausführen können. Auf diese Weise wird sichergestellt, dass die Website oder Anwendung für alle Benutzer zugänglich und funktionsfähig ist, unabhängig von ihrem Gerät oder Browser (MDN, [2023c](#)). Anstelle einer vollständigen AR-Lösung könnte mit einer einfacheren 3D-Ansicht des Objekts gestartet und diese nach und nach erweitert werden, sobald die Technologie von Flutter Web und der Web XR API ausgereifter ist. Hierbei könnte die **three_dart**-Bibliothek zum Einsatz kommen, die das Rendern von 3D-Objekten in Dart unterstützt (Pub, [2023h](#)).

5.1.6 Implementierung von AR-Basismodulen

Um mit der WebXR Device API interagieren zu können, werden in einem Core-Modul mehrere Klassen und Funktionen der API definiert. Dies erfolgt mithilfe der JS-Bibliothek, die die Interaktion zwischen Dart und JavaScript ermöglicht.

Im Core-Modul wird auf die XR-Eigenschaft zugegriffen, um auf die API zugreifen zu können. Die JS-Bibliothek verwendet die `@JS`-Annotation, um Dart anzuweisen, auf ein spezifisches JavaScript-Objekt zuzugreifen. In diesem Fall wird Dart angewiesen, auf das `xr`-Objekt des globalen `navigator`-Objekts in einem Webkontext zuzugreifen. Die folgende externe Getter-Funktion namens `xrSystem` gibt ein `XRSystem` oder `null` zurück. Das Schlüsselwort `external` bedeutet, dass die Implementierung dieses Getters nicht in Dart selbst geschrieben wird, sondern außerhalb liegt, in diesem Fall in JavaScript. Zudem wird im Core-Modul der AR-Funktionalität die JavaScript-Klasse `XRSystem` definiert. Die Schnittstelle XR-System stellt eine Methode bereit, die den Zugriff auf ein XR-Session Objekt ermöglicht, das eine WebXR Session repräsentiert. Diese Methode nimmt einen Parameter an, der den angeforderten Modus der Session festlegt (MDN, [2023m](#)). Die MDN Web Docs (MDN, [2023i](#)) beschreiben eine Session wie folgt: “Die `XRSession` Schnittstelle stellt eine laufende XR-Sitzung dar und bietet Methoden und Eigenschaften zur Interaktion und Steuerung der Sitzung.” Eine weitere Methode des XR-Systems überprüft, ob das aktuelle Gerät einen bestimmten immersiven Modus (AR, VR) unterstützt (MDN, [2023l](#)).

Listing 5.4: Zugriff auf das `XRSystem`.

```
@JS('navigator.xr')
external XRSystem? get xrSystem;

@JS("XRSystem")
class XRSystem {
  external XRSession requestSession(String mode);
  external bool isSessionSupported(String mode);
}
```

Die Klasse `XRController` kümmert sich um die Anwendungslogik der AR-Funktionalität des Plugins und um die Interaktion mit anderen erforderlichen Modulen. Zum Beispiel wird dem Controller eine Instanz des `RendererController` für die Three.js Bibliothek übergeben und er enthält Variablen, die für die Verwendung der WebXR Device API benötigt werden. In diesem Fall greift der XR-Controller auf den Getter des Core-Moduls zurück, der die Klasse `XRSystem` zurückgibt und prüft, ob die XR-Eigenschaft verfügbar ist.

Listing 5.5: Die Klasse XRController.

```
class XRController {
  final RendererController rendererController;
  final SceneController sceneController;
  final CameraController cameraController;

  late Object? gl;
  late XRSession xrSession;
  late XRReferenceSpace xrReferenceSpace;
  late XRWebGLLayer baseLayer;

  XRController(
    this.rendererController, this.sceneController, this.
      cameraController);

  bool isWebXrSupported() => xrSystem != null;
}
```

Der `XRController` stellt daraufhin die asynchrone Methode `requestSession` bereit, die ein `Future` zurückgibt und die beiden Methoden aus dem WebXR-Code-Modul implementiert. Zuerst wird geprüft, ob das aktuell verwendete Gerät eine `XRSession` vom Typ AR unterstützt, indem sie die `isSessionSupported` Methode aufruft. Wenn dies nicht der Fall ist, wird ein Fehler ausgelöst und die Methode beendet. Wenn die Unterstützung gewährleistet werden kann, wird mithilfe der Funktion `setupXrDomOverlay` ein DOM-Overlay für die AR-Ansicht erstellt. Diese Implementierung erfolgt jedoch erst im zweiten Sprint der Entwicklung. Als nächstes wird versucht, eine WebXR-Session mit dem Modus `immersive-ar` anzufordern. Hierbei werden die zuvor erstellten Session-Optionen verwendet. Der Aufruf `xrSystem!.requestSession` gibt ein JavaScript-Promise zurück. Daher wird die Methode `promiseToFuture` verwendet, um ein JavaScript-Promise in ein Dart-Future zu konvertieren (Dart API, [2023b](#)). Sobald die Session erfolgreich angefordert wurde und einer globalen Variable im `XRController` zugewiesen wurde, wird die Methode `setupXRSession` aufgerufen, um weitere Einrichtungsaufgaben durchzuführen.

Listing 5.6: Implementierung der `requestSession`-Methode.

```
Future<void> requestSession() async {
  if (!(await isSessionSupported())) {
    throw Exception('WebXR Session not supported');
  }

  final Map<String, dynamic> sessionOptions =
    setupXrDomOverlay();

  try {
    xrSession = await promiseToFuture(
      xrSystem!.requestSession("immersive-ar", jsify(
        sessionOptions)));

    await setupXRSession();
  } catch (e) {
    throw Exception('Error requesting session: $e');
  }
}
```

Die asynchrone `setupXRSession` Methode beinhaltet mehrere Schritte, um eine `XRSession` zu konfigurieren. Die Methode `initializeGL` wird aufgerufen, um den WebGL-Kontext zu initialisieren oder zu konfigurieren, um 2D- und 3D-Grafiken im Webbrowser zu rendern. In diesem Fall wird der Variable `gl`, die den WebGL-Kontext repräsentiert, der WebGL-Kontext des Three.js Renderers zugewiesen. Der `XRWebGLLayer` stellt eine Verbindung zwischen dem WebGL-Kontext und der WebXR-Session her. Er fungiert als Ebene, auf der die 3D-Objekte mithilfe des WebGL-Kontexts gerendert werden. Insbesondere bietet er Zugriff auf den WebGL-Framebuffer und den Viewport, um den Zugriff auf den Kontext zu erleichtern (MDN, [2023o](#)). Die Methode `createXRWebGLLayer` wird aufgerufen, um ein solches Layer-Objekt zu erstellen und das resultierende Objekt in der Variable `xrLayer` zu speichern. Als Parameter werden der Klasse `XRWebGLLayer` die aktuelle `XRSession` und die WebGL-Kontext Variable `gl` übergeben. Im nächsten Schritt wird ein Objekt vom Typ `XRRenderStateInit` initialisiert, das Informationen über die Darstellung von Inhalten in einer WebXR-Session enthält. In diesem Fall wird es mit dem zuvor erstellten `xrLayer` initialisiert. Anschließend wird die Funktion `updateRenderState` aufgerufen, die von der `XRSession` bereitgestellt wird und den Render-Zustand der WebXR-Session mit den im `renderStateInit`-Objekt definierten Einstellungen aktualisiert (MDN, [2023k](#)). Schließlich wird die Methode `requestReferenceSpace` asynchron aufgerufen, die in der globalen Variable `xrReferenceSpace` den Referenzraum speichert. Ein Referenzraum vom Typ `XRReferenceSpace` definiert in WebXR die Koordinaten des Raums, in dem der Benutzer sich bewegt. Dies ist wichtig, um zu verstehen, wie virtuelle Objekte im physischen Raum po-

sitioniert werden sollten (MDN, [2023g](#)). Zusammenfassend wird die Methode `setupXRSession` verwendet, um eine WebXR-Session einzurichten, indem ein WebGL-Layer erstellt, der Render-Zustand konfiguriert und ein Referenzraum angefordert wird.

Listing 5.7: Implementierung der `setupXRSession`-Funktion.

```
Future<void> setupXRSession() async {  
  initializeGL();  
  XRWebGLLayer xrLayer = createXRWebGLLayer();  
  XRRenderStateInit renderStateInit =  
    createXRRenderStateInit(xrLayer);  
  updateRenderState(renderStateInit);  
  await requestReferenceSpace();  
}
```

Nachdem erfolgreich eine Session erstellt wurde und die benötigten Konfigurationen abgeschlossen wurden, wird die Funktion `startFrameHandler` aufgerufen. Diese Funktion ruft die `requestAnimationFrame` Funktion auf, die von der Klasse `XRSession` bereitgestellt wird und fordert somit die nächste Animationsschleife für die XR-Session an. Die Funktion teilt dem Browser mit, dass eine Animation ausgeführt werden soll und plant einen Callback (Rückruf), der ausgeführt wird, sobald der Browser bereit ist, die virtuelle Umgebung der Session auf das XR-Display zu rendern. In diesem Fall wird die Methode `frameHandler` als Callback verwendet, die ein Objekt vom Typ `XRFrame` als Eingabeparameter erhält. Dieses Objekt beschreibt den Zustand aller verfolgten Objekte für die Session und enthält einen Zeitstempel (MDN, [2023j](#)). Die Methode `frameHandler` sollte mit der Methode `allowInterop` umhüllt werden, die von der JS-Bibliothek bereitgestellt wird. Dies ist erforderlich, wenn eine Dart-Funktion als Argument an eine JavaScript-API übergeben wird. Dadurch kann die Funktion von JavaScript aus aufgerufen werden (Pub, [2023b](#)). Im Körper der `frameHandler` Methode wird eine Rekursion gestartet, indem der `startFrameHandler` erneut aufgerufen wird, um die nächste Animationschleife anzufordern und somit die Funktion vor jedem Repaint des Browsers ausgeführt wird. Als nächstes wird im `frameHandler` Callback die Methode `bindGraphicsFramebuffer` aufgerufen, um sicherzustellen, dass alle Grafiken, die in der aktuellen XR-Session gerendert werden sollen, in das richtige Framebuffer geschrieben werden, das dem `baseLayer` der XR-Session zugeordnet ist. Nach Abschluss dieser Methode wird versucht, die aktuelle Pose des Betrachters (Position und Ausrichtung des Benutzers) mit `getViewerPose` zu erhalten (MDN, [2023f](#)). Wenn eine Pose vorhanden ist, wird angenommen, dass es nur eine Ansicht gibt. Die Methode `configureRendererForView` wird dann für diese Ansicht aufgerufen, um den Renderer entsprechend zu konfigurieren.

Listing 5.8: Implementierung der `startFrameHandler`-Funktion.

```
void startFrameHandler() {
    xrSession.requestAnimationFrame(allowInterop(
        frameHandler));
}

void frameHandler(double time, XRFrame xrFrame) {
    startFrameHandler();
    bindGraphicsFramebuffer(xrSession);

    final XRViewerPose? pose = getViewerPose(xrFrame);

    if (pose != null) {
        // In mobile AR, we only have one view.
        configureRendererForView(pose.views[0]);
    }
}
```

Nachdem eine Pose verfügbar ist, wird der Funktion `configureRendererForView` ein `XRFrame` namens `view` übergeben. Zuerst wird der sichtbare Bereich für diesen Frame mithilfe der Methode `getViewport(view)` der Schnittstelle `XRWebGLLayer` abgerufen, die ein Objekt vom Typ `XRViewport` zurückgibt. Ein `viewport` definiert, welcher Teil der Szene auf dem Bildschirm angezeigt wird und in welcher Größe (MDN, [2023p](#)). In diesem Fall werden vermehrt Module von der 3D-Grafikbibliothek Three.js verwendet. Zum Beispiel wird die Größe des Renderers mithilfe der Methode `setSize` festgelegt, indem die Breite und Höhe des `viewport` als Parameter übergeben werden. Anschließend wird die Methode `configureCameraForView` aufgerufen, um die Kamera basierend auf dem übergebenen `view` zu konfigurieren. Schließlich wird die Szene mit der konfigurierten Kamera gerendert. Die `render` Methode, die vom `rendererController` von der Three.js Implementierung bereitgestellt wird, nimmt zwei Parameter: die aktuelle Szene, die gerendert werden soll, und die Kamera, die zum Rendern der Szene verwendet wird.

Listing 5.9: Implementierung der `configureRendererForView`-Funktion.

```
void configureRendererForView(XRFrame view) {
    XRViewport viewport = baseLayer.getViewport(view);
    rendererController.setSize(viewport.width, viewport.
        height);

    configureCameraForView(view);

    rendererController.render(
        sceneController.scene, cameraController.
            perspectiveCamera);
}
```

Die Methode `configureCameraForView` nimmt auch einen Parameter namens `view`, der vom Typ `XRFrame` ist. Innerhalb der Methode wird die Transformation des `view` als Liste von `double`-Werten abgerufen. Diese Liste repräsentiert die 4x4 Transformationsmatrix des `view` (MDN, [2023h](#)). Zusätzlich wird auch die Projektionsmatrix des `view` abgerufen, die auf die zugrundeliegende Ansicht angewendet werden soll. Diese Matrix definiert, wie die 3D-Objekte auf eine 2D-Oberfläche projiziert werden und sollte verwendet werden, um eine Perspektive für alles in der Szene zu integrieren (MDN, [2023n](#)). Anschließend wird auf die aktuelle Instanz der Three.js Kamera zugegriffen, die eine `perspectiveCamera` repräsentiert. Danach wird die Transformationsmatrix des aktuellen `view` verwendet, um die interne Transformationsmatrix der Kamera mithilfe der Methode `fromArray` zu setzen. Der gleiche Vorgang wird mit der Projektionsmatrix der `view` durchgeführt, um die Projektionsmatrix der Kamera festzulegen. Abschließend wird die Methode `updateMatrixWorld(true)` aufgerufen, um das aktuelle Frame neu zu berechnen, ohne auf das Update durch den Renderer im nächsten Frame warten zu müssen (Three.js Dokumentation, [2023c](#)). Zusammenfassend ist diese Methode dazu verantwortlich, dass die Three.js Kamera mit den Daten eines `XRFrame` konfiguriert wird, sodass sie die 3D-Szene aus der Perspektive dieses Frames rendern kann. Da in AR/VR-Anwendungen die Kameraperspektive aufgrund der Bewegungen des Benutzers ständig aktualisiert werden muss, ist diese Methode besonders wichtig.

Listing 5.10: Implementierung der `configureCameraForView`-Funktion.

```
void configureCameraForView(XRFrame view) {
    List<double> matrix = view.transform.matrix;
    List<double> projectionMatrix = view.projectionMatrix;

    // Use the view's transform matrix and projection matrix
    // to configure the THREE.camera.
    final PerspectiveCamera camera = cameraController.
        perspectiveCamera;

    camera.matrix.fromArray(matrix);
    camera.projectionMatrix.fromArray(projectionMatrix);
    camera.updateMatrixWorld(true);
}
```

5.1.7 Implementierung von Three.js-Basismodulen

Während der Implementierung der WebXR API-Basismodule für AR-Funktionalitäten wurde versucht, die `three_dart`-Bibliothek mit der AR-Funktionalität zu integrieren, um das Anzeigen von 3D-Objekten in der AR-Umgebung zu ermöglichen. Die `three_dart`-Bibliothek repräsentiert eine Dart-Implementierung der renommierten 3D-Bibliothek Three.js und ist für die Webplattform von Flutter verfügbar. Nach erfolgreichen Tests dieser Bibliothek, die das Rendering eines 3D-Objekts in einer Flutter-Web-Anwendung involvierten, wurde jedoch festgestellt, dass eine Integration mit den spezifisch erstellten AR-Modulen des Plugins nicht realisierbar ist.

Dieses Problem tritt auf, da `three_dart` unabhängig von der Plattform auf die OpenGL API mittels `dart:ffi` zugreift (Wasabia, 2023). WebGL basiert zwar auf OpenGL-Technologie, jedoch war in diesem speziellen Fall der Zugriff auf den für die Erstellung eines WebGL-Renderers in Three.js erforderlichen WebGL-Kontext nicht möglich. Um dieses Problem zu lösen, wurde die Three.js-Bibliothek direkt mithilfe des JS-Packages implementiert. Um das AR-Plugin nutzen zu können, muss der Benutzer die Three.js-Bibliothek in der `index.html`-Datei des Flutter-Projekts mittels eines Script-Tag von einem CDN laden.

Zur Interaktion mit der Three.js Bibliothek werden verschiedene Klassen und Funktionen der Bibliothek in Dart mithilfe der JS-Bibliothek definiert, die als Wrapper für eine JavaScript-Klasse mit demselben Namen dienen. Für den Einsatz von Three.js sind Objekte wie `Renderer`, `Szene` und `Kamera` relevant, die miteinander interagieren, um ein 3D-Objekt im Browser darzustellen. Der folgende Code-Abschnitt definiert beispielsweise eine Dart-Klasse na-

mens **WebGLRenderer**. Diese Klasse ermöglicht über WebGL das Rendering der erstellten Szene im Browser und agiert als Wrapper zur Interaktion mit der **WebGLRenderer**-Klasse aus der Three.js-Bibliothek in einer Dart-Umgebung. Dabei implementiert **WebGLRenderer** alle Methoden und Eigenschaften der JavaScript-Klasse, die für den vorliegenden Anwendungsfall notwendig sind.

Listing 5.11: Darstellung der **WebGLRenderer**-Klasse.

```
@JS('WebGLRenderer')
class WebGLRenderer {
  external factory WebGLRenderer([Object options]);
  external html.CanvasElement get domElement;
  external void render(Scene scene, dynamic camera);
  external XR get xr;
  external void setSize(num width, num height);
  external set autoClear(bool value);
  external clear([bool color, bool depth, bool stencil]);
  external WebGL2RenderingContext getContext();
  external void dispose();
}
```

Die Klasse **RendererController** ist verantwortlich für die Initialisierung und Verwaltung eines WebGL-Renderers und eines Canvas-Elements. Sie enthält Variablen wie `texttt_gl`, die den aktuellen WebGL-Kontext speichern und über einen Getter für andere Klassen zugänglich sind. Beim Aufruf des Konstruktors **RendererController** wird die Methode `_initRenderer` automatisch bei der Instanziierung ausgeführt.

Listing 5.12: Implementierung der `RendererController`-Klasse.

```
class RendererController implements RendererOperations {
    late WebGLRenderer renderer;
    late Object? _gl;
    final String _createdViewId = 'canvas';

    Object? get glObject => _gl;

    final html.CanvasElement _canvas = html.CanvasElement()
        ..style.width = '100%'
        ..style.height = '100%';

    RendererController() {
        _initRenderer();
    }
}
```

Die Methode `_initRenderer`, die während des Konstruktionsprozesses der `RendererController`-Klasse aufgerufen wird, instanziiert die `WebGLRenderer`-Klasse. Zuerst wird der `_gl`-Variable der aktuelle WebGL-Kontext zugeordnet, welcher zum `_canvas`-Element gehört. Bei dem Aufruf der `getContext`-Methode werden als Parameter der gewünschte Kontexttyp und ein Optionsobjekt übergeben. Dadurch wird sichergestellt, dass der erstellte WebGL-Kontext mit WebXR kompatibel ist. Anschließend werden Konfigurationswerte in einem Optionsobjekt festgelegt, die der Renderer als Konstruktorparameter annimmt. Diese beinhalten den WebGL-Kontext und das dazugehörige Canvas-Element. Danach wird eine neue Instanz der Klasse `WebGLRenderer` erstellt und der Renderer konfiguriert. Es wird festgelegt, dass der Zeichenpuffer nicht automatisch zwischen Frames gelöscht wird, sodass mehrere Szenen in einem Frame gerendert werden können. Zusätzlich wird die WebXR-Funktionalität des Renderers aktiviert.

Listing 5.13: Implementierung der `_initRenderer`-Funktion.

```
void _initRenderer() {
    try {
        _gl = _canvas.getContext('webgl', {'xrCompatible':
            true});
        if (_gl == null) throw Exception('Failed to get WebGL
            context');

        final Object options = jsify({
            'context': _gl,
            'canvas': _canvas,
            'alpha': true,
            'preserveDrawingBuffer': true,
        });

        renderer = WebGLRenderer(options);
        renderer.autoClear = false;
        renderer.xr.enabled = true;
    } catch (e) {
        rethrow;
    }
}
```

Nachdem der `Renderer` initialisiert wurde, greift die Klasse `RendererController` auf zwei weitere Methoden zu, die in der Wrapper-Klasse der `WebGLRenderer`-Klasse definiert sind. Die Methode `render` ist dafür verantwortlich, eine 3D-Szene auf dem Bildschirm zu rendern und nimmt als Parameter die aktuelle Szene und die Kamera an. Die andere Methode `setSize` ändert die Größe des `Renderers` und nimmt als Parameter zwei Werte an, die die Breite und Höhe darstellen.

Listing 5.14: Darstellung der Methoden `render` und `setSize`.

```
@override
void render(Scene scene, PerspectiveCamera camera) =>
    renderer.render(scene, camera);

@override
void setSize(num width, num height) => renderer.setSize(
    width, height);
```

Zusätzlich zum `RendererController` wurden weitere Controller implemen-

tiert, darunter der `SceneController` und der `CameraController`. Im Folgenden wird exemplarisch der `SceneController` erläutert. Szenen in Three.js dienen dazu, festzulegen, welche Objekte wo gerendert werden sollen. Hierzu zählen nicht nur Objekte selbst, sondern auch Lichter und Kameras (Three.js Dokumentation, [2023d](#)). Der `SceneController` definiert mehrere Variablen, unter anderem eine `scene` Variable vom Typ `Scene`, die zu einem späteren Zeitpunkt initialisiert wird, und eine private Variable `_objects`, welche eine Liste aller aktiven Objekte in der Szene hält. Der Konstruktor dieser Klasse initiiert den Aufruf `createScene` Methode, die für die Initialisierung der Szene verantwortlich ist. Des Weiteren bietet die Klasse einen Setter, der es erlaubt, die Hintergrundfarbe der Szene festzulegen. Durch die `addElement` Methode lässt sich ein Objekt zur Szene hinzufügen, welches in der Folge in der AR-Umgebung angezeigt wird. Dieses Objekt wird zudem zur `_objects` Liste hinzugefügt. Zusammenfassend dient der `SceneController` zur Verwaltung einer 3D-Szene und ermöglicht das Hinzufügen von Objekten in die AR-Umgebung.

Listing 5.15: Implementierung der `SceneController`-Klasse.

```
class SceneController implements ThreeScene {
    late Scene scene;
    final List<Object> _objects = [];

    SceneController() {
        createScene();
    }

    List<Object> get activeObjects => _objects;

    set backgroundColor(Color color) => scene.background =
        color;

    @override
    void addElement(Object object) {
        scene.add(object);
        _objects.add(object);
    }

    @override
    void createScene() {
        scene = Scene();
    }
}
```

5.1.8 Integration der Basismodule

Nachdem beide Basismodule für das Plugin implementiert wurden, wurden diese Module in der Web-spezifischen Implementierung der Klasse namens `FlutterWebXrWeb` verwendet. Diese Klasse erbt von `FlutterWebXrPlatform` und wird benötigt, wenn ein Flutter-Plugin für die Webplattform entwickelt wird. Der folgende Code zeigt diese Klasse, die drei Controller-Objekte instanziiert. Dadurch werden Rendering, Szenenverwaltung und Kamerasteuerung für die Three.js Bibliothek für den Einsatz in der AR-Umgebung vorbereitet. Der Konstruktor der Klasse versucht, ein `XRController`-Objekt zu instanziiieren, indem er die zuvor erstellten Controller-Objekte als Parameter übergibt. Die `registerWith` Methode wird verwendet, um die `FlutterWebXrWeb`-Klasse als spezifische Implementierung für die Web-Plattform zu registrieren.

Listing 5.16: Implementierung der `FlutterWebXrWeb`-Klasse.

```
class FlutterWebXrWeb extends FlutterWebXrPlatform {
    late XRController xrController;

    final RendererController rendererController =
        RendererController();
    final SceneController sceneController = SceneController();
    final CameraController cameraController = CameraController
        ();

    FlutterWebXrWeb() {
        try {
            xrController =
                XRController(rendererController, sceneController,
                    cameraController);
        } catch (e) {
            throw Exception('Failed to instantiate XRController:
                $e');
        }
    }

    static void registerWith(Registrar registrar) {
        FlutterWebXrPlatform.instance = FlutterWebXrWeb();
    }
}
```

Die Klasse bietet Funktionen, um zu überprüfen, ob die aktuelle Plattform WebXR unterstützt, WebXR-Sitzungen zu starten und zu beenden sowie 3D-Objekte zur Szene hinzuzufügen. Bevor das Plugin verwendet werden kann, müssen bestimmte Voraussetzungen erfüllt sein, die von der Funktion

`isWebXrAvailable` überprüft werden. Diese Funktion nutzt den `xrController`, um zu überprüfen, ob WebXR auf dem aktuellen Gerät/Browser verfügbar ist. Wenn die API unterstützt wird, wird die Methode `checkRequirements` aufgerufen und das Ergebnis zurückgegeben. Diese Funktion überprüft, ob die aktuellen Browser- und Geräteeigenschaften kompatibel mit den Anforderungen für die Verwendung von WebXR sind und nutzt die Dart Bibliothek `html`, um Web-Funktionen im Browser zu nutzen. Die Funktion gibt `true` zurück, wenn beide Bedingungen erfüllt sind, ansonsten `false`.

Listing 5.17: Verschiedene Funktionen der `FlutterWebXrWeb`-Klasse.

```
bool isBrowserCompatible(String userAgent) =>
    userAgent.toLowerCase().contains('chrome') ||
    userAgent.toLowerCase().contains('edg');

bool isMobileDevice() => html.window.matchMedia("(max-width:
767px)").matches;

@override
bool isWebXrAvailable() {
    bool isWebXrAvailable = xrController.isWebXrSupported();

    if (!isWebXrAvailable) return false;
    return checkRequirements();
}

bool checkRequirements() {
    String userAgent = getPlatformVersion() ?? 'Unknown
platform version';

    bool isCompatible = isBrowserCompatible(userAgent) &&
        isMobileDevice();
    return isCompatible;
}
```

Wenn alle Voraussetzungen erfüllt sind, kann die Methode `createCube` aufgerufen werden. Diese Methode verwendet das implementierte Three.js-Modul, um einen dreidimensionalen Würfel zu erstellen. Dabei wird ein zeitverzögerter Aufruf initiiert, der nach einer festgelegten Dauer eine bestimmte Funktion aufruft. Dies ermöglicht es der Kamera, genügend Zeit zu haben, um die Umgebung zu erfassen. Erst nachdem die Umgebung vollständig erfasst wurde, kann das Objekt an der festgelegten Position angezeigt werden. Nach dem zeitverzögerten Aufruf wird ein `Mesh` Objekt erstellt, das ein 3D-Objekt repräsentiert. Es enthält die geometrischen Informationen eines Würfels und verschiedene Oberflächeneigenschaften. Nachdem das Objekt erstellt wurde, wer-

den die Position und Rotationswerte festgelegt, damit das Objekt später zentriert in der AR-Umgebung angezeigt wird. Schließlich wird das 3D-Objekt der aktuellen Szene, also der visuellen 3D-Umgebung, durch die Instanz des `SceneController` hinzugefügt.

Listing 5.18: Implementierung der `createCube`-Funktion.

```
@override
void createCube() {
    Future.delayed(const Duration(seconds: 2), () {
        final BoxGeometry geometry = BoxGeometry(0.2, 0.2,
            0.2);
        final List<MeshBasicMaterial> materials =
            createMaterials();

        final Mesh object = Mesh(geometry, materials);
        object.position.z = -1;
        object.rotation.x += 7;
        object.rotation.y += 7;

        sceneController.addElement(object);
    });
}
```

Die asynchrone Methode `startSession` der Klasse `FlutterWebXrWeb` ruft als erstes die Methode `requestSession` auf dem Objekt `xrController` asynchron auf und wartet auf deren Abschluss. Diese Funktion startet eine Anfrage, um eine XR-Session zu starten. Nachdem die XR-Session erfolgreich erstellt wurde und alle Vorbereitungen für die AR-Funktionalität abgeschlossen wurden, wird erneut die Methode `startFrameHandler` auf dem Objekt `xrController` aufgerufen. Diese Methode startet einen Frame-Handler, der für die Handhabung der Frames in einer XR-Session verantwortlich ist.

Listing 5.19: Implementierung der `startSession`-Funktion.

```
@Override
Future<void> startSession() async {
    try {
        await xrController.requestSession();
        xrController.startFrameHandler();
    } catch (e) {
        throw Exception('Failed to start xr session');
    }
}
```

Nach Abschluss des ersten Sprints und dem Starten des Plugins ist das folgende Ergebnis sichtbar [5.1](#).

Hier ist das AR DOM Overlay!

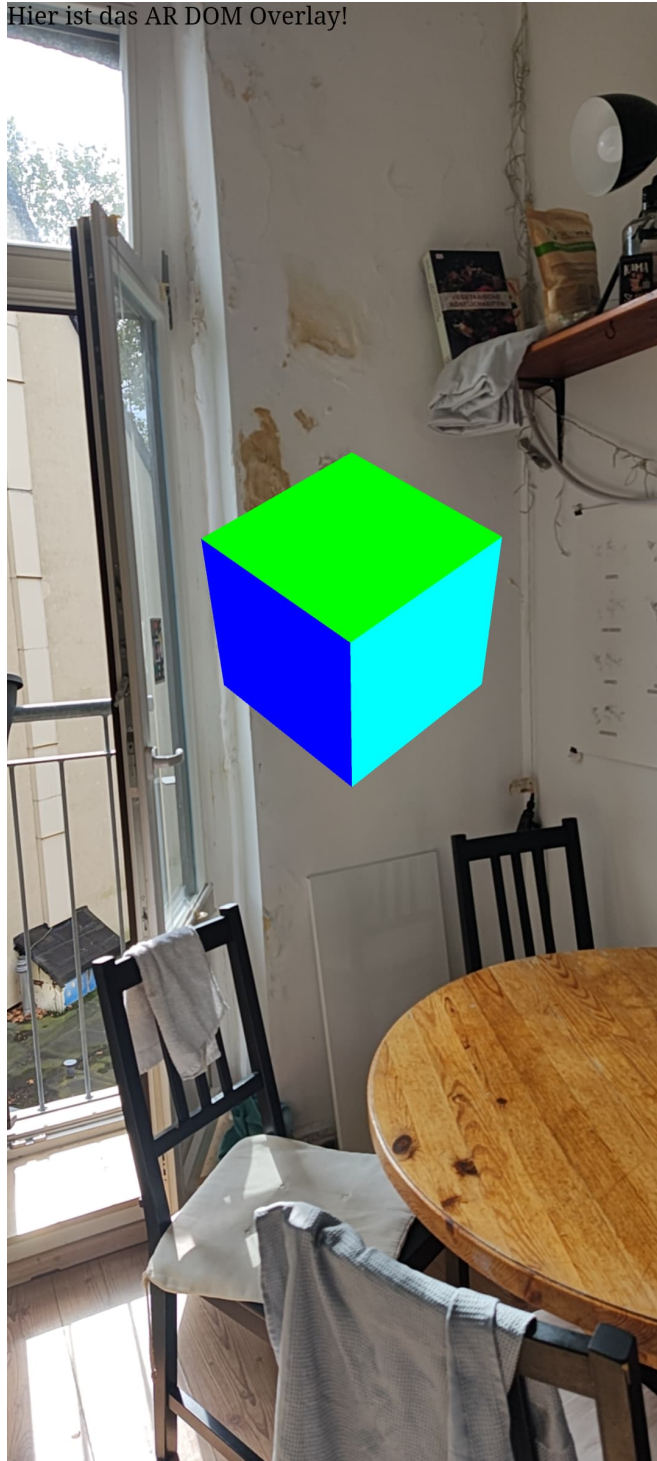


Abbildung 5.1: Erster Prototyp des Plugins.

5.2 Sprint 2

Nachdem ein erster Prototyp erfolgreich entwickelt wurde, sollten im zweiten Entwicklungssprint Optimierungen am Prototyp vorgenommen werden. Zudem wurde das Ziel definiert, das Plugin für den Paketmanager zu veröffentlichen und für die Flutter-Community öffentlich zugänglich zu machen. Um diese Ziele zu erreichen, wurden skalierbare Three JS und Web XR Module entwickelt und die gesamte Plugin-Architektur angepasst. Darüber hinaus wurden Widgets für die Darstellung und Auswahl von 3D-Objekten in Flutter Web erstellt und die AR-Funktionalitäten erweitert. Dadurch ist es nun möglich, mehrere 3D-Objekttypen in der AR-Umgebung zu importieren und darzustellen. Für die Veröffentlichung wurden API-Dokumentation und eine README-Datei erstellt, die die verfügbaren Funktionen und ihre Anwendung ausführlich beschreiben. Eine Anleitung zur Integration und Nutzung des Plugins wird ebenfalls bereitgestellt.

5.2.1 Optimierung der Module und Architektur

Um eine skalierbare Architektur zu gewährleisten, wurden die jeweilige Anwendungslogik für die Three.js Bibliothek und die WebXR-API in eigene Module ausgelagert und die Plugin-API verbessert. Ein Modul ist unterteilt in Interfaces, Interops und Models. Das Submodul Interfaces ist dafür zuständig, dass Methoden definiert werden, die von anderen Klassen implementiert werden müssen, wenn das Interface als Erweiterung genutzt wird. Folgend ist das **RendererOperations** Interface dargestellt, das vom **RendererController** implementiert wird. Dieses Interface stellt allgemeine Methoden bereit, die der **RendererController** enthalten muss.

Listing 5.20: Darstellung des **RendererOperations** Interface.

```
abstract class RendererOperations {  
    void initRenderer(html.CanvasElement canvas);  
    void render(Scene scene, PerspectiveCamera camera);  
    void animate(Scene scene, PerspectiveCamera camera,  
        dynamic object,  
        double xValue, double yValue);  
    void setSize(num width, num height);  
}
```

Das Interop-Submodul dient als Brücke zwischen Flutter/Dart und JavaScript. Es ermöglicht es Dart-Code, JavaScript-Bibliotheken und -Funktionen direkt aufzurufen und mit ihnen zu interagieren. In diesem Submodul werden alle Dart-Abbildungen (Wrapper) von JavaScript-Klassen in passende Kategorien eingeordnet, sodass diese einfach erweiterbar sind. Als Beispiel ist der folgenden

Codeabschnitt zu betrachten. In diesem werden verschiedene Geometrien wie die `BoxGeometry`, die `ConeGeometry` und die `ExtrudeGeometry` der Three.js Bibliothek in einem weiteren Submodul zusammengefasst. Eine `Geometry` in Three.js bezeichnet die Struktur oder Form eines 3D-Objekts.

Listing 5.21: Implementierung von verschiedenen `Geometry`-Objekten in Three.js

```
@JS('BoxGeometry')
class BoxGeometry {
  external BoxGeometry(num width, num height, num depth);
}

@JS('ConeGeometry')
class ConeGeometry {
  external ConeGeometry(num radius, num height, num
    radialSegments);
}

@JS('ExtrudeGeometry')
class ExtrudeGeometry {
  external ExtrudeGeometry(dynamic shape, Object?
    extrudeSettings);
}
```

In dem dritten Submodul befinden sich die Controller, die als zentrale Steuereinheit innerhalb des Plugins fungieren. Sie sind verantwortlich für die allgemeine Koordinaten und Steuerung von Prozessen. Ein Controller soll zudem zwischen Modell und Ansicht (oder anderen Systemkomponenten) vermitteln und dient oft als Bindeglied zwischen verschiedenen Teilen einer Softwareanwendung. Er ist auch verantwortlich für Initialisierung und Konfiguration, Verarbeitung von Benutzereingaben oder Anforderungen und die Fehlerbehandlung.

Die Plugin-API, die Entwicklern zur Verfügung steht, wurde durch die Spezifizierung und Erweiterung seiner Funktionalität optimiert. Da dieses Plugin in Zukunft weiterentwickelt und skalierbar sein soll, spielt die Gesamtarchitektur des Plugins eine entscheidende Rolle. Laut Singh und Hassan (Singh & Hassan, 2015) sollten die Programmier-, Wartungs- und Supportphasen des Softwareentwicklungszyklus ohne Stress oder Probleme verlaufen, wenn das Architekturdiesign den entsprechenden Regeln entspricht. Diese Regeln werden von den SOLID-Designprinzipien definiert, die als Standard in der Softwareentwicklung dienen und eine gute Richtlinie für Designentscheidungen bieten, um die Wiederverwendbarkeit, Wartbarkeit und Skalierbarkeit der Software zu verbessern. Daher wurden diese Designprinzipien zur Optimierung der Plugin-Architektur

gewählt. Durch die Aufteilung der verschiedenen Verantwortlichkeiten in einzelne Module wird ein modulares Design gefördert, das somit den Sinn des Single Responsibility Prinzip (SRP) erfüllt. Dies gewährleistet Flexibilität und Skalierbarkeit. Auch das Open/Closed Prinzip (OCP) wird durch die Optimierung erfüllt. Es besagt, dass Softwareeinheiten erweiterbar sein sollten, ohne den vorhandenen Code zu ändern. Dadurch können zusätzliche AR-Funktionalitäten hinzugefügt werden, ohne den bestehenden Code zu ändern. Durch die Einhaltung der SOLID-Prinzipien kann ein System effizienter und einfacher optimiert werden.

5.2.2 Entwicklung von Widgets

Ein weiterer Schritt der Optimierung beinhaltete die Erstellung von Widgets für die Darstellung und Auswahl von 3D-Objekten in Flutter Web. Dazu sollte die AR-Funktionalität erweitert werden, um verschiedene 3D-Objekttypen zu importieren und in der AR-Umgebung anzuzeigen. Es wurden zwei Widgets entwickelt, um verfügbare 3D-Objekte in der Flutter Web App darzustellen und auszuwählen.

Ein Widget namens `ObjectGrid` zeigt eine Rasteransicht von 3D-Modellen an. Dieses Widget nimmt als Parameter eine Liste von Objekten mit dem Typ `ThreeModel` an, welche die 3D-Objekte repräsentieren. In der Build-Methode wird die Bildschirmgröße aufgerufen, um die Abmessungen jedes Gitterelements zu berechnen. Dies stellt sicher, dass die Gitterelemente sich gut an verschiedene Bildschirmgrößen anpassen. In einem `Column` Widget wird eine Reihe angezeigt, die ein Icon und Titel enthält. Darunter wird ein `GridView` Widget dargestellt, das die 3D-Modelle zeigt, die in der übergebenen Liste enthalten sind. Jedes Modell wird in einem `ObjectContainer`-Widget dargestellt, wobei jedes Gitter zwei Elemente pro Zeile hat.

Listing 5.22: Darstellung des ObjectGrid Widget.

```

class ObjectGrid extends StatefulWidget {
  // List of 3D models to be displayed in the grid.
  final List<ThreeModel> models;

  const ObjectGrid({super.key, required this.models});

  @override
  State<ObjectGrid> createState() => _ObjectGridState();
}

class _ObjectGridState extends State<ObjectGrid> {
  @override
  Widget build(BuildContext context) {
    // Getting the screen size
    final Size size = MediaQuery.of(context).size;

    // Calculating the height and width for each grid item
    final double itemHeight = size.height / 2.5;
    final double itemWidth = size.width / 2;

    return Padding(
      padding: const EdgeInsets.all(16.0),
      child: Column(
        children: [
          // Header Row with an icon and title
          Row(
            children: [
              const Icon(Icons.grid_view),
              const SizedBox(width: 12.5),
              Text("3D-Objects", style: Theme.of(context).
                textTheme.titleLarge),
            ],
          ),
          const SizedBox(height: 12),
          // Grid displaying 3D models
          Expanded(
            child: GridView.count(
              crossAxisCount: 2,
              childAspectRatio: (itemWidth / itemHeight),
              children: widget.models.map((model) {
                return ObjectContainer(model: model);
              }).toList(),
            ),
          ),
        ],
      ),
    );
  }
}

```

Das zweite Widget, genannt **ThreeScene**, integriert 3D-Modelle in eine Flutter Anwendung mithilfe der Bibliothek Three.js. Es bietet die Möglichkeit, entweder ein direkt bereitgestelltes 3D-Objekt oder ein Modell von einem bestimmten Pfad zu verwenden. Jedes dieser Modelle benötigt eine eindeutige ID, die **createdViewId** genannt wird, um in der Flutter-Anwendung dargestellt zu werden. Es wird ein HTML-Canvas-Element erstellt, auf dem die 3D-Szene gerendert wird. Zusätzlich gibt es verschiedene Controller, die unterschiedliche Aspekte der 3D-Szene steuern. Zum Beispiel die Steuerung der Kamera und das Rendering des Modells. Sobald dieses Widget initialisiert ist, werden eine Reihe von Vorbereitungen getroffen. Das Canvas wird für die Verwendung im Plattformansichtssystem von Flutter registriert. Die Kamera wird in der 3D-Szene positioniert und das 3D-Modell, abhängig davon, ob es direkt oder über einen Pfad bereitgestellt wurde, wird geladen und zur Szene hinzugefügt. Wenn weder ein Objekt noch ein Pfad angegeben ist, wird ein Fehler ausgelöst. Nachdem das 3D-Modell geladen oder hinzugefügt wurde, wird eine Animation gestartet, die das Modell in der Szene bewegt. Im endgültigen Schritt, in der **build**-Methode, wird das gerenderte 3D-Modell über ein **HtmlElementView**-Widget in der Flutter-Anwendung sichtbar gemacht, indem die zuvor definierte eindeutige ID verwendet wird.

Listing 5.23: Darstellung des **ObjectGrid** Widget.

```
class ThreeScene extends StatefulWidget {
  /// A unique ID associated with the view created for
  /// displaying the 3D model.
  final String createdViewId;

  /// An optional mesh object to be displayed within the
  /// scene.
  final Mesh? object;

  /// An optional path to a 3D model that should be loaded
  /// and displayed.
  final String? path;

  /// Constructs an instance of 'ThreeScene'.
  const ThreeScene(
    {super.key, required this.createdViewId, this.object,
    this.path});

  @override
  State<ThreeScene> createState() => _ThreeSceneState();
}

class _ThreeSceneState extends State<ThreeScene> {
```

```

/// HTML canvas element where the 3D scene will be
    rendered.
final html.CanvasElement canvas = html.CanvasElement();

/// Controller responsible for rendering the 3D scene.
late RendererController rendererController;

/// Controller for managing the 3D scene.
final SceneController sceneController = SceneController();

/// Controller for handling camera-related operations.
final CameraController cameraController =
    CameraController(matrixAutoUpdate: true);

/// Controller to assist with loading 3D models.
final LoaderController loaderController = LoaderController
    ();

@override
void initState() {
    super.initState();
    rendererController = RendererController(canvas: canvas);

    _registerCanvas();
    _configureCamera();
    _loadAndAddElementToScene();
}

/// Registers the canvas to be used within Flutter's
    platform view system.
void _registerCanvas() {
    // Register div as a view and ensure the div is ready
    // before we try to use it
    // ignore: undefined_prefixed_name
    ui.platformViewRegistry
        .registerViewFactory(widget.createdViewId, (int
            viewId) => canvas);
}

/// Configures the camera's position within the 3D scene.
void _configureCamera() {
    cameraController.setPosition(null, null, 6);
}

/// If an object is provided, adds it directly to the
    scene. Otherwise, loads a 3D model from the specified
    path
/// and then adds it to the scene.
Future<void> _loadAndAddElementToScene() async {
    if (widget.object == null && widget.path == null) {

```

```

        throw Exception('You can only pass either an object or
            a path');
    }

    if (widget.object != null) {
        sceneController.addElement(widget.object!);
        _startAnimation(widget.object, 0.04, 0.04);
        return;
    }

    if (widget.path != null) {
        try {
            final model = await loaderController.loadModel(
                widget.path!);

            sceneController.addElement(model);
            _startAnimation(model, 0, 0.04);
        } catch (error) {
            domLog('Error loading model: $error');
        }
    }
}

/// Starts the animation of the object in the scene.
void _startAnimation(dynamic model, double x, double y) =>
    rendererController.animate(sceneController.scene,
        cameraController.perspectiveCamera, model, x, y);

@override
Widget build(BuildContext context) {
    return HtmlElementView(viewType: widget.createdViewId);
}
}

```

Der Prototyp sollte durch die Optimierung außerdem die Funktionalitäten erhalten, verschiedene 3D-Objekttypen in der AR-Umgebung darzustellen. Darüber hinaus sollten die Methoden innerhalb des Plugins erweitert werden, um die direkte Erstellung von einfachen 3D-Modelle wie einem Würfel, einem Kegel oder einem Herz mithilfe von Three.js zu ermöglichen.

5.2.3 3D-Modellformate und Three.js

Three.js ermöglicht grundsätzlich den Import von 3D-Modellen. Es gibt jedoch unzählige Dateiformate für 3D-Modelle zur Auswahl. Jedes dieser Formate hat unterschiedliche Zwecke, Funktionen und Komplexitäten. Obwohl die Three.js Bibliothek viele Dateiformate unterstützt, kann die Wahl des richtigen Formats und Workflows Zeit und Frustration sparen. Einige Formate sind ineffizient für

Echtzeiterlebnisse oder werden derzeit nicht vollständig unterstützt. In der Dokumentation der Bibliothek wird empfohlen, das glTF-Format (GL Transmission Format) zu verwenden, da es darauf ausgelegt ist, Ressourcen zur Laufzeit bereitzustellen. Dadurch ist es kompakt in der Übertragung und schnell beim Laden (Three.js Dokumentation, [2023b](#)).

Um das glTF-Format in Three.js zu verwenden, muss der `GLTFLoader` als Add-On explizit importiert werden. Die Klasse `LoaderController` bietet einen Mechanismus zur Steuerung des Ladevorgangs von 3D-Modellen. In dieser Klasse wird eine Instanz des `GLTFLoader` erstellt, der speziell für das Laden von GLTF-Dateien entwickelt wurde. Die Methode `loadModel` nimmt den Pfad einer GLTF-Datei als Parameter und initiiert den Ladevorgang. Um asynchronen Code besser zu handhaben, wird ein `Completer`-Objekt verwendet. Dieses Objekt gibt ein zukünftiges Ergebnis oder einen Fehler zurück. Beim Aufruf von `loader.load` werden drei Callback-Funktionen übergeben: Eine für den erfolgreichen Abschluss des Ladevorgangs, eine für den Fortschritt des Ladevorgangs und eine für den Fall, dass ein Fehler auftritt. Je nach Ergebnis gibt die Instanz des `Completers` entweder das vollständig geladene Modell oder einen Fehler zurück. Zusätzlich gibt es einen Fortschritts-Callback, der den aktuellen Stand des Ladevorgangs angibt. Abschließend gibt die Methode das Future-Objekt des `Completer` zurück, das über den Erfolg oder Misserfolg des Ladevorgangs informiert.

Listing 5.24: Darstellung der LoaderController-Klasse

```
class LoaderController {
    final GLTFLoader loader = GLTFLoader();

    Future<dynamic> loadModel(String path) {
        final Completer<dynamic> completer = Completer<dynamic>
            >();

        loader.load(
            path,
            allowInterop((gltf) {
                completer.complete(gltf.scene);
            }),
            allowInterop((progress) {
                // Optional: progress callback
            }),
            allowInterop((error) {
                completer.completeError(error);
            }),
        );

        return completer.future;
    }
}
```

Weitere Verbesserungen des Prototyps umfassen die Implementierung neuer Methoden, die die direkte Erstellung einfacher 3D-Modelle ermöglichen. Nach der Optimierung stehen Funktionen zur Verfügung, um einen Würfel mit einer bestimmten Seitenlänge und bestimmten Oberflächen zu erstellen. Darüber hinaus gibt es eine Funktion, um einen Kegel basierend auf einem bestimmten Radius, einer Höhe und mehreren Oberflächen zu erstellen. Es gibt auch eine Funktion zur Erstellung eines Herz-Objekts, das festgelegte Proportionen enthält und mit einer bestimmten Farbe erstellt werden kann. Die folgende Abbildung zeigt einige der Optimierungen des Prototyps. Verschiedene 3D-Objekte werden mithilfe des `ObjectGrid`-Widget und des `ThreeScene`-Widget angezeigt [5.2](#). Darunter befinden sich auch Objekte, die direkt mit der Three.js-Bibliothek erstellt werden können, sowie ein GLTF-Modell, das mit dem GLTF-Loader importiert und geladen wurde [5.3](#).

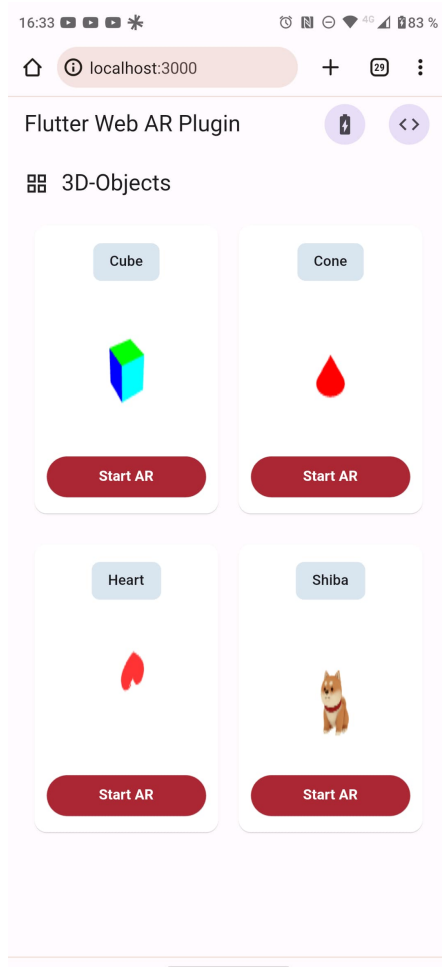


Abbildung 5.2: ObjectGrid-Widget in der Verwendung.

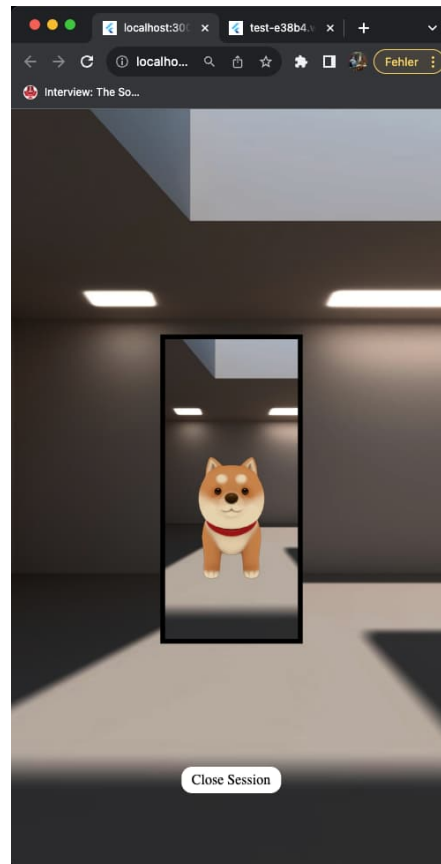


Abbildung 5.3: AR-Ansicht einer glTF-Datei mithilfe des GLTFLoader.

5.2.4 Veröffentlichung des Plugins

Nach Abschluss der Optimierungen war ein erster experimenteller Prototyp des Plugins fertig, der einige der definierten Anforderungen erfüllte. Der nächste Schritt bestand darin, das Plugin zu veröffentlichen und es der Flutter-Community zugänglich zu machen, um die Lücke im Flutter-Plugin-Ökosystem zu schließen. In Flutter können Plugins über den Paketmanager Pub installiert werden, der bereits ausführlicher erläutert wurde. Entwickler können ihre Plugins auf der pub.dev Plattform veröffentlichen und nach anderen Plugins suchen, die sie in

ihren eigenen Projekten verwenden können. Um ein Plugin zu veröffentlichen, müssen bestimmte Voraussetzungen erfüllt sein, wie zum Beispiel die Erstellung einer API-Dokumentation und einer README-Datei.

Für die API-Dokumentation wurde die Inline-Code-Dokumentation mithilfe des KI-Tools ChatGPT von OpenAI für die Hauptklassen des Plugins erstellt. Eine API-Referenzdokumentation wird automatisch aus dem Dart-Quellcode mit dem Befehl `dart doc` generiert. Standardmäßig handelt es sich bei den Dokumentationsdateien um statische HTML-Dateien, die im Verzeichnis `doc/api` platziert werden (Dart Dokumentation, 2023b). Die Dart-Dokumentation beschreibt weitere Schritte, die vor der Veröffentlichung unternommen werden müssen. Es ist wichtig, das pubspec-Format und die Konventionen für das Paket-Layout einzuhalten. Diese Konventionen müssen eingehalten werden, damit andere Benutzer das Plugin verwenden können. Zusätzlich gibt es einige Vorschläge, um es Benutzern leichter zu machen, das Plugin zu verstehen und damit zu arbeiten. Im Allgemeinen zielt der Paketmanager darauf ab, sicherzustellen, dass das Plugin optimal in das Flutter-Ökosystem integriert wird. Es gibt einige wichtige Dateien, die Pub verwendet, um die Plugin-Seite auf der Plattform pub.dev zu erstellen. Diese Dateien umfassen die README-Datei, die den Hauptinhalt auf der Plugin-Seite enthält und Anweisungen zur Integration und Verwendung des Plugins bereitstellt. Die CHANGELOG-Datei wird ebenfalls in einem Tab auf der Plugin-Seite angezeigt und beschreibt die Änderungen der neuen Version. Die pubspec-Datei ist die dritte wichtige Datei, die zum Beispiel den Namen, die Beschreibung und das Code-Repository des Plugins darstellt. Wenn alle Vorbereitungen erledigt wurden, kann mit dem Befehl `dart pub publish` das Plugin auf der Plattform pub.dev veröffentlicht werden (Dart Dokumentation, 2023d). Die folgende Abbildung 5.4 zeigt das veröffentlichte Plugin auf der Plattform pub.dev und die dazugehörigen Informationen, die von den genannten erforderlichen Dateien bereitgestellt werden. Das Plugin kann hier gefunden werden.

flutter_web_xr 0.0.2

Published 4 days ago • @felixguenther.com [Dart 3 compatible](#)

[SDK](#) [FLUTTER](#) [PLATFORM](#) [WEB](#)

1

[Readme](#) [Changelog](#) [Example](#) [Installing](#) [Versions](#) [Scores](#) [Admin](#) [Activity log](#)

Flutter Web XR Plugin

This plugin is in beta. Use it with caution.

This Flutter plugin enables developers to create augmented reality (AR) experiences in their Flutter web applications. With this plugin, you can integrate interactive AR content such as 3D models, animations, and visual effects into your Flutter web projects.

Getting Started

Add the Flutter package to your project by running:

```
flutter pub add flutter_web_xr
```

Add the following Script tags to your index.html file in the flutter web directory to import the Three.js library and the GLTFLoader module:

1 130 0%
LIKES PUB POINTS POPULARITY

Publisher

@felixguenther.com

Metadata

Plugin for creating Augmented Reality experiences for Flutter Web.

[Repository \(GitHub\)](#)
[View/report issues](#)

Topics

[#augmented-reality](#)
[#flutter-web](#) [#webxr](#)

Documentation

[API reference](#)

Abbildung 5.4: Flutter Web XR Plugin auf pub.dev.

Herausforderungen

In dem folgenden Abschnitt werden die Herausforderungen und Hürden erläutert, die bei der Entwicklung eines Augmented Reality Plugins für Flutter Web aufgetreten sind. Dabei wird auch darauf eingegangen, wie diese Hürden gelöst werden konnten.

Vor Beginn der Entwicklung des Plugins bestand die Wahrscheinlichkeit, dass während der Entwicklung Herausforderungen und Hindernisse auftreten könnten. Die WebXR API birgt ein gewisses Risiko, da sie sich noch in der Entwicklung befindet und noch nicht die experimentelle Phase verlassen hat. Die API entwickelt sich stetig weiter und hat einige Einschränkungen hinsichtlich der Geräte und Browser-Kompatibilität (MDN Web Docs, 2023). Diese Fakten verdeutlichen anfängliche Herausforderungen bei der Verwendung der WebXR API. Trotz der Stabilität weist Flutter Web immer noch einige Probleme auf. Insbesondere stellt das Fehlen von Web APIs für die Entwicklung dieses Plugins ein Hindernis dar und die damit verbundene Integration einer Web API mithilfe der JS-Bibliothek. Weitere Herausforderungen umfassen Probleme mit der Browserkompatibilität und gelegentlich längere Ladezeiten. Die folgende Liste stellt alle Herausforderungen chronologisch dar.

- Das `Navigator`-Objekt der `html`-Bibliothek von Flutter unterstützte ursprünglich nicht die WebXR API, wodurch diese nicht zugänglich war. Dieses Problem wurde behoben, indem die API mithilfe der JS-Bibliothek zugänglich gemacht wurde.
- Die Einarbeitung mit der JS-Bibliothek hat viel Zeit in Anspruch genommen, da es an komplexeren Anwendungsbeispielen mangelt und Sonderfälle auftreten können. Zudem können Probleme bei der Einbindung externer JavaScript-Bibliotheken auftreten. Die Annotation `@staticInterop` der JS-Bibliothek, die statische Interoperabilität zwischen Dart und JavaScript ermöglicht, hat ebenfalls Probleme verursacht. Dabei war es nicht möglich, auf die entsprechende Erweiterung zuzugreifen, die die Eigenschaften und Methoden der statischen Klasse enthält. Aus diesem Grund wurde die `@staticInterop` Annotation nicht implementiert, da sie noch experimentell ist.
- Eine weitere Herausforderung bestand darin, das AR-Modul, das die WebXR API enthält, mit der `three_dart` Bibliothek zu integrieren, die ursprünglich verwendet wurde. Diese Integration ist fehlgeschlagen, da `three_dart` unabhängig von der verwendeten Plattform auf die OpenGL API mittels `dart:ffi` zugreift und einen WebGL-Renderer erstellt, der anstelle

eines WebGL-Kontexts einen OpenGL-Kontext besitzt (Wasabia, 2023). Dieser Kontext ist entscheidend für Integration der WebXR API. Um dieses Problem zu lösen, wurde direkt die Three.js-Bibliothek mithilfe des JS-Packages implementiert, um den richtigen Kontext abrufen zu können. Um das AR-Plugin nutzen zu können, muss der Benutzer die Three.js-Bibliothek in der `index.html`-Datei des Flutter-Projekts mittels eines Script-Tag von einem CDN laden.

- Der Import mehrerer Module über ein Content Delivery Network (CDN) in der `index.html`-Datei verursachte allgemeine Probleme, wie zum Beispiel das Einbinden der Three.js-Bibliothek und Add-ons wie dem GLTF Loader. Dabei trat der Fehler auf, dass das CDN keine Modul-Auflösung unterstützte und somit die Bibliotheken nicht verwendbar waren. Diese Hürde wurde durch die Verwendung eines CDNs mit verbesserter Modul-Auflösung gelöst.
- Bei der Implementierung des GLTF Loader, um 3D-Modelle mit dem glTF-Dateiformat zu importieren und darzustellen, sind während lokaler Tests keine Probleme aufgetreten. Jedoch tritt im Produktionsmodus ein unerklärlicher Fehler auf. Dieser Fehler besagt, dass eine unbekannte Variable keine Funktion ist. Es wird vermutet, dass dies auf die verwendete Three-Version zurückzuführen ist, die über das CDN importiert wird. Es wurden verschiedene CDN-Anbieter und Three-Versionen getestet, jedoch konnte bisher keine Lösung für den Fehler in der Produktionsumgebung gefunden werden.
- In Bezug auf das DOM-Overlay gibt es einen Fehler, wenn die Anwendung den AR-Modus startet. Es scheint, dass die CSS-Styles nicht korrekt angewendet werden und das Overlay-Element möglicherweise CSS-Styles aus einer tieferen Ebene annimmt. Die Funktionalität kann jedoch gewährleistet werden, sodass das DOM-Overlay verwendet werden kann, um den AR-Modus zu verlassen.
- Eine grundlegende Hürde sind Einschränkungen in Bezug auf die Geräte- und Browser-Kompatibilität. Die Verwendung der WebXR-API beschränkt das Plugin auf das Android-Betriebssystem in Kombination mit dem Google Chrome-Browser. Allerdings wurde bei ersten Tests festgestellt, dass es auch bei dieser Kombination zu Problemen kommen kann. Beim Starten der AR-Session wurde teilweise der Bildschirm des Geräts schwarz oder es wurde lediglich die Kamera geöffnet, ohne den AR-Inhalt anzuzeigen. Diese Tests wurden auf Samsung- und Huawei-Geräten durchgeführt.

Fazit und Ausblick

Im folgenden Kapitel werden die erzielten Forschungsergebnisse dargelegt, die sich mit der Entwicklung und Konzeption eines Augmented Reality Plugins für Flutter Web befassen. Das Kernziel war es, ein funktionales AR-Plugin für einen spezifischen Anwendungsfall zu schaffen. Dieses Ziel wurde erreicht. Es bleibt, die gewonnenen Erkenntnisse zu reflektieren und Herausforderungen zu reflektieren. Mit einem Blick in die Zukunft kann das Potenzial und die Möglichkeiten für weiterführende Arbeiten und Forschungen in diesem Bereich erkundet werden. Dadurch wird nicht nur die Relevanz der erzielten Ergebnisse betont, sondern es wird auch ein Ausgangspunkt für die Weiterentwicklung des AR-Plugins geschaffen.

7.1 Fazit

Im Fazit werden die Kernergebnisse zusammengefasst, insbesondere in Bezug auf das identifizierte Problem und die Zielsetzung der Arbeit. Gleichzeitig wird die Bedeutung der Ergebnisse im Kontext des Flutter-Ökosystems bewertet und eventuelle Limitationen oder Schwächen der Arbeit behandelt.

7.1.1 Zusammenfassung der Hauptergebnisse

Das in dieser Arbeit behandelte Problem wurde erfolgreich gelöst. Durch die Entwicklung und Veröffentlichung steht jetzt ein Plugin im Flutter-Ökosystem zur Verfügung, das Augmented Reality Erlebnisse für Flutter Web ermöglicht. Das Plugin ist öffentlich zugänglich und kann mit dem Pub-Paketmanager in einer Flutter-Webanwendung installiert und implementiert werden, um eine Flutter-Webanwendung mit AR-Funktionalität zu erweitern. Das Ergebnis dieser Arbeit erweitert die Grenzen des Möglichen und verbessert das Potenzial von Flutter als Werkzeug für die Erstellung interaktiver, immersiver Web-Erlebnisse. Neben den bereits verfügbaren AR-Plugins für IOS und Android vervollständigt dieses Plugin die Reihe von Flutter-Plugins für Augmented Reality für Webanwendungen. Darüber hinaus erfordert das Ziel dieser Forschung auch, dass das Plugin einen spezifischen Anwendungsfall unterstützt, der in eine bestehende Dating-Anwendung integriert wird. In diesem Anwendungsfall können Benutzer virtuelle Geschenke an ihre Matches senden. Diese Geschenke werden mithilfe von Augmented Reality in der Umgebung der Benutzer angezeigt. Dieses spezifische Ziel wurde weitgehend erreicht und wird anhand der definierten Anforderungen für das Plugin gemessen.

Vier von fünf der funktionalen Anforderungen konnten erfüllt werden. Benutzer können aus einer Liste von virtuellen Geschenken auswählen. Für die

3D-Inhaltsverwaltung können einfache 3D-Inhalte (z.B Würfel, Kegel, Herz) direkt innerhalb der Anwendung erstellt werden. Dazu bietet das Plugin eine Importfunktion, die das Hinzufügen von 3D-Modellen in glTF-Dateiformaten unterstützt. Benutzer haben die Möglichkeit, ein virtuelles Geschenk in ihrer Umgebung mithilfe der AR-Funktionalität zu betrachten, jedoch fehlt noch die Möglichkeit, das virtuelle Geschenk gezielt in der Umgebung zu platzieren. Die Überprüfung der AR-Kompatibilität wird ebenfalls durch das Plugin gewährleistet, da es automatisch die Verfügbarkeit und Kompatibilität von AR-Funktionen auf dem jeweiligen Gerät erkennt. Eine funktionale Anforderung, die nicht umgesetzt werden konnte, ist die Interaktion mit AR-Objekten. Benutzer sollten die Möglichkeit haben, mit virtuellen Geschenken zu interagieren, beispielsweise um sie zu öffnen, ihre Größe zu ändern oder sie in der AR-Umgebung zu verschieben.

Bezüglich der nicht-funktionalen Anforderungen wurden sieben von acht erfüllt. Die AR-Darstellung funktioniert reibungslos und ohne Verzögerungen. Die Benutzerfreundlichkeit kann durch eine intuitive und leicht verständliche Benutzeroberfläche für die AR-Funktion gewährleistet werden. Darüber hinaus ist das Plugin in der Lage, eine wachsende Anzahl von Benutzern und Geschenken zu unterstützen. Die Anforderung der Erweiterbarkeit wird durch eine modulare Architektur erfüllt, die zukünftige Erweiterungen und Integrationen mit anderen Systemen erleichtert. Die Three.js-Bibliothek kümmert sich um die hochwertige Darstellung von 3D-Objekten und erfüllt somit die ästhetische Anforderung. Es steht umfassendes und verständliches Dokumentationsmaterial zur Verfügung, um das Plugin zu nutzen, was den Benutzern einen schnellen Einstieg ermöglicht. Regelmäßige Updates und Support sind ebenfalls gewährleistet, da das Plugin in Zukunft weiterentwickelt werden soll. Eine nicht-funktionale Anforderung, die nicht erfüllt werden konnte, war die Kompatibilität. Das Plugin funktioniert nur eingeschränkt auf Smartphones und Webbrowsern.

Der spezifische Anwendungsfall konnte größtenteils umgesetzt werden, da das Plugin die meisten Anforderungen erfüllen konnte. Dadurch musste der definierte Fallback-Fall nicht verwendet werden, und das Problem konnte mit dem ursprünglichen Plan gelöst werden. Die Anforderungen führten jedoch zu einigen technischen Herausforderungen, die viel Zeit in Anspruch nahmen und oft den Entwicklungsprozess verzögerten.

- Das Plugin ist hier verfügbar: pub.dev/packages/flutter_web_xr
- Github-Repository: github.com/Felix2019/flutter-web-ar-plugin

7.1.2 Bewertung der Ergebnisse

Die erzielten Ergebnisse dieser Forschungsarbeit stellen einen bedeutenden Fortschritt im Flutter-Ökosystem dar. Das entwickelte AR-Plugin für Flutter Web schließt eine Lücke im Ökosystem, die zuvor bestand. Dadurch werden nicht nur

die Möglichkeiten für Entwickler verbessert, die mit Flutter arbeiten, sondern auch die Attraktivität von Flutter als Plattform für zukunftsorientierte Webanwendungen. Diese Arbeit zeigt, dass Flutter auch im Web-Bereich Potenzial hat und komplexere Funktionalitäten wie Augmented Reality integrieren kann. Je umfangreicher die Funktionalität von Flutter Web wird, desto mehr kann es mit beliebten Web-Frameworks wie React, Angular oder Vue.js konkurrieren. Darüber hinaus ist die Integration von AR in Webanwendungen ein aufstrebendes Feld, und diese Forschungsarbeit positioniert Flutter als wichtigen Akteur in diesem Bereich.

Allerdings gibt es auch einige Limitationen und Schwächen, die berücksichtigt werden sollten. Wie bereits erwähnt, funktioniert das Plugin nur eingeschränkt auf Smartphones und Webbrowsern. Das könnte die Akzeptanz und Verbreitung des Plugins in der Flutter-Community einschränken, da eine universelle Kompatibilität oft ein Schlüsselkriterium ist. Dazu kommt die fehlende Möglichkeit, mit AR-Objekten zu interagieren. Derzeit fehlen grundlegende Interaktionsfunktionen, um den Benutzern ein vollständig immersives Erlebnis zu bieten. Die Relevanz und Nützlichkeit des Plugins wird stark von der kontinuierlichen Weiterentwicklung und dem Hinzufügen neuer Funktionen abhängen, obwohl das Plugin eine solide Grundlage bietet. Abschließend kann gesagt werden, dass diese Forschungsarbeit einen wertvollen Beitrag zum Flutter-Ökosystem und zur Web-Entwicklung im Allgemeinen leistet. Trotz einiger Limitationen bietet sie eine solide Grundlage, auf der zukünftige Fortschritte aufbauen können, um Flutter als eine konkurrenzfähige Plattform für AR-Webanwendungen zu etablieren.

7.2 Ausblick

Das Cross-Plattform Framework Flutter wird kontinuierlich weiterentwickelt, darunter fällt auch die Web-Plattform des Frameworks. Hier entstehen zunehmend Plugins, die Funktionalitäten für Flutter Web bereitstellen. Für den Bereich der erweiterten Realität (AR) hat dieses Plugin die Grundlage für Flutter Web geschaffen und bietet AR-Grundfunktionen. Das Plugin wurde mit dem Ziel entwickelt, aktiv von der Flutter-Community weiterentwickelt zu werden, um es zu verbessern und den Bedürfnissen der Entwickler anzupassen. Es basiert auf Technologien wie der WebXR API und der JS-Bibliothek, die experimentell sind und kontinuierlich weiterentwickelt werden. Daher ist eine ständige Weiterentwicklung des Plugins notwendig, um von den neuesten Entwicklungen und Verbesserungen der verwendeten Technologien zu profitieren und somit das Flutter-Ökosystem weiter zu bereichern. Die Hauptaufgabe für die Zukunft besteht darin, die Funktionalität des Plugin auszubauen. Dazu gehören die direkte Erstellung weiterer 3D-Modelle, der Import weiterer 3D-Modelle mit verschiedenen Dateiformaten, Interaktionsmöglichkeiten mit 3D-Objekten und die Optimierung der Benutzererfahrung hinsichtlich Performance und Stabilität. Für diese neuen Funktionen muss auch die Three.js Bibliothek enger integriert werden,

um die Qualität und Vielfalt der AR-Erfahrungen zu erhöhen. Die Ergebnisse dieser Arbeit zeigen die Möglichkeiten und den Nutzen eines solchen AR-Plugins und eröffnen neue Perspektiven für die weitere Forschung und Entwicklung in diesem Bereich. Es gibt mehrere Bereiche, in denen weitere Forschung nützlich wäre. Erstens, die Untersuchung der Interaktion zwischen Dart und JavaScript, um die Performance und Integration von AR in Flutter Web zu optimieren. Zweitens, die Erforschung der WebXR API in Bezug auf ihre Entwicklung und Kompatibilität mit verschiedenen Geräten und Browsern. Die Durchführung dieser Forschungsarbeit war eine lehrreiche Erfahrung, die mit vielen Teilerfolgen und Herausforderungen verbunden war. Sie bot die Gelegenheit, tief in die Welt von Flutter Web und AR einzutauchen und die Möglichkeiten und Grenzen dieser Technologien zu erkennen. Diese Arbeit dient als Sprungbrett für weitere Forschungen und Entwicklungen in diesem Bereich.

Abbildungsverzeichnis

1.1	Marktumsatz der Mobile-AR weltweit von 2021 bis 2026. (ARtil- lery Intelligence, 2023)	6
1.2	Can I use WebXR Device API (Can I Use, 2023)	7
2.1	Suchfunktion der Plattform pub.dev (Pub, 2023f)	13
3.1	WebXR API Emulator in Google Chrome (Mozilla Mixed Reality, 2023)	16
3.2	Architectural overview: platform channels. (Flutter Dokumenta- tion, 2023e)	22
5.1	Erster Prototyp des Plugins.	53
5.2	ObjectGrid-Widget in der Verwendung.	63
5.3	AR-Ansicht einer glTF-Datei mithilfe des GLTFLoader.	63
5.4	Flutter Web XR Plugin auf pub.dev.	65

Listings

3.1 FlutterWebXr Klasse.	18
3.2 FlutterWebXrPlatform Klasse.	20
3.3 MethodChannelFlutterWebXr Klasse.	23
3.4 MainActivity Klasse.	24
3.5 FlutterWebXrWeb Klasse.	25
5.1 getBatteryLevel()-Methode.	34
5.2 Veranschaulichung der JS Bibliothek anhand eines Beispiels.	34
5.3 Darstellung der getBatteryLevel-Funktion.	35
5.4 Zugriff auf das XRSystem.	38
5.5 Die Klasse XRController.	39
5.6 Implementierung der requestSession-Methode.	40
5.7 Implementierung der setupXRSession-Funktion.	41
5.8 Implementierung der startFrameHandler-Funktion.	42
5.9 Implementierung der configureRendererForView-Funktion.	43
5.10 Implementierung der configureCameraForView-Funktion.	44
5.11 Darstellung der WebGLRenderer-Klasse.	45
5.12 Implementierung der RendererController-Klasse.	46
5.13 Implementierung der _initRenderer-Funktion.	47
5.14 Darstellung der Methoden render und setSize.	47
5.15 Implementierung der SceneController-Klasse.	48
5.16 Implementierung der FlutterWebXrWeb-Klasse.	49
5.17 Verschiedene Funktionen der FlutterWebXrWeb-Klasse.	50
5.18 Implementierung der createCube-Funktion.	51
5.19 Implementierung der startSession-Funktion.	52
5.20 Darstellung des RendererOperations Interface.	54
5.21 Implementierung von verschiedenen Geometry-Objekten in Three.js	55
5.22 Darstellung des ObjectGrid Widget.	57
5.23 Darstellung des ObjectGrid Widget.	58
5.24 Darstellung der LoaderController-Klasse	62

Literatur

- Alsop, T. (2023). *Augmented reality (AR) - statistics & facts* [Abgerufen am 7. Oktober 2023]. <https://www.statista.com/topics/3286/augmented-reality-ar/#topicOverview>
- Artillery Intelligence. (2023). *Mobile augmented reality (AR) market revenue worldwide from 2021 to 2026* [Abgerufen am 25. Juli 2023]. <https://www.statista.com/statistics/282453/mobile-augmented-reality-market-size/>
- Can I Use. (2023). *Can I use WebXR Device API* [Abgerufen am 10. August 2023]. <https://caniuse.com/?search=webxr>
- Dart API. (2023a). *dart:html library* [Abgerufen am 1. September 2023]. <https://api.dart.dev/stable/3.1.0/dart-html/dart-html-library.html>
- Dart API. (2023b). *promiseToFuture;T; function* [Abgerufen am 25. September 2023]. <https://api.dart.dev/stable/2.8.1/dart-js-util/promiseToFuture.html>
- Dart Dokumentation. (2023a). *C interop using dart:ffi* [Abgerufen am 3. September 2023]. <https://dart.dev/guides/libraries/c-interop>
- Dart Dokumentation. (2023b). *dart doc* [Abgerufen am 20. Oktober 2023]. <https://dart.dev/tools/dart-doc>
- Dart Dokumentation. (2023c). *JavaScript interoperability* [Abgerufen am 3. September 2023]. <https://dart.dev/web/js-interop>
- Dart Dokumentation. (2023d). *Publishing packages* [Abgerufen am 20. Oktober 2023]. <https://dart.dev/tools/pub/publishing>
- Dart Team. (2023). *Dart Programming Language Specification* [Abgerufen am 6. September 2023]. <https://spec.dart.dev/DartLangSpecDraft.pdf#External%20Functions>
- Flutter API. (2023a). *Class MethodChannel* [Abgerufen am 31. August 2023]. <https://api.flutter.dev/javadoc/io/flutter/plugin/common/MethodChannel.html>
- Flutter API. (2023b). *promiseToFuture;T; function* [Abgerufen am 6. September 2023]. <https://api.flutter.dev/flutter/dart-js-util/promiseToFuture.html>

- Flutter Dokumentation. (2023a). *Developing packages plugins* [Abgerufen am 3. September 2023]. <https://docs.flutter.dev/packages-and-plugins/developing-packages>
- Flutter Dokumentation. (2023b). *Flutter Favorite program* [Abgerufen am 27. Juli 2023]. <https://docs.flutter.dev/packages-and-plugins/favorites>
- Flutter Dokumentation. (2023c). *Support for WebAssembly (Wasm)* [Abgerufen am 8. Oktober 2023]. <https://docs.flutter.dev/platform-integration/web/wasm>
- Flutter Dokumentation. (2023d). *Web FAQ* [Abgerufen am 12. September 2023]. <https://docs.flutter.dev/platform-integration/web/faq#which-web-browsers-are-supported-by-flutter>
- Flutter Dokumentation. (2023e). *Writing custom platform-specific code* [Abgerufen am 31. August 2023]. <https://docs.flutter.dev/platform-integration/platform-channels?tab=type-mappings-swift-tab#publish>
- Google. (2023). *Build apps for any screen* [Abgerufen am 8. Oktober 2023]. <https://flutter.dev>
- Günthner, F. (2023). *Entwicklung einer produktionsreifen Progressive Web App mit dem Cross-Platform Framework Flutter* [Abgerufen am 25. Juli 2023]. <https://drive.google.com/file/d/1g0p2afaJWmL5J9aKgsMmmmHsgIOlZmA/view?usp=sharing>
- Hasnany, M. (2021). *Flutter web support hits the stable milestone* [Abgerufen am 27. Juli 2023]. <https://medium.com/flutter/flutter-web-support-hits-the-stable-milestone-d6b84e83b425>
- JetBrains. (2021). *Cross-platform mobile frameworks used by software developers worldwide from 2019 to 2021* [Abgerufen am 25. Juli 2023]. <https://www.statista.com/statistics/869224/worldwide-software-developer-working-hours/>
- Khronos Group. (2023). *WebGL* [Abgerufen am 8. September 2023]. <https://www.khronos.org/webgl/>
- Klessascheck, M. (2023). *Funktionale Anforderungen versus nicht-funktionale Anforderungen* [Abgerufen am 12. September 2023]. <https://www.johner-institut.de/blog/iec-62304-medizinische-software/funktionale-und-nicht-funktionale-anforderungen>
- MDN. (2023a). *Navigator* [Abgerufen am 3. September 2023]. <https://developer.mozilla.org/en-US/docs/Web/API/Navigator?retiredLocale=de>

- MDN. (2023b). *Package management basics* [Abgerufen am 3. August 2023]. https://developer.mozilla.org/en-US/docs/Learn/Tools_and_testing/Understanding_client-side_tools/Package_management
- MDN. (2023c). *Progressive Enhancement* [Abgerufen am 6. September 2023]. https://developer.mozilla.org/en-US/docs/Glossary/Progressive_Enhancement?retiredLocale=de
- MDN. (2023d). *WebGL: 2D and 3D graphics for the web* [Abgerufen am 21. August 2023]. https://developer.mozilla.org/en-US/docs/Web/API/WebGL_API?retiredLocale=de
- MDN. (2023e). *WebXR Device API* [Abgerufen am 20. August 2023]. https://developer.mozilla.org/en-US/docs/Web/API/WebXR_Device_API
- MDN. (2023f). *XRFrame: getViewerPose() method* [Abgerufen am 26. September 2023]. <https://developer.mozilla.org/en-US/docs/Web/API/XRFrame/getViewerPose>
- MDN. (2023g). *XRReferenceSpace* [Abgerufen am 26. September 2023]. <https://developer.mozilla.org/en-US/docs/Web/API/XRReferenceSpace>
- MDN. (2023h). *XR rigid transform: matrix property* [Abgerufen am 27. September 2023]. <https://developer.mozilla.org/en-US/docs/Web/API/XRRigidTransform/matrix>
- MDN. (2023i). *XRSession* [Abgerufen am 25. September 2023]. <https://developer.mozilla.org/en-US/docs/Web/API/XRSession>
- MDN. (2023j). *XRSession: requestAnimationFrame() method* [Abgerufen am 26. September 2023]. <https://developer.mozilla.org/en-US/docs/Web/API/XRSession/requestAnimationFrame>
- MDN. (2023k). *XRSession: updateRenderState() method* [Abgerufen am 25. September 2023]. <https://developer.mozilla.org/en-US/docs/Web/API/XRSession/updateRenderState#>
- MDN. (2023l). *XRSystem: isSessionSupported() method* [Abgerufen am 25. September 2023]. <https://developer.mozilla.org/en-US/docs/Web/API/XRSystem/isSessionSupported>
- MDN. (2023m). *XRSystem: requestSession() method* [Abgerufen am 25. September 2023]. https://developer.mozilla.org/en-US/docs/Web/API/XRSystem/requestSession#requesting_a_session_with_a_dom_overlay

- MDN. (2023n). *XRView: projectionMatrix property* [Abgerufen am 27. September 2023]. <https://developer.mozilla.org/en-US/docs/Web/API/XRView/projectionMatrix>
- MDN. (2023o). *XRWebGLLayer* [Abgerufen am 25. September 2023]. <https://developer.mozilla.org/en-US/docs/Web/API/XRWebGLLayer>
- MDN. (2023p). *XRWebGLLayer: getViewport() method* [Abgerufen am 26. September 2023]. <https://developer.mozilla.org/en-US/docs/Web/API/XRWebGLLayer/getViewport>
- Microsoft. (2023). *Was ist Augmented Reality oder AR?* [Abgerufen am 25. Juli 2023]. <https://dynamics.microsoft.com/de-de/mixed-reality/guides/what-is-augmented-reality-ar/>
- Mozilla Mixed Reality. (2023). *WebXR API Emulator* [Abgerufen am 21. August 2023]. <https://chrome.google.com/webstore/detail/webxr-api-emulator/mjddjgeghkdijejnciaefnkjmkafnnje?hl=de>
- Ochuko, K. (2019). *ARKit And ARCore Everything You Need To Know* [Abgerufen am 5. September 2023]. <https://medium.com/@kevwe/arkit-and-arcore-everything-you-need-to-know-bea25b36a21c>
- Pub. (2023a). *flutter_unity_widget* [Abgerufen am 7. September 2023]. https://pub.dev/packages/flutter_unity_widget
- Pub. (2023b). *JS* [Abgerufen am 20. August 2023]. <https://pub.dev/packages/js>
- Pub. (2023c). *The official package repository for Dart and Flutter apps* [Abgerufen am 25. Juli 2023]. <https://pub.dev>
- Pub. (2023d). *Package scores & pub points* [Abgerufen am 26. Juli 2023]. <https://pub.dev/help/scoring>
- Pub. (2023e). *plugin_platform_interface* [Abgerufen am 27. August 2023]. https://pub.dev/packages/plugin_platform_interface
- Pub. (2023f). *Pub Paketsuche* [Abgerufen am 26. Oktober 2023]. <https://pub.dev/packages?q=flutter+web+xr>
- Pub. (2023g). *Search* [Abgerufen am 26. Juli 2023]. <https://pub.dev/help/search>
- Pub. (2023h). *three_dart* [Abgerufen am 7. September 2023]. https://pub.dev/packages/three_dart


- Singh, H., & Hassan, S. I. (2015). Effect of SOLID Design Principles on Quality of Software: An Empirical Assessment [Abgerufen am 22. August 2023]. <https://www.ijser.org/researchpaper/Effect-of-SOLID-Design-Principles-on-Quality-of-Software-An-Empirical-Assessment.pdf>
- Statista Market Insights. (2023). *AR Software* [Abgerufen am 7. Oktober 2023]. <https://www.statista.com/outlook/amo/ar-vr/ar-software/worldwide#revenue>
- Terkelsen, H. (2019). *How to Write a Flutter Web Plugin* [Abgerufen am 26. August 2023]. <https://medium.com/flutter/how-to-write-a-flutter-web-plugin-5e26c689ea1>
- Three.js Dokumentation. (2023a). *Fundamentals* [Abgerufen am 21. August 2023]. <https://threejs.org/manual/#en/fundamentals>
- Three.js Dokumentation. (2023b). *Loading 3D models* [Abgerufen am 19. Oktober 2023]. <https://threejs.org/docs/#manual/en/introduction/Loading-3D-models>
- Three.js Dokumentation. (2023c). *Object3D* [Abgerufen am 27. September 2023]. <https://threejs.org/docs/#api/en/core/Object3D.updateMatrixWorld>
- Three.js Dokumentation. (2023d). *Scene* [Abgerufen am 27. September 2023]. <https://threejs.org/docs/#api/en/scenes/Scene>
- Unity Technologies. (2023). *Machen Sie mehr mit Unity* [Abgerufen am 7. September 2023]. <https://unity.com/de>
- Wangoo, A. (2023). *Enabling smooth communication between javascript and dart in Flutter* [Abgerufen am 28. August 2023]. <https://www.droidcon.com/2023/08/07/enabling-smooth-communication-between-javascript-and-dart-in-flutter/>
- Wasabia. (2023). *Flutter GL* [Abgerufen am 8. September 2023]. https://github.com/wasabia/flutter_gl

Eidesstattliche Erklärung

Ich, **Felix Günthner**, geboren am **07.03.1998** in **Freiburg**, erkläre hiermit eidesstattlich, dass ich die vorliegende Arbeit mit dem Titel „**Augmented Reality für Flutter Web: Entwicklung eines Plugins anhand eines Praxisbeispiels**“ selbstständig und ohne unerlaubte fremde Hilfe verfasst habe.

Ich habe keine anderen als die angegebenen Quellen und Hilfsmittel benutzt und alle wörtlich oder inhaltlich übernommenen Stellen als solche kenntlich gemacht. Die Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Köln, 28.10.2023


.....
Unterschrift