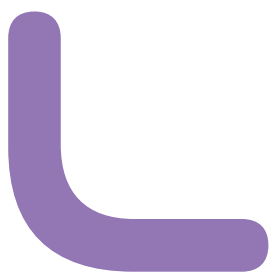


Interdisziplinäre Bachelorarbeit von  
Jacob Wolyniec und Christine Koppe

Interaktive  
Applikation  
zum  
Darstellen  
von lokalen  
Artefakten



**B**

Bachelorarbeit

## **Interaktive Applikation zum Darstellen von lokalen Artefakten**



Hochschule für Künste Bremen  
Studiengang Digitale Medien

Erstprüfer: Nuri Ovüc  
Zweitprüfer: Udo Frese

**Christine Koppe**

Böttcherei 55  
28844 Weyhe  
Matrikelnummer: 32839



Universität Bremen  
Studiengang Digitale Medien

Erstprüfer: Udo Frese  
Zweitprüfer: Nuri Ovüc

**Jacob Oskar Wolyniec**

Fährer Flur 13e  
28755 Bremen  
Matrikelnummer: 4400483

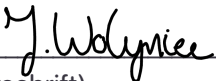
Abgabedatum: 01.06.2021

## Erklärung


Ich versichere, den Bachelor-Report oder den von mir zu verantwortenden Teil einer Gruppenarbeit\*) ohne fremde Hilfe angefertigt zu haben. Ich habe keine anderen als die angegebenen Quellen und Hilfsmittel benutzt. Alle Stellen, die wörtlich oder sinngemäß aus Veröffentlichungen entnommen sind, sind als solche kenntlich gemacht.

\*) Bei einer Gruppenarbeit muss die individuelle Leistung deutlich abgrenzbar und bewertbar sein und den Anforderungen entsprechen.

Bremen den, 01.06.2021

  
\_\_\_\_\_  
(Unterschrift)

Bremen den, 01.06.2021

  
\_\_\_\_\_  
(Unterschrift)

## Vorwort



Diese Bachelorarbeit ist ein interdisziplinäres Projekt, das von Studierenden des Studiengangs Digitale Medien an der Universität Bremen und Hochschule für Künste Bremen, namentlich Christine Koppe (HfK) und Jacob Oskar Wolyniec (Uni Bremen), bearbeitet wurde. Dieses Dokument wurde aufgrund besonderer Prüfungs-umstände bilingual in den Sprachen Deutsch und Englisch verfasst. Die individuelle Leistung ist jeweils deutlich namentlich in der Fußzeile erwähnt. Zusammen erarbeitete Themen, sowie die Texte von Christine liegen in Deutscher Sprache vor, Jacobs Hauptanteil in Englisch. Dabei wurde der amerikanische Sprachstil gewählt. Einzig das Wort Artefakt wurde in der britischen Schreibweise "artefact" anstatt der amerikanischen Variante "artifact" verwendet, da diese sich sonst von der Schreibweise in der praktischen Arbeit unterscheiden würde. Weiterführend bezeichnen wir vorwiegend Elke Munderloh und Nadine Scheffler als "Stakeholder" des Projektes.

Bei der Bachelorarbeit haben wir darauf geachtet unsere Texte gendergerecht zu formulieren. Wir haben die Doppelpunkt Schreibweise gewählt und versucht sie konsequent anzuwenden. Wenn uns dies nicht immer gelingt, wollen wir in keinen Fall ein Geschlecht ausschließen oder diskriminieren.

# Zusammenfassung

Die vorliegende interdisziplinäre Bachelorarbeit behandelt das Thema **“Interaktive Applikation zum Darstellen von lokalen Artefakten”**. Bei dem Projekt haben wir gemeinsam als Team, Christine und Jacob, einen Piloten (Prototyp) einer mobilen Applikation entwickelt. Zusammen haben wir die Rahmenbedingungen geschaffen, jedoch haben wir uns im späteren Verlauf in zwei Aufgabenbereiche aufgeteilt: Christine war für die Konzeption und das Design zuständig und Jacob für die Entwicklung und technische Umsetzung der Applikation. Von Dezember 2020 bis Mai 2021 beschäftigten wir uns mit der Konzeption, dem Design und der Implementierung der Applikation und dem Schreiben der Bachelorarbeit. Das Bürgerhaus Obervielnd hatte nach einem Treffen das Interesse einer App geäußert. Ziel der App soll es sein, dass längst geschehene Ereignisse nochmal aufgearbeitet und lebhaft dargestellt werden. Alle weiteren Bedingungen waren der kreativen Freiheit ausgesetzt. Aufgrund unserer vorherigen Projekten an der HfK und Uni Bremen zum Thema “AR” und “Interaktionsdesign” wurden wir motiviert, uns dem Thema anzunehmen. Wir entwarfen einen Zeitplan und erarbeiteten mit Strategien zur Ideenfindung eine Produktvision bis hin zur einer Informationsarchitektur. Aus der Designperspektive hat Christine ein detailliertes Screen-Design mit App Icon und passenden Styleguide entwickelt. Dabei hat Sie Gestaltgesetze beachtet und ein eigenen Stil für Farbe, Icons und Typografie kreiert. Jacob hat die technische Umsetzung der Applikation in der Unity Engine übernommen und das gemeinsame Konzept umgesetzt. Dabei hat er nach den Vorbildern gängiger Praktiken und Methoden, die im Zusammenhang mit der Unity Engine verwendet werden, gearbeitet und die UI-Screens gebaut und Skripte für die Funktionalität programmiert.

Das Resultat des Projektes zeigt eine erste funktionierende Applikation, in der die wichtigsten Grundfunktionen des Konzeptes umgesetzt wurden. Gemäß dem Konzept wurden der Applikation Möglichkeiten für die Weiterentwicklung und Skalierbarkeit der Inhalte gegeben.

**Keywords:** AR, Interaktionsdesign, App, UI, UX, Unity, ScriptableObject, Artefakte

# Abkürzungsverzeichnis

<b>App</b>	Applikation / Anwendungssoftware
<b>BGO</b>	Bürgerhaus Gemeinschaftszentrum Obervieland
<b>CSS</b>	Cascading Style Sheets
<b>d.h.</b>	das heißt (deutsch: Zum Beispiel)
<b>e.g.</b>	for example
<b>GPS</b>	Global Positioning System (deutsch: Globales Positionsbestimmungssystem)
<b>GUI</b>	graphical user interface (deutsch: grafische Nutzerschnittstelle)
<b>HEX</b>	Hexadezimal-Code (Farbwert)
<b>i.e.</b>	that is
<b>otf</b>	OpenType Font (digitale Schrift)
<b>PNG</b>	Portable Network Graphics
<b>px</b>	Pixel (deutsch: Bildpunkt)
<b>RGB</b>	Rot, Grün, Blau (Farbraum)
<b>SO</b>	ScriptableObject
<b>UI</b>	User Interface (deutsch: Benutzeroberfläche)
<b>UX</b>	User Experience (deutsch: Nutzererfahrung)
<b>z.B.</b>	zum Beispiel

**G**

# Inhaltsverzeichnis

## Das Projekt

Einleitung	3
Quartier AR Pilot	5
Die Hintergründe	6
Demografischer Wandel	8
Zeitlicher Ablauf	9

## Verwandte Arbeiten und Inspirationen

Pokemon GO	13
Civilisations AR	15
Snapchat	17
iPhone AR Apple	19

## Mobile Applikation

Mobile Apps	23
Augmented Reality	24
Interaktion im Alltag	28
Physisches Umfeld	29

## Interdisziplinäres Team

Rollen im Team	33
Aufgabenbereiche	35

## Konzeption

Design Prozess	39
Produktvision	41
Methoden	44
Anforderungsmanagement	47
Personas	49
Navigationsplan	51



Christine	<b>User Interface</b>	
	Styleguide	55
	Touch-Gesten	57
	Elementgrößen	58
	Wireframes	59
	<b>Visuelles Design</b>	
	Farben	65
	Typographie	67
	Icons	69
	Elemente	71
	<b>App Design</b>	
	App Icon	75
	Prototype	77
Produktmerkmale	97	
Klick-Dummy	99	
Jacob	<b>Unity Engine</b>	
	Why did we choose Unity?	103
	Unity Editor	104
	Scripting in Unity	107
	<b>Architecture</b>	
	ArtefactSO	115
	LocationSO	116
	UI System	117
	Map and Locations	124
	ImageTrackingScene	131
	Gallery	139
General Information	141	

**Conclusion**

Reflexion, Problems, Technical Outlook \_\_\_\_\_ 145

**Integration und Fazit**

Integration of Content \_\_\_\_\_ 151

Verwaltung der Inhalte vom BGO \_\_\_\_\_ 154

Working Conditions \_\_\_\_\_ 155

Fazit \_\_\_\_\_ 158

Literaturverzeichnis \_\_\_\_\_ 159

Danksagung \_\_\_\_\_ 164

Anhang \_\_\_\_\_ 166

K

# Das Projekt

**Einleitung**  
**Quartier AR Pilot**  
**Die Hintergründe**  
**Demografischer Wandel**  
**Zeitlicher Ablauf**

# Einleitung

Als Studierende der Digitalen Medien beschäftigen wir uns seit mehreren Jahren mit Computern, Technologien, Medien, Konzepten, Theorie und Kunst. Kurse fordern von uns eine kritische Auseinandersetzung mit weltbewegenden Themen und die praktische Umsetzung unserer eigenen Ideen. Wir haben uns in dieser Bachelorarbeit zur Herausforderung gemacht, eine App zu entwickeln, die für jung und alt, zugeschnitten auf den Lebensort, eine Interaktion erzeugt. Mit dem Thema "Interaktive Applikation zum Darstellen von lokalen Artefakten" befassen wir uns mit den Objekten der Stadtgeschichte, die wir immersiv auf dem Bildschirm wiederaufleben lassen wollen. Wir haben beschlossen unsere Bachelorarbeit als Gruppenprojekt zu verwirklichen, um als interdisziplinäres Team ein rundes Konzept aufstellen zu können, das möglichst alle wichtigen Aspekte der App, sowohl gestalterisch als auch technisch, berücksichtigt. Für die Entwicklung eines solchen Projektes werden verschiedene Fähigkeiten benötigt, die wir in das Projekt einbringen:

**Christine Koppe** (26 Jahre alt) 8. Semester studiert an der Hochschule für Künste den Studiengang Digitale Medien Bachelor. In der Studienzeit hat sie sich auf das gestalten von Benutzeroberflächen spezialisiert. Experimentell erforscht sie auch die VR/AR Technologien und hat Kunst im öffentlichen Raum erstellt. Ihre Schwerpunkte in der Bachelorarbeit ist die konzeptionelle Arbeit, Projektmanagement und das ausgestalten des Produktes. Schon in früheren Projekten haben Jacob und Christine zusammengearbeitet und konnten ihre Fähigkeiten immer sehr gut ergänzen.

**Jacob Wolyniec** (23 Jahre alt) studiert im 10. Semester Digitale Medien an der Universität Bremen. Im Laufe seines Studiums hat er eine starke Verbindung zur HfK aufgebaut, da er dort seine Kreativität in den Projekten der Kurse frei ausleben und mit neuen Technologien experimentieren kann. Im Laufe der Zeit hat er sich ein breites Spektrum an Fähigkeiten erarbeitet, das von AR- und VR- über WebGL-Anwendungen bis hin zu Photogrammetrie und point cloud Animationen reicht. Die Unity Engine ist dabei seine favorisierte Entwicklungsumgebung, da sie für die meisten seiner Projekte die perfekte Grundlage bietet. Schon seit 2017 begleitet ihn die Game Engine durch sein Studium. Er ist für den Großteil der technischen Umsetzung und Architektur des Projektes zuständig.

Der Ursprung des Projektes einer interaktiven Applikation zu erstellen kam von Elke Munderloh, die Leiterin des Begegnungszentrums im Bürgerhaus Obervieland und Nadine Scheffler, die Projektleiterin von Kattenturm Quartier - gemeinnützige Gesellschaft mbH. Sie kamen mit Ihrem Anliegen zur Hochschule für Künste Bremen

und der Universität Bremen und wollte die Entwicklung einer Smartphone Applikation in die Hände der Studierenden der Hochschul-Institutionen geben, um kooperativ Ihre Idee verwirklichen zu lassen. Letztendlich wurden wir auf dieses Projekt aufmerksam gemacht und bekundeten unser Interesse, da wir bereits schon an experimentellen App-Projekten gearbeitet haben und uns die Idee mit ihren Hintergründen zusagten.

Bei einem ersten Kennenlernen im Café wurde uns näher gebracht, dass das Ziel der Projektes ist, die Stadtteilgeschichte Obervielands abzubilden. Genauere Vorstellungen einer Umsetzung gab es zu dem Zeitpunkt nicht, weshalb sie froh war mit uns gemeinsam ein konkretes Produkt Ihrer Idee entsprechend zu erstellen.

Elke begann uns von Ihren täglichen Konfrontationen mit der Vergangenheit des Stadtteils durch den Kontakt zu den Senioren:innen im Bürgerhaus zu erzählen und wie sehr diese in Erinnerungen an vergangene Zeiten schwelgen. Dabei wurde ihr bewusst, wie wichtig es sein könnte, diese Zeit noch einmal, wenn auch nur künstlich, aufleben zu lassen. Daraufhin entschloss Sie sich dazu die alten Archive des Stadtteils digital aufarbeiten zu lassen und zeitgemäß darzustellen. Die Bereitschaft der Menschen im Bürgerhaus sich an der Umsetzung des Projektes zu beteiligen war groß, indem sie zum Beispiel helfen die Archive nach brauchbaren Artefakten zu durchsuchen.

→ Als "**(lokale) Artefakte**" bezeichnen wir Objekte aus den Archiven des Stadtteils, aber auch immaterielle Güter wie beispielsweise Geschichten und Erzählungen von Zeitzeugen, der Stadtteilgeschichte.

Über die Digitalisierung der Inhalte wird im Kapitel "Integration" näher eingegangen. Überall in Kattenturm gibt es historisch wichtige Orte, die die Geschichte des Stadtteils und die des Bürgerhauses erzählen. Um ein entscheidendes Begeisterungsmerkmal zu schaffen, wollen wir die App mit zeitgemäßer Technologie erweitern. Dabei kam die Idee, dass man eine interaktive Landkarte des Stadtteils stellt, auf der es die Artefakte in Form von AR-Events zu entdecken gibt. Sofern das Material der Stadtteilgeschichte aus dem Archiv dies hergibt, wollen wir die Artefakte nicht nur auf Bilder zeigen, sondern zusätzlich als detaillierte 3D Modelle. Durch Augmented Reality (auf deutsch: Erweiterte Realität) lassen sich diese Modelle digital in die Welt projizieren - eine Technologie auf die wir im Späteren genauer eingehen werden.

Damit sollen die Nutzer:innen motiviert werden, an die Originalschauplätze zurückzukehren, um dort die AR-Events an lokalen Merkmalen, zum Beispiel bildhaften Darstellungen wie Mosaik, in der echten Umgebung auszulösen, sodass jeder die

Geschichte in der App nochmal hautnah miterleben kann.

Im Bürgerhaus wird Generationsübergreifend gearbeitet und so war auch der Wunsch da die App für Kinder attraktiv zu gestalten. Die Artefakte werden spielerisch aufgearbeitet, um die Versäulung der Generationen aufzuheben, damit eine Schnittstelle von jung und alt noch mehr gefördert wird.

## Der Kern unseres Produktes ist die Darstellung von historischer Artefakten und lokaler Stadtteilgeschichte.

Wir entwarfen einen Zeitplan und erarbeiteten Personas bis hin zur einer Produktion. Aus der Designperspektive hat Christine eine detailliertes Screen-Design mit App Icon und passenden Styleguide entwickelt. Dabei hat Sie Gestaltgesetze beachtet und ein eigenen Stil für Farbe, Icons und Typografie kreiert. Jacob hat die technische Umsetzung der Applikation in der Unity Engine übernommen und das gemeinsame Konzept umgesetzt. Dabei hat er nach den Vorbildern gängiger Praktiken und Methoden, die im Zusammenhang mit der Unity Engine verwendet werden, gearbeitet und die UI-Screens gebaut und Skripte für die Funktionalität programmiert.

## QuartierAR Pilot

Mit dem Pilotprojekt „QuartierAR“, in Form einer interdisziplinären Bachelorarbeit, soll ein einfacher erster funktionierender Prototyp für das Testen und Veranschaulichen erstellt werden. Ziel ist es den Stakeholdern eine App zu präsentieren, dessen Funktionsumfang und Handhabung bereits ein sehr lebendiges bzw. natürliches Bild der späteren App vermitteln soll.

Als Grundlage für die weitere Entwicklung soll zunächst ein System entwickelt werden, dass alle wichtigen Kernfunktionen bietet und gleichzeitig skalierbar ist. Dies bedeutet, dass sowohl der Funktionsumfang als auch die Inhalte der App erweiterbar sein sollen. Der Pilot ist ein experimenteller Ansatz, um die technologischen Grundlagen zu erkunden und eine Einschätzung für die Machbarkeit und mögliche Probleme und weitere Anforderungen zu gewinnen.



## Skalierbarkeit

Idealerweise sollte diese frühe Ausführung technisch so einfach wie möglich gestaltet werden, um den Entwickler:innen und Designer:innen zukünftig Zeit und Aufwand einzusparen und um, das Verwalten der Inhalte ohne viel Vorwissen zu ermöglichen. Direkt eine einwandfreie App zu entwickeln sei selbstverständlich nicht möglich, aber einen optimalen Arbeitsfluss zu schaffen ist für uns ein erstrebenswertes Ziel. Auf der Basis des Piloten soll im nächsten Schritt die finale App entwickelt werden.

## Die Hintergründe

Das Fördergebiet Kattenturm ist ein Ortsteil des Bremer Stadtteils Obervieland, welches sich bis zu seiner heutigen Struktur in den sechziger / siebziger Jahren entwickelte. Im Zentrum des Ortes sind viele Großwohnraumsiedlungen zu sehen.

Das Klinikum "Links der Weser" ist der größte Arbeitgeber im Ortsteil.

*Kattenturm ist ein kinderreiches, multikulturelles und lebenswertes Quartier, es leben hier überdurchschnittlich viele alleinerziehende Elternteile.*

Soziale Stadt Bremen, 2021

Durch Fehlplanungen und Defizite in der sozialen Infrastruktur in den achtziger Jahren, schrumpfte das Versorgungsangebot und Leerräume in Mietwohnungen nahmen zu. Daraufhin wurden in den neunziger Jahren, mit dem Einsatz von öffentlichen und privaten Hilfsquellen, viele Mängel beseitigt und die Demoralisierung des Quartiers aufgehoben. Trotzdem gehört Kattenturm immer noch zu den benachteiligten Gebieten in Bremen.

Um ein Gefühl für die Gegend von Kattenturm zubeikommen, haben wir die Menschen im Bürgerhaus befragt. Des weiteren wurden wir an alle Orte, die auch in der App zu finden sind, herumgeführt. Bei den Erzählungen haben wir schon gemerkt, mit wie viel Freude und Herz die sozialen Projekte gefördert werden. „Bürgerhaus Gemeinschaftszentrum Obervieland e.V.“ besteht seit 1977 im Herzen von Kattenturm. Es ist ein fester Bestandteil des sozialen und kulturellen Lebens im Stadtteil. Es dient als Treffpunkt aller Altersgruppen und organisiert Kurse in den unterschied-

lichsten Bereichen wie Gesundheit, Unterhaltung, Kultur und vieles mehr.

Elke Munderloh beschreibt selber das Bürgerhaus als eine "bunte Kultur". Im Interview erzählt Sie uns, dass es viel Spaß macht in diesem Stadtteil zu arbeiten.

*"Der Stadtteil ist total spannend, weil er sehr innovativ ist und hier ein großer Bevölkerungsquerschnitt lebt"*

Elke Munderloh, 2021

Auch der Leiter des Bürgerhauses Stefan Markus ist sehr vernetzt mit dem Stadtteil. Ihm liegt viel daran die Lebensqualität vor Ort für alle Bürger:innen angenehmer zu gestalten. Bei allen Veranstaltungen wird stets darauf geachtet generationsübergreifend zuarbeiten. Es gibt Eltern-Kind Gruppen, eine Jugendabteilung und einen Kreativbereich für Erwachsene und Senior:innen. Seit zwei Jahren gibt es für die ältere Bevölkerung eine Ausgabe der Senioren Tafel. Dort helfen mehr als 100 Ehrenamtliche aus dem Stadtteil jede Woche Essen auszuteilen. Die Tafel ist für bewegungseingeschränkte und ältere Menschen gedacht. Das Bürgerhaus veranstaltet viele Kurse für Senior:innen. Elke leitet diesen Bereich. Ihr ist es wichtig, dass Ältere im Rentenalter nicht ihren Fokus verlieren.

*"Senior:innen sollen nach der Arbeit, also in der Rente eine neue Bedeutung finden und wir helfen ihnen dabei"*

Elke Munderloh, 2021

Das Angebot ist vielseitig. Spielgruppen, ehrenamtliche Tätigkeiten, Beratungen und Lernkurse wie "Zehn Finger schreiben" und "Computer 1x1" werden häufig besucht.

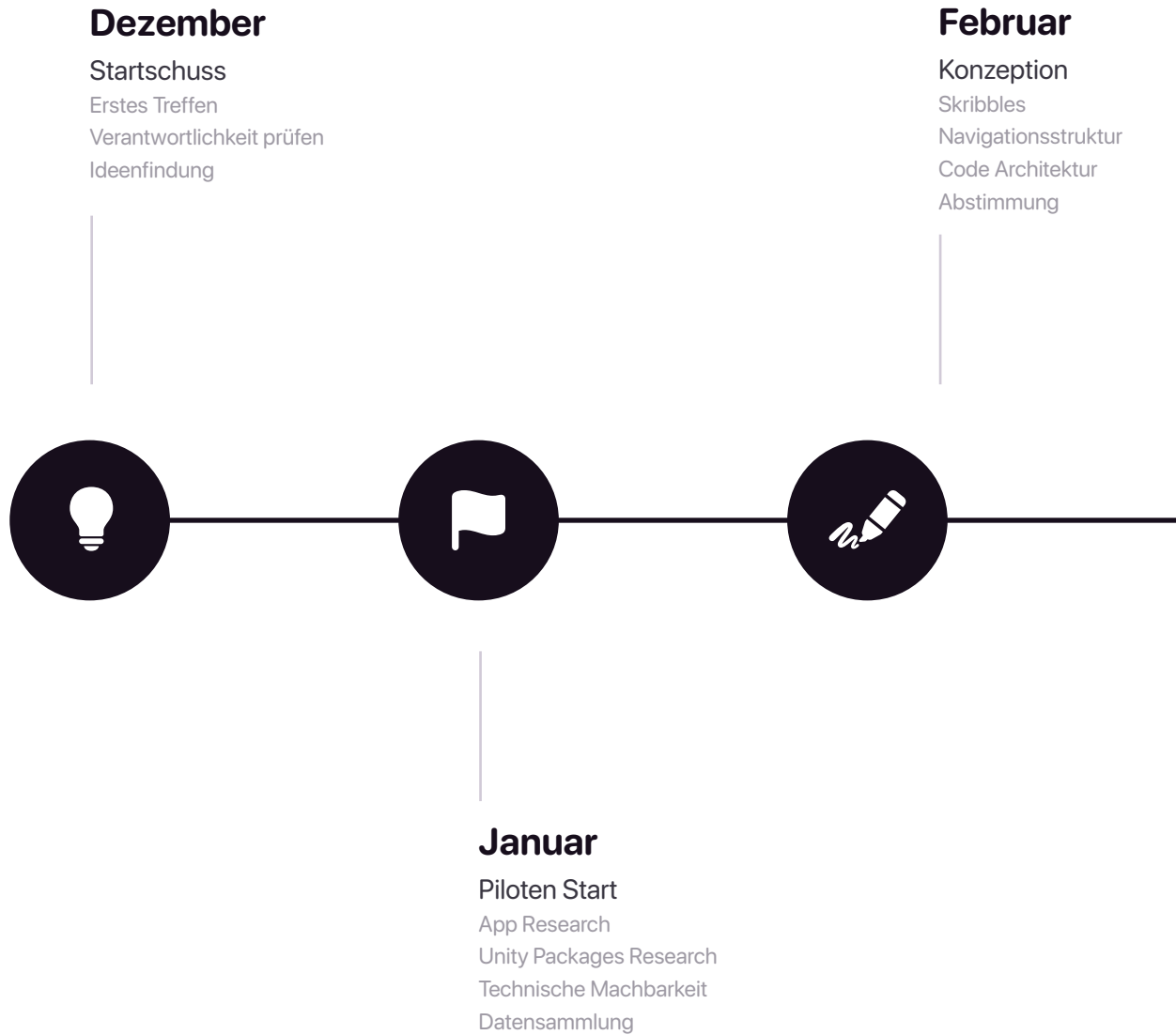
## Demografischer Wandel

Die älteren Teile der Bevölkerung gelten nicht erst seit kurzer Zeit als wichtige Zielgruppe in der digitalen Welt, zugleich ist sie jedoch die vermutlich am schwierigsten zu erreichende. Alte Menschen sollte man die Technikaffinität nicht von vornherein absprechen, auch wenn sie in Teilen von körperlichen und/oder geistigen Defiziten gezeichnet sind. Ihr Interesse an Technologie ist nicht gering, doch in unseren Augen fehlen entsprechende Angebote, die dieser Zielgruppe gerecht wird, besonders in Hinsicht auf die Interaktion. Für alte Menschen mag ein Smartphone oder eine App vielleicht eine große und neue Herausforderung sein, jedoch auch eine Chance weiterhin am aktuellen Geschehen der Welt teilzuhaben - besonders in Zeiten der COVID-19-Pandemie. Alte Menschen lernen langsamer als junge Menschen, doch auch das ist kein Grund als Entwickler:in vor der Herausforderung zurückzuweichen und sie in die Entwicklung unserer App besonders miteinzubeziehen.

Durch Elke und Ihre Arbeit im Quartier ist uns dies sehr bewusst geworden. Wir entwickelten einen gewissen Respekt dafür, dass die ältere Generation trotz gewisser Barrieren Ihr Interesse an unserer App und den Ihnen völlig fremden Konzepten und Technologien bekundete.

Wir wollen besonders den ehrenamtlichen Mitarbeiter Hans Dieter Oehlke erwähnen. Durch seine Arbeit konnten wir die App mit echtem Leben füllen. Seine Geschichte als ehrenamtlicher Mitarbeiter im Bürgerhaus begann vor ca. 40 Jahren. Er organisierte Fotogruppen für Jugendliche mit etlichen Ausstellungen und Reisen. Sein Engagement zieht auch viel Verantwortung mit sich und ging soweit, dass er Vorsitzender im Bürgerhaus wurde. Als Beiratsmitglied der SPD setzte er sich auch für Bildung im Rahmen des Schulelternsprechers ein. Er hat die Geschichte des Stadtteils miterlebt und auch selber viel Fotodokumentiert. Aus diesem Wissen und der Fülle an Fotos konnten wir eine detailreiche Zeitgeschichte abbilden. Hans Dieter ist die Quelle des Wissen über den Stadtteil und hat auch bei unseren Projekt mit viel Engagement Informationen zusammengetragen, die wir in unserem Projekt darstellen können.

# Zeitlicher Ablauf



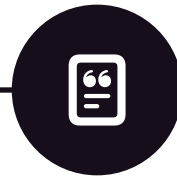
Christine

Das Projekt

## April

### Implementierung

Design Integration  
Programmierung  
Content Integration  
Schnittstellen  
Zwischenstand präsentieren



## März

### Prototype

Wireframes  
Styleguide  
Programmierung  
Backend  
Abstimmung

## Mai

### Finalisieren

Programmierung  
Testing  
Dokumentation

# Verwandte Arbeiten und Inspirationen

In dem folgenden Kapitel geben wir einen Überblick darüber, welche AR Apps uns inspiriert haben. Jede App wird kurz beschrieben und erwähnt unsere Verbindung zu unserem Projekt. Es gibt heutzutage schon eine ganze Menge Augmented Reality Apps in den digitalen App Stores. Die beliebtesten unter Ihnen sind seit jeher Spiele, das prominenteste unter Ihnen ist vermutlich Pokémon GO.

**Pokemon GO**  
**Civilisations AR**  
**Snapchat**  
**iPhone AR Apple**

## Pokemon GO



Kategorie: Action & Abenteuer

Downloads: 100 Mio. +

Bewertung: 4,1 Sterne

Altersfreigabe: ab 6 Jahren

Pokémon Go wurde am 13. Juli 2016 veröffentlicht und allein im Google Play Store über 100 Millionen Male heruntergeladen. Ich selbst (Jacob) habe es im ersten Jahr seit der Veröffentlichung intensiv gespielt und geliebt. Mit einer gigantischen generationsübergreifenden Fanbasis und der neuartigen Gameplay Idee war der Erfolg der App absehbar.

Das eigentliche Spielprinzip von Pokémon GO ist eine Art digitale GPS-Schnitzeljagd. Die Spielenden sind hauptsächlich damit beschäftigt durch die reale Welt zu gehen und somit die eigene Spielfigur auf der digitalen Karte auf dem Smartphone zu bewegen, welche ein genaues Abbild der realen Welt durch Google Maps ist. Dabei werden in der Nähe befindliche virtuelle Pokemon angezeigt, die, sobald sie in Reichweite der eigenen Spielfigur sind, angetippt werden können, um gefangen zu werden. Das Fangen der Pokémon wird durch ein Minispiel realisiert, bei dem die Spielenden den ikonischen Pokéball mit einer Wischgeste über den Bildschirm auf das digitale Wesen werfen. Wird das Pokémon getroffen öffnet, sich der Pokéball und versucht dieses in sich einzuschließen. Je präziser der Ball auf das Pokémon geworfen wurde, desto höher ist die Chance es zu fangen. Die Spielenden haben die Wahl, ob sie diese Minispiel in einer virtuellen Umgebung spielen möchten, oder als Augmented Reality Feature mit der Smartphone Kamera in der Szenerie der realen Welt. Außerdem gibt es im Spiel die Möglichkeit Fotos von den gefangenen Pokémon zu schießen, ebenfalls im Kontext der realen Welt. Auf dem Bild sieht es so aus als hätte das Pokemon die reale Welt tatsächlich betreten. Das Pokémon Universum hat, in meinen Augen, erneut eine Generation geprägt, diesmal war es jedoch die der AR-Technologie. Die Funktion die Pokémon im Kontext der realen Welt anzuzeigen war eine große Inspiration für unsere Idee für die Darstellung unserer Artefakte. Diese sind jedoch durch das Image Tracking an ein bestimmtes lokales Merkmal





Google Playstore, o.J. a

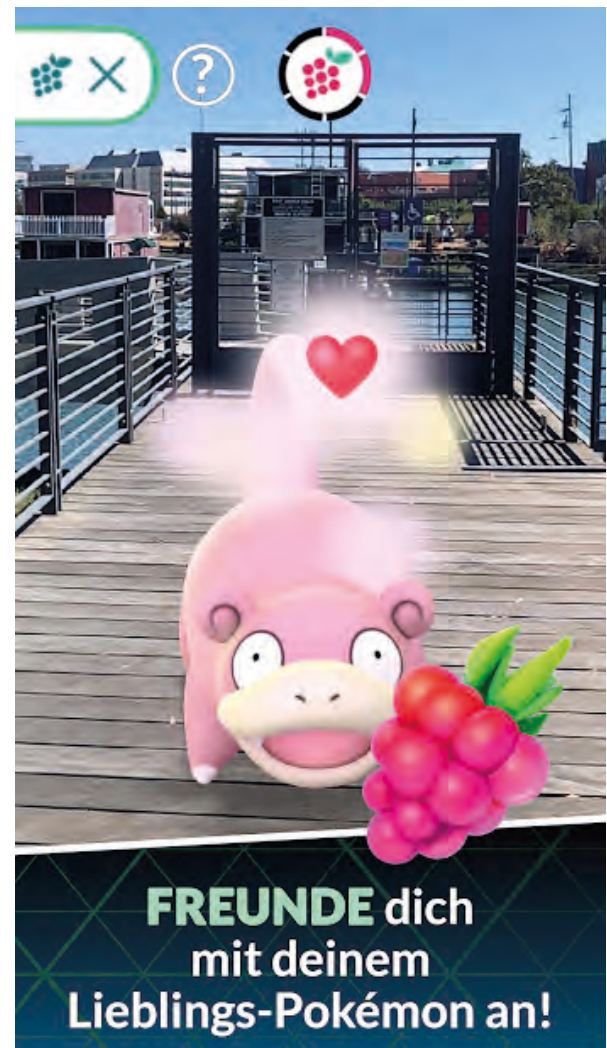
an dessen Position gebunden (sofern sie nicht über die Galerie aufgerufen werden). Dennoch war Pokémon GO die erste AR-Referenz mit der ich unsere App-Idee bei den Menschen im BGO bewarb.

Des Weiteren hat Pokémon GO auch gezeigt, dass man die Spielenden mit besonderen oder seltenen Spiel Objekten an bestimmte Orte locken kann. Diese wären beispielsweise Arenen, in denen man mit seinen eigenen Pokémon ge-

Jacob

gen die anderer Spielenden antreten kann, oder auch seltene Pokémon Exemplare, von denen man meist von anderen Spielenden erfährt, falls man gut mit der Spielgemeinschaft im eigenen Umfeld vernetzt ist. In meinem Stadtteil gibt es eine große Gemeinschaft aktiver Spielende, die regelmäßig unterwegs sind, um seltene Pokémon zu finden und fangen.

Auch die Sammlung der Pokémon, die man bereits gefangen hat, hatte Rückblickend einen Einfluss auf die Gestaltung unserer Artefakt Galerie.



Verwandte Arbeiten und Inspirationen

## Civilisations AR



Kategorie: Action & Abenteuer

Downloads: 332

Bewertung: 3,7 Sterne

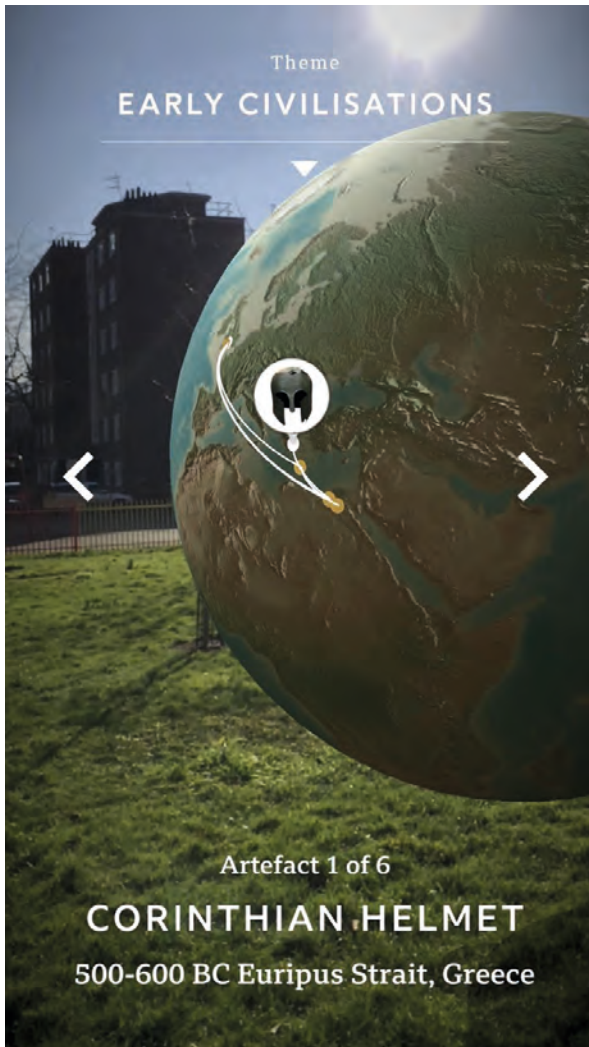
Altersfreigabe: ab 0 Jahren

Civilisations AR ist eine für BBC entwickelte AR App, die Artefakte vergangener Zivilisationen zeigt. Nutzer haben die Möglichkeit auf einer in AR dargestellten Erdkugel einen markierten Ort auszuwählen und dann dessen Artefakt in Originalgröße auf dem Smartphone anzuschauen.

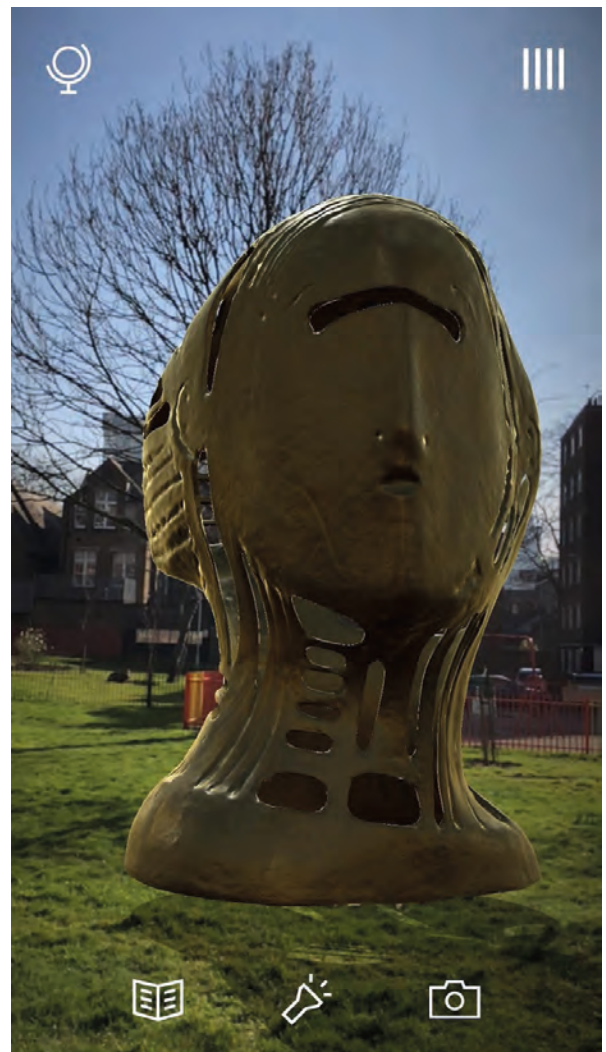
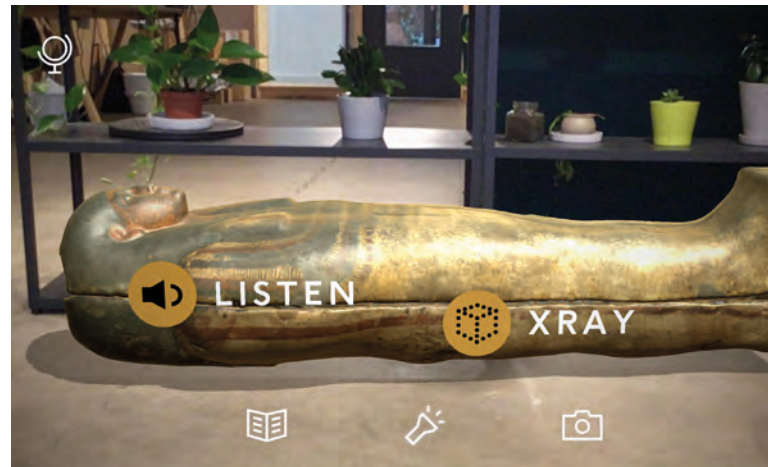
Die Artefakte werden auf einer vorher durch das AR-System getrackten ebenen Fläche platziert und können darauf bewegt, gedreht, skaliert und dann von allen Seiten auf dem Bildschirm betrachtet werden. Die App bietet mit den Artefakten nicht nur ein realitätsnahes eingescanntes 3D Objekt, sondern auch einen ausführlichen Beschreibungstext und Funktionen zum Entdecken zusätzlicher Inhalte und Darstellungen für die Artefakte. Interessante und kreative Spielereien wären beispielsweise das Polieren eines alten Korinther Helmes oder das Betrachten der Mumie im Inneren eines Sarkophags durch dessen Wände.

Civilisations ist in meinen Augen die interessanteste App, da sie unserer Idee bereits ein konkretes Vorbild ist, an dem wir uns speziell für die Darstellung und Interaktion mit unseren Artefakten orientieren können.





Google Playstore, o.J. b



Jacob

Verwandte Arbeiten und Inspirationen

## Snapchat



Kategorie: Soziale Netzwerke

Downloads: 25.997.072

Bewertung: 4,3 Sterne

Altersfreigabe: ab 12 Jahren


Snapchat ist eine Foto-App, die Augenblicke mit Freunden teilt. Die vielfältige Auswahl an Filtern und Objekten, die in der Kamera Szene platziert werden, macht die App so interessant. In Sekundenschnelle wird ein Foto oder Video mit Text erstellt. Die Auswahl der Filter variiert ständig, so kann der:die Benutzer:in immer wieder etwas neues entdecken. Die Anzeigedauer der Bilder kann limitiert werden, so dass es besonders reizvoll ist, kurze Momente festzuhalten und sie nie wiederzusehen. Diese App hat uns im Zuge der Interaktion mit der Kamera bedeutend inspiriert.



Google Playstore, o.J. c

# Mache Snaps

und teile den Moment



A screenshot of the Snapchat app interface. At the top, the time is 3:19 and the battery is at 99%. The main image shows a woman with curly hair and glasses holding a dog. Both are wearing large, round glasses. The dog is also wearing glasses. The text "SEE YOU SOON!" is overlaid in pink, bubbly font. On the right side, there are icons for text, drawing, and filters. At the bottom, there is a "Send To" button.

# Hab Spaß


und gestalte deine Welt neu



A screenshot of the Snapchat app interface. At the top, the time is 3:19 and the battery is at 99%. The main image shows a man in a grey shirt and dark pants standing in a grassy field, reaching out to touch the nose of a large, white, winged unicorn. The unicorn has a blue horn and is breathing a blue flame. At the bottom, there are icons for various filters, including a unicorn, a smiley face, and a red face. At the very bottom, there are icons for "Create", "Scan", "Browse", and "Create".

# Finde heraus,

was deine Freunde machen



A screenshot of the Snapchat app interface showing a map of New York. The map is labeled "New York" and shows various streets and landmarks. There are several location pins with avatars and names, indicating where friends are located. The pins include "David & 3 more Now", "Liz 1m", "John & Stacy 3m", "Leo 12m", and "Zoe 4h". There is also a red car icon and a person on a skateboard icon. At the top, there is a search bar and a settings icon.

# Chatte

mit deinen echten Freunden



A screenshot of a Snapchat chat conversation. The chat is titled "Besties" and shows a conversation between "ME" and "JOHN". The messages are:

- ME: Heyyyyyy! Is anyone at the lake yet??
- JOHN: Yep! Just got here! OMG is Daisy coming too???
- LIZ: Almost there
- JOHN: At Burrows Road & Bayside Drive!

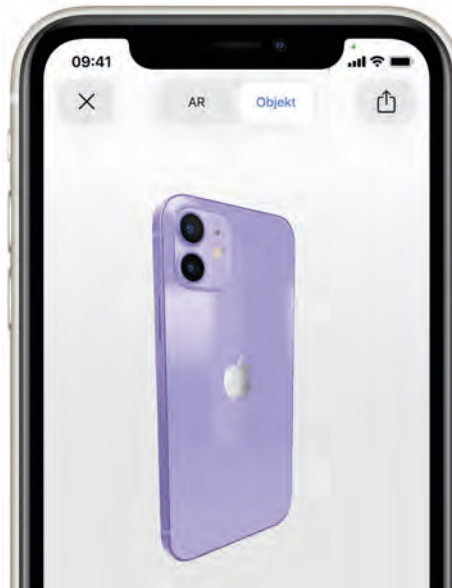
At the bottom, there is a "Snapchat" logo. The chat interface includes a search bar, a camera icon, and a settings icon at the top.

## iPhone AR Apple



### Erlebe sie mit AR aus jedem Winkel.

Öffne diese Seite in Safari auf  
deinem iPhone oder iPad.

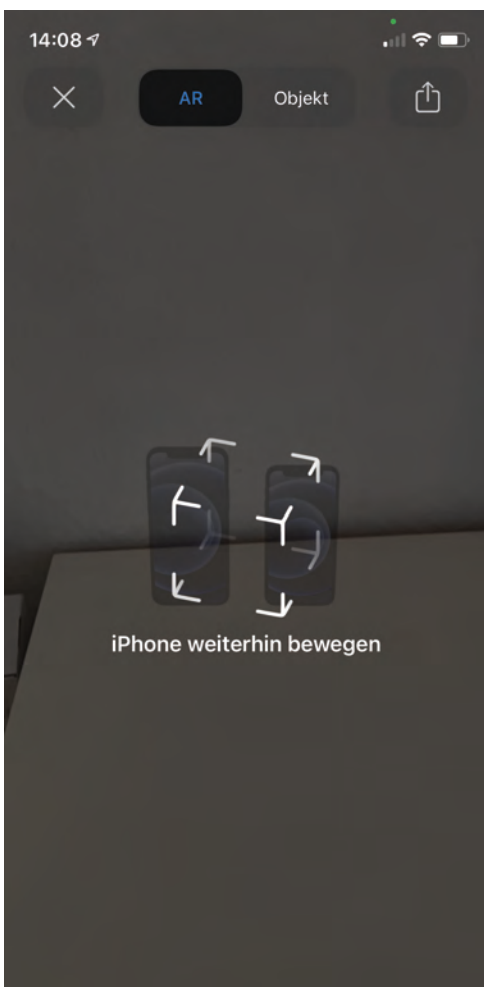


Apple, o.J.

Auf der Internetseite von Apple.com können Benutzer:innen das neue iPhone betrachten. Für diese Funktion wird der Browser „Safari“ benötigt und ist tatsächlich auch nur für Apple Benutzer:innen vorgesehen. Für iPhone oder iPad wird ein Link auf der Seite in der Kategorie „Erlebe sie mit AR aus jedem Winkel.“ aufgezeigt, um sich das neue iPhone in AR darstellen zu lassen. Das ist typisch provokant von der Marke Apple, aber uns geht es um die Umsetzung. Wenn die Kamera geöffnet erscheint, wird der:die Benutzer:in mit einem Beschreibungstext kurz und übersichtlich zur Interaktion geleitet. Der Workflow hat uns bei unserer Ideenfindung sehr begeistert.



Eigene Screenshots aus iPhone AR Apple



# Mobile Applikation



**Mobile Apps**  
**Augmented Reality**  
**Interaktion im Alltag**  
**Physisches Umfeld**

## Mobile Apps

Die Kinder der heutigen Generation wachsen schon mit Apps auf. Sie wischen, schwenken über den Bildschirm als wäre es das natürlichste der Welt. Diese Fähigkeit wurde aber nicht in die Wiege gelegt. Meist werden die Gesten durch Beobachtung erlernt. Beispielsweise wenn die Mutter über das Smartphone einen Termin reserviert oder der Vater den Wikipedia Eintrag von Kronkorken vorliest.

Abhängig von Altersgruppen werden Smartphones benutzt. Für Kinder sorgt es für Unterhaltung. Erfüllt die App dieses Ziel, spielt es zufrieden weiter. Für ältere Altersgruppen verschiebt sich oder eher vervielfältigt sich die Verwendung im Alltag. Mittlerweile gibt es für viele Gebiete die passende App. Ob Online Banking, die Suche nach dem passenden Kühlschrank oder die richtige Laufstrecke. Wenn diese App unseren Ansprüchen nicht entspricht oder zu kompliziert ist, wird diese gelöscht und im Store die nächst beste runtergeladen.

*Laut eMarketer hören fast 34 Prozent aller Nutzer nach nur einem einzigen Tag mit der Nutzung einer App auf.*

(vgl. Adobe (o.J.), 2018)

Das verdeutlicht wie wichtig der erste Eindruck der App ist. Um bei diesem Projekt den richtigen ersten Eindruck zu vermitteln und auch langfristig die App attraktiv zu machen, haben wir Formen des Design Sprints angewendet. Wir haben Methoden eingesetzt, um natürliche Benutzererlebnis zu generieren. Weiter gehe ich auf das Thema auf im Kapitel „Konzeption“ ein. Um Momente richtig in Szene zu setzen benötigt der:die Benutzer:in Hilfe. Da ist es wichtig, wenn Sie sich verloren fühlen, Hinweise zu geben wie man zum Ziel kommt. (Zum Beispiel im Szenario: Ein:eine Benutzer:in möchte ein AR-Event ausprobieren und öffnet den Menüpunkt „Scanne deine Umgebung“. Die Kamera wird auf dem Gerät geöffnet und als Hinweistext auf dem Screen steht: „Bewege deine Kamera auf ein Objekt“. Damit wird eine Hilfestellung gegeben, um zum Ziel zukommen). Wir haben uns für eine mobile Applikation entschieden, weil es für den:der Nutzer:in als persönlich und natürlich empfunden wird. Die Benutzeroberfläche zeigt nur kontextuelle relevante Information an. Es wird **keine Werbung** geschaltet und Daten werden auch **offline** (ohne Internet) gezeigt. Native Elemente wie aufrufen der Kamera und **Push Notifications** (Benachrichtigung) können mit implementiert werden. Viele Faktoren müssen beim kreieren einer Software beachtet werden. Im folgenden Text gehe ich auf positive Konditionierung und darauf ein, ob die physische Wirkung auch eine Wirkung auf das Produkt hat. Aber zuerst erläutert Jacob, was AR ist und warum es ein Mehrwert für unser Projekt darstellt.

## Augmented Reality

The digital world is no longer just on our screen, it transcends the boundaries of the digital world and increasingly enters the real world by us sharing information with it and allowing it to be presented in the context of our reality. The rapid development of sensors and input devices as well as the increasing performance of microcontrollers and wireless connections play a decisive role. Nevertheless, virtual worlds with increasing immersion, driven by developments in the gaming industry, are also contributing to virtuality taking on more and more an almost physical form. (Paradiso & Landay, 2009)

*“We call the ubiquitous mixed reality environment that comes from the fusion of these two technologies cross-reality.”*

(Paradiso & Landay, 2009)

**Cross Reality (XR)** for short), a term coined in 2009 by Joseph A. Paradiso, Professor of Media Arts and Sciences, and James A. Landay, Professor of Computer Science and Engineering, describes a broad field of technological mediation of combined virtual and real environments.

**Augmented Reality (AR)** is one of these technologies.

It can be described as computer-aided tracking of the real world and its extension, modification, or addition by virtual objects (Markgraf, 2018).

The augmentation of the real world only refers to the image of the environment on the screen of the AR-enabled device, since nothing is projected or inserted into the real world. Users see the augmented world merely through a gateway that makes digital information visible. It superimposes the real, camera-captured environment with digitally linked information (Object Tracking | Fraunhofer IGD, o. D.).

*“Augmented Reality enhances a user’s perception of and interaction with the real world. The virtual objects display information that the user cannot directly detect with his own senses. The information conveyed by the virtual objects helps a user perform real-world tasks.”*

(Azuma, 1997)

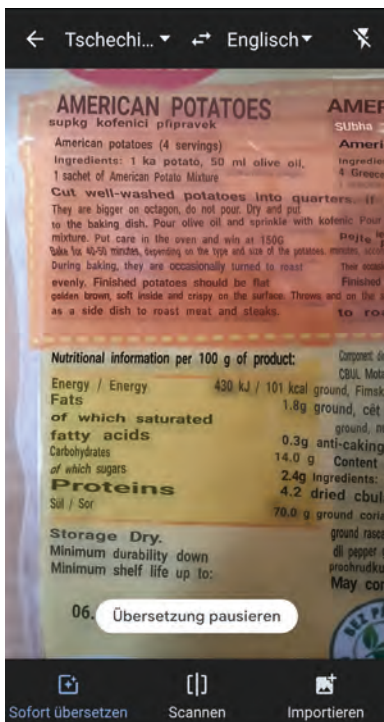
AR is described here primarily as humanitarian aid in the perception and interaction with his environment. When Ronald T. Azuma published these lines in 1997, the devices for this technology were still large, unwieldy, and therefore impractical, as shown by the “Touring Machine”. (Feiner et al., 1997)

A supportive function can only be achieved if the technology empowers the users doing their task, rather than hindering them at it. Therefore, there was no added value at that time. Nevertheless, the concepts were future-oriented, and we can see the result today.



Feiner et al., 1997

## The Added Value of Augmented Reality



During my semester abroad in Prague, I often used the functions of the Google Translate app to translate ingredients on food packaging, for example. The AR function of the app allowed me to translate the texts captured with the camera on the products in real-time. This method offers added value compared to the manual input of text into the app since it provides a faster input method and thus speeds up the translation process. Especially if you are not used to typing foreign words on the keyboard.

Today's ubiquitous presence of smartphones and tablets enables almost anyone who owns such a device to benefit from AR technology. The devices are small, handy, and fit in just about any pocket.

But the mere possibility of using AR technology should not blindly invite you to display any content in this way. Augmented reality is only useful if it is more than a game and visually complements the exhibits, i.e. not only shows texts about the object (Kopp, 2018).

In our project, the presentation of artefacts from the district history of Kattenturm is in the foreground. Instead of just offering pictures, texts, and audio recordings, we want to stage the artefacts in three dimensions at their places of origin. The added value arises when we transform viewing the artefacts for the users into an experience that seems to go beyond the purely digital level and becomes spatial, thus conveying a sense of the artefacts' zeitgeist.

## Image Tracking

To display our artefacts in AR, a kind of tracking is required for the destination where the artefact is to appear on the screen.

In general, there are two types of AR solutions – marker-based augmented reality and marker-free (sensor-based) augmented reality (CONNECTED REALITY, 2020). We use a marker-based process, called image tracking, to place the 3D objects at their corresponding locations in the district. It is a type of image processing in which a reference image is compared with input data from a camera to recognize and track it. QR codes are a common example of this.

For the project, we have identified striking surfaces and reliefs in places that are appropriate as reference images. For this purpose, they should be easily recognizable by the AR system and at the same time be located on the site for the long term.

For the Cato Bontjes van Beek Platz, the information board in the corner of the place is well suited. Factors such as time of day and weather are important as they influence the appearance and brightness of the marker. A constant recognition of the marker is therefore not always guaranteed, which has to be considered.

Image tracker used for the "Cato Bontjes van Beek Platz" in the project.



To be able to display the artefacts regardless of location and without markers, we rely on the markerless tracking of surfaces. The AR system can detect flat surfaces, create its own virtual markers on them, and also place 3D objects there.



Image of the plane-tracking from an early stage prototype of our app.

## Interaktion Im Alltag

Es beginnt schon am frühen Morgen. Der Wecker klingelt und wird ausgeschaltet. Der Alltag ist geformt von **Erlebnissen** die wir unterbewusst ganz selbstverständlich ausführen (Garrett, 2010). Eine Interaktion mit dem Lichtschalter, die Tür öffnen und den Knopf auf der Kaffeemaschine. Je nachdem wie der Alltag aussieht, könnte man die Liste weiter fortsetzen. Jede dieser kleinen Interaktion erzeugt **Emotionen**. Diese können positiv oder negativ (Der Wecker wird eher negativ wahrgenommen), stark oder schwach wirken. Solche Erlebnisse, gerade wenn wir sie konditionieren, speichert unser Gehirn als **Erfahrung** im emotionalen Gedächtnis ab.

*„Erlebnisse sind emotional bewertete Ereignisse. Die Emotionen sind positiv, wenn die Erwartungen übertroffen werden.“*

(vgl. Ballmer et al., 2002)

Bei einem Spiel, einer Software oder auch bei einer Applikation, wie in unserem Fall, werden emotionale Kontakte verknüpft. Wenn eine Funktion Glücksgefühle auslöst, hat eine App höhere Chancen Erfolg bei dem:der Nutzer:in zu haben. (Beispiel: Referenz App – „Pokemon Go“ Mit dem Pokéball erfolgreich ein Pokemon fangen.) Wenn der Moment positiv war, möchte man das Erlebte wiederholen. Dopamin wird ausgeschüttet und steigert die Kreativität und Lernfähigkeit. Gelingt eine Funktion nicht, kann es auch zum Gegenteil führen (Norman, 20124). Der:die Benutzer:in benutzt das Produkt nur noch ungerne und verbindet damit Stress und fühlt sich entmutigt.

Um Erfolg mit dem Produkt zu erzielen, brauchen die Benutzer:innen möglichst viele positive Erlebnisse, damit die Erwartungen erfüllt oder sogar übertroffen werden. Dieser Faktor haben wir bei unserem Ziel mitbeachtet. Im Kapitel „App Design“ gehen wir auf die „Produktmerkmale“ unserer Piloten App ein. Wir erläutern, ob die App Begeisterungsmerkmale, Leistungsmerkmale, sowie Basismerkmale besitzt.



## Physisches Umfeld

Wir Menschen machen vieles vom **Wetter** abhängig. Natürlich gibt es Ausnahmen, aber die bestätigen meist die Regel. Hier ein paar Beispiele: Mit dem Fahrrad zur Arbeit wird nur bei trockenem Wetter gefahren, Eis schmeckt nur in der Sonne gut, Drachen steigen lässt sich am besten, wenn es windig ist. So hat die physische Situation auch starken Einfluss auf das Produkt. Es ist quasi ein Zeitpunkt in einem bestimmten Umfeld, wie die Benutzer:innen das Produkt verwendet. Für das Design ist es besonders wichtig, sich bewusst zu sein, dass das Ziel aus unterschiedlichen Situationen erreichbar sein muss (vgl. Ho, 2016). Welche Situationen sind gemeint? Die Software selbst hat keine Berührungspunkte mit dem **physischen Umfeld**, aber der:die Benutzer:in schon. Beispielsweise dann, wenn die App bei schlechten **Lichtverhältnissen** benutzt wird, ist es wichtig darauf zu achten, dass genug Kontrast für die Lesbarkeit geboten wird. Der Standort bei einer Interaktions-Applikation spielt natürlich auch eine Rolle. Ist das Erlebnis **Ortsabhängig**, oder kann man alles auch aus der Couch aussteuern? Das Thema Location (der Ort der Anwendung) haben wir berücksichtigt und eine Lösungsidee im Menü implementiert.

*„Die Erwartungen an ein Produkt sind abhängig von der Situation, in der es verwendet wird. Dies muss im Design berücksichtigt werden.“*

(vgl. Christian Moser, 2012)

Zusätzlich kann das soziale Umfeld Einfluss auf das Ergebnis nehmen. Da wir keinerlei Daten von dem:der Benutzer:in fordern, besteht keine Gefahr der Sicherheit. Beim Bedienen muss, um das Erlebnis zu garantieren, die Kamera und Audio-Signale innerhalb der Applikation zugelassen werden. Dazu erscheint jeweils ein Dialog mit der Anfrage: **„Darf diese App auf ihre Kamera zugreifen“**. Dies muss bestätigt werden, um die Aktion auszuführen. Dazu mussten wir nichts programmieren, diese Anfrage führt das Smartphone nativ pro Anwendung selber aus.





# Interdisziplinäres Team

# **Rollen im Team**

## **Aufgabenbereiche**



## Rollen im Team

Um eine gute App zu entwickeln, erfordert es viel **Fachwissen**. Daher haben wir uns interdisziplinär zusammengefunden, damit wir mehr **Kompetenzen** abdecken können. Da es bei der Applikation auch darum geht sie zeitnah einzusetzen, haben wir eng mit den Stakeholdern zusammengearbeitet. Je größer das Team, desto mehr Aufwand entsteht bei der Kommunikation. Wir als Entwickler-Team müssen uns regelmäßig absprechen aber auch immer unseren aktuellen Stand den Stakeholdern transparent offenlegen. Im Kapitel „Integration und Fazit“ wird im Abschnitt „Working Conditions“ im Detail erklärt ,wie unsere Zusammenarbeit und unser Arbeitsrhythmus intern als Entwickler-Team, aber auch die Arbeitsweisen mit den Stakeholdern war.



### Designer

Christine Koppe  
Konzeption  
UI / UX Design  
Projektmanagement  
Contentmanagement  
Front-End



### Entwickler

Jacob Wolyniec  
Technische Architektur  
Software Entwicklung  
Frontend / Backend  
Qualitätssicherung



### Stakeholder

Elke Munderloh  
Verwaltung  
Content Creator  
Anforderungsmanagement



### Stakeholder

Nadine Scheffler  
Verwaltung  
Content Creator  
Anforderungsmanagement



### Weitere Rollen

Stefan Markus, Hans Dieter Oehlke,  
Sabrina Jambor  
Leitung  
Content Creator  
Archiv

# Aufgabenbereiche

## Projektleitung / Management

Der:die Projektleiter:in koordiniert das Team, priorisiert Themen, organisiert Termine und hält das Zeitmanagement im Blick. Verhandelt mit den Stakeholdern und ist verantwortlich für die Qualität (Garcia et al., 2020).

## Informationsarchitektur

Sammelt Informationen, analysiert und strukturiert diese für die technische Basis. Dafür wird großes abstraktes Denken und viel Analyse benötigt.

## UI / UX Design

Der:die Designer:in gestaltet grafisch den Dialog zwischen Benutzer:in und der App. Analysiert die Bedürfnisse und erstellt konzeptionell alle Rahmenbedingungen und hält diese in Screens fest. Die sogenannte Benutzeroberfläche (User Interface) stellt Informationen optisch ästhetisch dar.

## Software Entwicklung

Verantwortlich für die passende Architektur der Software. Setzt Anforderungen um und programmiert den funktionsfähigen Code. Erstellt Methoden für einfache Darstellung von Inhalten (Garcia et al., 2020).

## Qualitätssicherung

Untersucht die Software nach Fehlern. Testet die App komplett nach jedem Szenario durch und dokumentiert Auffälligkeiten. Wenn Fehler vorhanden sind, werden diese mit dem Team kommuniziert und Lösungen erarbeitet (Garcia et al., 2020).

## Stakeholder

Der Kunde ist bei unserer Arbeit der sogenannte Stakeholder. Sie vertreten die Interessen der Institution mit deren Erwartungen. Sie haben Einfluss auf das Projekt und sind Anspruchsberechtigt. Vom Projektmanager werden sie regelmäßig über den Zwischenstand informiert werden. Der Stakeholder ist am Ende des Projekts der Abnehmer der App (Garcia et al., 2020).

## Content Creator

Wir haben den Begriff etwas abgewandelt und für unseren Kontext wie folgt zugeschnitten: Ein Content Creator sammelt und erstellt die multimedialen Inhalte von den Artefakten und Geschichten, durchsucht Archive, interviewt Verantwortliche und digitalisiert zum Teil die Daten. Ein Content Creator erfasst Texte, sortiert Bildmaterial und kontrolliert diese auf der Datenschutzebene und stellt alles zur Verfügung. (Garcia et al., 2020)

# Konzeption

Christine



**Design Prozess**

**Produktvision**

**Methoden**

**Anforderungsmanagement**

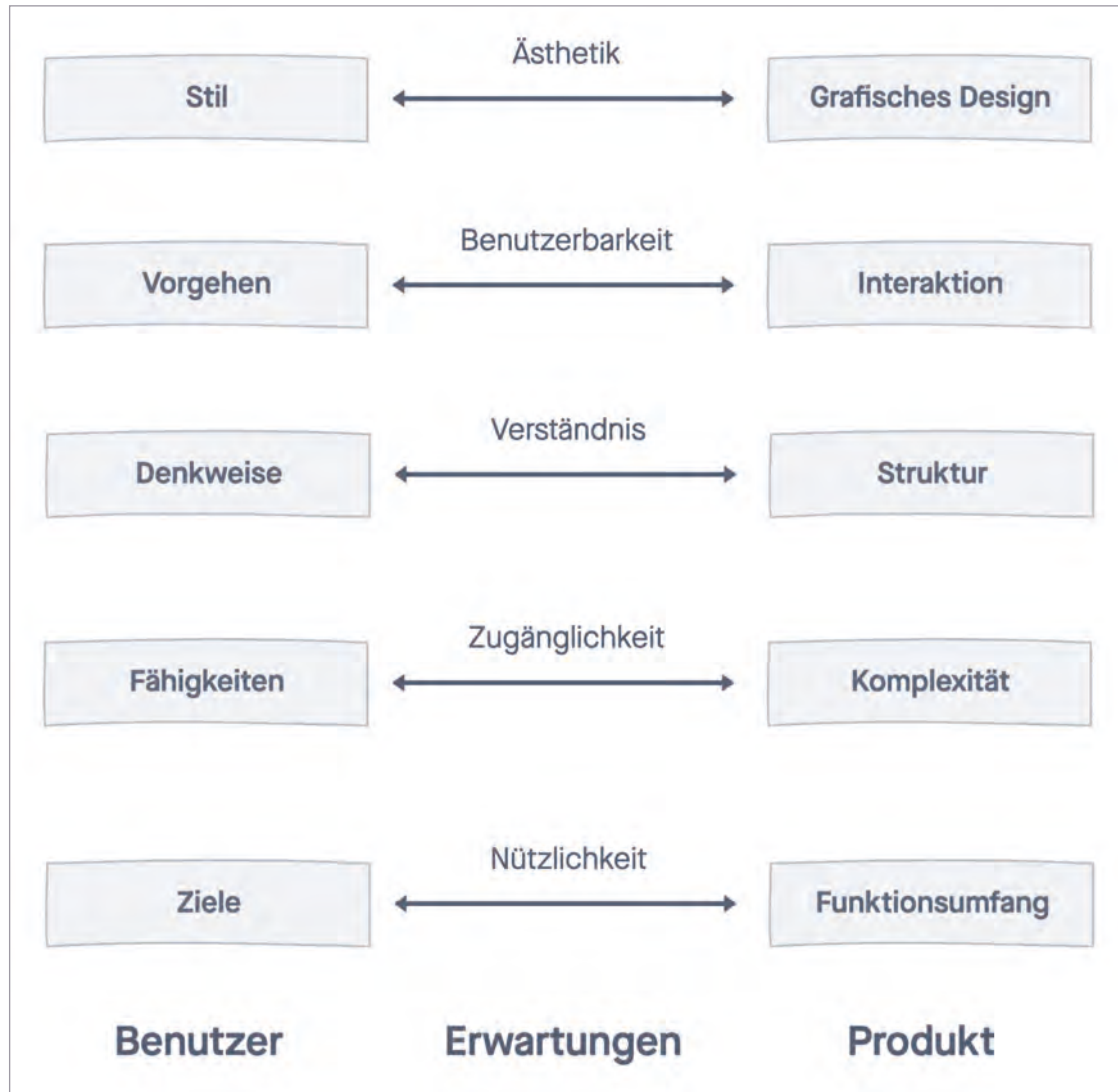
**Personas**

**Navigationsplan**

## Design Prozess

In der Passage „Mobile App“ habe ich schon erwähnt, dass natürliche Erlebnisse ein wichtiger Punkt für den Erfolg des Produkts ist. Positive Emotionen erzeugen Begeisterung für das Produkt. Doch wie gestaltet man ein gutes Produkt? Um so etwas kümmert sich der:die User Experience Designer:in. Oft haben die Benutzer:innen keine Vorstellung welche Möglichkeiten das Produkt haben kann. Jede Interaktion mit einem Produkt assoziiert Erfahrungen und das Ziel dabei ist es eine gute und sinnvolle Verbindung herzustellen. Also müssen die Erwartungen untersucht werden. Dazu gehört eine Marktrecherche, Trends und Analyse der aktuellen technischen Machbarkeit (Park & Maurer, 2009).

Als User Experience Designer:in benötigt man ebenfalls Wissen über die Technologie, um ein Verständnis für die Implementierung zu bekommen. Für die Analyse werden viele Information benötigt. Die untere Abbildung visualisiert eine Informationsarchitektur, die sämtliche Aspekte zeigt, die beachtet werden müssen.



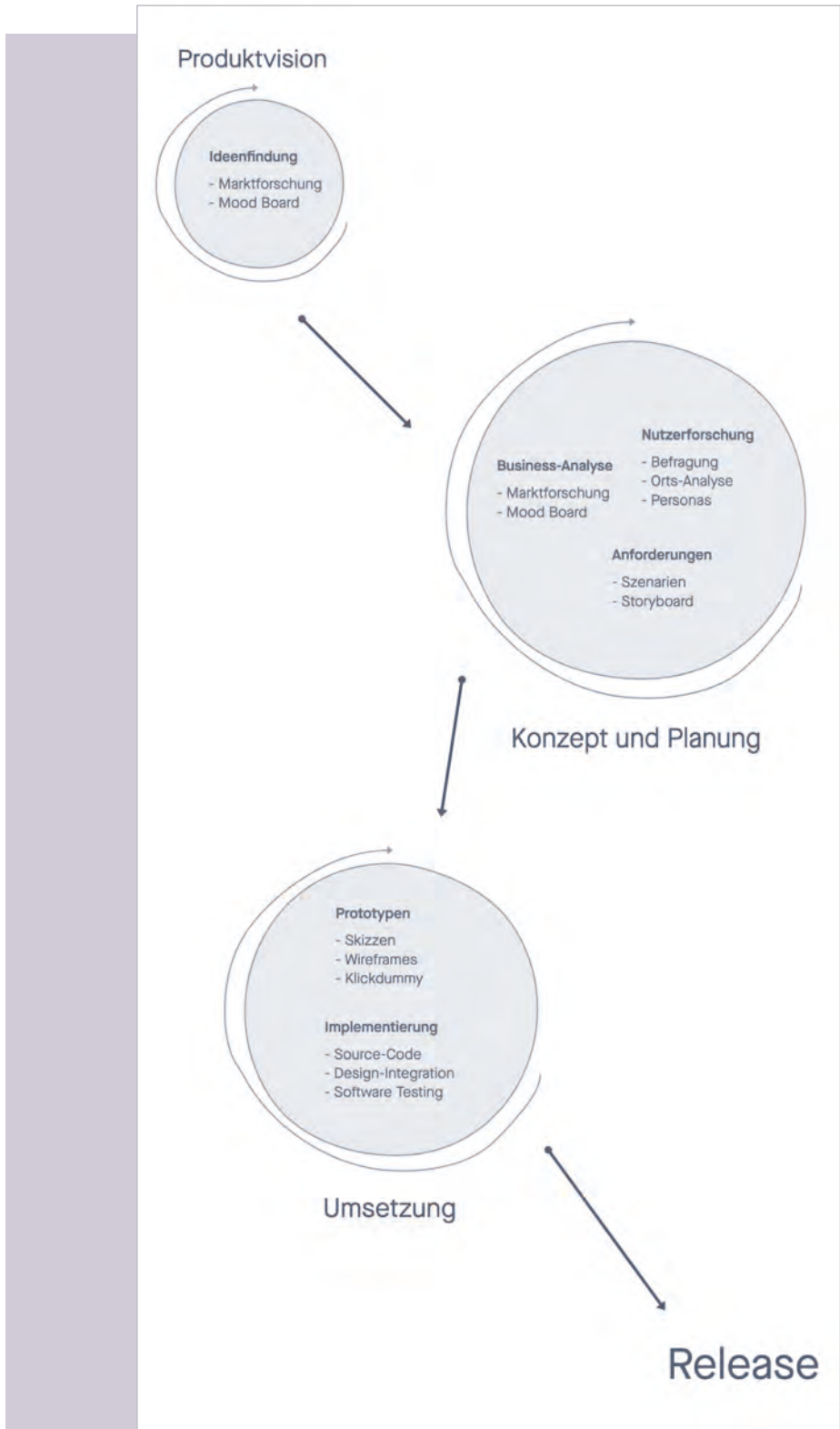
**Produkteigenschaften, selbsterstellte Grafik**

Können jeweils die Berührungspunkte von dem Benutzer:in erfüllt werden, ist das Begegnung mit dem Produkt positiv

## Produktvision

Nach der ersten Iteration der Forschung und Analyse, haben wir angefangen Lösungsansätze zu definieren. In dieser Phase haben wir uns oft mit den Verantwortlichen getroffen. Wie schon erwähnt hat sich das Team wie folgt zusammengesetzt: Jacob und ich (Christine), die die App konzeptionieren und umsetzen. Elke Mund-erloh (Leiterin der Senior:innen Begegnungszentrum im Bürgerhaus Obervieland und unsere erste Ansprechperson), Nadine Scheffler (Projektleitung des Quartiers Kattenturm), Stefan Markus (Leiter und Geschäftsführer des Gemeinschaftszentrum Obervieland), Hans Dieter Oehlke (Ehrenamtlicher Helfer beim Erstellen der Inhalte) und Sabrina Jambor (Ehrenamtliche Helferin und unterstützt Elke bei ihren Aufgaben).

Bei den ersten Treffen haben wir viel über die Bedürfnisse der Stakeholder geredet und was sie von uns und vom Produkt erwarten. Das Ziel war es eine gemeinsame Produktvision zu definieren, um ein gemeinsames Verständnis der App zu schaffen. Zusammen haben wir diskutiert, was der:die Benutzer:in für eine Erwartungshaltung einer neuen App haben kann. Neue Konzepte und Technologien waren gewünscht. Nach einigen Gesprächen haben Jacob und ich gemerkt, dass eine außerplanmäßige Herausforderung dazu gekommen ist. Bei diesem Projekt ging es nicht nur darum den:die Endnutzer:in glücklich zu machen, sondern auch unsere Stakeholder. Die sind diejenigen, die das Produkt abnehmen.



Von der Produktversion zum Release, selbsterstellte Grafik

Design Prozess mit seinen Iterationen

Christine

Konzeption

*„Die App ist eine wunderbare Ergänzung darzustellen, was war eigentlich früher, was ist heute und was ist in 5 Jahren geplant. Aber auch explizit auf den Stadtteil bezogen, weil wir machen so viele Stadtteil Rundgänge in verschiedenen Bezügen. Man kann mal zeigen, wie hat sich der Stadtteil entwickelt.*

*Die Entwicklung, nicht nur wenn man in die Vergangenheit guckt und Fotos von damals sieht, dass Kattenturm auch schöner geworden ist. So den Werdegang einer Stadt in einer App nochmal wahrzunehmen find ich richtig genial.“*

Stefan Markus, 2021

## Methoden

In diesem Abschnitt möchte ich nochmal auf die Ideenfindung eingehen. Die Rahmenbedingungen, dass es eine interaktive App werden soll, war gegeben. Das Produkt hatte das Ziel die Versäulung der Generation aufzuheben. Es soll eine Kombination von alten Wissen transportiert werden, aber zugleich sollen diese spielerisch erforscht werden können.

Bei den Befragungen und Treffen konnten wir ein gemeinsames Verständnis für das Projekt schaffen. Der Zuspruch war groß und wir beide waren motiviert etwas Innovatives und extra für den Stadtteil zugeschnitten, zu erschaffen.

Also haben wir uns der Herausforderung gestellt und wollen etwas Besonderes für die Bürger:innen in Obervieland kreieren. Innovation entsteht durch Kreativität. Bestehendes Wissen mit neuen Ideen zu kombinieren. Durch weitere Gespräche hat sich herauskristallisiert, dass nicht nur der Stadtrundgang wichtig ist für Bürger:innen.

Wie wir auf in der „Einleitung“ schon erklärt haben, arbeitet das Bürgerhaus mit vielen Senior:innen zusammen. Es gibt ältere Bürger:innen die nicht mehr selbstständig mobil unterwegs sein können und die Sorge haben, nicht mehr Teil der Gemeinde zu sein. Die App soll das Leben hier im Stadtteil widerspiegeln und der:die Benutzer:in auch von zuhause im Wohnzimmer erreichbar sein. Durch die aktuelle Pandemie sind Kontakte eingeschränkt, Gruppentreffen finden nicht statt und die Menschen müssen zuhause bleiben. Um nicht zu vereinsamen und soziale Kontakte zu pflegen, musste vieles digital erfolgen.

## MindMapping

Noch nie war die Nachfrage so groß von Senior:innen so groß, ob sie ein Kurs für „Zoom“ oder andere Online Videoprogramme erlernen können. Technologien wird also immer mehr von Älteren angenommen. Auch das Bürgerhaus möchte den aktuellen Zeitgeist leben, und da sie nicht mehr viel persönlich mitteilen können, soll die App als Sprachrohr dienen. Uns war also klar: Die App soll nicht nur ein Stadtrundgang sein, sondern Information für die Bürger:innen unabhängig vom Standort leicht verfügbar machen.

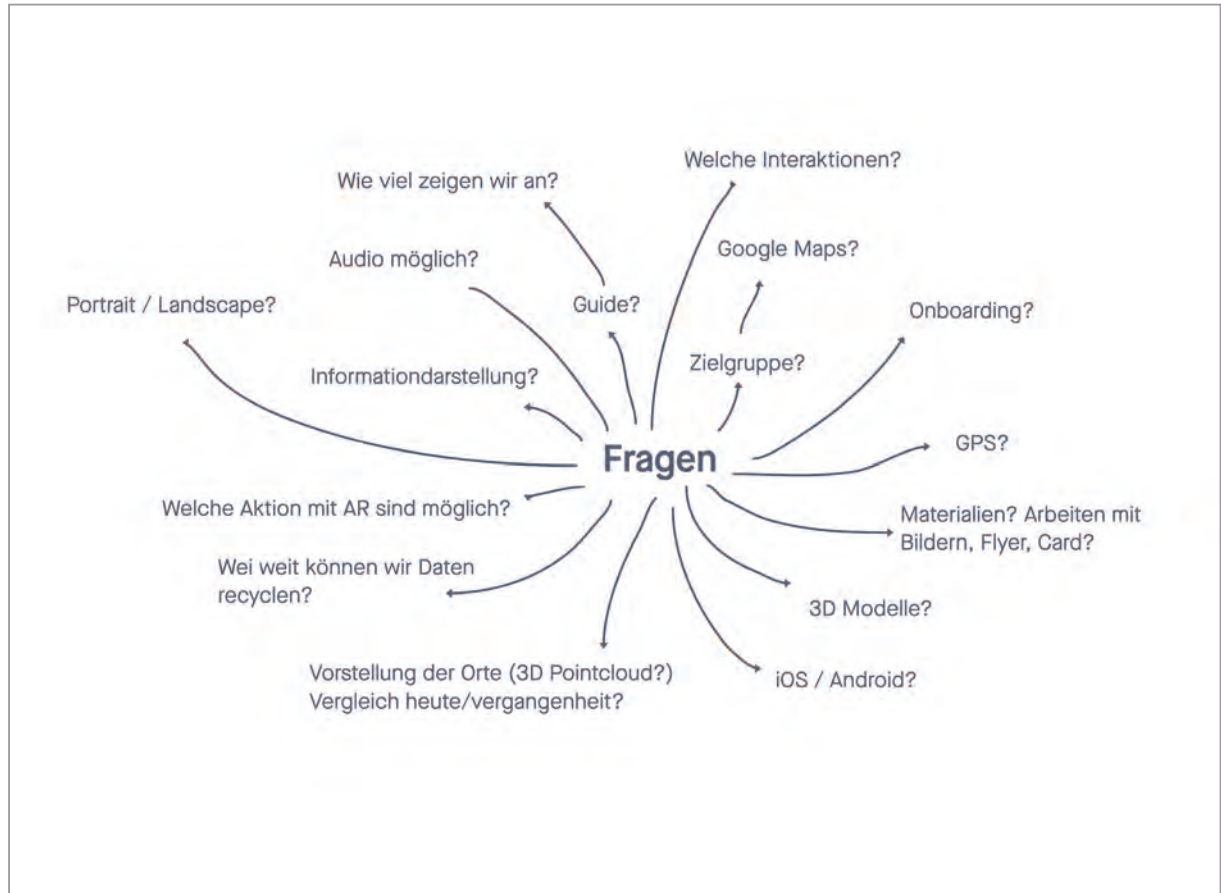
*„Wenn man der Pandemie was Positives abgewinnen will, dann ist es jetzt der Zwang, das sich ältere Menschen mit den digitalen Medien beschäftigen müssen“*

Stefan Markus, 2021

Mit dem ganzen Wissen haben wir mit Kreativitätsmethoden wie „Brainstorming“ und „Mind-Mapping“ gearbeitet. Dabei haben wir Ideen generiert und uns offene Fragen notiert. Ziel war es in kurzer Zeit viele Gedankenansätze zu entwickeln und danach die Machbarkeit einzuschätzen.

Nicht jede Idee hat das Potential, dass sie erfolgreich bei den Benutzer:innen ankommt. Deshalb haben wir diese Pläne mit einer Erfolgchance bewertet, bevor wir diese weiter konkretisiert haben. Unser oberstes Vorhaben ist es am Ende eine funktionsfähige erfolgreiche App zu realisieren. So waren unsere Hauptaugenmerke auf **Innovation**, zeitliche und technische **Machbarkeit** und verfügbare **Ressourcen** gerichtet. Nach diesen Kriterien haben wir potentielle Eingebungen gefiltert und weiter ausdefiniert.





MindMapping Visualisierung mit Fragestellungen, selbsterstellte Grafik

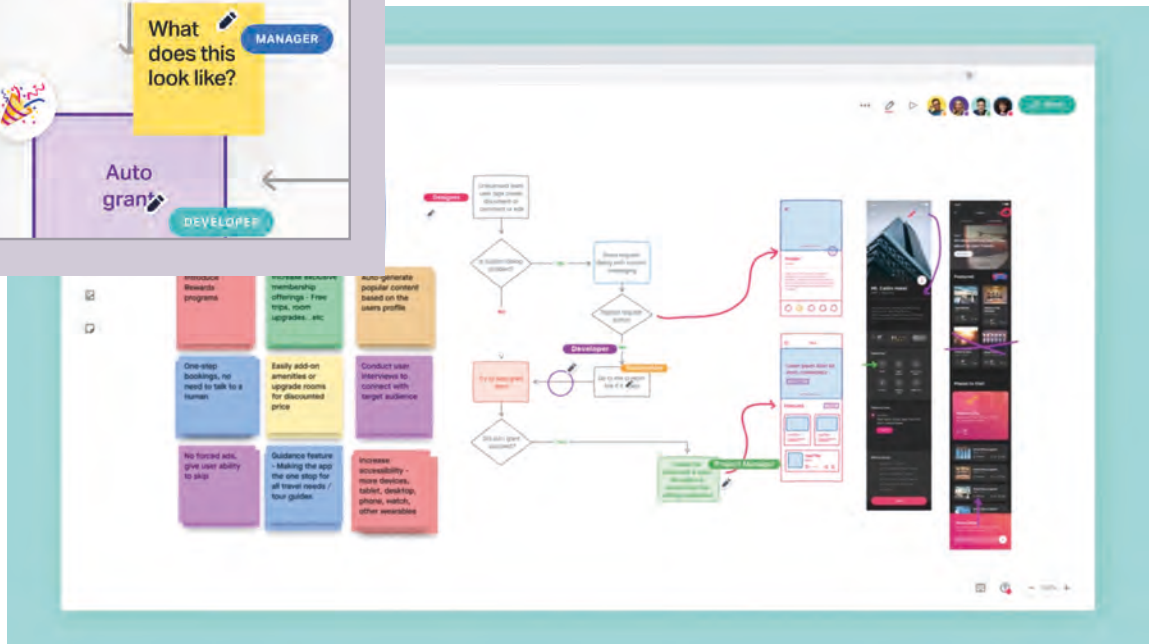
# Anforderungsmanagement

Da wir jetzt eine gute Übersicht haben, für wen und was das Produkt ist, war unser nächster Schritt die genauen **Anforderungen** zu definieren. Anforderungsmanagement ist eine wichtige Regelung in einem Projekt, die allgemein eine wesentliche Rolle spielt. Es wird sichergestellt, dass zu entwickelnde Produkt, im Einklang mit **Funktionalität** und **Qualität** den Ansprüchen von den Stakeholdern ausreicht. Dafür braucht man einen klaren Prozess. Das Management wird in zwei Teile unterteilt. Zuerst werden die Anforderungen dargelegt. Dabei geht es darum, alle relevanten Merkmale zu erheben und zu dokumentieren. Als Quellen der Anforderungen dienen die Produktvision, das technische Wissen, Marktrecherche, Experten-Interviews, Personas und alle weiteren Vorgaben. Im zweiten Schritt werden die gesammelten Rahmenbedingungen verständlich formuliert und in eine einheitliche Form gebracht. Bei der Aufbereitung haben wir nochmal alle Daten überprüft.

Die Dokumentation aller Arbeitsschritte fand mit einem Whiteboard Programm statt. „Invision“ ist so genannte Online Echtzeit Tafel die man endlos erweitern kann. Dort haben wir mit Text, Grafiken, Bilder und all unsere Daten zusammentragen und stetig ergänzt. Angefangen beim Brainstorming, bis hin zur Präsentation der aktuellen Stände, haben wir alles dort dokumentiert und veranschaulicht.



Invision, 2021



# Personas

Um die Ideen, die wir nun fokussiert hatten, auch richtig für die Zielgruppen zuschneiden zu können, haben wir als nächstes Personas angefertigt. Zusammen mit den Stakeholdern haben wir zwei fiktive Personen entwickelt (Primäre und sekundäre Persona), die stellvertretend für eine Gruppe von Benutzer:innen mit deren Bedürfnisse stehen. Primäre Personas repräsentieren die wichtigsten Benutzer:innen. Für diese wird in erster Linie das Produkt erstellt. Die Sekundären Personas stellen weitere wichtige Personengruppen da. Für Sie kann das Produkt weitere Funktionen beinhalten. Ziele und Verhaltensmuster werden aus Fakten und Diskussionen abgeleitet. Sie verleihen etwas Abstraktem ein Gesicht (vgl. Faily et al., 2011). Das hilft dem Team ein gemeinsames Verständnis für die Bedürfnisse der:die Benutzer:in zu bekommen. Folgende Merkmale sind beim Modellieren einer Persona wichtig:

- Name,**
- Beruf / Aufgaben**
- Demographische Angaben (Altersgruppe)**
- Fähigkeiten, Kenntnisse und Erfahrungen**
- Motivation, Hobbys**
- Ziele, Erwartungen und Wünsche an das Produkt**

Für diesen Prozess haben wir eine Vorlage mit Platzhaltern entwickelt und zusammen zwei finale Personas erarbeiten zu können.

## Beispiel Persona

The form is a template for creating a persona. It consists of the following elements:

- A square placeholder for a profile picture containing a question mark.
- A text input field labeled "Name" with three dots indicating a placeholder.
- A text input field labeled "Über die Persona" with three dots indicating a placeholder.
- A text input field labeled "Ziel / Motivation" with three dots indicating a placeholder.
- A text input field labeled "Konflikte / Probleme" with three dots indicating a placeholder.
- A list of demographic and professional attributes on the left side, each followed by three dots:
  - Beruf ...
  - Alter ...
  - Ort ...
  - Betriebssystem ...


Selbsterstellte Mustervorlage für die Personas

Christine

Konzeption


Die fertigerstellten Personas haben wir im großen Team überprüft, damit das Verständnis für alle bekannt ist. Dieses Modell hat großen Einfluss auf weitere Designentscheidungen und Eigenschaften des Produkts.

### Primäre Persona

	Name	Frau Meier	
	Über die Persona	Wohnt seit 1980 in Kattenturm	
<p><i>"Die Geschichte des BGO find ich total klasse und möchte das gerne meinen Enkeln zeigen"</i></p>	Ziel / Motivation	Möchte die Bilder der letzten Veranstaltung sehen	Skill-level
			Anfängerin
Beruf	Rentnerin		
Alter	73		
Ort	Kattenturm		
Betriebs-system	Android		
	Konflikte / Probleme	Frau Meier kann kleine Schrift nicht gut lesen. Außerdem fällt es ihr teilweise schwer zu verstehen wie man bei Apps wieder zum vorherigen Bildschirm zurückkommt und schließt dann häufig die komplette App um von vorne anzufangen.	

Ausgearbeitete Personas, selbsterstellte Grafiken

### Sekundäre Persona

	Name	Michael Ritter	
	Über die Persona	Verheiratet, zwei Kinder, wohnt seit vielen Jahren mit seiner Familie in Kattenturm	
<p>Beruf</p>	Versicherungs- kaufmann	Ziel / Motivation	Aufmerksam geworden? Über einen Post auf der Facebook-Seite des Bürgerhauses
	Alter	53	
Ort	Kattenturm		
Betriebs-system	Android		
	Konflikte / Probleme	Erwartungen: Hat Lust Kattenturm über die App neu kennenzulernen, die Inhalte sollten das kurzweilig, interessant und vielseitig aufbereitet werden (nicht zu viel Text, optisch ansprechend, am besten auch mit akustischen Hinweisen).	

# Navigationsplan

Die **Hauptfunktionen** der App haben wir nun festgehalten. Zum Teil soll der:die Benutzer:in interaktiv den Stadtteil erkunden und neue / alte Artefakte durch AR kennenlernen und dabei sich Informiert fühlen. Daneben soll die App auch die aktuellen Nachrichten und die nächsten Termine verlinken.

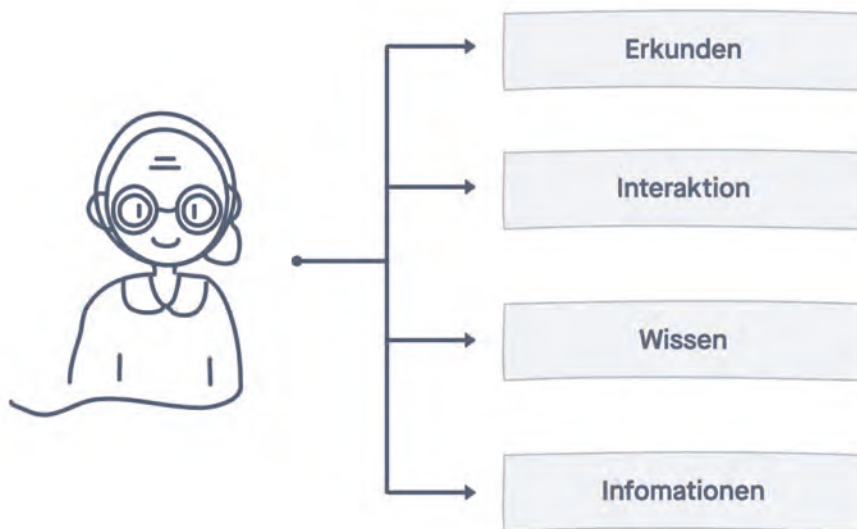
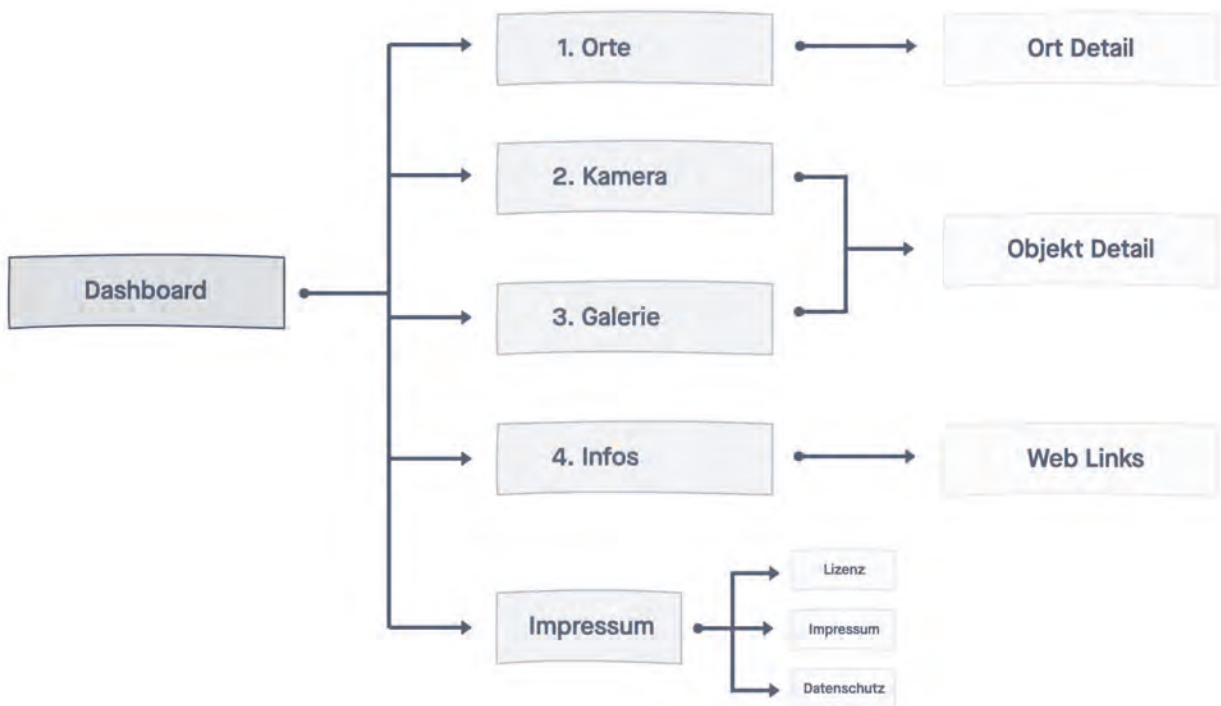


Abbildung der Anforderungen mit der zugehörigen Persona, selbsterstellte Grafik

In diesem Teil der Bachelorarbeit war der Kern die Kommunikation. Wir haben als Team viele kritische Fragen gestellt und versucht diese auch zu beantworten. Wir haben viele Ideen verworfen und einen Plan kreiert, damit wir endlich mit unserer Hauptarbeit starten können: Design und Entwicklung. Bis hierhin hat das ganze Team eng zusammengearbeitet. Als nächstes war es wichtig Arbeitspakete zu schnüren, um aus der Produktvision auch ein spannendes sehenswertes Produkt zu formen. Ab diesem Zeitpunkt haben wir parallel gearbeitet. Jacob hat begonnen die Software Architektur aufzusetzen. Ich (Christine) habe mich mit dem konzeptionellen Navigationsplan und mit dem Interface Design beschäftigt.

Ein Navigationsplan dokumentiert die Informationsarchitektur. Es stellt konzeptionell, ohne grafisches Design, die Verbindungen der einzelnen Inhaltsseiten dar. Der Plan zeigt einfach und verständlich die Struktur und den Umfang der geplanten App. Diesen Schritt haben wir wieder mit unseren Stakeholdern besprochen, ob es ihren Erwartungen für eine App entspricht. Durch diesen Plan kann sich Jacob eine technische Architektur erarbeiten und die Schnittstellen ableiten (Brancheau et al., 1989). Zum Erstellen eines Navigationsplan habe ich verschiedene Elemente verwendet. Inhaltsseiten sind in rechteckige Blöcke dargestellt, die mit einem Titel die Hauptfunktion beschreibt. Die Hauptpunkte sind mit eindeutigen Nummern beschriftet. Mit Grautönen werden die Hierarchien aufgezeigt. Die Pfeile zeigen den Navigationspfad an.



Navigationspfad der App, selbsterstellte Grafik

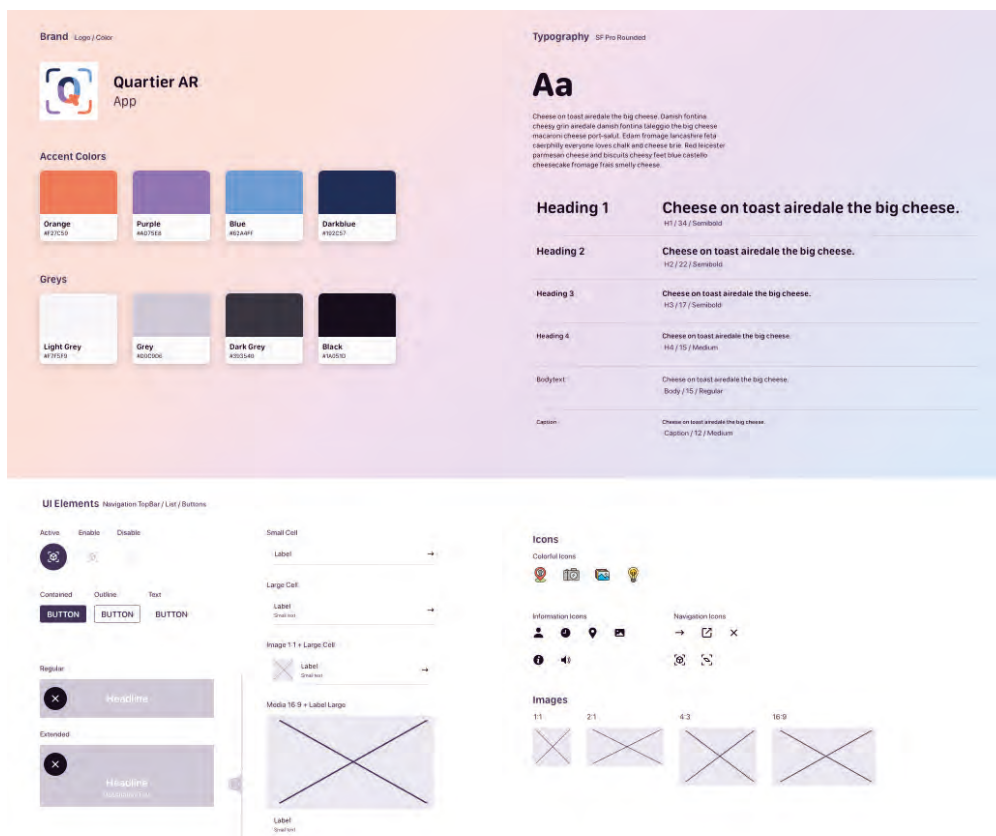
# User Interface



**Styleguide**  
**Touch-Gesten**  
**Elementgrößen**  
**Wireframes**

# Styleguide

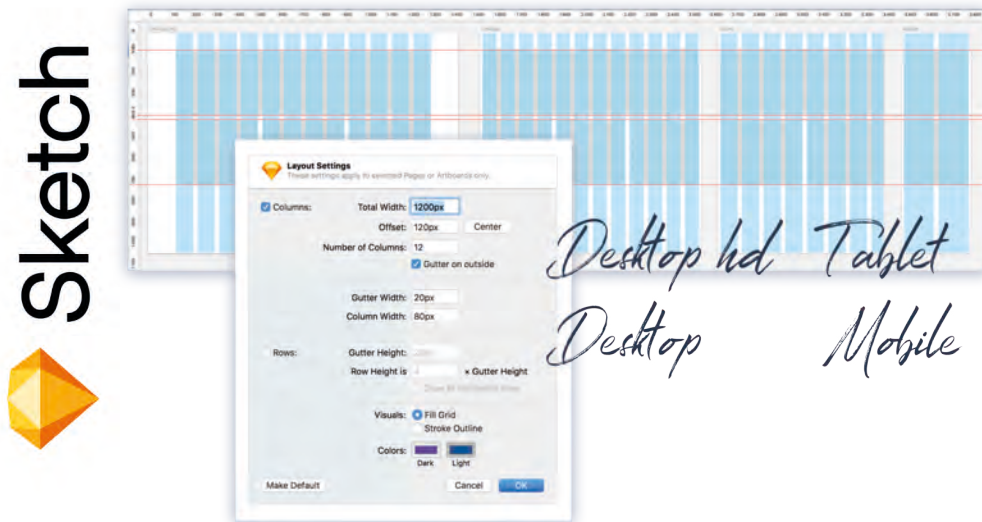
Der **Styleguide** verleiht einem Produkt seinen einmaligen Charakter. Es dient als Vorlage für die Umsetzung der UI Elemente und verweist auf alle Designelemente. Jede Designentscheidung ruht einer Reihe von komplexen Systemen und Überlegungen. Ein Styleguide beinhaltet die Navigation, Interaktionssystem / Wireframes, Formen, Farben, Symbole und Schriften. Um ein Design von vorne bis zum letzten Screen Konsistenz zu halten wird so ein Guide erstellt. Das Dokument wächst organisch mit. Die Ausarbeitung des Designs entsteht Schritt für Schritt (Moore & Arar, 2019).



Styleguide des Projektes, selbsterstellte Grafik

## Design Interface Programm

Als grafisches Programm habe ich „**Sketch**“ gewählt. Das Tool hat sich auf Interface Design spezialisiert, ähnlich wie bei Photoshop für Bildbearbeitung, ist hier Gestaltung von Benutzeroberflächen im Vordergrund. Es bietet eine große Auswahl an Mobilien Auflösungen und eine große Auswahl an vektorbasierten Werkzeugen an. Es ist strukturiert wie eine Front-End Entwicklung und Elemente können direkt mit zum Beispiel CSS/HEX Werte ausgegeben werden. (Sketch, 2021).



Sketch, 2021

## Touch-Gesten



Bei der Umsetzung der Screens, gibt es auch wie bei der Konzeption, Faktoren die vorher schon beachtet werden sollten. Ich habe mich mit Themen wie **Touch-Gesten** auf Smartphones und deren grafischen Auflösungen beschäftigt.

Ein Smartphone bedient man ausschließlich nur mit Touch. Hierzu gibt es wesentliche Unterschiede zur Bedienung auf Web-Oberflächen, die mit Maus und Tastatur benutzt werden. Der Interaktionsstil von Touch ist eine direkte Manipulation (Parhi et al., 2006). Ein Objekt wird mit dem Finger direkt berührt. Bei der Maus wird der Zeiger pixelgenau auf ein Element platziert und löst eine Aktion aus. Durch Hover-Effekte und Tool-Tipps wird nochmal signalisiert oder beschrieben, was auf der Aktion für ein Auslöser liegt. Bei Touch ist es reduzierter aufgebaut (keine Hover-Effekte) und muss daher schon optisch klar sein, was der jeweilige Button oder ein Icon auslöst.

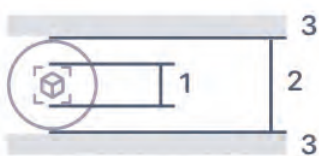
## Elementgrößen

Der Durchmesser einer Fingerkuppe beträgt durchschnittlich 15-20mm. Damit müssen die Aktions-Elemente deutlich größer sein als auf einem Computer Bildschirm. Die empirische Studie (Parhi, 2006) untersuchte die Thematik und hat folgende fünf Grundregeln zusammengetragen:

- **Normale Elemente** Minimalgröße von 9mm / 26px
- **Wichtige Elemente** deutlich größer als 9mm / 26px
- **Bei Ausnahmefällen** kann die Höhe 7mm/20px ausmachen aber dann muss das Element mindestens 15mm/44px breit sein.
- **Genügend Abstand** zwischen den Elementen erhöht die Treffsicherheit
- **Der visuelle Teil** eines Elements soll nicht kleiner als 4,2mm/12px (60% der Minimalgröße) sein. Sonst wird es nicht als Touch-Aktion wahrgenommen.

Die Elemente sind im Design nochmal für die Zielgruppe optimiert. Im Design gibt es größere Elemente und berührbare Fläche mit breiteren Abständen.

Minimalegröße laut empirischer Studie



- 1 - Sichtbare Größe / 12px
- 2 - Berührbare Größe / 26px
- 3 - Abstand / 6px

Elementgröße in der Quartier App



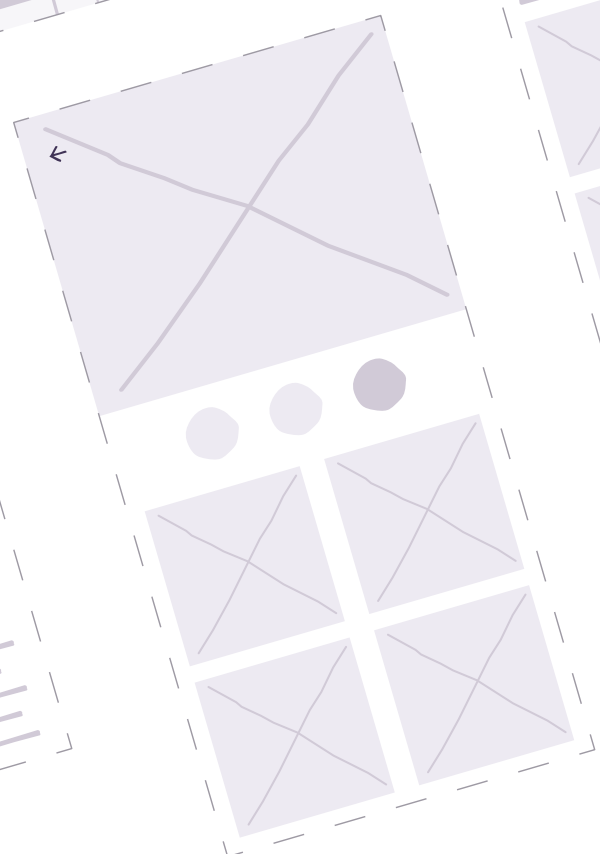
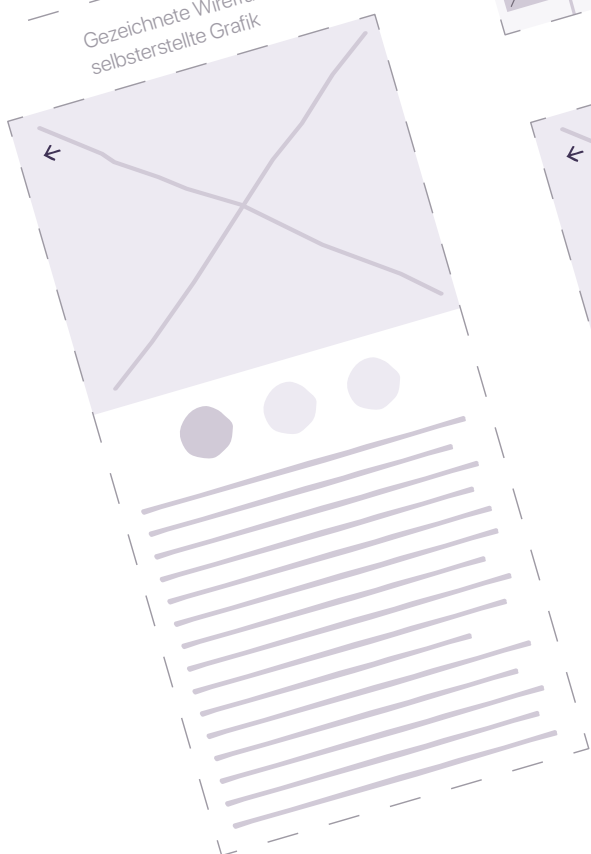
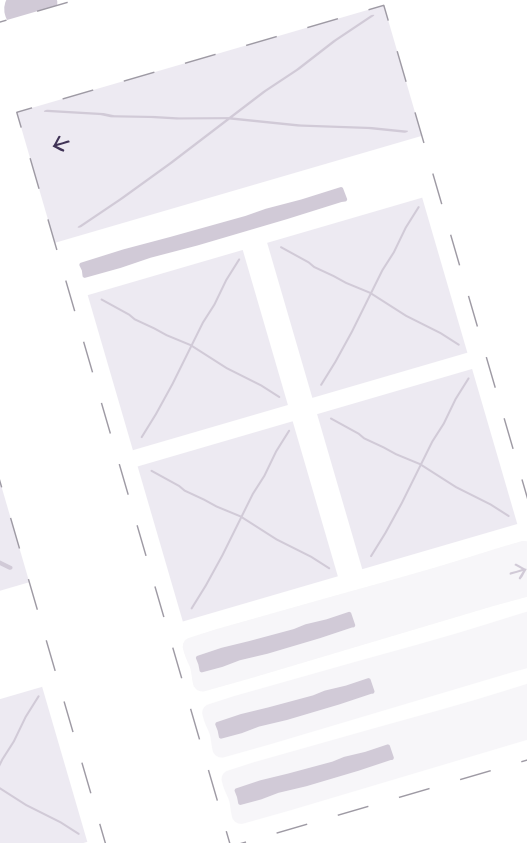
- 1 - Sichtbare Größe / 24px
- 2 - Berührbare Größe / 48px
- 3 - Abstand / 8px

Berührbare Elemente im Vergleich, selbsterstellte Grafik

# Wireframes



Gezeichnete Wireframes der App,  
selbsterstellte Grafik



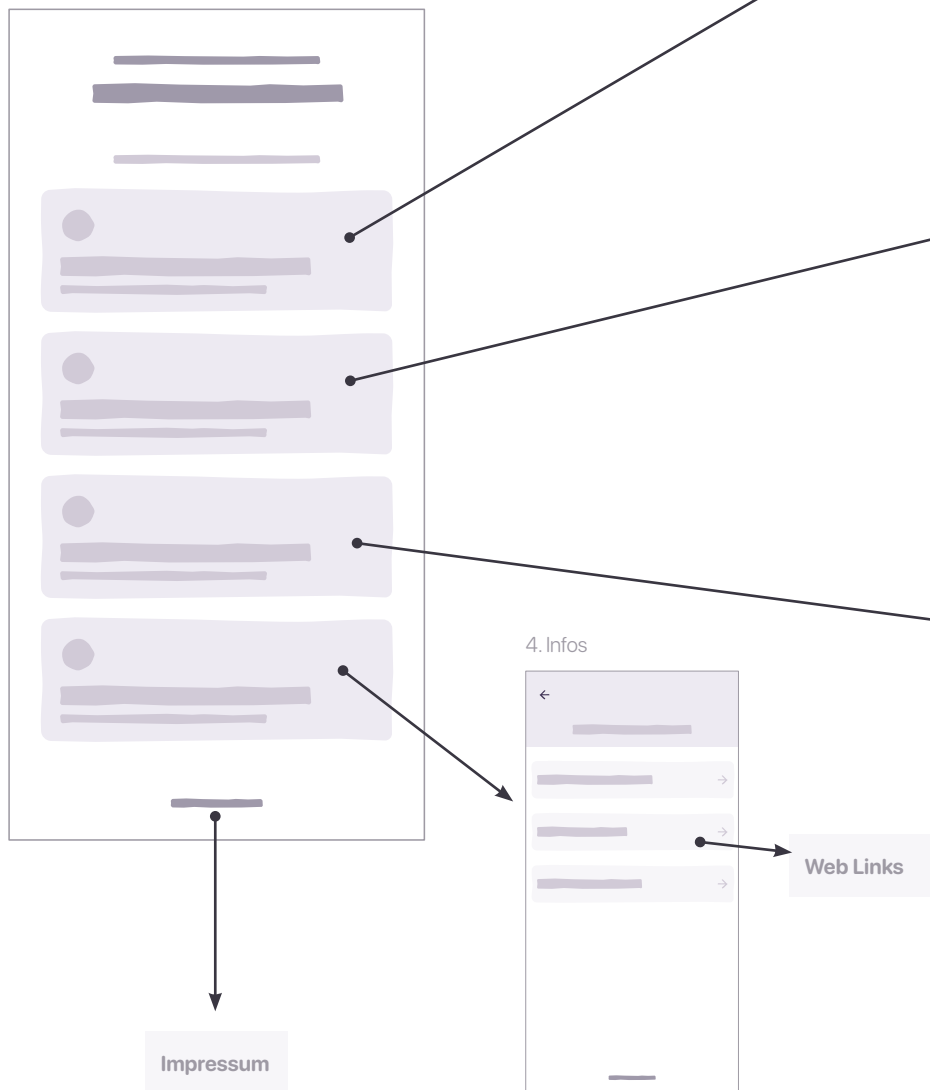


Als erstes, bevor ich mit den Designelementen, wie Farb-, Bild und Schriftwelten beginne, habe ich **Wireframes** händisch gezeichnet und digital ausgearbeitet. Wireframes (auf Deutsch Drahtgerüst) ist eine reduzierte Darstellungsform der Screens mit den wichtigsten Bausteinen. Details wie Icons oder Bilder werden nur Schemenhaft aufgezeichnet. Text Elemente sind als graue Blöcke dargestellt. Titel sind in größeren Rechtecken und Fließtext in dünneren Linien gezeichnet. Icons und Symbole, die ich noch optisch definieren muss, sind als Ovale zu sehen. Die nativen Navigationselemente wie der „Pfeil Zurück“ habe ich schon platziert, weil die Funktion und Position gleichbleibt. Platzhalter für Bilder sind in einer gängigen Darstellung als Rechteck mit zwei durchgezogenen Linien, die sich diagonal durchkreuzen.

## Interaktionsmodell

Mit den Wireframes habe ich ein **Interaktionsmodell** erstellt. Eine Erweiterung der Navigationsstruktur mit einfachen Screens. Dieses Modell ist der Dialog zwischen Benutzer:in und der Gliederung. Durch die reduzierte Darstellung kann der:die Betrachter:in möglichst schnell Erkenntnisse gewinnen.

### Dashboard

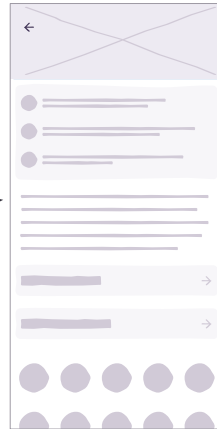




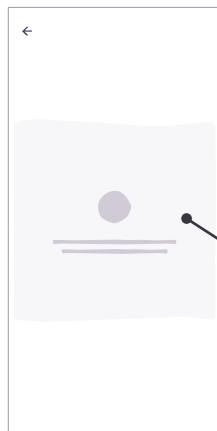
1. Orte



Ort Detail



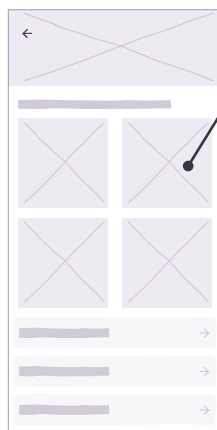
2. Kamera



Artefakt Detail



3. Galerie



Christine

Visuelles Interaktionsmodell / Navigation der App, Selbsterstelle Grafik

User Interface

# Visuelles Design

Die **visuelle Gestaltung** der Benutzeroberfläche ist die Schnittstelle von System zum Endnutzer:in. Die gezeichneten Wireframes werden in diesem Schritt gestylt. Für die Geräte Auflösung habe ich eine Android Standardgröße von 360px x 760px gewählt, dies entspricht ein Samsung Galaxy S10 (Design, o. J.). Beim Aufbau der Screens, habe ich mich an die **Android Material Guidelines** orientiert. Mit Hilfe von Rastern, Abständen und Anordnungen habe ich die App mit Farben, Formen, Schriften, Symbolen und Bilder gefüllt. Ein Design bezweckt aber nicht nur, dass etwas gut aussieht, sondern kann auch die **Usability** (deutsch: Benutzerfreundlichkeit) verbessern. Durch Ordnung und Struktur kann die Orientierung gelenkt werden. Farbkontraste erhöhen die Lesbarkeit und erregen Aufmerksamkeit. Symbole und Bilder können Texte ersetzen und reduzieren eine komplexe Informationsflut.

**Farben**  
**Typographie**  
**Icons**  
**Elemente**

# Farben

Die psychologische Wirkung der **Farbe** auf Menschen ist nicht zu unterschätzen. Die Verknüpfung der Farbe entsteht durch verschiedenen Erlebnisse wie aus der Natur, Kindheit oder Religion. So spielt bei der Gestaltung die Farbauswahl auch eine bedeutende Rolle. Durch Farbe lenke ich die Aufmerksamkeit auf die wesentlichen Punkte. Zudem unterstreicht die Farbgebung den Charakter des Produkts und verstärkt das Erlebnis. Es verleiht der App die visuelle Identität (Gegenfurtner & Goldstein, 2014). Für das Design habe ich ein Farbkonzept entwickelt. Das bedeutet, ich habe eine Liste von **Primären** und **Sekundären** Farben definiert, die ich durchgängig im Design verwende. Alle Farben habe ich als RGB und **HEX-Code** Format angegeben (Paletton - The Color Scheme Designer, o. J.).

Bei der Farbauswahl achte ich darauf, dass diese gut aufeinander abgestimmt sind und miteinander harmonieren, aber sich auch klar voneinander abgrenzen. Dabei habe ich mir die Farbkombination Regeln des „**Komplementärkontrast**“ und den „**tetradischen Farbklang**“ zu Nutze gemacht. Der Komplementärkontrast verstärkt die Leuchtkraft der einzelnen Farben. Die beiden Werte liegen im Farbkreis fast gegenüber. Der tetradische Farbklang beinhaltet zwei Paare von komplementären Farben. Es wird eine primäre Farbe entwickelt und die anderen werden abgestuft durch Helligkeit oder Sättigung manipuliert, um die erste Farbe zu unterstützen (Farben in Religion und Kultur, o. J.). Bei der Entstehung der App Farbwelt habe ich freie Hand und bin an kein Corporate Design gebunden. Für jede Thematik/Menüpunkt in der App habe ich mich für eine individuelle Farbe entschieden, die deren Handlung unterstreichen soll. Die Bedeutung der ausgewählten Farben gehe im Screen-Design „Dashboard“ weiter ein.

Accent Colors



**Orange**  
#F27C59



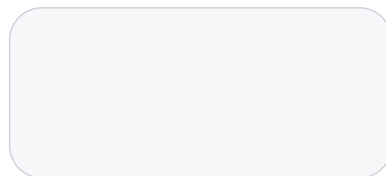
**Purple**  
#AD75E8



**Blue**  
#62A4FF



**Darkblue**  
#192C57



**Light Grey**  
#F7F5F9



**Grey**  
#D0C9D6

Shades



**Darkgrey**  
#393540



**Black**  
#1A051D

# Aa

Cheese on toast airedale the big cheese. Danish fontina  
cheesy grin airedale danish fontina taleggio the big cheese  
macaroni cheese port-salut. Edam fromage lancashire feta  
caerphilly everyone loves chalk and cheese brie. Red leicester  
parmesan cheese and biscuits cheesy feet blue castello  
cheesecake fromage frais smelly cheese.

## Heading 1

**Cheese on toast airedale the big cheese.**

H1 / 34 / Semibold

## Heading 2

**Cheese on toast airedale the big cheese.**

H2 / 22 / Semibold

## Heading 3

**Cheese on toast airedale the big cheese.**

H3 / 17 / Semibold

## Heading 4

**Cheese on toast airedale the big cheese.**

H4 / 15 / Medium

## Bodytext

Cheese on toast airedale the big cheese.

Body / 15 / Regular

## Caption

Cheese on toast airedale the big cheese.

Caption / 12 / Medium

Aufstellung des Schriftstils, selbsterstellte Grafik

# Typographie

Die **Schrift** ist neben den Farben ebenfalls elementar für das Design. Texte müssen leicht zu lesen sein und sollten zum ästhetischen Bild der App passen. Jede Schriftart bekommt durch Proportionen, Formen, Strichstärken und Serifen seinen individuellen Charakter. Viele Schriften gibt es in unterschiedlichen Ausführungen. Fett, kursiv oder schmal sind gängige Schriftschnitte.

Bei der Schriftwahl habe ich darauf geachtet, dass sie gut lesbar auf Smartphone Bildschirmen ist und zur App passen. San Francisco Compact Rounded ist abgerundet und wirkt dadurch organisch und verspielt. Dadurch passt diese Schrift optisch sehr gut ins Bild und zu der Thematik. Diese Schriftart Familie „**San Francisco**“ wurde extra für Bildschirme entwickelt. Die besitzt keine Serifen und hat eine konstante Strichstärke, die die Lesbarkeit auch auf kleinen Geräten garantiert. Die Schriftgröße ist in der App ist größer als der Standard damit die ältere Zielgruppe auch den Text lesen kann. Überschriften wie H1, H2, H3, H4 gliedern den Text und bieten einen guten Einstiegspunkt. Je wichtiger die Überschrift, desto größer auch der Schriftgröße. Die Schriftart ist im Web frei verfügbar und ich habe diese als **.otf Format** in das **Unity Projekt** importiert. So ist die Schrift in der App mit implementiert und wird auf jedem Smartphone richtig angezeigt.

# Icons

**Icons** können für unterschiedliche Ziele verwendet werden. Den Zweck der Icons ordnet man mit der **Pragmatik** ein:

- **Indikativ** – weißt auf etwas hin (Beispiel: Fast leerer Akku)
- **Suggestiv** – empfiehlt eine Handlung (Beispiel: Zeichen auf Lebensmittel zwecks Mülltrennung von Plastik/Papier)
- **Imperativ** – gibt ein Verhalten vor (Beispiel: Ein Schild mit Geschwindigkeitsbegrenzung im Straßenverkehr)

Die **Sigmatik** gibt die Beziehung zum Zeichen an.

Den Sachverhalt kann man auch in drei unterschiedlichen Weisen aufklären.

- **Icon** - ist eine einfache Abbildung eines Gegenstandes (Beispiel: Lupe steht stellvertretend für die Suche)
- **Symbol** - was stellvertretend für das Bezeichnete steht (Beispiel: Kreuz für Religion)
- **Index** - zeigt eine räumliche oder zeitliche Nähe auf (Beispiel: Ein Rauch Bild als Zeichen für Feuer) (Moser, 2012).

Auch in der App verwenden wir Icons, um den Text zu reduzieren, optisch spannender und die Bedeutung schneller greifbarer zu machen. Bei der grafischen Umsetzung (**die Syntaktik**) habe ich darauf geachtet, dass die Icons gewissermaßen prägnant und leicht zu verstehen sind. Stilmittel wie Form, Farbe, Position und Material macht ein Icon aus. Bis auf die Menüpunkt Icons, habe ich die Icons einfarbig mit ausgefüllten Flächen gestaltet. Das bietet den nötigen Kontrast. Alle Ecken und Linien sind wie bei der Schrift abgerundet, damit ein einheitliches Bild erschaffen wird. Auf Perspektive und detaillierte Realitätstreue habe ich verzichtet, um die Grafiken nicht unnötig kompliziert zu machen.



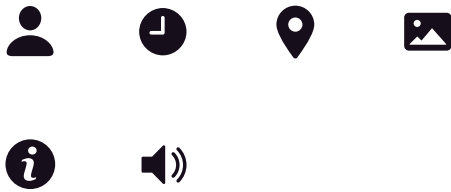
## Icons

### Colorful Icons

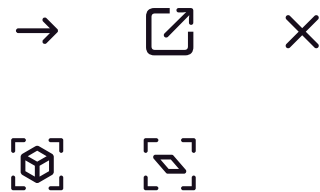


flaticon, 2021

### Information Icons



### Navigation Icons



Icons, selbsterstellte Grafiken

Optisch unterscheide ich, ob das Icon gewisse Information anzeigt oder als unterstützendes Navigations-Element aufgeführt wird. Der Pfeil und das Kreuz sind auf dem Screen wiederkehrende Navigationspunkte und etwas filigraner gezeichnet, als die flächigen Informationselemente. Die Icons sind in Sketch als Vektoren entworfen und ich habe sie alle als PNG's exportiert und in Unity eingefügt. Die Icons sind alle im quadratischen Format angelegt und die Auflösung beträgt 24 x 24px, wie ich im Kapitel „Elementgröße“ schon erwähnt habe. Bei den farbigen Icons mache ich eine Ausnahme mit der Auflösung von 32 x 32px, weil diese mehr Details und Farbigkeit besitzen und durch eine größere Darstellung die Sichtbarkeit garantieren.

## UI Elements Navigation TopBar / List / Buttons

Active    Enable    Disable



Contained    Outline    Text

**BUTTON**    **BUTTON**    **BUTTON**

Regular



Extended



Small Cell



Large Cell

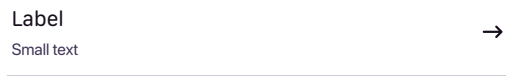
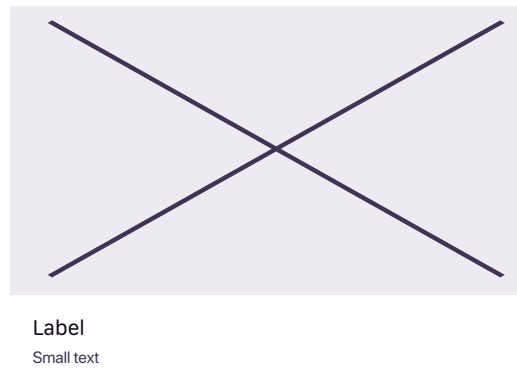


Image 1:1 + Large Cell



Media 16:9 + Label Large



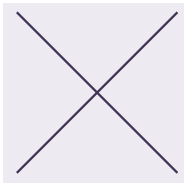
UI Elemente, selbsterstellte Grafiken

# Elemente

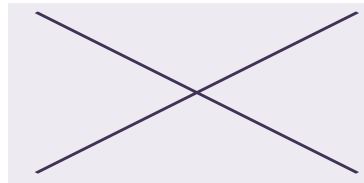
Den Kopfbereich, Buttons, Bildformate und Zellen habe ich in derselben Weise dargestellt, um die Design Sprache weiterhin konsistent zu halten. Gewisse Elemente habe ich für unterschiedliche Statute wie aktiv, selektiert und deaktiviert erstellt, die vorkommen können.

## Images

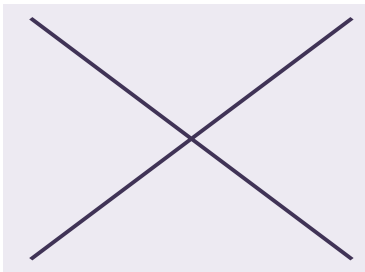
1:1



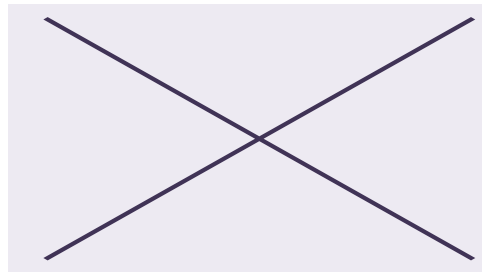
2:1



4:3



16:9



# App Design

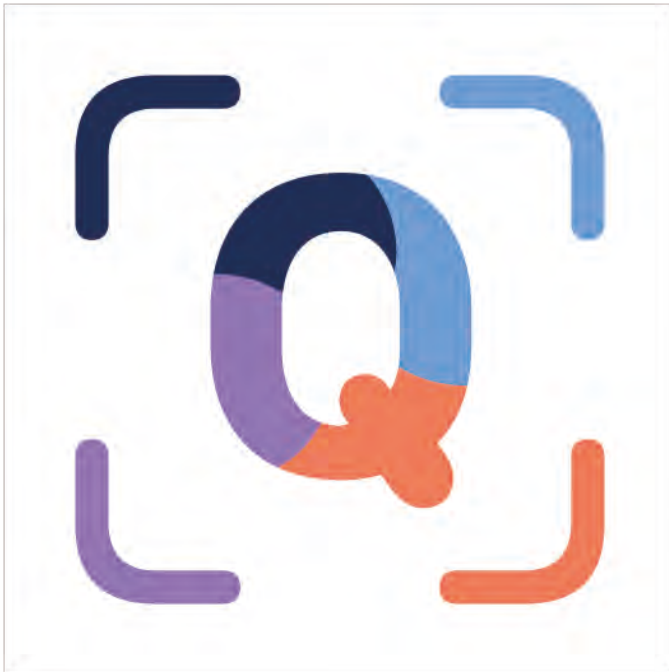
**App Icon**

**Prototype**

**Produktmerkmale**

**Klick-Dummy**

## App Icon



App Icon der App, selbsterstellte Grafik



Das **App Icon** ist ein wichtiger Bestandteil der App. Es kommuniziert mit einer einzigen Grafik die Kernfunktionen und die Identität. Bevor der:die Benutzer:in die App im Playstore runterladen sehen sie dieses Icon. Wie bei den normalen Icons, ist es hier ebenso wichtig nicht zu komplex zu werden. Es sollte auf den ersten Blick freundlich und einfach wirken (designenlassen, o. J.).

Bis das finale Icon fertig war, habe ich auch hier viel ausprobiert und kombiniert. Mir ist es wichtig, die Farbgebung und die Aktion in einer kombinierten Weise aufzugreifen. Zuerst habe ich nur mit den Farben gespielt, aber das war mir nicht Aussagekräftig genug. Das finale Icon ist eine spielerische Verknüpfung aus Wort, Aktion und Farbgebung geworden, die sich auf jedem Format sehen lassen kann. Aufgrund der aufkommenden Kosten bei der Entwicklung für Apple Plattformen (macOS, iOS), haben wir vorerst nur für eine Umsetzung der App für das Android System entschieden. Je nach Endgerät können die App Icons anders dargestellt werden. Formen wie Quadrate, Kreis, vertikale und horizontale Rechtecke sind möglich (Design, o. J.). Das ist das sogenannte „Adaptiv Icon“. Um alle Formen abdecken zu können, wird ein Vordergrund und ein Hintergrund benötigt. Der Hintergrund ist einfarbig und wird der gewünschten Form zugeschnitten. Der Vordergrund beinhaltet die Vektorgrafik und in der Größe etwas variiert, damit es immer vollständig angezeigt und nicht abgeschnitten wird.



App Icon Prozess, selbsterstellte Grafik

# Prototype

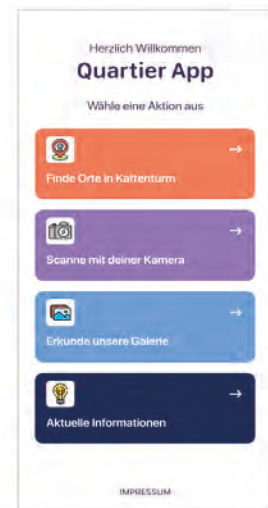
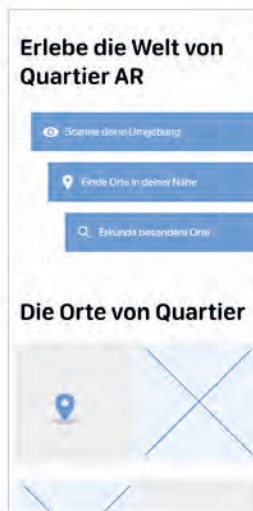
Bis zum fertigen Interface Design ist ein langer Prozess. Die Ausarbeitung eines ganzen Designsystems ist eine komplexe Aufgabe. Als Grundlage habe ich mich an Gestaltgesetze orientiert. Viele gestalterische Ansätze wurden verworfen, technologische Machbarkeit musste stets geprüft werden und die Stakeholder wollen auch Einfluss nehmen. Hinzu kommt noch, dass es bei einem visuellen Design kein Richtig oder Falsch gibt, es ist nur eine Frage des Stils. Schrittweise habe ich die einzelnen Designelemente ausgearbeitet (Garrett, 2010). In unterschiedlichen Versionen habe ich Text und Bild angeordnet um die Stimmung einzufangen.

In dieser Grafik zeige ich auf, wie zum Beispiel der Findungsprozess des Dashboards (der Hauptnavigationspunkt der App) entstanden ist. Den ersten Entwurf habe ich nach einer Iteration verworfen, weil nicht alle Menüpunkte sofort ersichtlich sind. Am Ende habe ich mich für ein vertikales Button Menü entschieden.

Erste Entwürfe....



....bis zum finalen Design



Designprozess vom Dashboard, selbstgestellte Grafik





## Die Screens

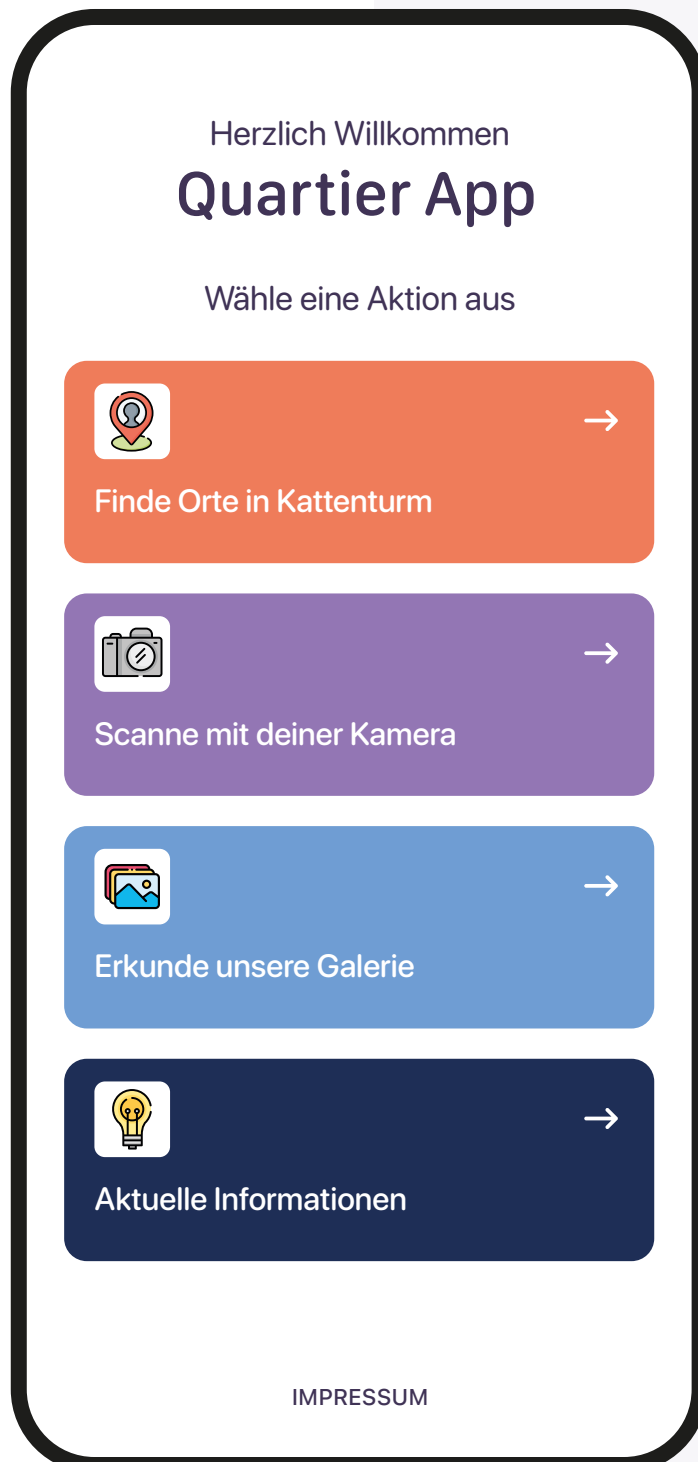
Auf den folgenden Seiten präsentiere ich die App Screens in der Navigationspfad Reihenfolge und erkläre den Hintergrund zu meinen gestalterischen Entscheidungen. Es beginnt mit dem Splashscreen und endet mit dem Impressum.

### Der Splashscreen

Wenn der:die Benutzer:in die App startet, müssen viele Daten geladen werden. Um diese Zeit zu überbrücken, wird oft ein **Splashscreen** angewendet. Es ist sozusagen der Startbildschirm der App. Um die Aufmerksamkeitsspanne zu erhalten wird während der Ladezeit ein Bild angezeigt. Abhängig vom Endgerät kann die Ladezeit abweichen. Es gibt eine Faustregel die besagt, dass der Startbildschirm unter 3 Sekunden von der:die Betrachter:in erfasst werden soll (badmin, 2019). Daher zeigt der Splashscreen nur wenige Informationen an. Ich habe mich für den Projektnamen mit Untertitel und die farbigen Menü-Icons entschieden. Das lockert die Situation auf und der:die Benutzer:in erkennt die Icons im Dashboard wieder.



Splashscreen der App



Das Dashboard ist der Einstieg in die App

# Dashboard

Das **Dashboard** ist der Einstieg der App. Es ist der erste funktionale Screen, den der:die Benutzer:in sieht, wenn man die auf das App Icon klickt. Um auf den ersten Blick den:die Benutzer:in nicht mit Informationen zu überfluten, habe ich ein einfachen Aufbau des Menüs gewählt. Die vier Menüpunkte sind vertikal untereinander platziert und farblich abgegrenzt. Auf jedem Button ist ein Icon, Titel und Navigationspunkt. Auf einem weißen Panel ist das farbige Icon zentriert positioniert. Die Icons sollen die Aktion des Buttons verdeutlichen. Wenn der:die Benutzer:in es nicht dadurch einschätzen kann, beschreibt der Titel kurz die Interaktion. Der Pfeil rechts suggeriert, dass ein neuer Screen dadurch geöffnet wird. Die ersten beiden Punkte sind Interaktionen, die nur Ortsgebunden funktionieren. Um die Vielfalt der App zu entdecken, muss der:die Benutzer:in die Orte erkunden und Bilder/Gegenstände mit der Smartphone Kamera scannen. Damit die App durchgehend genutzt wird, sind die zwei unteren Menüpunkte unabhängig von Ort und Zeit benutzbar. Sie verweisen auf die Artefakte und Programminformationen, was für die Bürger:innen interessant sein könnte.

## Bedeutungen der Farben



### Orange

Die Natürliche Assoziation ist die Farbe ein Symbol für Wärme, Freude und Aktivität. Orange ist kraftvoller als Gelb und wird oft als Warnfarbe verwendet. Diese Farbe erregt Aufmerksamkeit.



### Purple

Auch Violett genannt, steht für Blüten, Mystik, Spiritualität und Macht. Spirituell stellt sie auch die Fusion zwischen von Körper und Himmel dar.



### Blau / Dunkelblau

Diese kühle Farbe strahlt Ruhe, Vertrauen und Seriosität aus. In orientalischen Ländern steht die Farbe für die Unsterblichkeit. Sehr häufig wird Blau als Informationsfarbe verwendet.



### Grau

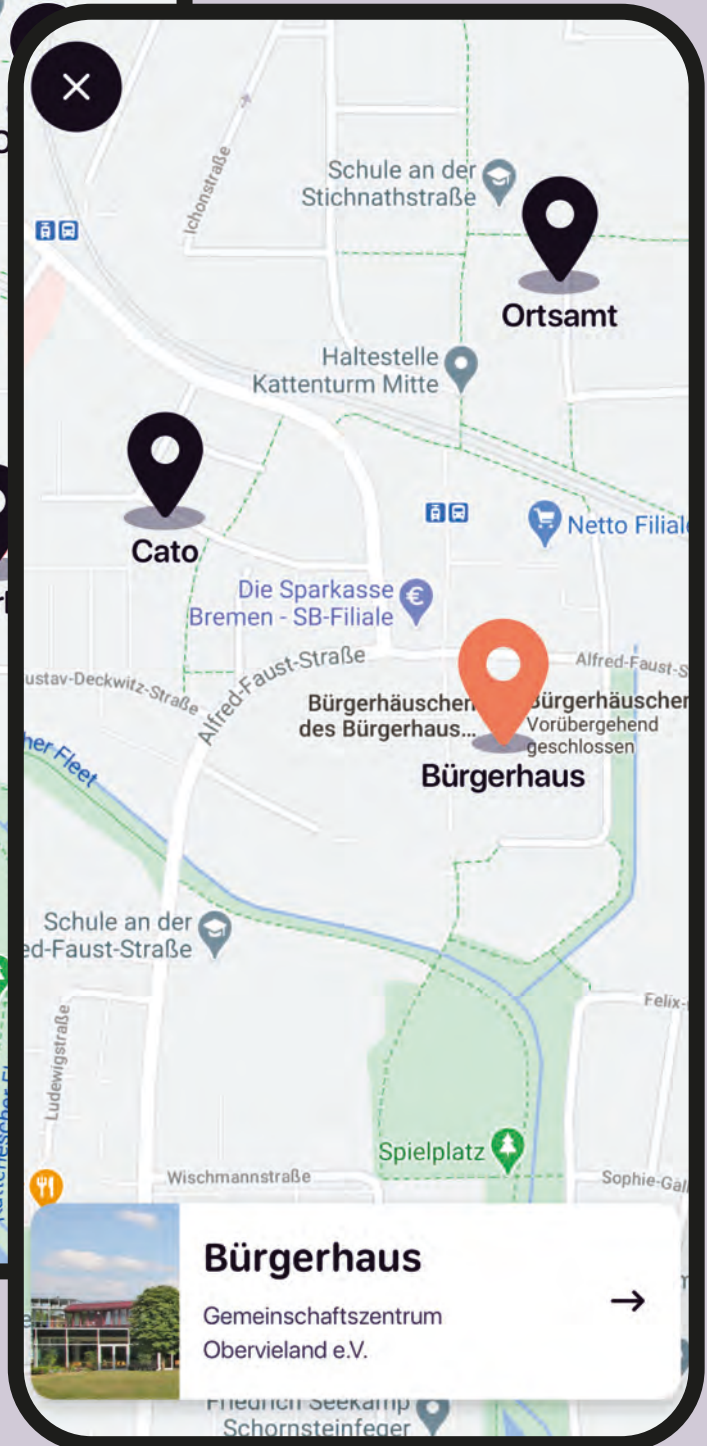
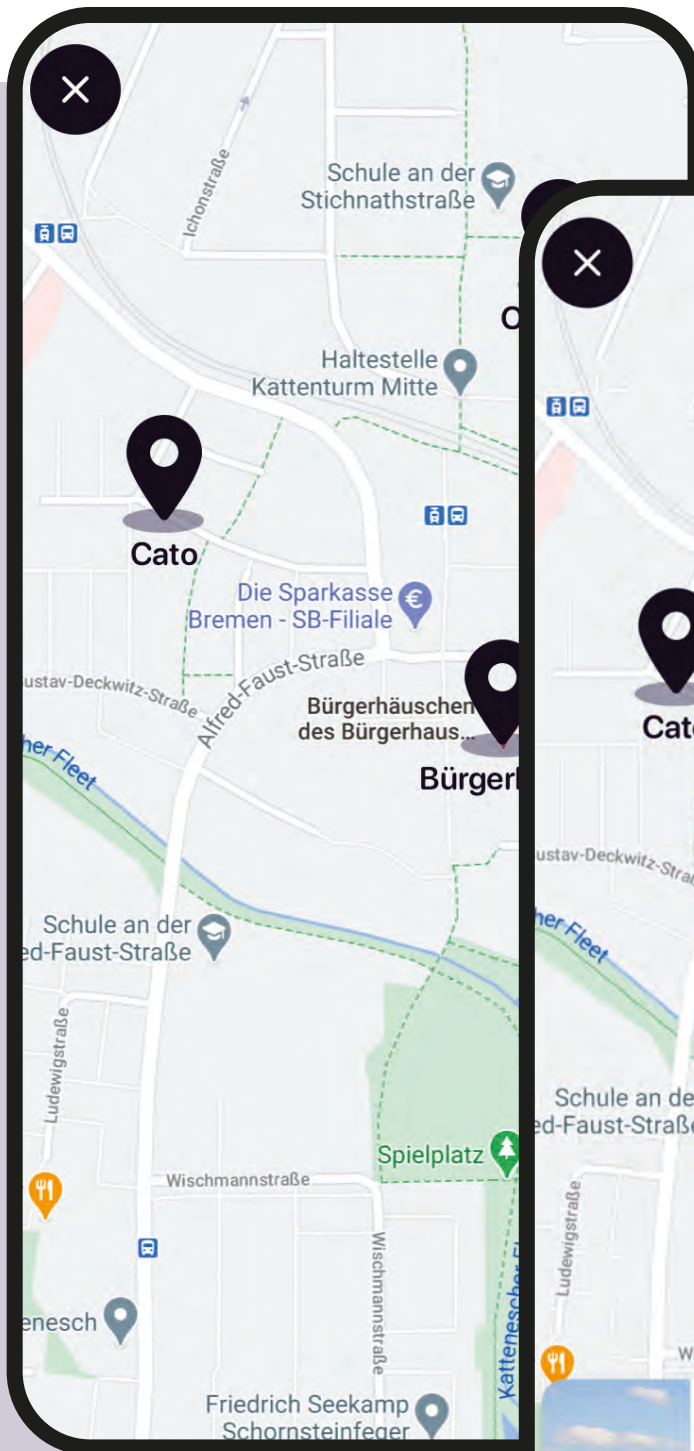
Um die Schrift oder optisch etwas zu trennen, habe ich zu den Farben zusätzlich auch Grautöne definiert. Grau wirkt neutral, formell und sachlich. Bei der Auswahl habe ich darauf geachtet, auch wenn es neutrale Töne sind, sie warm und wohlrig wirken zu lassen.

# Menüpunkt 01

## Die Stadtkarte von Kattenturm

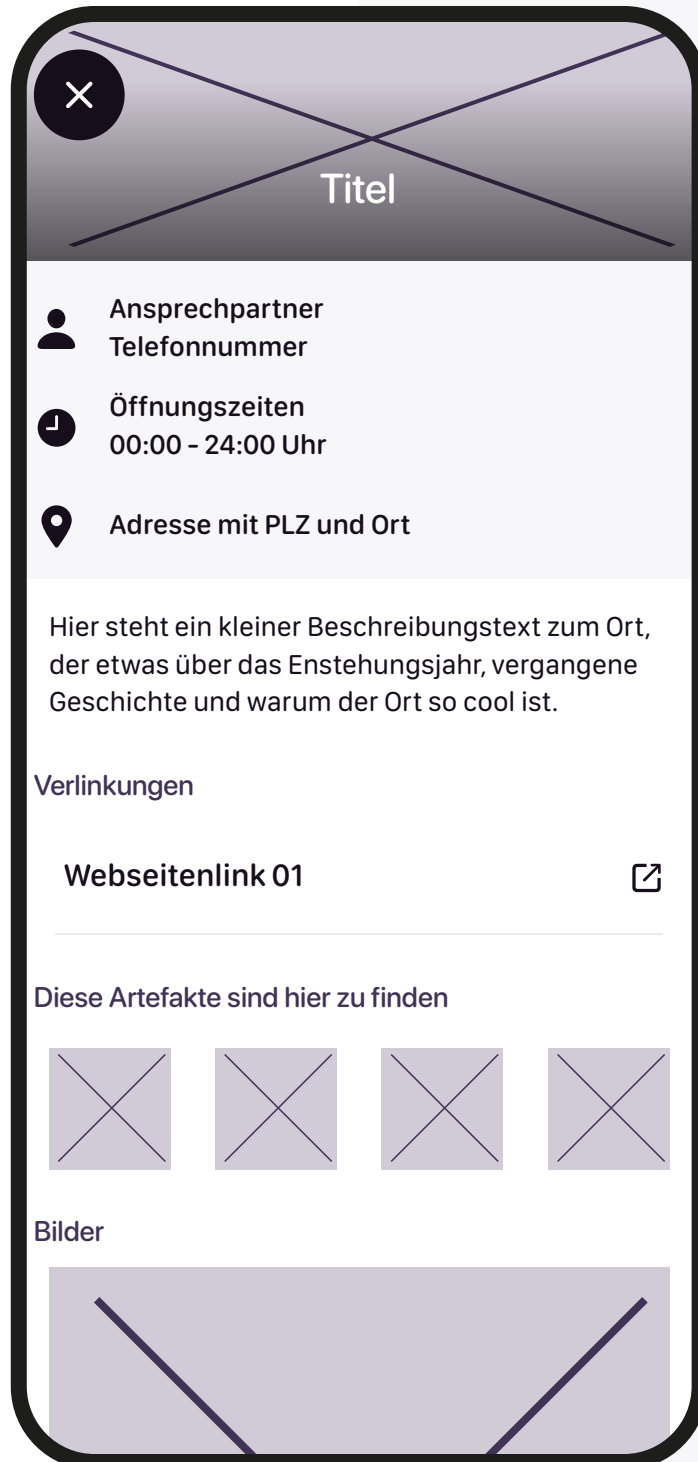
Wer schon mal Google Maps oder andere Navigations-Apps geöffnet hat, kennt das Prinzip. Es ist die Stadtkarte von Kattenturm dargestellt. Die Karte geht über den Screen hinaus. Durch Touch-Gesten, wie links oder rechts ziehen, wird die Karte erweitert und ein weitere Straßen werden sichtbar. Die Map-Pin Icons mit zusätzlichen Titel, zeigt die Orte, dies es aktuell in der App zu finden sind. Wir haben uns auf drei Orte konzentriert. Das Bürgerhaus, das Denkmal „Cato Bontjes van Beek“ und das Ortsamt. Wenn der:die Benutzer:in auf ein Map-Pin Icon klickt, wird dieser farblich selektiert und zeigt eine kleine Info-Panel an, die ankündigen soll, was sich dahinter verbirgt. Das Info-Panel zeigt ein entsprechendes Bild zum Ort an, den Titel und eine kurze Beschreibung. Rechts ist wieder der Navigationspfeil zu sehen, der anzeigt, dass dadurch ein neuer Screen geöffnet wird.





Erster Menüpunkt: Finde Orte, selbsterstellte Grafik

Stadtkarte Google Maps, 2021



Default Frame für den Ort, selbsterstellte Grafik



# Ortsdetail

Dieser Screen beinhaltet viele Information zum Ort. Im Kopfbereich befinden sich ein Bild, was im vorherigen Screen schon angekündigt wurde. Darauf folgt farblich abgehoben und mit Informations-Icon versehen, die Ansprechpartner:innen, Öffnungszeiten und nochmal die genaue Adresse. Darunter gibt es einen kleinen Beschreibungstext, der knapp formuliert wird. Mit Headlines werden die nächsten Themen auf dem Screen präsentiert. Verlinkungen, AR Artefakte und Bilder zum Ort werden untereinander angezeigt.

## Vorlagen erstellen

Ohne echte Bilder kann man sich so einen Screen kaum richtig vorstellen. Da es jedoch unser Ziel ist, eine App zu entwickeln, die echte Inhalte besitzt, habe ich eine Vorlage erarbeitet. So konnten sich die Stakeholder Zeit lassen und in Ruhe mit der Vorlage alle benötigten Inhalte raussuchen.

Nachdem Sie die Vorlagen ausgefüllt und mir alle Dateien geschickt haben, konnte ich schon alles in meine Designs fließen lassen. Das gab mir die Bestätigung, ob die Screens auch mit echten Daten funktionieren und noch ästhetisch wirken.

**Vorlage - Ort**

Bild für den Kopfbereich  
(jpg. png)

Name des Ortes  
...

Ansprechpartner      Öffnungszeiten      Adresse  
...      (wenn vorhanden)      ...

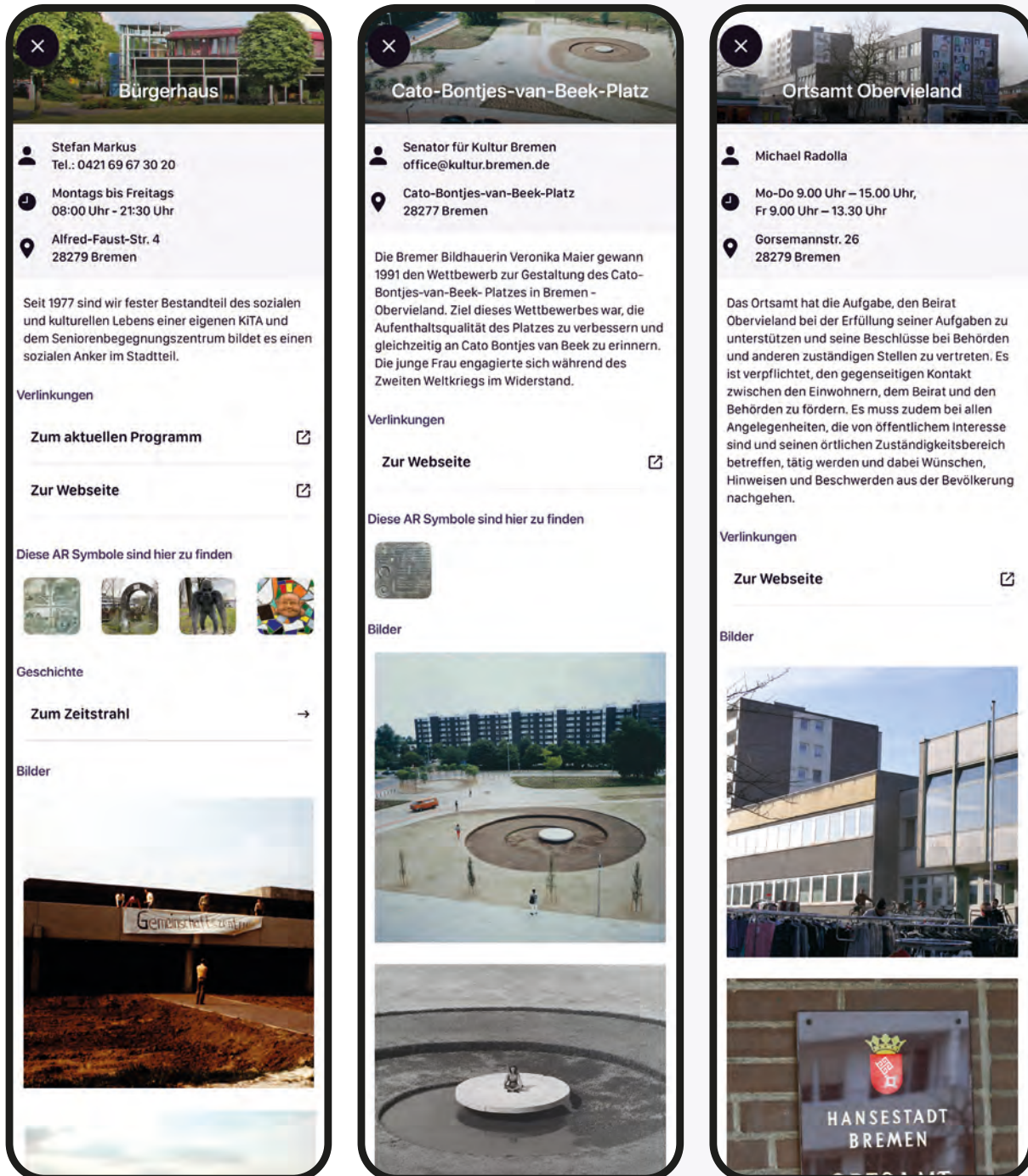
Beschreibungstext  
...

Weitere Bilder zum Ort

Bild-01.jpg    Bild-02.jpg    Bild-03.jpg    Bild-04.jpg    Bild-05.jpg

Selbsterstellte Mustervorlage für die Orte, selbsterstellte Grafik

# Ortsdetails mit Inhalt



Screens der Orte, selbsterstellte Grafik

# Individuelle Erweiterungen

Dem Bürgerhaus ist es wichtig, dessen Zeitwandel festzuhalten. Aktuell haben sie alle relevanten Geschehnisse auf einer übergroßen Plakatwand händisch aufgeschrieben. Da haben wir uns entschieden, diesen "Zeitstrahl" zusätzlich zu digitalisieren. So passt es ebenfalls zum modernen Zeitgeist. Im Ortsdetail „BGO“ gibt es eine Verlinkung mit dem Titel „Geschichte“, die auf den Zeitstrahl verweist. Bei der Digitalisierung haben die Stakeholder netterweise alle Geschehnisse nochmal Textuell zusammengefasst. So konnte ich mich auf das gestalterische fokussieren. Aus technischen Gründen haben wir uns entschieden, die App stets im Portrait Modus (Hochkant) zu lassen. Somit wird auch der Zeitstrahl horizontal vorgelegt. Die Jahreszahlen sind als kleine farbigen Flaggen gezeichnet, damit diese direkt ins Auge fallen. Die Farbe habe ich im Wechselspiel angeordnet damit, wie bei einer klassischen Tabelle auch, die Zuordnung schneller erfasst werden kann. Wichtige Ereignisse werden mit einem Titel gekennzeichnet und die entsprechende Information dazu in kleineren Fließtexten darunter angezeigt.



Der Zeitstrahl vom BGO, selbsterstellte Grafik



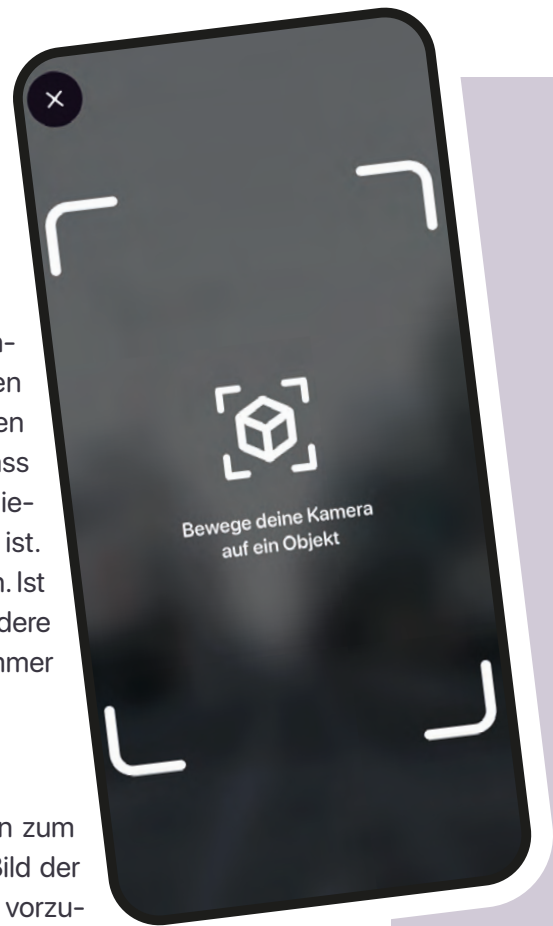
## Menüpunkt 02

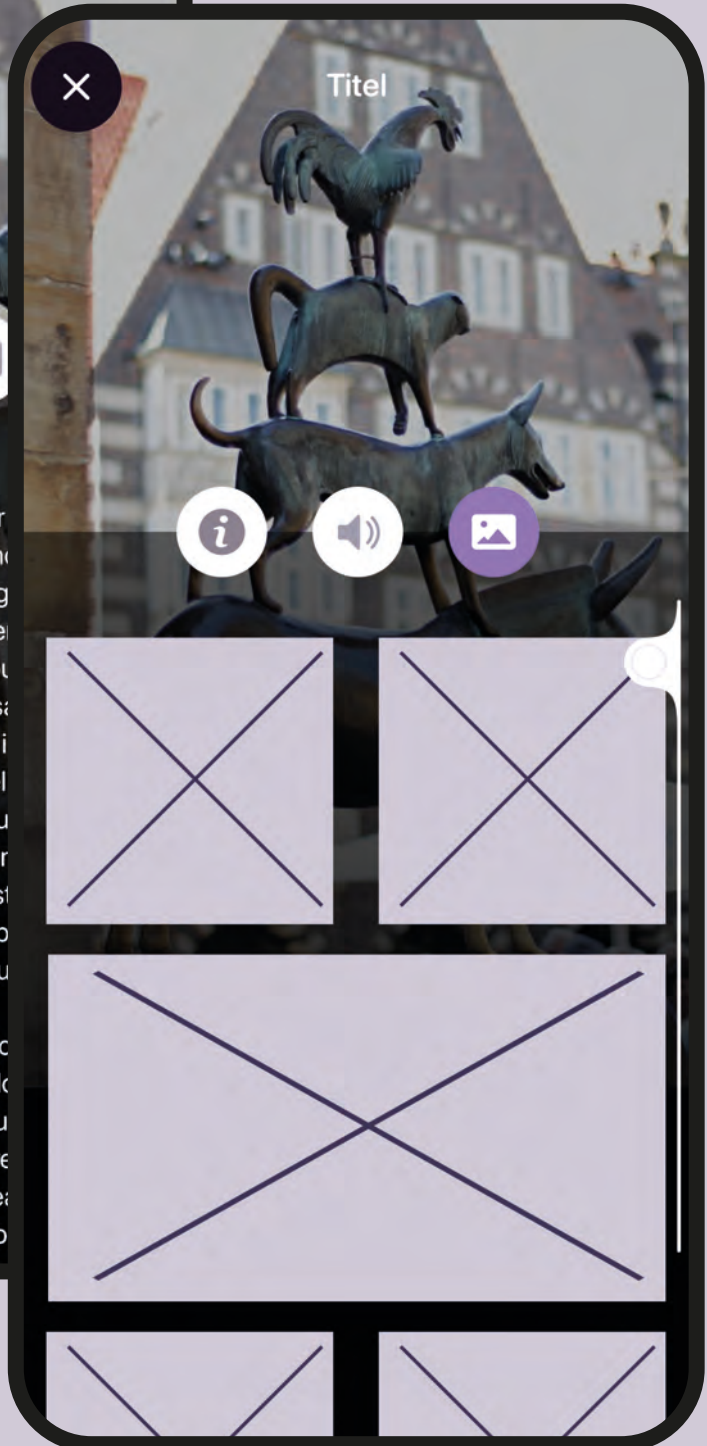
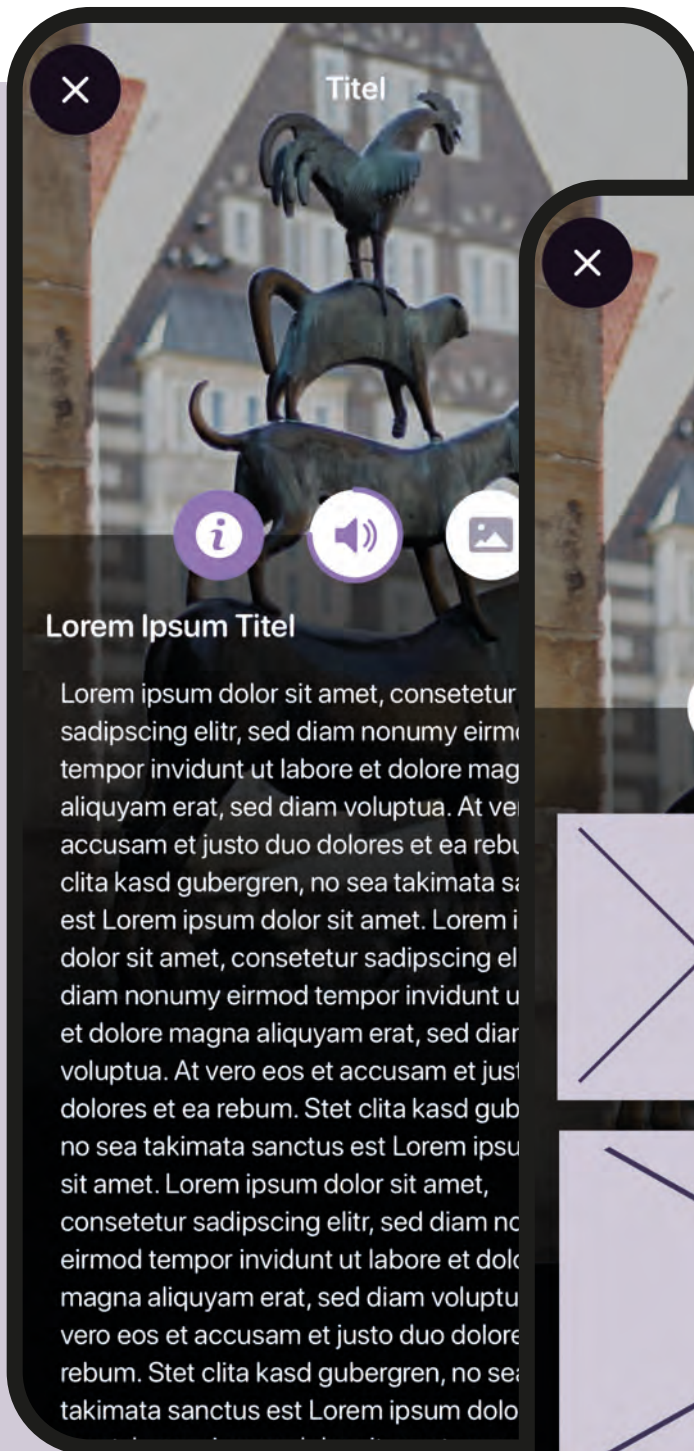
### Scanne deine Umgebung

Der zweite Button ist, wie schon erwähnt auch ein ortsgebundener Interaktionspunkt. Wenn der:die Benutzer:in auf den zweiten Menüpunkt klickt, öffnet sich die Kamera. Nun können AR-Artefakte gescannt werden. Da wir davon ausgehen, dass der:die Benutzer:in sich erstmal richtig zum Objekt positionieren muss, teilen wir textuell auf dem Screen mit, was zu tun ist. Mit einem Navigations-Icon zeigen wir beispielhaft die Aktion. Ist kein Artefakt in der Nähe oder man möchte doch lieber andere Funktionen der App anschauen, gelingt der:die Benutzer immer über das Kreuz oben links zum vorherigen Screen zurück.

### Artefakt Detail

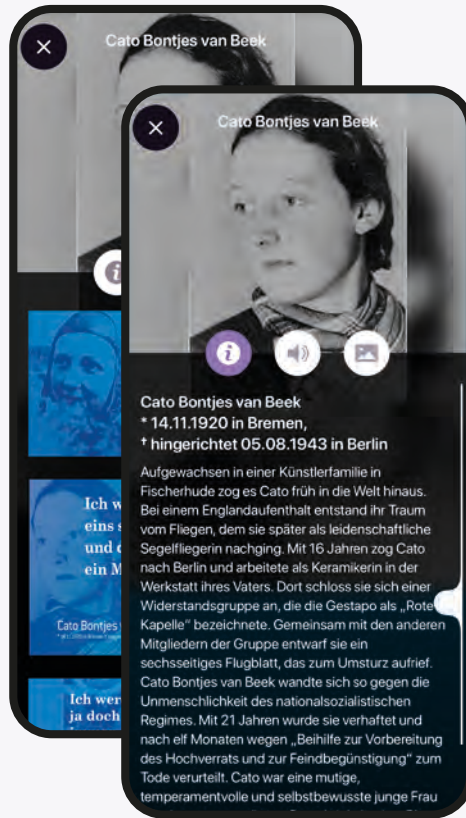
Ist der Scann Vorgang erfolgreich, erscheint ein Info-Screen zum Artefakt. In diesem Screen habe ich mit einem Platzhalter Bild der Bremer Stadtmusikanten gearbeitet, um sich das besser vorzustellen. Der Inhalt ist über dem Artefakt auf einen halbtransparenten schwarzen Overlay platziert. Damit ist alles leserlich aber das Objekt wird nicht vollständig verdeckt. Die drei Informations-Icons auf dem Overlay zeigen an, was es hier zu entdecken gibt. Der erste Punkt ist die reine Textinformation zum Artefakt. Falls die Schrift zu klein oder der:die Benutzer:in nicht bereit ist den Text zu lesen, zeigt der zweite Punkt ein Audio-Symbol an, der die Informationen vorspielt. Dafür haben die Stakeholder den Text extra eingesprochen. Ähnlich wie bei einer Musik-App zeigen wir an, wie viel Hörfortschritt noch bleibt, indem ein Kreis um das Icon zuläuft. Weitere spannende Bilder zum Artefakt und deren Geschichte wird im dritten Punkt angezeigt. Wie bei dem Screen der Orte, habe ich auch hier eine Vorlage für die Stakeholder erstellt, um die passenden Inhalte zu erhalten.



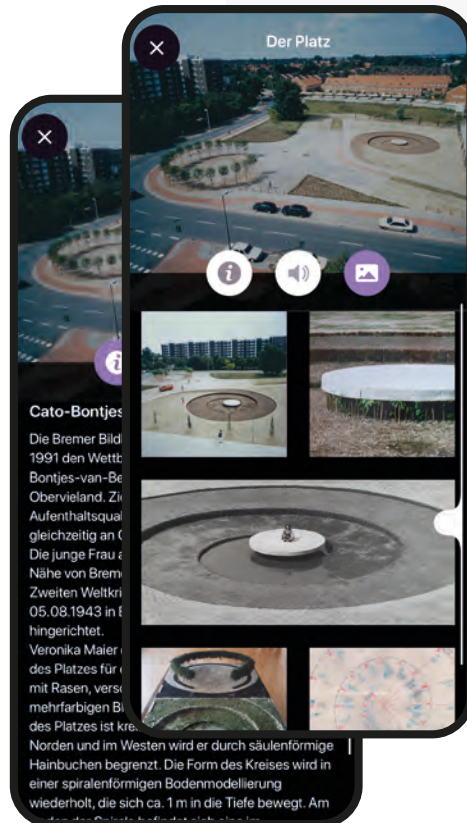


Default Frame für ein Artefakt, selbsterstellte Grafik, Bild Bremerstadtmusikanten, 2021

# Echte Artefakte mit Inhalt



Screens von Artefakten,  
selbsterstellte Grafik  
Bilder BGO





Hier zeige ich Beispiele der Artefakte mit echten Inhalten von einem historischen Mosaik-Gemälde im Bürgerzentrum, der Cato-Bontjes-van-Beek-Platz und der Friedenskämpferin Cato Bontjes van Beek.

Wie bei dem Screen der Orte, habe ich auch hier eine **Vorlage** für die Stakeholder erstellt, um die passenden Inhalte zu erhalten.

Vorlage - Material / Objekt

Bild vom Objekt  
(jpg, png)

Name des Objekts  
...

Beschreibungstext  
...

Weitere Bilder

Bild-01.jpg   Bild-02.jpg   Bild-03.jpg   Bild-04.jpg   Bild-05.jpg

Selbsterstellte Mustervorlage für die Artefakte,  
selbsterstellte Grafik

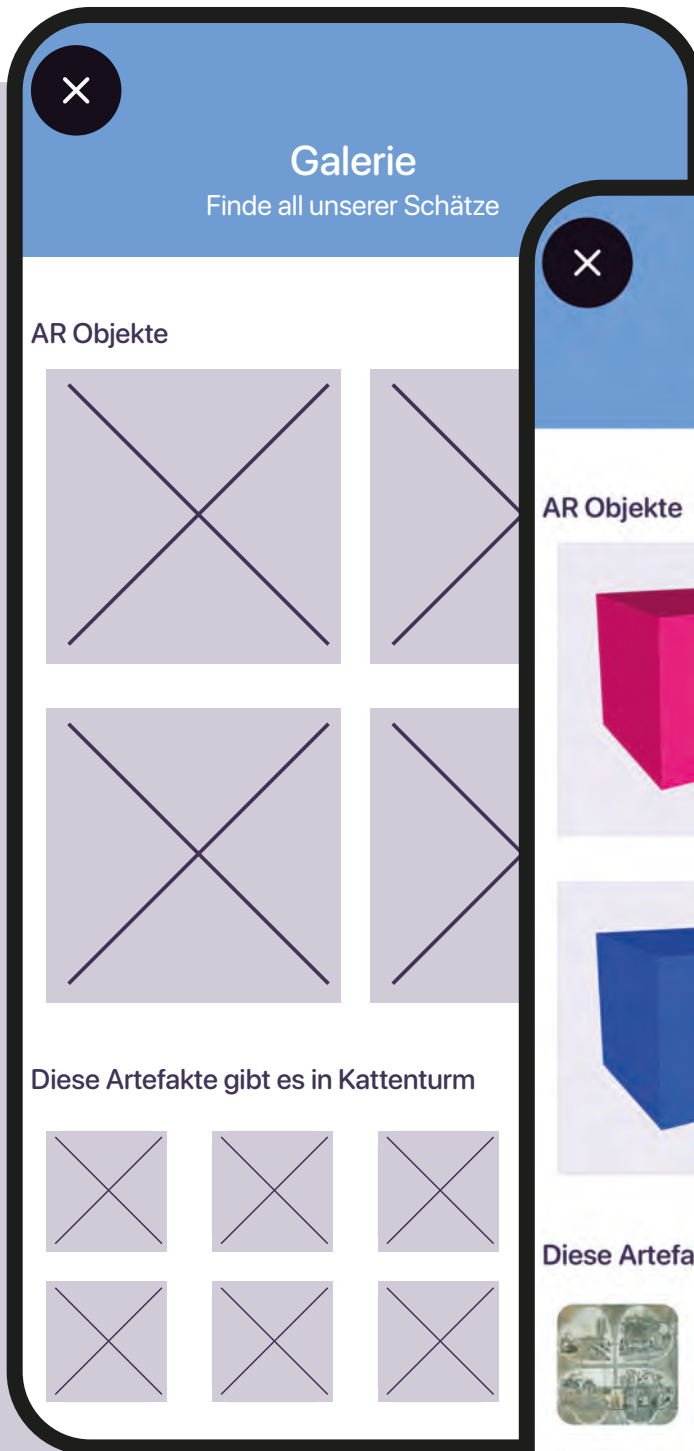
## Menüpunkt 03

### Erkunde unsere Galerie

Das ist der erste Menüpunkt von beiden ortsunabhängigen Buttons. Hier werden alle Inhalte nochmal bildlich dargestellt, um den ganzen Umfang der App auf einen Blick erfassen zu können. Die gescannten 3D Objekte werden als erstes angezeigt. Da der Aufwand für solche Objekte sehr groß ist, wird es im Piloten nicht viele davon geben. Dieser Bereich ist erstmal mit Platzhaltern gefüllt und wird erst in der Phase nach der Bachelorarbeit erarbeitet. Durch die Zusammenarbeit mit den Stakeholdern konnten wir viele Artefakte rund um Kattenturm definieren, die meisten im Bürgerhaus selbst. Weil die Anzahl so groß ist, habe ich mich für eine kleine Bildgröße entschieden, damit der:die Benutzer:in auf einem Blick die Vielfalt der Artefakte erfassen kann.







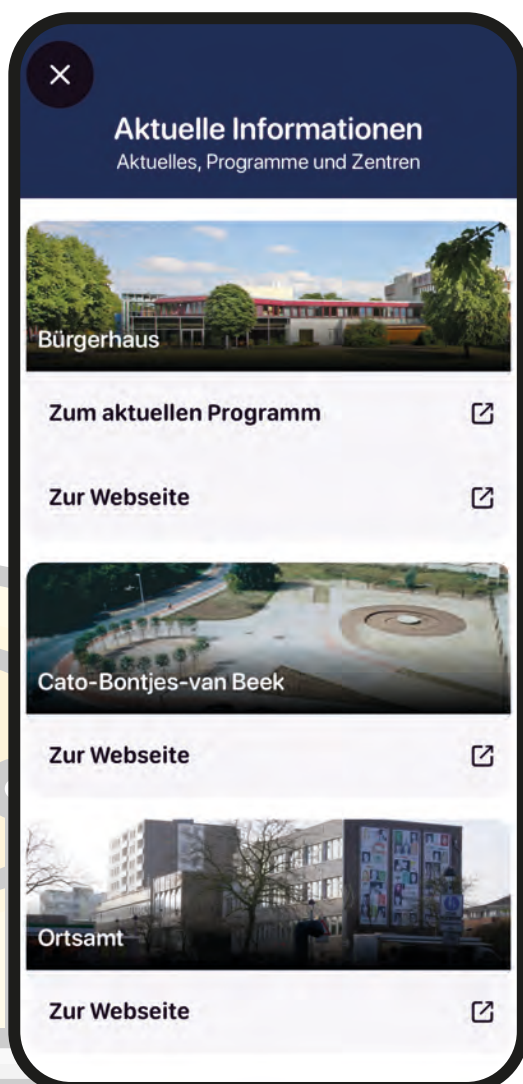
Erster Mneüpunkt: Finde Orte, selbsterstellte Grafik



# Menüpunkt 04

## Aktuelle Informationen

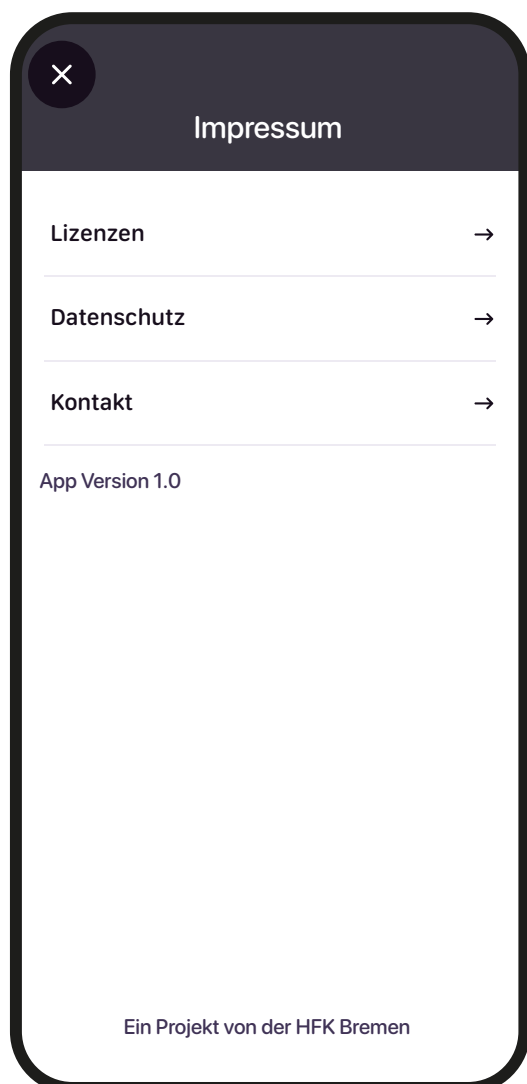
Der letzte Punkt listet alle Orte auf und bietet deren externe Links, die beispielsweise zu aktuellen Programmen oder Webseiten führen. Hier kann der:die Benutzer:in durchgehend aktuelle Informationen zum Stadtteil erhalten. Durch ein jeweiliges Headerbild, gewinnt man sofort die Erkenntnis, um welchen Ort es sich handelt.



Letzter Menüpunkt: Informationen,  
selbsterstellte Grafik

## Impressum

Webseiten und Apps sind dazu verpflichtet ein **Impressum** zu führen. Im Impressum steht wer für die Inhalte verantwortlich ist (Was ist ein Impressum?, 2018). Ich unterteile die Themen in Punkte wie: Lizenzen, Datenschutz, Kontaktdaten von den Entwicklern und Herausgeber und Versionsnummer der App.



Impressum,  
selbsterstellte Grafik

# Produktmerkmale

Jedes Erlebnis ist in gewisser Weise einmalig. Wenn dieselbe Person das gleiche Produkt in einer ähnlichen Situation nochmal verwendet, hat sich etwas Elementares verändert: Die Erfahrung (Moser, 2012). Eine Ansammlung von Wissen was beim zweiten Durchführen angewendet wird. Laut Professor Dr. Noriaki Kano von der Universität Tokio, können die ausschlaggebenden Merkmale in drei Kategorien eingeteilt werden:

## Begeisterungsmerkmale

Emotionen die nicht erwartet wurden. Der „Wow“-Effekt.

Beispiel Backofen: Besitzt eine zusätzliche Mikrowellenfunktion. Das spart Platz und weitere Küchengeräte.

## Leistungsmerkmale

Die Qualität und Performance

Beispiel Textprogramm: Leistung wie Rechtschreibprüfung inklusive, die die Arbeit erspart.

## Basismerkmale

Voraussetzungen die erwartet werden. Negative Gefühle, wenn diese fehlen. Beispiel Wasserkocher: Bei dem Gerät wird erwartet, dass es das Wasser im Gefäß heiß kocht.

### Begeisterungsmerkmale

AR Tracking Event  
3D Modell werden gezeigt  
Texte (Audio) werden vorgelesen

### Leistungsmerkmale

Kamera wird geöffnet  
Lokale Events  
Informationen werden gezeigt  
Vielfalt an Orten

### Basismerkmale

App wird geöffnet  
Buttons geben Klick-Feedback  
Navigation (Menü) funktioniert

Um den Erfolg der App zu garantieren, reicht kein einmaliger „Wow“-Effekt. Sondern die Zufriedenheit entsteht durch nachhaltige Begeisterung. Das bedeutet, die Daten müssen gepflegt und ganz wichtig: erweitert werden. Das bedeutet nicht nur im gleichen Muster hinzugefügt, zusätzlich mit neuen Interaktionen und Events gefüttert werden. Nach diesem Prinzip habe ich die App konzipiert. Es gibt Funktionen die begeistern, leistungsstarke native Elemente werden eingebunden und das Design versucht jeden Randfall darzustellen.

## Themen, die wir aus zeittechnischen Gründen nicht berücksichtigt/realisieren konnten

### User Tests

Ein User-Test überprüft die Benutzbarkeit des Produkts. Bei dem Test werden dem:der Benutzer:in kleine Aufgaben gestellt, die an einem Klick-Dummy oder am fertigen Produkt gelöst sollen. Jeder Schritt wird dabei beobachtet und dokumentiert. Bei der Beobachtung können Fehler vom System abgeleitet werden. Außerdem kann geschaut werden, ob noch Unklarheiten bei der Bedienung der Benutzeroberfläche besteht. (Holzinger, 2005)

### GPS-Tracking

Ein Feature, das wir gerne implementiert hätten, war das GPS Tracking, mit dem wir die genaue Lokalisierung des Smartphones anzeigen können. So hätten wir den genauen Standort mit den lokalen Artefakten anzeigen können. Leider ist der verbundene Aufwand, der von vielen Faktoren abhängig ist, schwer einzuschätzen und in der Zeitspanne der Bachelorarbeit nicht zu garantieren.

### Globalisierung / Internationalisierung

Globalisierung bedeutet, das Produkt an verschiedene Kulturen anzupassen. (Moser, 2012) Der größte Punkt dabei ist, alle Texte zu übersetzen. Zusätzlich gibt es noch kulturelle Aspekte, wie Leserichtung, Zahlenformate oder Währung, die man beachten sollte.

### „Teilen“ Funktion

Aus sozialen Netzwerken wie Facebook oder Whatsapp ist die Teilen-funktion bekannt. Der:die Benutzer:in gefällt ein Bild, Video oder Text und möchte diesen Inhalt mit anderen teilen. Diese Anwendung wäre eine tolle Chance die Reichweite der App zu vergrößern - Artefakte über WhatsApp teilen, um Freund:innen davon zu erzählen.

### Weitere Interaktionsmöglichkeiten

3D Artefakte im Raum platzieren und ein Foto auslösen. So könnte jede schöne Statue, die vor dem Bürgerhaus steht, ins eigene Wohnzimmer gestellt und der Moment festgehalten werden.

Hinter bunten Buttons versteckt sich eine unerwartete Reaktion, die auf das 3D Artefakt angewendet wird. Das Artefakt könnte in tausend Partikel zerspringen und sich zufällig wieder zusammensetzen, mit einer Sprachverzerrung Gesprochenes wiedergeben oder weitere Ideen für eine abstrakte Darstellung.

Stempel / Münzen könnten bei jedem Ort oder Artefakt zu finden und einzusammeln sein. Durch Gamefikation könnte der Spielspaß der App nochmal gesteigert werden.



Weitere Features,  
selbsterstellte Grafik

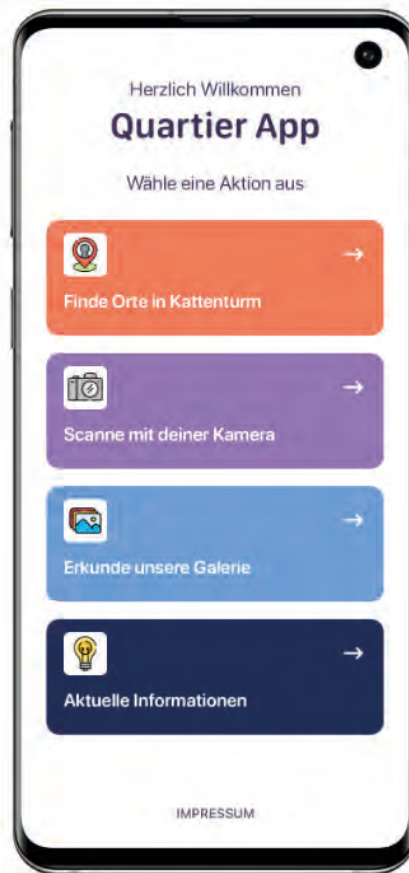
# Klick-Dummy

Mit dem Programm „Invision“ habe ich noch einen **Klick-Dummy** entwickelt, damit die ersten Erfahrungen der Benutzer:innen eingefangen werden können. Mit einem Plug-in für mein grafisches Programm „Sketch“ exportiere ich alle fertigen Screens und verlinke jede Interaktion die vorgesehen ist. So sind alle Screens miteinander verknüpft. Der Klick-Dummy wird sogar in einem Smartphone Rahmen dargestellt. So fühlt sich die Interaktion nochmal echter an.

<https://invis.io/73Z4R6SU8TQ>



QR-Code zum Klick-Dummy,  
selbsterstellte Grafik



Screenshot vom Klick-Dummy  
Invision, 2021

### Jacob's Arbeitsbereiche

Vom Konzept zur fertigen App gibt es keinen direkten Weg. Die Entwicklung ist ein Prozess, dem durchaus noch Änderungen vorbehalten sind. Dennoch ist es wichtig sich an das Konzept zu halten und Rücksprachen mit dem Designteam zu halten - und dieses sogar mit einzubeziehen.

In unserem Fall sind wir ein flexibles Team. Da wir uns beide auch mit den Aufgaben der:des anderen beschäftigt haben, konnten wir nicht nur das Konzept gemeinsam gestalten, sondern auch die Umsetzung dessen. Im folgenden erklärt Jacob die Unity Engine als essenzielles Werkzeug des Projektes, geht auf das Konzept der Projektarchitektur ein und beschreibt insbesondere den Aufbau der App ausführlich.

# Unity Engine



**Why did we choose Unity?**

**Unity Editor**

**Scripting in Unity**

# Unity Engine

The **Unity Engine** is a game engine created by Unity Technologies and was released in 2005 (Wikipedia-Autoren, 2011). It is a highly flexible development environment for all sorts of 2D and 3D applications mainly focusing on games but is also used in mobile development, visualization, and more.

One of its greatest advantages, in general, is its great platform diversity. Unity can deploy to over 25 different platforms (Unity Technologies, o. D.-e). Not only has it proven to be one of the most popular game engines for game development on mobile platforms (Unity Technologies, o. D.-d), but also advertises to be the most popular platform for AR and VR development (Unity Technologies, o. D.-a).

Unity has a lot of useful plugins/packages to extend the basic functionality provided by default. For the use in this project, the most important extension is the **AR Foundation** plugin that enables the usage of AR features on various mobile devices running the Android operating system or iOS. AR Foundation is an interface that unites core functionalities of ARCore (Google Android) and ARKit (Apple) besides other XR frameworks. By appearing like one joint library, the usage in Unity, as well as the deployment to both platforms, is facilitated (Unity Technologies, o. D.-b).

Programming in Unity is done in C#, an "object-oriented, and type-safe programming language" (Microsoft-Autoren, 2021) developed by Microsoft.

## Why did we choose Unity?

In my opinion, Unity's biggest rival is the Unreal Engine by Epic Games that focuses on high-quality graphics rendering in real-time while maintaining good overall performance. However, it was not suited for our project since it does not offer as much support for the development of applications for mobile devices as we are used to with Unity, especially considering AR features.

The Unity Engine has become my personal favorite engine, it was easy and fun to learn throughout my studies since 2017. To me, it has already proven to be a reliable and flexible tool fulfilling my needs as a multidisciplinary developer. It offers great modularity and access to new and interesting technologies. Additionally, it can be adapted through configurable structures, such as the **ScriptableObject**, to fit the needs of software developers and designers. With packages and plugins, immediate access to AR features is gained and the quick deployment of our application to Android and iOS is possible. Furthermore, the app can be built directly onto a device from the editor and tested on it.

Programming in C# feels similar to Java and other common object-oriented programming languages, which made it easy for me to learn although it never felt like doing something entirely new to me. The key factor for our decision was our beforehand experience with Unity. Both Christine and I have been working with Unity in the past and are familiar with the engine and the editor interface.

To make the understanding of how the architecture inside the Unity project works and the underlying system of Unity easier, the user interface of the Unity editor will be explained subsequently.

## Unity Editor

The **Unity Editor** is the graphical user interface (**GUI**) of the engine. It is organized through a flexible multi-window system. The different tools of the editor are displayed in separate windows and usually fulfill a specific task. These different tools of the editor are always accessible via the menu bar. They can be added and removed, but also rearranged and scaled. The resulting window layouts can be saved at the top right under „Layout“, which can prove to be useful if, for example, they have been specifically arranged for certain work steps and are needed again at a later time.

I will now briefly explain the most important tools and windows to give a basic overview of the editor and to establish a common level of knowledge. In this explanation, I will slowly move on to how I developed the app.

## Scene View

The **Scene View** shows the visible representation of a scene. Every scene needs to contain at least one active camera to be visible to the user at runtime. Objects and the environment of the scene are rendered in real-time with customizable, editor-specific render settings. Changes to the visual appearance and quality of the scene and project, such as lighting or shaders, can be viewed here.

Developers can navigate the scene view camera through the scene, add and select objects, edit their position, rotation and scale using either transform gizmos or shortcuts to build the scene.

The viewing of the scene, lighting, and audio can be changed. These settings do not affect the actual scene properties but are a useful tool to evaluate certain aspects of the scene.

## Game View

The **Game View** represents the final look of the built application. It needs at least to have one active Camera in the scene to render it. By entering the **Play mode** the application starts playing. Controls and UI are fully operational now and the application is ready for testing. Any changes made during the Play mode are temporary and will be reset on exiting the Play mode.

## Hierarchy

The **Hierarchy** window shows every **GameObject** inside a scene as a hierarchy of its contained GameObjects. The visual representation of such GameObjects is a mini icon followed by the GameObject's name.

The hierarchy can show multiple scenes at once and each of their GameObjects as well below their corresponding scene root. GameObjects can be parented to each other; those relations are visible in the hierarchy window by an inset of the GameObject's name. GameObjects in the scene view and hierarchy are both linked to the actual scene file and therefore can be added or deleted in both the SceneView and Hierarchy at the same time.

## Project Window

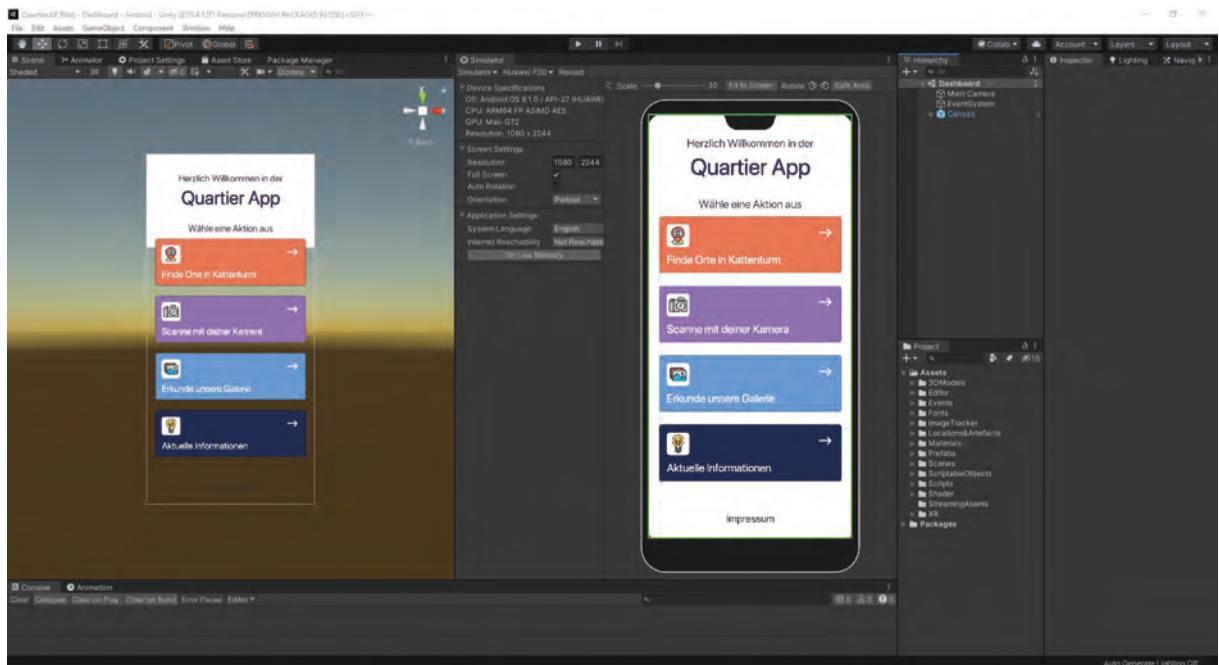
The **Project window** displays the project folder structure and its files. This is the main window to navigate, manage and organize assets in the project. It offers several ways to search and filter for certain assets and has a changeable layout to fit user preferences.

## Inspector

The **Inspector** shows the properties of almost all kinds of objects, files, and assets in the Unity Editor and is also an interface to edit these properties. The appearance of assets and scripts can be altered by using Editor Scripts which help make custom classes suitable for the inspector that contains types that are usually not recognized by it. The way certain kinds of objects are displayed in the Editor can differ from each other depending on their type. It is also possible to select multiple objects of the same kind to compare their properties and change them all at once.

## Console and Debugging

The **Console** is an output log window showing debug logs, errors, and warnings from the Editor when the code gets compiled or during its runtime. Programmers can use them as well and implement their own debug messages, thus enhancing the debuggability of the program.



This is how I used to arrange my window layout in the Unity Editor

# Scripting in Unity

## GameObjects

*“GameObjects are the fundamental objects in Unity that represent characters, props, and scenery. They do not accomplish much in themselves but they act as containers for Components, which implement the functionality.”*

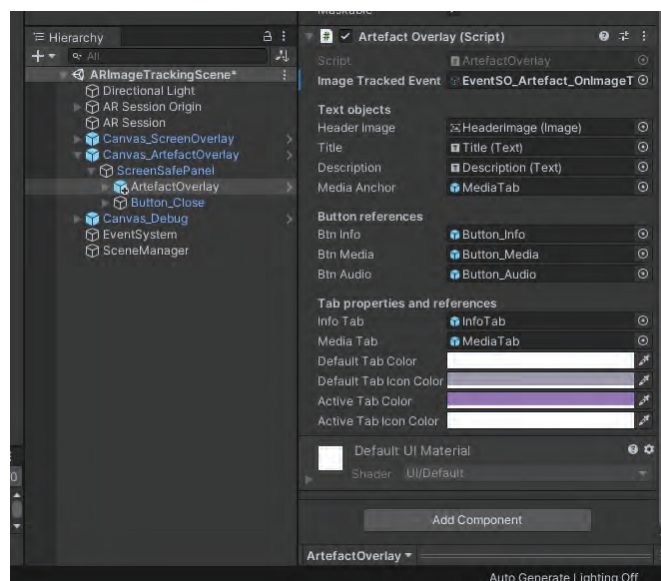
(Unity Technologies, 2021a)

GameObjects are all objects inside a Unity scene. At their creation, they always come with a Transform component by default, which can not be detached. This **Transform** component determines the position, rotation, and scale of the GameObject in the scene and handles the parenting of GameObjects.

GameObjects can be dragged and dropped into the project window to create a Prefab from them. Prefabs are reusable assets that are created in the scene at first and then saved somewhere inside the project folder with all their components, property values, and child GameObjects (Unity Technologies, 2017b). To extend the functionality of GameObjects, developers can attach more components to them.

## Monobehaviour Scripts

Unity is a component-based system. These components are Unity Scripts that derive from the **Monobehaviour** class. The Monobehaviour class is the base class for every Unity Script by default and brings a lot of functionality to a script, e.g. for getting references to other components or providing useful hooks for certain event functions like „Start“ and „Update“ (Unity Technologies, o. D.-g). These components are written as **C# Scripts** which often have only one class that is dedicated to fulfilling a certain task.



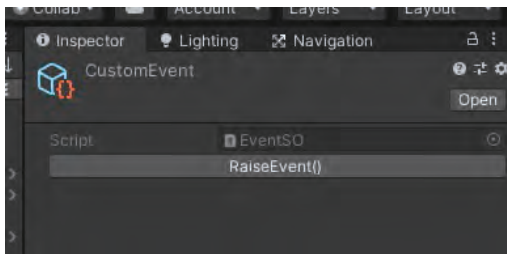
An example of a GameObject with a script attached

Monobehaviour scripts are also represented in the Inspector and can be specified with an Editor script. Fields of known types are serialized, therefore already visible and editable by default in the Inspector, if they are public or explicitly marked with the **[SerializeField]** attribute.

## Editor Scripts

*“Editor Scripting can help you customize and extend the Unity editor to make it easier to use on your projects.”*

(Unity Technologies, o. D.-c)



An EventSO asset's appearance in the Inspector

**Editor scripts** derive from the **Editor class**. They enable developers to add more functionality to the editor, for example, to change the Inspector for certain objects. They can be attached to a custom component using the **[CustomEditor]** attribute. To show an example, I used such a script to add a button to a ScriptableObject type of script to call its function from the editor.

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

// https://www.youtube.com/watch?v=ra031hhE_Kk

[CreateAssetMenu(menuName = "EventSO", fileName = "NewEventSO")]
public class EventSO : ScriptableObject
{
    List<EventListener> listeners = new List<EventListener>();

    public void Raise()
    {
        for (int i = listeners.Count-1; i >= 0; i--)
        {
            listeners[i].OnEventRaised();
        }
    }

    public void Register(EventListener listener)
    {
        if(!listeners.Contains(listener))
            listeners.Add(listener);
    }

    public void Unregister(EventListener listener)
    {
        listeners.Remove(listener);
    }
}
```

The EventSO ScriptableObject class

## ScriptableObjects

The **ScriptableObject** class derives from the base Unity Object. ScriptableObjects cannot be attached to GameObjects like MonoBehaviour scripts but can be stored as an asset in the Project folder. These assets can then be used to provide values without having a class instance; they are persistent data objects that can be referenced at runtime and editor-time by scripts, even if they are attached to uninstan-  
tiated Prefabs.

They are not restricted to only grouping and preserving values; they can also contain code that can be executed and have their variables appear in the Inspector. ScriptableObjects get serialized by the editor so if their fields are public or have the attribute **[SerializeField]**, they will appear in the Inspector although types unknown to the engine still will not show. The appearance of ScriptableObjects in the Inspector can be altered by Editor Scripts, so it is possible to display types that are unknown to the editor, too.

The held values will remain changed after exiting Play mode even when changed during the program runtime. Only ScriptableObjects created during runtime are not retained afterward. (Unity Technologies, 2018)





# Architecture



**ArtefactSO**

**LocationSO**

**UI System**

**Map and Locations**

**ImageTrackingScene**

**Gallery**

**General Information**

# Architecture

*"I'm gonna talk a lot about my approach to game architecture using ScriptableObjects, it's kinda the glue between everything."*

(Ryan Hipple, 2017)

ScriptableObjects are an important part of the Unity Engine and very essential to our project. They enable the content management of our app from within the Unity Editor and establish connections between different components. The simplicity of ScriptableObjects and their flexibility give us many advantages as programmers and designers alike while working with them.

Whenever I think about approaching software architecture, especially when working with Unity, I remember the Unite talk from the Unite Austin 2017 event by Ryan Hipple who works as the principal engineer at Shell Games. In his talk about ScriptableObjects, he came up with his three pillars of game engineering that I use as a breakdown for every game-related project:

## **Modular, Editable, and Debuggable.**

**Modularity** means avoiding rigid structures that are dependent on each other. Modular systems should be able to operate on their own and communicate with other systems without having hard references between them. In Unity, this means that scripts should focus on solving simple tasks, as already stated, and rely on other components as little as possible. Also, Prefabs should be enclosed parts, working mostly individually with every functionality they need already contained inside of their structure.

**Editability** aims at the properties of components and assets. These should be "as much [...] data driven as possible." (Unity Technologies, o. D.-f) Making things editable in the Inspector helps not only programmers during development, but also designers to adapt and optimize the experience of the application by changing values directly in the editor - especially if that's possible at runtime.

**Debugging** the program is a crucial step, and modularity again is key to test and debug single parts isolated from the rest of its environment in the application. It is the result of a project being set up for modularity and editability.

With those principles, I started thinking about how I should approach the actual project. Since we have designed four basic tasks for the app I should try to keep them separated.

Knowing that I could make use of multi-occurring structures and UI elements I tried using Prefabs and different Prefab variants whenever it seemed to make sense. Also, I had trouble debugging mobile applications and especially AR apps in the past, so this time I had to provide myself with a possibility to do so. The solution to this problem will be shown later in "Testing and Debugging".

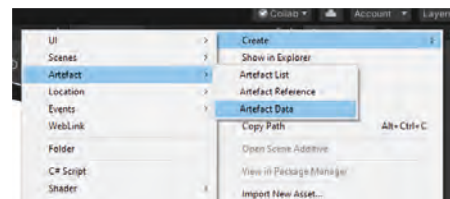
Personally, I think **scalability** should be added to the previously mentioned pillars, as this is something worth considering in any project containing data-driven or generative content. Providing scalability without any extra coding is something that can be achieved through the usage of ScriptableObjects, which are the main key to all these pillars and our project, as will be explained now.

Since our project is supposed to be a scalable solution, i.e. content can be added simply at any time, I created data structures for this content that fulfill our requirements. In our case, this content can be roughly divided into two types: locations and artefacts. Each artefact is located at one of these locations or in its immediate vicinity. For example, artefacts of the BGO are those that can be found in the building or the outdoor area surrounding it – artefacts and locations are therefore related to each other.

# ArtefactSO

Starting with the artefacts, I created a new Unity script, the **ArtefactSO**, where the SO stands for ScriptableObject. The ArtefactSO extends the ScriptableObject class and therefore can be referred to as ScriptableObject. Every artefact has its data stored in an ArtefactSO asset. A new asset can be created inside the project window in the Unity Editor by right-clicking and selecting "Create", then "Artefact" and choosing "Artefact Data" from the context menu.

This is possible due to the **[CreateAssetMenu]** attribute that I added before declaring the class in its script. I also specified the name of the menu and a default name for new assets. This enabled us to create new data assets from inside the editor without the need to instantiate new classes through code and importing the data from an external file or database.



The content for these data assets can then be written or dragged directly into public or serialized fields which will pop up in the Inspector when an ArtefactSO asset is selected. With this, a decent workflow for content management was created, which offers control over the content to everyone familiar with the Editor GUI.

The ArtefactSO contains fields for the name of the artefact, its description, and also a title image, as well as an icon and a reference to the trackable image. Furthermore, there are also fields for a Prefab to be instantiated when the corresponding image gets tracked, an audio clip, and two separate lists for images and video clips.

```
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.Video;

[CreateAssetMenu(fileName = "NewArtefactData", menuName = "Artefact/Artefact Data")]
/// <summary>
/// A container for artefact data.
/// </summary>
public class ArtefactSO : ScriptableObject
{
    [Header("Artefact information")]
    public string title;
    public string description;
    public Sprite headerImage;
    public Sprite icon;
    public Sprite arTracker;

    [Header("Media")]
    public GameObject objectPrefab;
    public AudioClip audioClip;
    //public List<AudioClip> audioClips;
    public List<Sprite> images;
    public List<VideoClip> videoClips;
}
```

## LocationSO

I created a similar ScriptableObject for the locations we want to display on the map, called **LocationSO**. It works similarly to the ArtefactSO but has other properties to suit different needs. Its main role is to provide data for a certain GameObject called LocationOverlay which shows all the information about a given location, but I will explain more about this later. First I am going to give some more context about the overall UI architecture of the application.

```
using System.Collections.Generic;
using UnityEngine;

[CreateAssetMenu(fileName = "NewLocationData", menuName = "Location/Location Data")]
/// <summary>
/// A container for location data.
/// </summary>
public class LocationSO : ScriptableObject
{
    [Header("Meta Data")]
    public string title;
    public string description;
    public Sprite icon;
    public Sprite headerImage;

    [Header("Contact")]
    public string contactPerson;
    public string openingHours;
    public string address;

    [Header("WebLinks")]
    public List<WeblinkSO> weblinkButtons;

    [Header("Artefact Symbols")]
    public List<ArtefactSO> artefacts;

    [Header("Media")] // Only pictures?
    public List<Sprite> pictures;

    [Header("CustomSections")]
    public List<CustomSectionSO> customSections;
}
```

The LocationSO ScriptableObject class

## UI System

The UI architecture of our app was probably the most time-consuming and complex task in this project. Unity's UI system was not completely new for me, but I have to admit that I only barely touched its surface before I started to have a closer look at it for this project. First I want to introduce a few important components of Unity's UI system, starting with the **Canvas**:

The Canvas is a component that acts as a frame for UI elements. It works as a 2D layer on top of the scene camera by default which is just right for our UI design.

*"All UI elements must be children of a GameObject that has a Canvas component attached."*

(Unity Technologies, 2020a)

This is why we can use them as a root GameObject for all our screens to keep the hierarchy of the scene clean and structured. Since we are working on mobile devices, the output display for our app may vary quite a lot from device to device. We have to keep in mind that different devices not only have different screen sizes but also different screen resolutions, different aspect ratios, and a different amount of pixels per inch (display sizes are often measured in inches). Unity's Canvas component provides us with a lot of helpful settings and additional components such as the CanvasScaler.

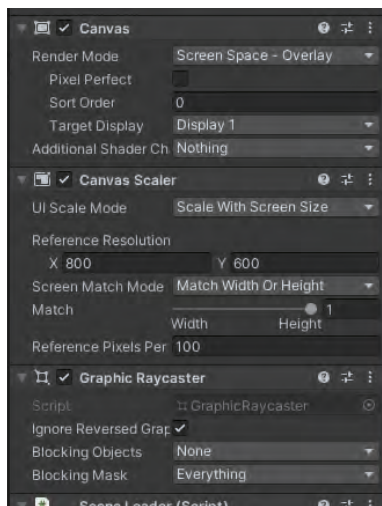
*"[It is a component] used for controlling the overall scale and pixel density of UI elements in the Canvas. This scaling affects everything under the Canvas, including font sizes and image borders."*

(Unity Technologies, 2020b)

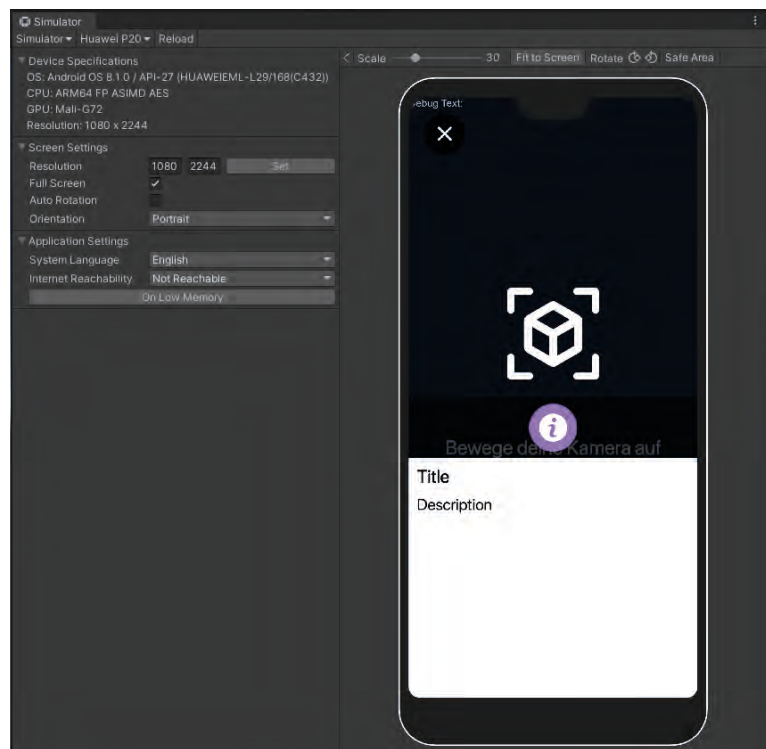
It was quite helpful to have this tool available, but finding the right settings was difficult in the beginning. Unity's "Designing UI for Multiple Resolutions" guide was a good way to start and it helped me understand how this component is to be used (Unity Technologies, 2017a), but I didn't know what our app would look like when I only started to implement it, so I decided to start building some UI first and then test different settings throughout the process. I had to make sure that the size of the elements stays almost the same or at least won't get too small so people with bad eyesight could still see them on the screen. With some time and ongoing changes during the development, I found a good setup that satisfied me on almost all devices I simulated on the **Device Simulator**.



Unity's **Device Simulator** package is an indispensable extension for every mobile project since it lets developers test different device settings and display formats inside the editor. It works as a special Game View only made for mobile testing without needing to build onto a real target device.



A GameObject with a Canvas, CanvasScaler, and GraphicRaycaster component



The Device Simulator

But even with that, we might run into problems with devices that don't have a perfectly rectangular screen or even a notch. To solve this problem, I had to add a script to the Canvas that resizes the UI to the space available on the phone that was not hidden by a notch or camera, the so-called "screen safe area". This safe area is returned by the `safeArea`-accessor of the `Screen` class which gives access to the device's display information. The `ScreenSafeArea` script resizes a UI panel right beneath the Canvas to the size returned by the accessor. Now that I have set up my Canvas with a screen safe area, the base structure for all future UI modules was created and saved as a Prefab. At this point, I was ready to construct and implement the app. Starting with the main menu, or "**Dashboard**" as we called it, I had a quite simple task: Building a UI that has four buttons that let the user reach each of our app's main functions. At this point, we had to decide how we want to separate different parts of our app and how to technically reach them.

## Multiple Canvases or Nested Panels

One way to achieve this would be to create separate Canvases that all represent one part of the app. The whole app, or at least most of it, would be inside this one scene. Transitions between these parts could be done by using multiple animations and Animator components to move the Canvases in and out of the screen area that is visible to the user or simply enable and disable GameObjects for now. This would lead to an overloaded scene in the editor. It wouldn't be a problem for the machine but the human user. That is because we are going to use a huge amount of UI components and encapsulated GameObjects where developers would easily get lost with too many Canvases and underlying UI panels.

I decided to put every enclosed app functionality in its own Unity scene so Christine and I could work on different parts without interfering with each other. Although we are using Git as our version control system and therefore could have used different branches to avoid this, it wouldn't really help in our workflow because we would have needed to receive the changes we made inside the project quite frequently. Different branches would lead to a lot of merging with only a single scene file, which is not easy as I know from previous projects. That would be unnecessary work as this can be avoided. Another benefit of this approach is the modularity and the ability to test single features of our app in its own environment without having to take other modules into account that might have an impact on something I could only guess at this point - this would cause even more problems while I was already busy fixing problems.

The slight disadvantage that comes with this approach is short loading times that might come up during the change of the scenes. This is something I wanted to ignore for prototyping, maybe even later too, as long as they did not become problematically long or even hindering in later stages of development.

## Changing Scenes

So each of the four menu points in the scene "Dashboard" should change the scene. To do this a script with a public function is needed that can be called by interacting with the button. Changing scenes in Unity is done by calling the public static function `LoadScene` on the `SceneManager` class from the `UnityEngine.SceneManagement` namespace which takes at least either the build index of the scene or the scene name (which is its asset path) as an argument.

So, I created a MonoBehaviour inheriting script that provides a public function that passes arguments to the `SceneManager.LoadScene()` function and then I set it up on each of those buttons to work with it. This was necessary since the **UnityEvent**, which is used by the Button script for interaction, can only reference functions that are a component of a `GameObject`, that therefore must be a MonoBehaviour inheriting class, and the `SceneManager` is not one of them.

This Script is called "**SceneLoader**". I attached it to the Canvas Prefab I created earlier, so all scripts on `GameObjects` below the Canvas Prefab now can change the scene by calling `transform.root.GetComponent<SceneLoader>().ChangeScene`. Or if they use a `UnityEvent` they just need a reference to the Canvas `GameObject` to call it. Still, the problem lies in the reference of the scene itself. Both the build index and scene name can change during development, which would break any reference I would try to establish to the new scene. If we had a lot of scenes or a lot of components using their names or build index, we would need to update all of them again to get things back to work.

```
using UnityEngine;
using UnityEditor;

[CreateAssetMenu(fileName = "NewSceneReference", menuName = "Scenes/SceneReference")]
public class SceneReferenceSO : ScriptableObject
{
    [SerializeField] public SceneAsset sceneAsset;

    [SerializeField] public string scenePath;

    public string GetScenePath()
    {
        //return AssetDatabase.GetAssetPath(sceneAsset);
        return scenePath;
    }
}
```

The SceneLoader script, taking a SceneReferenceSO as an argument for the ChangeScene function instead of a rigid value

Fortunately for us, `ScriptableObjects` can be used to simplify this problem. I created a new `ScriptableObject` called **SceneReferenceSO**. It holds only a string for the `scenePath` which is applied through an `EditorScript` that makes the string take the scene asset as an argument.

It then extracts the path from the asset into the string variable. So if we now use the string from the `ScriptableObject` instead of the name or the build index of the actual scene asset, we would only have to update the `ScriptableObject` once after changing its name or path.

Whether it is just an overlay or a whole scene, the user also needs to have the ability to exit it or “go back” to the previous screen. Every screen has a previous screen except for the Dashboard; and since they all use the Canvas Prefab I created earlier as a base, I could easily edit it and add a UI button to either close it or change the scene. I only had to add the SceneLoader script to the Canvas and make a reference to it on the button component, then choose the LoadScene function and pass the SceneReferenceSO asset of the previous scene as an argument.

```
using UnityEngine;
using UnityEditor;

[CustomEditor(typeof(SceneReferenceSO), true)]
public class SceneReferenceEditor : Editor
{
    public override void OnInspectorGUI()
    {
        var reference = target as SceneReferenceSO;
        var oldScene = AssetDatabase.LoadAssetAtPath<SceneAsset>(reference.scenePath);

        serializedObject.Update();

        EditorGUI.BeginChangeCheck();
        var newScene = EditorGUILayout.ObjectField("scene", oldScene, typeof(SceneAsset), false) as
SceneAsset;

        if (EditorGUI.EndChangeCheck())
        {
            var newPath = AssetDatabase.GetAssetPath(newScene);
            var scenePathProperty = serializedObject.FindProperty("scenePath");
            scenePathProperty.stringValue = newPath;
        }
        serializedObject.ApplyModifiedProperties();
    }
}
```

```
using UnityEngine;
using UnityEngine.SceneManagement;

public class SceneLoader : MonoBehaviour
{
    public void ChangeScene(SceneReferenceSO scene)
    {
        SceneManager.LoadScene(scene.GetScenePath(), LoadSceneMode.Single);
    }
}
```

The SceneReferenceSO with its Editor script

## Testing and Debugging

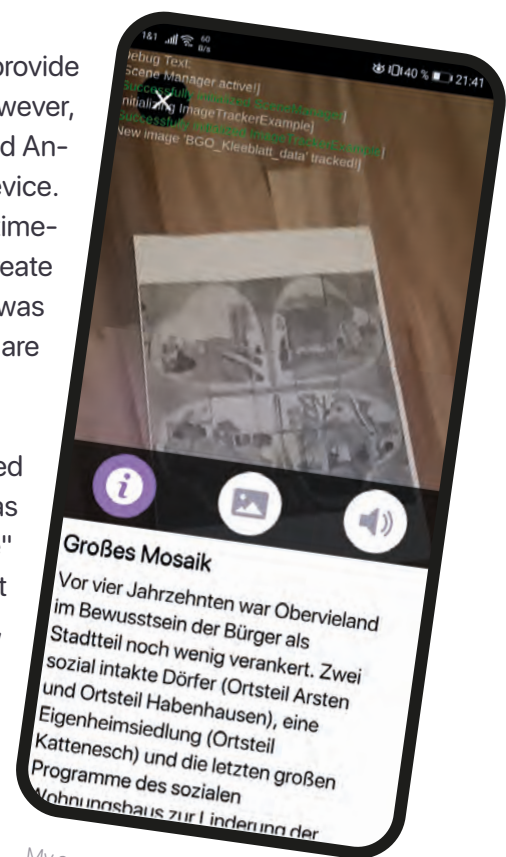
Since we are working on a mobile platform and with AR on top of that, I had to create a different way to debug the app for the following reasons:

Developing mobile applications in Unity means that our target platform is a mobile device, a smartphone. Unity can build and transfer the application directly onto a device matching the current target platform specified in the editor build settings (e.g. Android). This is a fast and handy way to test the app on a native device and shows how the app behaves on a real device.

Builds running on an external device, such as a smartphone, will not display output logs in the editor. AR needs the device camera which is not available in the editor. An external webcam can't be used either, the device simulator has no functionality for this yet.

The "Unity Remote 5" app for Android was created to provide testability on devices being connected to the editor, however, this does not work anymore for newer versions of Unity and Android, which brings me back to testing the app on a real device. This made the process of finding faults in my script very time-consuming and inefficient, so I came up with the idea to create my own debug screen for use on a real device. This idea was originally inspired by the Unreal Engine, where debug logs are usually displayed right in the viewport.

So I created a new Canvas with an Image and Text nested inside and a new script called **MobileDebugScreen**. It has a simple public static function called "AddDebugLogLine" which takes a string as an argument and adds it to the Text element in a new line on the screen. It is a hacky solution, but it worked great for me and has potential for more functionalities and use in other projects.



My custom on-screen debugger during real device testing



```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;

public class MobileDebugScreen : MonoBehaviour
{
    [SerializeField] public Text debugText;
    static Text currentText;

    public static void AddDebugLogLine(string logMessage)
    {
        if (currentText)
            currentText.text += $"{logMessage}\n";
    }

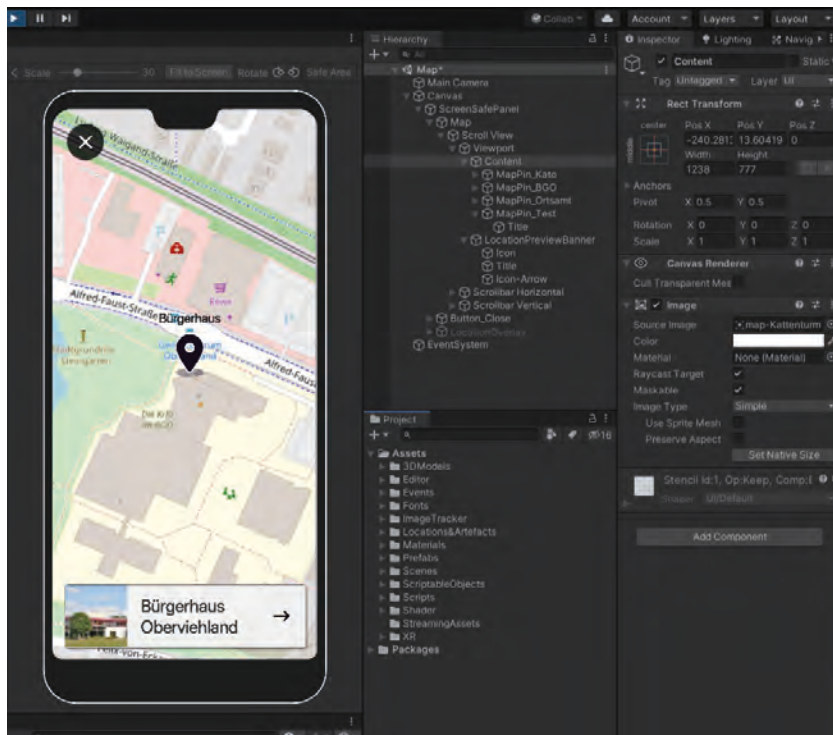
    private void OnEnable()
    {
        currentText = debugText;
    }
}
```

A hacky but handy on-screen debugging solution working with a static function

Starting with the requirements for our product, we have decided on two main methods of content-to-the-user-communication, both considering artefacts, and locations. This results in the four following main functionalities and scenes of the app: The first one would be a playful and explorative approach, mainly addressing the younger users. The main goal is to motivate the users to explore locations within the target area, a map to show these locations is needed. With this, we want to have a map in our application showing various locations the users can go to. They will be able to see the information about the location and the available artefact trackers there beforehand. At the real locations, users can then search for the artefact images and activate the artefacts in AR by scanning the tracker with the device camera. The second is an informative approach where we simply want to display the wealth of artefacts in our app and make them accessible from everywhere. Users should be able to experience the artefacts even from home, just in case, it is not possible to reach the locations of the artefacts out in the environment.

## Map & Locations

The first of our four main functionalities is the **map**. This map scene has to show a map of the area around Kattenturm and **locations** which we chose to show in our prototype. Those locations are visualized as map pins that should let a small preview banner appear on interaction which I called **LocationPreviewBanner**. Another interaction with the banner should let appear an overlay that shows all the information about the location on the whole screen. I called it **LocationOverlay**.



The active MapPin is indicated by the PreviewBanner at the bottom of the screen

For that, I started with the Canvas Prefab and created a new UI panel below the ScreenSavePanel in the object hierarchy and called it Map. This contains a Scroll-View, a UI element that works as a 2D scrolling area with optional scroll bars. This will be the base for the map since we needed it to be bigger than the screen to cover the whole area of the Kattenturm map image and its locations. Unity's ScrollView element already has a script with settings to configure the behavior of the scrolling. The Content GameObject created within this Prefab is going to be the actual map element, so I added an Image component and resized it to the map image.

The map pins are interactable UI elements displayed on the map to represent the locations, so they need to know which location they should represent. I wanted to construct them as Prefabs because they need to be reusable parts of the map system. They have to preserve modularity and independence as the project grows and more locations are being added to it. Therefore I created a Prefab GameObject with a new script called **MapPin**.

```
using UnityEngine.UI;
using UnityEngine;

public class LocationPreviewBanner : MonoBehaviour
{
    public static MapPin activeMapPin;

    [SerializeField] Text title;
    [SerializeField] Text description;
    [SerializeField] Image icon;
    [SerializeField] LocationOverlay placeInfo_ViewPanel;

    public void SetInformation(string title, string description, Sprite icon)
    {
        this.title.text = title;
        //this.description.text = description;
        this.icon.sprite = icon;
    }

    public void EnableLocationOverlay()
    {
        placeInfo_ViewPanel.SetLocationAndEnableOverlay(activeMapPin.placeData);
    }
}
```

The MapPin can also disable the LocationPreviewBanner

This Prefab has an Image and Text component for the icon and title of the location, a Button component for touch interaction, as well as the **MapPin** script. This class holds references to a LocationSO and the LocationPreviewBanner script that is attached to the LocationPreviewBanner GameObject.

On interaction with the icon, the script calls the "ToggleInfo" function on the LocationPreviewBanner script to either enable or disable it. If it gets enabled the field "activeMapPin" on the **LocationPreviewBanner** will be set to the current MapPin script and some of its location data will be passed to be displayed on the banner.



The **LocationPreviewBanner** script holds references to some Text and Image components to preview some location information passed on its activation by the MapPin. On interaction with the LocationPreviewBanner the **LocationOverlay** gets enabled through the script which also passes the LocationSO asset from the active MapPin.

```
using UnityEngine.UI;
using UnityEngine;

public class MapPin : MonoBehaviour
{
    public LocationSO placeData;
    public LocationPreviewBanner previewBanner;

    private void Awake()
    {
        GetComponentInChildren<Text>().text = placeData.title;
    }

    public void ToggleInfo()
    {
        if (LocationPreviewBanner.activeMapPin && LocationPreviewBanner.activeMapPin.Equals(this))
        {
            LocationPreviewBanner.activeMapPin = null;
            previewBanner.gameObject.SetActive(false);
        }
        else
        {
            LocationPreviewBanner.activeMapPin = this;

            // Override information
            previewBanner.SetInformation(placeData.title, placeData.description, placeData.icon);

            // Activate
            previewBanner.gameObject.SetActive(true);
        }
    }
}
```

The public static MapPin is an older piece of code that should be changed to a private reference with accessor functions

## LocationOverlay

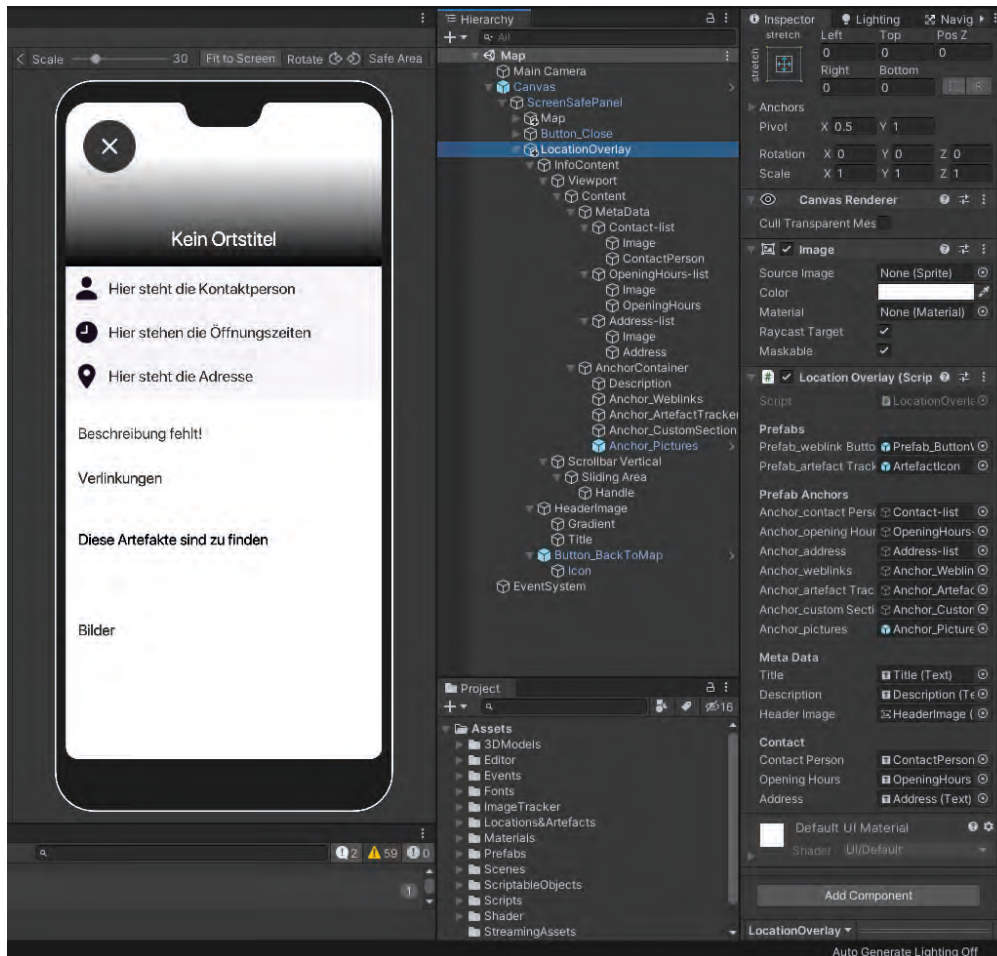
The **LocationOverlay** consists of multiple nested UI panels. It is one of the most complex UI screens that I have built for this project. The LocationOverlay script is located on its root GameObject. The script holds references to all necessary UI elements that might contain information about the location and manages their content. Since the header image (which includes the title) will not scroll with the rest of the content, it is placed on the same level in the hierarchy as the InfoContent GameObject, beneath the LocationOverlay.

The InfoContent has a ScrollArea component. Its Content GameObject uses a VerticalLayoutGroup component and a ContentSizeFitter component. The first one is to make offsets and alignments for child objects adjustable while the second makes it adapt its size to fit the content which is changing dynamically. Those two components are very helpful and are used many times for the overlays of this project. Nested inside, I divided the rest of the information into two panels: MetaData and AnchorContainer. This is necessary to highlight the first group as we decided to do in our design process.

The MetaData contains UI elements for the contact person, opening hours, and the address of the location. Not all LocationSO assets will contain data for all of those fields, for example, Cato Bontjes van Beek Platz which has no opening hours. In this case, the script will disable those UI elements and MetaData will resize itself. The AnchorContainer GameObject features the description of the location followed by GameObjects that serve as anchors for further elements. Each of those also has some kind of layout group component, a ContentSizeFitter, since they are dynamic parts, as well as a Text component to display a title over the section. The GameObjects Anchor\_Weblinks and Anchor\_ArtefactTracker serve as parents for WebLink-Button Prefabs and the artefact tracker icons, which are both specified in the LocationSO.

The Anchor\_CustomSection is an optional and highly dynamic anchor, which I included to feature special sections unique to some locations such as the BGO. The BGO location asset has an additional Prefab to display the "Zeitstrahl" as a special element to it.

Ultimately, the Anchor\_Pictures GameObject serves as a parent for images and eventually videos of the location.



It took a long time to design and develop the complex LocationOverlay

```

using System.Collections.Generic;
using UnityEngine.UI;
using UnityEngine;

public class LocationOverlay : MonoBehaviour
{
    [Header("Prefabs")]
    [SerializeField] GameObject prefab_weblinkButton;
    [SerializeField] GameObject prefab_artefactTrackerIcon;

    [Header("Prefab Anchors")]
    [SerializeField] GameObject anchor_contactPerson;
    [SerializeField] GameObject anchor_openingHours;
    [SerializeField] GameObject anchor_address;
    [SerializeField] GameObject anchor_weblinks;
    [SerializeField] GameObject anchor_artefactTrackerIcons;
    [SerializeField] GameObject anchor_customSections;
    [SerializeField] GameObject anchor_pictures;

    [Header("Meta Data")]
    [SerializeField] Text title;
    [SerializeField] Text description;
    [SerializeField] Image headerImage;

    [Header("Contact")]
    [SerializeField] Text contactPerson;
    [SerializeField] Text openingHours;
    [SerializeField] Text address;

    private void OnEnable()
    {
        anchor_contactPerson.SetActive(true);
        anchor_openingHours.SetActive(true);
        anchor_address.SetActive(true);
        anchor_weblinks.SetActive(true);
        anchor_artefactTrackerIcons.SetActive(true);
        anchor_customSections.SetActive(true);
        anchor_pictures.SetActive(true);
    }

    private void OnDisable()
    {
        foreach (Transform child in anchor_weblinks.transform)
        {
            Destroy(child.gameObject);
        }

        foreach (Transform child in anchor_artefactTrackerIcons.transform)
        {
            Destroy(child.gameObject);
        }

        foreach (Transform child in anchor_customSections.transform)
        {
            Destroy(child.gameObject);
        }

        foreach (Transform child in anchor_pictures.transform)
        {
            Destroy(child.gameObject);
        }
    }

    public void SetLocationAndEnableOverlay(LocationSO placeInformation)
    {
        // Check if all anchors are set
        if (!anchor_contactPerson)
            throw new System.NullReferenceException("Contact anchor is not set!");
        if (!anchor_openingHours)
            throw new System.NullReferenceException("Opening Hours anchor is not set!");
        if (!anchor_address)
            throw new System.NullReferenceException("Address anchor is not set!");
        if (!anchor_weblinks)
            throw new System.NullReferenceException("Weblink anchor is not set!");
        if (!anchor_artefactTrackerIcons)
            throw new System.NullReferenceException("Artefact anchor is not set!");
        if (!anchor_customSections)
            throw new System.NullReferenceException("Custom Section anchor not set!");
        if (!anchor_pictures)
            throw new System.NullReferenceException("Picture anchor is not set!");

        // Check for necessary information
        if (placeInformation.title.Length == 0)
            Debug.LogWarning("Title is empty!");
        if (placeInformation.description.Length == 0)
            Debug.LogWarning("Description is empty!");
        if (!placeInformation.headerImage)
            Debug.LogWarning("No header image set!");

        // Set meta data
        title.text = placeInformation.title;
        description.text = placeInformation.description;
        headerImage.sprite = placeInformation.headerImage;

        // Set or hide contact details
        if (placeInformation.contactPerson.Length == 0)
            anchor_contactPerson.SetActive(false);
        else
            contactPerson.text = placeInformation.contactPerson;

        if (placeInformation.openingHours.Length == 0)
            anchor_openingHours.SetActive(false);
        else
            openingHours.text = placeInformation.openingHours;

        if (placeInformation.address.Length == 0)
            anchor_address.SetActive(false);
        else
            address.text = placeInformation.address;
    }
}

```

```

// Create weblink buttons or hide section
if (placeInformation.weblinkButtons.Count > 0)
{
    List<WeblinkSO> weblinkButtons = placeInformation.weblinkButtons;
    foreach (WeblinkSO weblink in weblinkButtons)
    {
        if (!weblink)
            continue;
        Debug.Log($"Creating a weblink button for {weblink.buttonText}");
        GameObject newWeblinkButton = Instantiate(prefab_weblinkButton,
        anchor_weblinks.transform);
        newWeblinkButton.GetComponentInChildren<Text>().text = weblink.buttonText;
        newWeblinkButton.GetComponent<Button>().onClick.AddListener(delegate {
        OpenURL(weblink); }); // To be saved!!
    }
}
else
{
    anchor_weblinks.SetActive(false);
}

// Create Artefact Symbols or Hide section
if (placeInformation.artefacts.Count > 0)
{
    List<ArtefactSO> artefacts = placeInformation.artefacts;
    foreach (ArtefactSO artefact in artefacts)
    {
        if (!artefact)
            continue;
        GameObject newARTrackerSymbol = Instantiate(prefab_artefactTrackerIcon,
        anchor_artefactTrackerIcons.transform);
        Destroy(newARTrackerSymbol.GetComponent<Button>());
        newARTrackerSymbol.GetComponent<Image>().sprite = artefact.icon; // SHOULD THIS BE
        // The TrackerImage instead of icon? TODO: Add the tracker to the ArtefactSO!
    }
}
else
{
    anchor_artefactTrackerIcons.SetActive(false);
}

// Create CustomSections or Hide section
if (placeInformation.customSections.Count > 0)
{
    List<CustomSectionSO> customs = placeInformation.customSections;
    foreach (CustomSectionSO custom in customs)
    {
        if (!custom)
            continue;
        GameObject customSection = Instantiate(custom.sectionPrefab,
        anchor_customSections.transform);
    }
}
else
{
    anchor_customSections.SetActive(false);
}

// Create Pictures or Hide section
if (placeInformation.pictures.Count > 0)
{
    List<Sprite> pictures = placeInformation.pictures;
    foreach (Sprite picture in pictures)
    {
        if (!picture)
            continue;
        GameObject newPicture = new GameObject();
        newPicture.transform.SetParent(anchor_pictures.transform);
        newPicture.transform.localScale = Vector3.one;

        AspectRatioFitter arf = newPicture.AddComponent<AspectRatioFitter>();
        arf.aspectMode = AspectRatioFitter.AspectMode.WidthControlsHeight;
        arf.aspectRatio = 1f;

        Image imageComponent = newPicture.AddComponent<Image>();
        imageComponent.sprite = picture;
    }
}
else
{
    anchor_pictures.SetActive(false);
}

gameObject.SetActive(true);
}

public void HidePlaceInfo()
{
    gameObject.SetActive(false);
}

private void OpenURL(WebLinkSO weblinkData)
{
    Application.OpenURL(weblinkData.url);
}
}

```

## Image Tracking Scene

The **ImageTrackingScene** will be the scene in which I build all the functionality for tracking images and displaying the artefacts' objects. The AR Foundation plugin, which is provided by Unity, already contains all the basic functionality needed to use AR features. To create an AR-ready scene, I only had to insert the ARSessionOrigin and ARSession Prefabs from the GameObject menu in the XR category. It is important to note that a correctly set up Camera is already a child object on the ARSessionOrigin GameObject and a Directional Light is required to light the virtual objects. After that, I added the **ARTrackedImageManager** script to the ARSessionOrigin. This script handles object tracking and provides information about tracked images. It needs a ReferenceImageLibrary and a TrackedImagePrefab which can be any Prefab.

The ReferenceImageLibrary is a ScriptableObject and can be created as an asset in the Project window. It contains a list of all image trackers that our app will be tracking. We have to make sure to add all image tracker and related artefacts to their respective lists because it will be necessary to compare them. The Tracked Image Prefab is created as soon as the ARTrackedImageManager has detected an image from the ReferenceImageLibrary. From here my work to stage the artefacts begins.

So, I first created a new "TrackedPrefab" GameObject, that's what I called it. It will be the base for the instantiation of all the artefact objects. The root GameObject contains no component, but two child objects. The one is a simple cube that is supposed to show me that a picture was recognized at all and the Prefab instantiated. I called the second "ScriptExecutor" which contains scripts to be executed. These cannot be located on the root GameObject, since no scripts get executed on it on instantiation!

The only script the ScriptExecutor contains now is the ImageTrackerExample script. Its task is to evaluate the tracking data of the ARTrackedImageManager and to deal with the tracking of images. Using the "OnImageChanged" delegate of the ARTrackedImageManager, the arguments given in the "OnImageChanged" function are used to check whether an image has been detected, updated, or lost. As soon as an image has been detected, an artefact with the same image stored is being searched in the ArtefactSOList asset. If this is found, the associated artefact is considered an "Event Artefact."



```

using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;
using UnityEngine.XR.ARFoundation;
using System;

public class ImageTrackerExample : MonoBehaviour
{
    SceneManager_ARImageTracking gameController;
    // InfoOverlay infoOverlay;
    ARTrackedImageManager trackedImageManager;
    [SerializeField] ArtefactListSO eventList;
    List<ArtefactSO> artefacts;

    ArtefactSO eventArtefact;
    GameObject eventGO;

    [SerializeField] EventSO_Artefact imageTracked;
    [SerializeField] EventSO_imageLost;

    void Awake()
    {
        gameController =
        GameObject.Find<GameObject>(WithTag("GameController")).GetComponent<SceneManager_ARImageTracking>();
        //gameController.GetIndicatorPanel().color = new Color(1f, 0f, 0f, 0.3f); // set red-30%
        //gameController.GetIndicatorPanel().color = new Color(1f, 0f, 0f, 0.3f); // set red-30%
        //gameController.AddDebugLogLine("Initializing ImageTrackerExample");
        gameController.RegisterImageTracker(this);
        DebugConsole.print("TrackedImagePrefab created!");

        if (eventList)
        {
            Debug.LogException(new System.Exception("No eventList set!"));
            gameController.AddDebugLogLine("<color=red>No eventList set!</color>");
            return;
        }

        artefacts = eventList.artefacts;
        trackedImageManager = FindObjectOfType<ARTrackedImageManager>(); // Bk bc slow, but perfect
        for (int i = 0; i < trackedImageManager.trackedImages.Count; i++)
        {
            if (trackedImageManager.trackedImages[i] != null)
            {
                Debug.LogException(new System.Exception("No trackedImageManager set!"));
                gameController.AddDebugLogLine("<color=red>No trackedImageManager set!</color>");
                return;
            }
        }

        //infoOverlay = gameController.GetInfoOverlay();
        //if (!infoOverlay)
        //{
            gameController.AddDebugLogLine("<color=red>No infoOverlay was passed!</color>");
            throw new MissingReferenceException("No infoOverlay was passed!");
        //}

        //gameController.SetIndicatorPanel().color = new Color(0f, 1f, 0f, 0.3f); // Reset red-30%
        //gameController.SetIndicatorPanel().color = new Color(0f, 1f, 0f, 0.3f); // Reset red-30%
        gameController.AddDebugLogLine("<color=green>Successfully initialized ImageTrackerExample</color>");
    }

    void OnEnable() => trackedImageManager.trackedImagesChanged += OnImageChanged;
    void OnDisable() => trackedImageManager.trackedImagesChanged -= OnImageChanged;
    void OnImageChanged(ARTrackedImagesChangedEventArgs eventArgs)
    {
        foreach (var newImage in eventArgs.added)
        {
            foreach (ArtefactSO artefact in artefacts)
            {
                if (artefact.name.Equals(newImage.referenceImage.name))
                {
                    gameController.AddDebugLogLine($"New image '{newImage.referenceImage.name}'
                    tracked!");
                    eventArtefact = artefact;
                    CreateARPrefab();
                    //SetEventData(eventArtefact);
                    gameController.ToggleEmptyScanOverlay(false);
                    imageTracked.Invoke(eventArtefact);
                    goto updatedImages;
                }
            }
        }

        updatedImages: foreach (var updatedImage in eventArgs.updated)
        {
            // handle updated event
        }

        foreach (var removedImage in eventArgs.removed)
        {
            if (removedImage.referenceImage.texture.Equals(eventArtefact.artTracker))
            {
                // Destroy ARPrefab and Show Info
                Destroy(eventGO);
                //RemoveEventData();
                // Debug: Call event to reset all passed data related to artefact eg. on InfoOverlay
            }
        }

        gameController.ToggleEmptyScanOverlay(true);
    }

    private void CreateARPrefab()
    {
        DebugConsole.print("Creating the eventGO!");
        eventGO = Instantiate(eventArtefact.objectPrefab, transform);
    }
}

```

The ImageTrackerExample script has an outdated name since I tried a different approach of the same functionality in another class, but failed, so I came back to using this early but working script instead that has not been renamed yet.

To transfer this artefact to other classes outwardly, a special connection must be made to them. I implemented such with ScriptableObjects following Ryan Hippler's example, because they are persistent assets and avoid rigid connections. Thus, the three pillars presented are addressed and another problem is elegantly solved:

Since the ImageTrackerExample script is based on a Prefab, it cannot get a direct reference to objects in the scene unless instantiated. When activating the Prefab, it would have to search specifically for other scripts on GameObjects, which is possible, but also rigid and unsafe, if there is no such reference available. The class should also fire events that signal when an image has been tracked or lost. In the first case, the ArtefactSO should also be passed. Since it is unclear which classes might need this information in the future and not all of them always have to search for connections to the Prefab, I created the EventSO as well as the **EventSO\_Artefact**, which is specifically responsible for the transfer of artefacts. Both are ScriptableObject classes, as the name suggests, and represent events that are equipped with the benefits of ScriptableObjects.

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

// https://www.youtube.com/watch?v=raQ3lJHE_Kk

[CreateAssetMenu(menuName = "Events/EventSO", fileName = "NewEventSO")]
public class EventSO : ScriptableObject
{
    List<EventListener> listeners = new List<EventListener>();

    public void Raise()
    {
        for (int i = listeners.Count-1; i >= 0; i--)
        {
            listeners[i].OnEventRaised();
        }
    }

    public void Register(EventListener listener)
    {
        if(!listeners.Contains(listener))
            listeners.Add(listener);
    }

    public void Unregister(EventListener listener)
    {
        listeners.Remove(listener);
    }
}
```

ImageTrackingScene-EventSO-Code

They work similarly but solve different problems. The EventSO works together with the EventListener class, following the example given by Ryan Hipple in his talk, which is just the observer pattern applied to ScriptableObject. The EventSO contains a list of EventListener.



EventListener can use the “Register” and “Unregister” functions to insert into and remove themselves from the list if they have the reference to this ScriptableObject. If the “Raise” function is called on the EventSO, the “OnEventRaised” function is called for all EventListeners entered in the list. On the EventListener, this function causes a **UnityEvent** to be invoked. This is serialized by the editor and thus displayed in the Inspector, so function references from other GameObject’s components can be called by that.

```

using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.Events;

// https://www.youtube.com/watch?v=raQ3iHHE_Kk

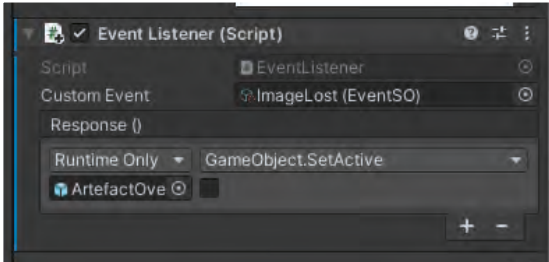
public class EventListener : MonoBehaviour
{
    [SerializeField] EventSO customEvent;
    [SerializeField] UnityEvent Response;

    private void OnEnable()
    {
        customEvent.Register(this);
    }

    private void OnDisable()
    {
        customEvent.Unregister(this);
    }

    virtual public void OnEventRaised()
    {
        Response.Invoke();
    }
}

```



An example of an EventListener that disables a GameObject when the ImageLost event gets invoked.

What the EventSO and the EventListener can’t do is pass arguments when invoking the Unity Event. For this, I created the **EventSO\_Artefact**, which makes this possible in a more rigid form. It includes a public delegate “OnEvent”, that expects an ArtefactSO as a function parameter, and a public event of these delegates.

Classes that have a reference to an EventSO\_Artefact asset can add a function matching the delegate to the event. Thus, the function is called and the ArtefactSO is passed when a class calls the public function "Invoke" on the EventSO\_Artefact. The ImageTrackerExample script uses this special EventSO\_Artefact to pass the ArtefactSO without detours once the associated image has been tracked.

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

[CreateAssetMenu(menuName = "Events/EventSO Artefact", fileName = "NewEventSO_Artefact")]
public class EventSO_Artefact : ScriptableObject
{
    public delegate void OnEvent(ArtefactSO artefact);
    public event OnEvent OnEventTrigger;

    [SerializeField] public ArtefactSO artefact;

    public void Invoke(ArtefactSO artefact)
    {
        this.artefact = artefact;
        OnEventTrigger.Invoke(this.artefact);
    }
}
```

A custom event structure as ScriptableObject for passing ArtefactSO references

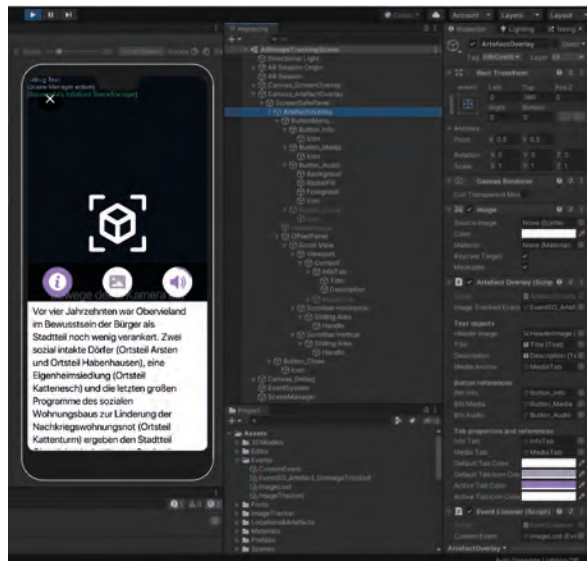
A recipient of this event is the **ArtefactOverlay** script. It is attached to a GameObject of the same name and inserts the information of the ArtefactSO into it. Similar to the LocationOverlay, this is also a complex UI element that has been extensively worked on.

## ArtefactOverlay

The **ArtefactOverlay** GameObject takes about half of the screen space, thus making it necessary to specify the offset in the transform component. Beneath the root GameObject, I added a panel for the menu buttons. It contains a `HorizontalLayoutGroup` attached to itself for ordering the child objects (four buttons for controlling the functions of the overlay).

On the same level as the menu panel, I had to add a specific panel layer to nest a `ScrollView` inside. This is necessary because I added an offset at the top of the `ArtefactOverlay` which seems to affect the behavior of the `ScrollView` if directly parented to the overlay root. Inside the `ScrollView`'s Content GameObject two more panels can be found. One is for the title and description of the artefact information, the other is only for displaying media content such as pictures and videos. Because we wanted to separate text and images from each other, as well as audio which I will talk about next, I created those two panels which can be switched by interacting with the menu buttons for information text and media. The third button will play an audio file if one is available for the given artefact. The `ArtefactOverlay` script provides all the functionality for this, as I already explained for the similar `LocationOverlay`, except for playing audio with the audio button. This one has its own script that can work on its own to preserve the modularity and debuggability of the system and its parts.

The **AudioButton** script including its GameObject has its own structure. The GameObject has four child GameObjects with Image components: a background, a radial fill image, a foreground, and an icon. With this, I was able to create a round button with a circle indicator for the listening process of the audio file. Since the `AudioButton` script fully manages the audio, it also updates the progress through an update routine if the audio source component is playing audio or has stopped.



The `ArtefactOverlay` showing artefact information and playing audio

```

using UnityEngine;
using UnityEngine.UI;

public class AudioButton : MonoBehaviour
{
    [Header("Editor references")]
    [SerializeField] private AudioClip audioClip;
    [SerializeField] private Image radialFill;
    [SerializeField] private Image audioIcon;

    [Header("Audio properties")]
    [SerializeField] private Color defaultColor;
    [SerializeField] private Color activeColor;

    private AudioSource audioSource;
    private Button audioButton;

    SceneManager_ARImageTracking gameController;

    private void Awake()
    {
        if(!radialFill)
            Debug.LogError("radialFill Image is not set!");

        InitializeAudioSystem();
        audioButton = GetComponent<Button>();
        audioButton.onClick.AddListener(delegate { PlayClip(); });

        radialFill.color = activeColor;
    }

    private void Update()
    {
        if(audioSource.isPlaying)
        {
            RadialFill();
        }
    }

    void InitializeAudioSystem()
    {
        audioSource = gameObject.AddComponent<AudioSource>();
        audioSource.playOnAwake = false;
        audioSource.Stop();
        //gameObject.AddComponent<AudioListener>();
    }

    public void SetAudioClip(AudioClip newAudioClip)
    {
        if(newAudioClip == null)
        {
            return;
        }
        audioClip = newAudioClip;
        audioSource.clip = audioClip;
    }

    public void PlayClip()
    {
        audioSource.Play();
        audioButton.onClick.RemoveListener(delegate { PlayClip(); });
        //audioButton.onClick.AddListener(delegate { PauseClip(); });
        audioButton.onClick.AddListener(delegate { StopClip(); });
        audioIcon.color = activeColor;
    }

    public void PauseClip()
    {
        audioSource.Pause();
        audioButton.onClick.RemoveListener(delegate { PauseClip(); });
        audioButton.onClick.AddListener(delegate { PlayClip(); });
        audioIcon.color = defaultColor;
    }

    public void StopClip()
    {
        audioSource.Stop();
        audioButton.onClick.RemoveListener(delegate { StopClip(); });
        audioButton.onClick.AddListener(delegate { PlayClip(); });
        radialFill.fillAmount = 0f; // Reset audio progress indicator
        audioIcon.color = defaultColor;
    }

    private void RadialFill()
    {
        float fillAmount = Mathf.InverseLerp(0f, audioClip.length, audioSource.time);
        radialFill.fillAmount = fillAmount;
    }
}

```

This script encloses the whole functionality for the AudioButton GameObject

```

using System.Collections.Generic;
using UnityEngine.UI;
using UnityEngine;

public class ArtefactOverlay : MonoBehaviour
{
    SceneManager_ARImageTracking gameController;

    [SerializeField] EventSO_Artefact ImageTrackedEvent;

    [Header("Text objects")]
    [SerializeField] private Image headerImage;
    [SerializeField] private Text title;
    [SerializeField] private Text description;
    [SerializeField] private GameObject mediaAnchor;

    [Header("Button references")]
    [SerializeField] private GameObject btnInfo;
    [SerializeField] private GameObject btnMedia;
    [SerializeField] private GameObject btnAudio;

    [Header("Tab properties and references")]
    [SerializeField] private GameObject infoTab;
    [SerializeField] private GameObject mediaTab;
    [SerializeField] private Color defaultTabColor;
    [SerializeField] private Color defaultTabIconColor;
    [SerializeField] private Color activeTabColor;
    [SerializeField] private Color activeTabIconColor;

    void Start()
    {
        // InfoOverlay, InfoOverlay = this;
        //gameController =
        gameController = FindObjectOfType<ITag>("GameController").GetComponent<SceneManager_ARImageTracking>;
        //gameController.GetComponent<ArtefactOverlay>();

        ImageTrackedEvent.OnEventTrigger += SetArtefactData;

        transform.root.gameObject.SetActive(false);

        private void OnEnable()
        {
            // Event registration
            //artEvents.Register(this);

            // Reset buttons and media
            btnAudio.SetActive(true);
            btnMedia.SetActive(true);
            foreach (Transform child in mediaTab.transform)
            {
                Destroy(child.gameObject);
            }

        private void OnDisable()
        {
            btnAudio.GetComponent<AudioButton>().StopClip();
            SetInfoTab(true);
            ResetInfo();
        }

        public void ResetInfo()
        {
            btnAudio.SetActive(true);
            btnMedia.SetActive(true);
            btnAudio.GetComponent<AudioButton>().StopClip();
            SetInfoTab(true);
            foreach (Transform child in mediaTab.transform)
            {
                Destroy(child.gameObject);
            }

            // <<summary>
            // Whether to show the info tab or the media tab.
            // True means show infoTab, false means not infoTab and therefore show mediaTab.
            // </summary>
            // <param name="showInfo">Whether to show the infoTab or instead the mediaTab/</param>
            public void SetInfoTab(bool showInfo)
            {
                if (showInfo)
                {
                    btnMedia.GetComponent<Image>().color = defaultTabColor;
                    btnMedia.transform.GetChild(0).GetComponent<Image>().color = defaultTabIconColor;
                    mediaTab.SetActive(false);
                    infoTab.SetActive(true);
                    btnInfo.GetComponent<Image>().color = activeTabColor;
                    btnInfo.transform.GetChild(0).GetComponent<Image>().color = activeTabIconColor;
                }
                else
                {
                    btnInfo.GetComponent<Image>().color = defaultTabColor;
                    btnInfo.transform.GetChild(0).GetComponent<Image>().color = defaultTabIconColor;
                    infoTab.SetActive(false);
                    mediaTab.SetActive(true);
                    btnMedia.GetComponent<Image>().color = activeTabColor;
                    btnMedia.transform.GetChild(0).GetComponent<Image>().color = activeTabIconColor;
                }
            }
        }
    }
}

```

```

public void SetArtefactData(ArtefactSO artefactData)
{
    MobileDebugScreen.AddDebugLogLine($"Set event data properties of '{arteffectData.name}' for
    infoOverlay");

    if (arteffectData == null)
    {
        MobileDebugScreen.AddDebugLogLine("<color=red>arteffectData is null!</color>");
        throw new System.ArgumentNullException("eventData must be passed!");
    }

    // infoOverlay.gameObject.SetActive(true);
    MobileDebugScreen.AddDebugLogLine("<color=white>Activated infoOverlay!</color>");

    if (arteffectData.Icon == null)
    {
        MobileDebugScreen.AddDebugLogLine("<color=white>(icon is null)</color>");
    }
    else
    {
        headerImage.sprite = arteffectData.Icon;
        MobileDebugScreen.AddDebugLogLine("Icon set");
    }

    if (arteffectData.title.Length > 0)
    {
        MobileDebugScreen.AddDebugLogLine("<color=white>Headline is empty!</color>");
        Debug.Log("Headline is empty!");
    }
    else
    {
        title.text = arteffectData.title; // Add and CHECK Title Function
        MobileDebugScreen.AddDebugLogLine("Headline set");
    }

    if (arteffectData.description.Length > 0)
    {
        MobileDebugScreen.AddDebugLogLine("<color=white>Description is empty!</color>");
        Debug.Log("Description is empty!");
    }
    else
    {
        description.text = arteffectData.description; // Add and CHECK Title Function
        MobileDebugScreen.AddDebugLogLine("Description set");
    }

    //gameController.GetComponent<IndicatorPanel>().color = new Color(1f, 0f, 0f); // set red failure
    screen

    if (arteffectData.audioClips)
    {
        btnAudio.SetActive(false);
        MobileDebugScreen.AddDebugLogLine("<color=white>No audio clips found, AudioButton
        disabled</color>");
        //gameController.GetComponent<IndicatorPanel>().color = new Color(0f, 0f, 0f); // set red failure
        screen
    }
    else
    {
        btnAudio.GetComponent<AudioButton>().SetAudioClip(arteffectData.audioClips);
        MobileDebugScreen.AddDebugLogLine("Audio clips set");
        //gameController.GetComponent<IndicatorPanel>().color = new Color(0f, 1f, 0f); // set red
        failure screen
    }

    if (arteffectData.images?.Count > 0 && arteffectData.videoClips?.Count > 0)
    {
        btnMedia.SetActive(false);
        MobileDebugScreen.AddDebugLogLine("<color=white>No images or video clips found, MediaButton
        disabled</color>");
    }
    else
    {
        if (arteffectData.images?.Count > 0)
        {
            List<Sprite> pictures = arteffectData.images;
            foreach (Sprite picture in pictures)
            {
                if (!picture)
                {
                    continue;
                }
                GameObject newPicture = new GameObject();
                newPicture.transform.SetParent(mediaAnchor.transform);
                newPicture.transform.localScale = Vector3.one;
                AspectRatioFitter arf = newPicture.AddComponent<AspectRatioFitter>();
                arf.aspectMode = AspectRatioFitter.AspectMode.WidthControlHeight;
                arf.aspectRatio = 1f;
                Image imageComponent = newPicture.AddComponent<Image>();
                imageComponent.sprite = picture;
            }
        }
        else
        {
            MobileDebugScreen.AddDebugLogLine("<color=white>No images set.</color>");
        }

        if (arteffectData.videoClips?.Count > 0)
        {
            MobileDebugScreen.AddDebugLogLine("<color=red>THERE SHOULD NOT BE ANY VIDEO YET!!!
            </color>");
        }
        else
        {
            MobileDebugScreen.AddDebugLogLine("<color=white>No video clips set.</color>");
        }

        MobileDebugScreen.AddDebugLogLine("<color=green>Successfully set event data.</color>");
    }
}

```

A quite similar code structure to the LocationOverlay

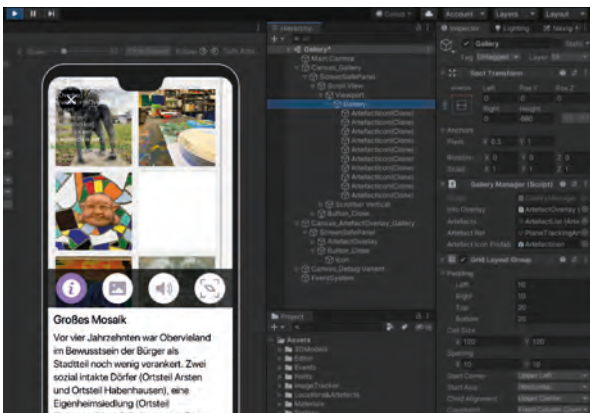


# Gallery

The **Gallery** is an overview of all the artefacts we have officially added to our app. These must be inserted into the ArtefactListSO asset "ArtefactList." In addition, it should be possible to view the artefacts from here in AR to offer an alternative to the local image trackers.

The scene contains two Canvas Prefabs: one for displaying the Gallery and one for the ArtefactOverlay, which will be reused in this scene. The Gallery GameObject is the former Content GameObject of a ScrollView and contained inside the Canvas Prefab. It has the **GalleryManger** script and a GridLayoutGroup component attached to it. The GalleryManager creates interactive icons from a Prefab for every artefact in "ArtefactList". These icons are effectively Button elements nested below the Gallery GameObject therefore automatically arranged by the GridLayoutGroup. They are set up to show the artefact's icon and have the GalleryManager's "SetArtefactAndEnableOverlay" function added as a delegate to their "OnClick" event on the Button component as a new listener. This function takes an ArtefactSO as an argument whose reference is stored within the delegate so no extra structure is needed to store it. It passes the artefact data to the ArtefactOverlay script and enables its whole Canvas to display the information.

So if the icon is touched by the user it displays the artefact information on the ArtefactOverlay.



An example of the Gallery scene in use

```
using UnityEngine;
using UnityEngine.UI;

[RequireComponent(typeof(GridLayoutGroup))]
public class GalleryManager : MonoBehaviour
{
    [SerializeField] private ArtefactOverlay infoOverlay;
    [SerializeField] private ArtefactListSO artefacts;
    [SerializeField] private ArtefactReferenceSO artefactRef;
    [SerializeField] private GameObject artefactIconPrefab;
    private GridLayoutGroup gridLayoutGroup;
    private RectTransform rectTransform;

    private void Awake()
    {
        gridLayoutGroup = GetComponent<GridLayoutGroup>();
        rectTransform = GetComponent<RectTransform>();

        LoadArtefacts();
    }

    (int spacings = (int)gridLayoutGroup.spacing.y * (artefacts.artefacts.Count / 2) +
    gridLayoutGroup.padding.top + gridLayoutGroup.padding.bottom);
    int artefactsHeight = (int)gridLayoutGroup.cellSize.x *
    (int)Mathf.Ceil(artefacts.artefacts.Count / 2f));
    rectTransform.sizeDelta = new Vector2(rectTransform.rect.width, artefactsHeight + spacings);
    }

    private void LoadArtefacts()
    {
        if (!artefacts)
            throw new System.Exception("List of artefacts is null");

        foreach (ArtefactSO artefactData in artefacts.artefacts)
        {
            GameObject newArtefact = Instantiate(artefactIconPrefab, transform);
            newArtefact.GetComponent<Image>().sprite = artefactData.icon;
            newArtefact.GetComponent<Button>().onClick.AddListener(delegate {
                SetArtefactAndEnableOverlay(artefactData); });
        }
    }

    private void SetArtefactAndEnableOverlay(ArtefactSO artefactData)
    {
        Debug.Log(artefactData.name);
        artefactRef.artefact = artefactData;
        if (!infoOverlay.isActiveAndEnabled)
            infoOverlay.gameObject.SetActive(true);
        infoOverlay.transform.root.gameObject.SetActive(true);
        infoOverlay.SetArtefactData(artefactData);
    }
}
```

The GalleryManager is a good example for a script focusing on one task: Instantiating artefact icon Prefabs

This ArtefactOverlay is a Prefab I created from the previous scene, however, I created a new Prefab variant to add a new feature to this particular ArtefactOverlay. This Prefab variant has a fourth menu button indicating plane tracking. It is set up to change the scene to the ARPlaneTrackingScene(Gallery) so the artefact model can be seen in AR on a tracked surface. This scene is almost identical to the ARImageTrackingScene but I removed the ArtefactOverlay and the ARTrackedImageManager script from the ARSessionOrigin and added the ARPlaneManager, ARRaycastManager, and PlaceObjectOnPlane scripts. The first and second script is part of the AR Foundation plugin handling the plane tracking while the third one only checks for any touch input to instantiate an artefact Prefab.

To instantiate the right artefact Prefab a reference to the corresponding artefact is needed, and since we are changing the scene it is also necessary to pass this information from the Gallery scene. So I created a new ScriptableObject class called "ArtefactReferenceSO" which has a serialized public ArtefactSO field.

```
using UnityEngine;

[CreateAssetMenu(fileName = "NewArtefactReference", menuName = "Artefact/Artefact Reference")]
public class ArtefactReferenceSO : ScriptableObject
{
    [SerializeField] public ArtefactSO artefact;
}
```

A ScriptableObject that holds a reference to another ScriptableObject.

I created a new asset from this class, the "PlaneTrackingArtefact" which is the reference to the artefact that is going to be instantiated in the other scene. Then I added a serialized ArtefactReferenceSO field to both the GalleryManager class and the PlaceObjectOnPlane class to drag and drop the created asset onto the scripts. Next, I made the GalleryManager set the value for the artefact reference inside the "SetArtefactAndEnableOverlay" function. The reference will be set whenever an artefact icon is interacted with. An artefact will always be set before the scene is changed because the button for changing is displayed after the corresponding function gets called. After changing the scene, the PlaceObjectOnPlane script can use the ArtefactReferenceSO asset to instantiate the referenced artefact's Prefab.

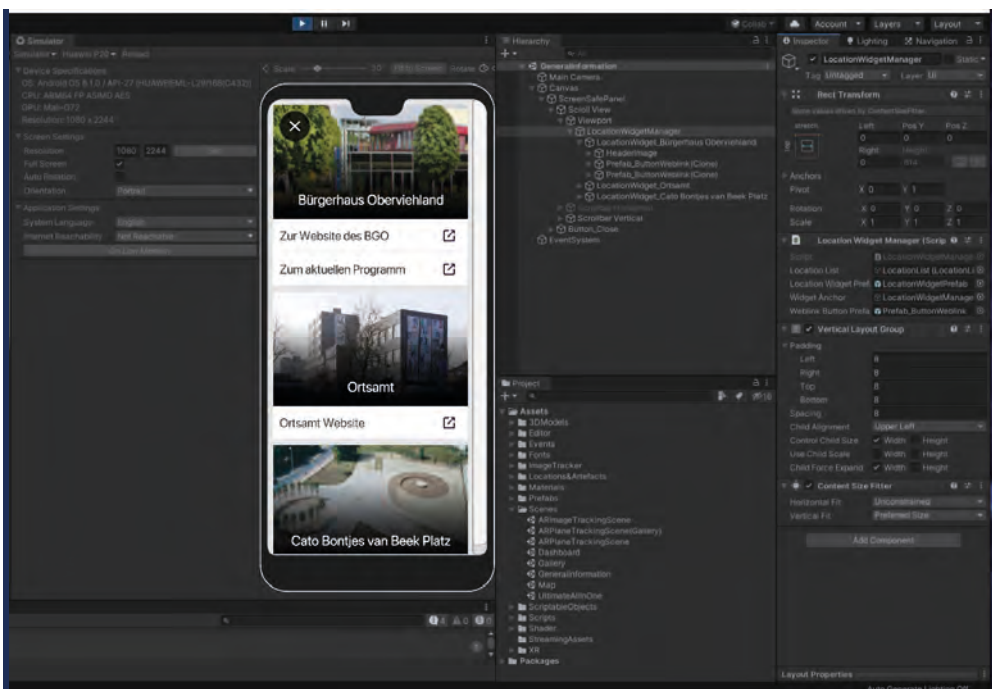
This feature was implemented in a very short time. The benefits of a modular design with Prefabs and ScriptableObjects in Unity are now clear to me. It's accelerating the development if small pieces can be reused to alter or build new modules.

## General Information

The **General Information** scene is similar to the Gallery. It shows a list of all locations we add to our app by adding LocationSO assets to the "LocationList". The scene has only one Canvas Prefab nesting a ScrollView. Beneath its Content GameObject, I added another GameObject named "LocationWidgetManager" which has a **LocationWidgetManager** script attached to it.

This script has four serialized fields to receive a LocationListSO asset and three GameObject references for a LocationWidgetPrefab, a WidgetAnchor, and a WebLinkButton Prefab.

In its "Awake" function, the script creates a location widget for each LocationSO in the "LocationList" by instantiating the LocationWidgetPrefab parented to the WidgetAnchor in the scene. The script then sets the information for the title and header and eventually creates web link buttons nested below the new Prefab instance.





```

using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;

public class LocationWidgetManager : MonoBehaviour
{
    [SerializeField] private LocationListSO locationList;
    [SerializeField] private GameObject locationWidgetPrefab;
    [SerializeField] private GameObject widgetAnchor;
    [SerializeField] private GameObject weblinkButtonPrefab;

    private void Awake()
    {
        foreach (LocationSO locationData in locationList.locations)
            CreateLocationWidget(locationData);
    }

    private void CreateLocationWidget(LocationSO locationData)
    {
        GameObject newLocationWidget = Instantiate(locationWidgetPrefab, widgetAnchor.transform);
        newLocationWidget.name = "LocationWidget_" + locationData.title;

        Image headerImage = newLocationWidget.GetComponentInChildren<Image>();
        Text locationTitle = newLocationWidget.GetComponentInChildren<Text>();

        headerImage.sprite = locationData.headerImage;
        locationTitle.text = locationData.title;

        // Create Weblink buttons
        if (locationData.weblinkButtons.Count > 0)
        {
            List<WeblinkSO> weblinkButtons = locationData.weblinkButtons;
            foreach (WeblinkSO weblink in weblinkButtons)
            {
                if (!weblink)
                    continue;
                Debug.Log($"Creating a weblink button for {weblink.buttonText}");
                GameObject newWeblinkButton = Instantiate(weblinkButtonPrefab,
                    newLocationWidget.transform);
                newWeblinkButton.transform.SetSiblingIndex(newLocationWidget.transform.childCount);
                newWeblinkButton.GetComponentInChildren<Text>().text = weblink.buttonText;
                newWeblinkButton.GetComponent<Button>().onClick.AddListener(delegate {
                    OpenURL(weblink); });
            }
        }

        private void OpenURL(WeblinkSO weblinkData)
        {
            Application.OpenURL(weblinkData.url);
        }
    }
}

```

# Conclusion

**Reflexion**  
**Problems**  
**Technical Outlook**

## Conclusion

Creating the architecture for the app prototype was a lot of fun in the first place, but also a challenge that brought me to my limits as a programmer and Unity developer. It was a long development journey that led to the current state of the project, filled with quite many small issues to solve. I have learned a lot of new things in many possible ways, as during this project I did not only encounter some difficulties but also have been forced to put a lot of time and effort into resolving these issues. This has "forced me to grow" in my role as a Unity developer and programmer, and I am very grateful for this kind of on-hand learning experience. I am looking forward to further continuing this project transcending the Bachelor Thesis.

A lot of in-between steps, small experiments with the AR Foundation functionalities, and a lot of bugs I had to fix are not listed explicitly. For example, I can quickly mention the struggle I had with the AR Foundation plugin and its `ARTrackedImageManager`. After about three hours of research, I found no easier way to check for a specific tracked image than comparing the name of the reference image with all my artefacts from the `ArtefactListSO`. Other Unity users online have complained about the same issue because it was not even possible to compare the tracked image itself as it gets encoded on the build.apk and therefore is not comparable to its original image anymore. Also, the tracked Prefab was a bit disappointing since it was only this one that would be instantiated if any image was tracked. I wonder if this was meant to be used differently, for example, to create multiple reference libraries so all images in this library would be equal target images for the same AR Event. In this case, multiple `ARTrackedImageManager` in one scene would be needed to handle each `ReferenceLibrary` with its corresponding `Tracked Prefab`. In our case, that would mean that we would have to create a reference library and a new `ARTrackedImageManager` instance in the scene for every image we want to track to display its own artefact. This seemed very unlikely to me so I didn't try it at all. It would break the app's capabilities as soon as we added more and more image trackers to it, so I was forced to find a way to compare and instantiate artefacts to avoid it. A problem that I did not pay much attention to until now is the cleanup of scenes. Sometimes this causes the app to crash when the `ARImageTrackingScene` is loaded for the second time on a native device. This did not happen in the editor due to sufficient resources on the desktop machine. As far as I investigated the problem, resetting the AR session should fix the problem.

Looking back on the history of our Git repository, I can see a lot of iterations of code and UI building. The project seemed quite overwhelming at the beginning and I did lack a clear concept since we could not apply the systems development life cycle

because of the special working conditions in the middle of the pandemic, so the step of the evaluation was completely missing. But without even noticing, I started working like I had planned another iterative development cycle all by myself, so the project took shape quickly and turned out much to my satisfaction. Starting with a lot of experimental builds to only test the raw features of AR Foundation, then learning to build and shape the UI in Unity, I finally found my way through the project. I could put all the small pieces together and had an almost working app early on. That is where I started to iterate the pilot from week to week bringing in more and more of our designs. Every small goal and milestone I set helped me to evolve the project for the next meeting.

In the end, the code itself for this project seems unimpressive to me, but its strength comes from its simplicity which I have developed over time. This simplicity could only be achieved by getting to know parts of Unity that were previously unknown to me and acquiring skills I wouldn't even have imagined I'd need before the project, of one is building complex UI structures. In its current state, the project still has some messy ruins of old code, improperly named variables and classes, and also some chaotic folders in the project window that I have to clean up. I will need to reorganize and rename those before I can continue adding more content or features to the app, but as said in the beginning it is a priority to create a working prototype despite it being messy on the backend. The most time-consuming work was finding a reliable UI structure that was working well with my code while making sure it's flexible enough to fit every device. I had to re-build a lot of screens and UI elements such as the Location- and ArtefactOverlay, and still there are some points I wish I had the time to optimize them even more. The proper anchoring of UI elements, using layout group components, as well as the nesting of such elements, was probably where I learned the most. The further I got with the project the faster the development got in which ScriptableObjects and Prefabs were the key factors. I never had a project before where the documentation was much more in danger of being unfinished than the project. It was hard for me to find the balance between providing sufficient documentation of the process, doing research for the project, and providing my experience.

Now that the foundation of the app, the pilot, has reached a state in which all functionalities are working and can be shown, more features can be implemented. I have some ideas to improve the experience of artefacts, as we barely focused on their appearance and interaction with the users up until now for given reasons. First of all,

I want to create more diverse artefact objects. We already have a bunch of scanned 3D objects, but it is also important to bring artefacts that are less spatial "to life" as well. For example, it was Nadine's wish to show a banner that was hanging over the Cato Bontjes van Beek Platz for an event, inside the app, but I wanted to put it back where it originally was - virtually. I had the idea to use the template of the banner to recreate it in Unity as a textured plane, animate it and show it in AR to remember the event. Also, more interaction directly with the artefact object is needed, as well as moving (animated) objects. This would make them more dynamic, which gives the artefacts additional character, depending on the single object. Another feature I would like to bring into the app is spatial sound, with additional speakers such as headphones, of course, to enable the sound dimension for our artefacts. Much of the immersion is achieved through the proper use of sound. We have a lot more ideas speaking only of the artefacts and presentation of their content. Aspects like gamification were also discussed, but it is dangerous to put more effort into these things without proper evaluation of the current state. We hope that this will be possible soon and look forward to summer, as we expect the pandemic will continue to be contained.

Working together with Christine was an overall very pleasant experience. She has contributed a lot, even though we did separate our tasks and have divided all the work between ourselves. Still, we always were in the picture about the work the other person did and the exchange of information worked very well. In general, we worked in a very organized and goal-oriented way, which reminds me of Scrum. All things aside, I am ultimately very proud of us for what we have managed to create in a limited time and during a time of general distress in this pandemic. I feel more confident than ever as a developer and have gained so much more confidence in my own skills so that I am not afraid of what is yet to come for me in the future.



# Integration und Fazit



## **Integration of Content**

**Verwaltung der Inhalte vom BGO**

**Working Conditions**

**Fazit**

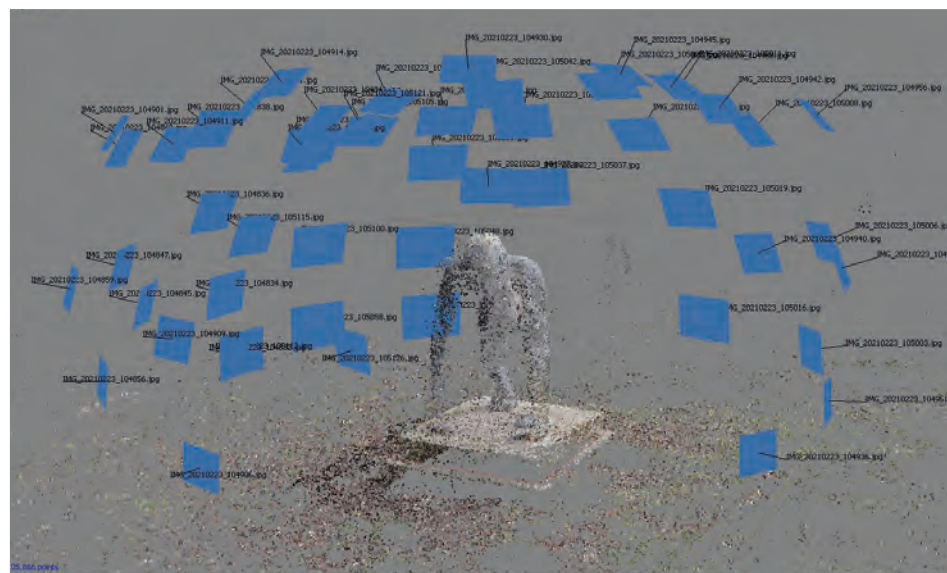
**Literaturverzeichnis**

## Integration of Content

The **digitization** of the artefacts is an important part of the project, as they should be in the foreground of the app. Each artefact in the app should have a 3D representation, which should be displayed on the image tracker. At the moment, we don't have enough material to represent like this, as creating 3D models requires access to real objects that we don't get because of the ongoing contact restrictions. Nevertheless, I had the opportunity to digitize a stele that was composed of individual parts that children of different cultures had designed in the workshop of the BGO. There is also a single part of an incomplete stele and a sculpture standing in front of the BGO.

### Photogrammetry

To digitize the objects, I used the method of **photogrammetry**, with which I have been familiar for almost as long as I have been with Unity. Overlapping images are taken from different perspectives around the object to be scanned to generate a 3D object using special programs. In the process, the scanning, i.e. the taking of the images, is the most important, since they are decisive for the success and quality of the model. Not only the composition of the frames around the object but also the influences from the environment play a role. Mirroring surfaces and shadows are disturbing factors because they influence the appearance of the model's shape in the images. The brightness should also be constant on all pictures, to be able to transmit the coloration of the model unadulterated later. The images must overlap for the most part so that as many striking key points as possible can be detected by the program on the image pairs. From the perspective differences of these key points, the program algorithm can determine the distance and position of these points in relative space.



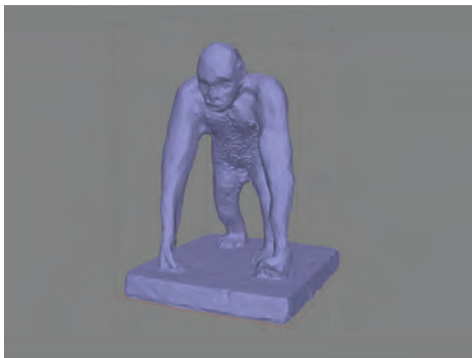
Arrangement of the images shot from the original object and the point cloud generated from them in Agisoft PhotoScan

In the first run, the images are arranged virtually according to the few key points, to generate as many key points as possible from the images in the second step, creating a **dense point cloud**.



The dense point cloud, which makes the object (monkey) already well visible

In the next step, these points are joined together to form the 3D mesh. After that, an additional texture can be created for the model. Between these steps, it is possible to make different corrections to the point clouds or the settings of the algorithms in some programs.



The 3D mesh of the object generated from the dense point cloud



The same 3D model, but with additionally generated texture

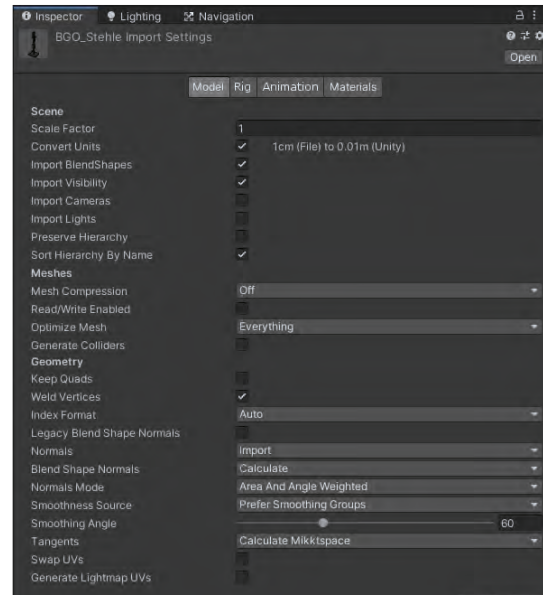
Ideally, a true-to-original digital image of the original object has now been created, which can be integrated into our app.

## Integration of 3D Models into Unity

Unity offers various import settings for importing 3D models, which can help to optimize models of different formats and properties and prepare them for use in Unity. Optimally, models should be exported uniformly for Unity beforehand, but this cannot always be guaranteed.

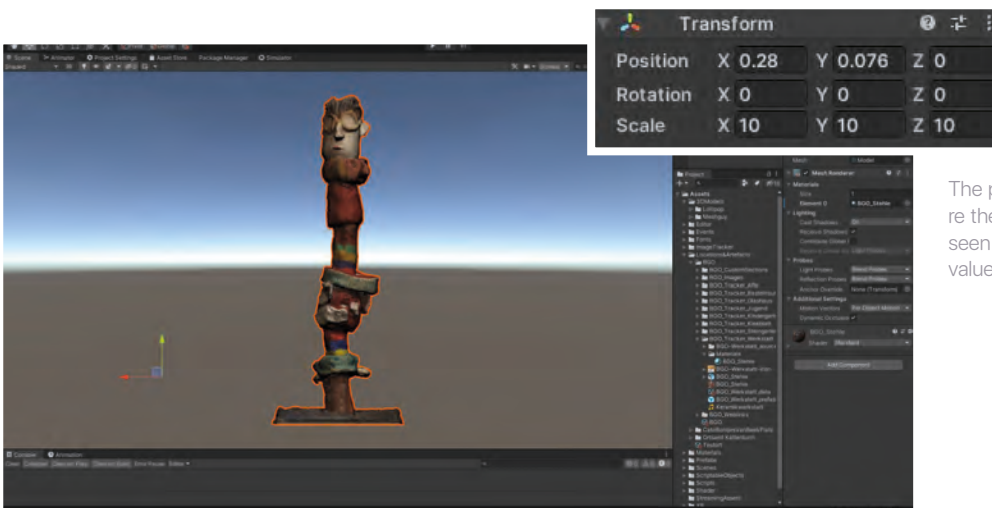
For the model to be used as an artefact in Unity, a Prefab must be made from the asset. To do this, it is temporarily inserted into a scene and prepared as follows:

It is optimal from the workflow point of view if the model is parented to a new GameObject at the origin of the scene with a uniform Transform component. This allows the model to be transformed relatively to it and preserves a uniform transformation as a Prefab.



An example of Unity's import settings for 3D models

When rotating and scaling it is important to consider the orientation and size of the image tracker of the artifact. Depending on whether the tracker is positioned horizontally or perpendicularly, the model also needs to be rotated accordingly so that it is displayed correctly when tracking. Size also plays a role, because the image trackers do not have a uniform size. They are entered in Unity with their real dimensions so that they are adapted to the Unity internal metric system. So the models must always assume their true size based on the dimensions in Unity to be displayed correctly. Once the model has been prepared, it can be stored as a Prefab in the project window and referenced in the artefact.



The prepared stele in Unity, where the offset of the model can be seen in the scene view and in the values of the transform

## Verwaltung der Inhalte vom BGO



### Integration von Text, Bild und Ton

Inhalte gab es von den Stakeholdern genug. Es wurde alles zusammengetragen, was die Ressourcen hergaben. Fotos aus vergangenen Zeiten, Anekdoten und Geschichten habe ich in Form von Text-Dateien bekommen. Diesen Berg an Informationen gilt es erstmal zu sortieren. Die Geschichten mussten zum Teil gekürzt und Bilder auf Rechte geprüft werden.

Die interessantesten Texte habe ich markiert und mit Bildern verknüpft. Die Auswahl habe ich mit den Stakeholdern abgestimmt und sie gebeten die finalen Texte von einem Sprecher als Audiodatei einzusprechen. Wenn die Informationen pro Ort oder Artefakt vollständig waren, habe ich diese in das Unity Projekt eingespeist. Jacob hat für jedes Thematik (**Titel, Text, Bilder, Audio, Webseiten**) Input-Felder angelegt, in der ich fließend die Daten einfügen konnte. Mit dem Styleguide und den Designs habe ich die Interface-Elemente detailgetreu in Unity übertragen. Schriftart und Größe angepasst, die Farbe der Button-Panels und deren Selektierung bestimmt und Layout Abstände positioniert. Dabei habe ich die Visualisierung regelmäßig auf unterschiedlichen Gerätegrößen getestet, um die richtige Darstellung zu garantieren.

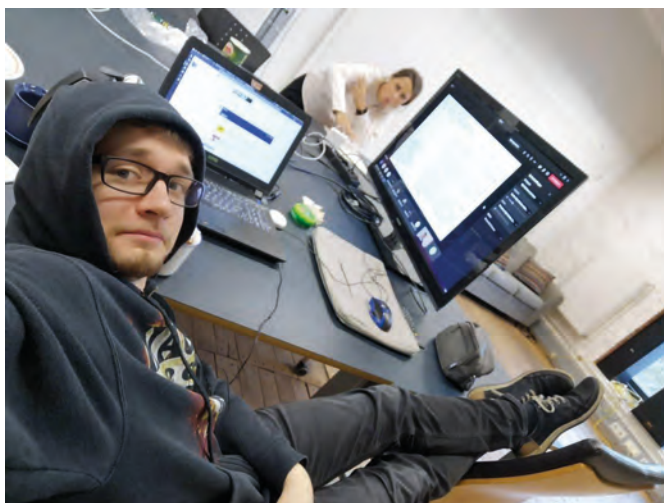


## Working Conditions

Throughout the work on the Quartier AR project, the enormous scale of the COVID-19 pandemic became apparent and also felt. I want to address this fact consciously first because the restrictions imposed by the government's countermeasures fundamentally influenced the initial situation of our working conditions. This has put us to a rather severe test in our project. The constantly changing circumstances forced us to set up new ways of working and we had to change our organization.

The distance that opened up between people and, of course, also the students during that time, was a real wedge in our student life. The many advantages offered by our familiar way of working in the HfK's premises had to be overcome with this. Before the pandemic, we were part of a lively group that enjoyed meeting up in room 2.11.070 in Speicher XI to work together and help each other wherever possible. The healthy sense of community that was a big part of our motivation just fell away. In the "home office" there was no more direct interaction and no separation of the working and living environment, which is very important for many. For me, it always took a lot of effort to sit down and start working. It sometimes took hours and a lot of preparation to get into a focused workflow. One's own home is, at least for me, not designed for concentrated work. My workstation at home is completely different and has a completely different motivation that is entirely tailored to my well-being and entertainment. That I was able to work hard on projects here for nights was because I had a corresponding compensation outside my room and the apartment. Moreover, the activities here were not characterized by any compulsion to perform, but by my interest and ambition. It took a lot of discipline to not be distracted by external influences. Whether it's due to housework, roommates, or other little things that one likes to put in between quickly to not have to work for a short time; acute procrastination has become an increasing hurdle during the pandemic.

In autumn it was possible to work together at the HfK a few times, which helped us a lot in the planning phase. However, with the tightening of the second wave measures, this option was lost. There were no more quick agreements, no direct help with technical problems, no "looking over the shoulder" or anything like that.



work together at the HfK

From now on, we were dependent on communication solely via the Internet. Various platforms became extremely relevant. The HfK organized its events via Microsoft Teams, the meetings with the contact persons in the BGO were held via Zoom and the HfK community moved to Discord. We sent short messages if possible via WhatsApp or by email, but it could not replace direct contact. The many communication channels quickly became very confusing. As a result, messages were sometimes forgotten, a reply was postponed or not answered in time, which became a problem for us in some situations, especially with time-consuming requests that could otherwise have been resolved in person. This also made planning more difficult, as each step needed at least a week in advance so that everyone involved had enough time to react. It was very frightening that some emails were only read and answered once a week. We were forced to think ahead and take hasty steps to prevent processes from dragging on too long.

Our time management was the key to success, but also our biggest weakness. Much was unpredictable, dates were constantly on the verge of new tightening of the exit restrictions, and a tense situation developed. We had to plan much more in advance and accept spontaneous changes regularly, which cost us extra time. We needed additional agreements with the stakeholders in the BGO. We were able to organize a few virtual meetings to present our stand with all the technical means we could use via videoconferencing, and then respond to their requests and feedback. A working concept such as Scrum was therefore hardly feasible, as we were unable to obtain a regularity or precise requirements from the stakeholders. We had to rely on our experience and also our intuition. Nonetheless, we unconsciously set up a certain cycle in our project management, whereby everyone set up specific working hours for themselves. Milestones and fixed deadlines helped us to live up to this, unfortunately also at the expense of project implementation in some places.

Also, the concept of the software development lifecycle could not be implemented correctly due to a lack of evaluation. There was a lack of possibilities and opportunities to run supervised test runs with different test ends, which is why the pilot also embodies a rather open concept, as already mentioned at the beginning. The only control structure for this was the presentation of the functions and the handling of the app, which we accompanied with many explanations and graphics to make the understanding as easy as possible.

With combined efforts and quick adaptation, we were able to continue the project plan. Biweekly meetings between Christine and me could then be organized again in spring, as we were able to book and use an external working space in compliance with safety regulations and hygiene concepts. We were able to make the most effective use of the common working hours. We made the greatest progress when we were able to work on our concepts in parallel.



work together at the external working space

Various tools, above all Git and Google Documents, helped us to research and align our knowledge and work levels. It was important to know what the other person was working on to avoid duplication and mistakes.

In the end, there was no time to incorporate the last open points into the work, but with the end of the Bachelor's thesis, the project is fortunately not yet finished.



# Fazit

Unsere Bachelorarbeit zum Thema „**Interaktive Applikation zum Darstellen von lokalen Artefakten**“ war vermutlich unser aufregendstes Projekt, bei dem wir sehr viel gelernt haben. Die größte Herausforderung war es zu einem Teil das ganze Team immer aktuell zu halten, auf die Wünsche der Stakeholder einzugehen aber auch eine Grenze zu finden. Zum anderen forderte die praktische Arbeit viel Zeit und Ausdauer, aber auch Lernbereitschaft und Expertise. Vieles davon erreichten wir durch den Entwicklungsprozess, der so auch unsere persönliche Entwicklung voran trieb. Als die Rahmenbedingungen feststanden, dass wir uns regelmäßig absprechen und weiter am Projekt arbeiten, konnten wir trotz der gegebenen Umstände gemeinsam produktiv sein. Wir haben Programme wie „Teams“ und „Invision“ schätzen gelernt, mit denen wir oft kommuniziert haben. Durch das interdisziplinäre Team waren viel mehr Möglichkeiten verfügbar. Beim Absprechen und diversen Diskussionsrunden sind immer ganz neue Ideen entstanden. So konnten wir unsere Sichtweisen fortwährend erweitern. Trotz fester Rollen gab es fließende Übergänge in unseren Tätigkeiten, die uns Freiheiten im Aufgabenbereich des anderen zuließen.

Die größte Motivation war dabei, dass die App einen Mehrwert für alle bieten soll. Unser Ziel für die Zukunft ist, dass die App eine langfristige Motivation zur Nutzung bietet und über die kommenden Jahre verbessert, erweitert und, in Hinsicht auf die Inhalte, gepflegt wird.

## Literaturverzeichnis / Christine

- Adobe.** (o. J.). Software und Mobile Apps für UI- und UX-Design | Adobe. Abgerufen 20. Mai 2021, von <https://www.adobe.com/de/creativecloud/ui-ux.html>
- Apple.** (o. J.). iPhone 12 und iPhone 12 mini. Apple (Deutschland). Abgerufen 20. Mai 2021, von <https://www.apple.com/de/iphone-12/>
- badmin.** (2019, August 16). Einsatz von Splash Screens in der App-Entwicklung. Bitforge - Die App- und Augmented Reality-Agentur in Zürich. <https://bitforge.ch/apps/splash-screen/>
- Ballmer, S., ContributorCEO, & Corporation, M.** (2002, Dezember 5). CES 2010: A Transforming Trend -- The Natural User Interface. HuffPost. [https://www.huffpost.com/entry/ces-2010-a-transforming-t\\_b\\_416598](https://www.huffpost.com/entry/ces-2010-a-transforming-t_b_416598)
- Brancheau, J. C., Schuster, L., & March, S. T.** (1989). Building and implementing an information architecture. ACM SIGMIS Database: the DATABASE for Advances in Information Systems, 20(2), 9–17. <https://doi.org/10.1145/1017914.1017916>
- Design.** (o. J.). Material Design. Abgerufen 20. Mai 2021, von <https://material.io/design>
- designenlassen.** (o. J.). App Icons designen – das musst Du beachten. designenlassen.de Blog. Abgerufen 20. Mai 2021, von <https://www.designenlassen.de/blog/app-icons-designen-das-muessen-sie-beachten/>
- farbenundleben.** (o. J.). Farben in Religion und Kultur. Abgerufen 20. Mai 2021, von [http://www.farbenundleben.de/kultur/religion\\_kultur.htm](http://www.farbenundleben.de/kultur/religion_kultur.htm)
- Flaticon.** (o. J.). Flaticon, the largest database of free vector icons. Flaticon. Abgerufen 21. Mai 2021, von <https://www.flaticon.com/>
- Garcia, L. A., Oliveira Jr, E., Leal, G. C. L., Morandini, M., & Urbanowski, S.** (2020). Adaptations of Scrum roles in software projects: Survey and representation tentative with feature models. Proceedings of the 34th Brazilian Symposium on Software Engineering, 47–51. <https://doi.org/10.1145/3422392.3422403>
- Garrett, J. J.** (2010). The Elements of User Experience: User-Centered Design for the Web and Beyond (2nd Aufl.). New Riders Publishing.
- Gegenfurtner, K. R., & Goldstein, E. B.** (2014). Wahrnehmungspsychologie: Der Grundkurs (K. N. Oettingen & G. Plata, Übers.; 9. Aufl. 2015 Edition). Springer.
- Google Bilder.** (o. J.). Stadtmusikanten-landscape.jpg (900×630). Abgerufen 21. Mai 2021, von <https://organicstrategies.de/wp-content/uploads/bb-plugin/cache/Stadtmusikanten-landscape.jpg>
- Google Play.** (o. J.). Google Play. Abgerufen 20. Mai 2021, von <https://play.google.com/store?hl=de&gl=DE>

- Ho, A. G.** (2016). Recognising the Importance of Visual Elements in Experience Design for Motivation. 한국HCI학회 학술대회, 127–133. <https://doi.org/10.17210/hcik.2016.01.127>
- Holzinger, A.** (2005). Usability engineering methods for software developers. *Communications of the ACM*, 48(1), 71–74. <https://doi.org/10.1145/1039539.1039541>
- Invision.** (o. J.). Free Online Whiteboard For Team Collaboration | InVision. Abgerufen 20. Mai 2021, von <https://www.invisionapp.com/freehand>
- Moore, R. J., & Arar, R.** (2019). *Conversational UX Design: A Practitioner's Guide to the Natural Conversation Framework*. Association for Computing Machinery.
- Moser, C.** (2012). User Experience Design. In C. Moser (Hrsg.), *User Experience Design: Mit erlebniszentrierter Softwareentwicklung zu Produkten, die begeistern* (S. 1–22). Springer. [https://doi.org/10.1007/978-3-642-13363-3\\_1](https://doi.org/10.1007/978-3-642-13363-3_1)
- Paletton.** (o. J.). Paletton—The Color Scheme Designer. Abgerufen 20. Mai 2021, von <https://paletton.com/#uid=1000u0kllllaFw0g0qFqFg0w0aF>
- Parhi, P., Karlson, A. K., & Bederson, B. B.** (2006). Target size study for one-handed thumb use on small touchscreen devices. *Proceedings of the 8th conference on Human-computer interaction with mobile devices and services*, 203–210. <https://doi.org/10.1145/1152215.1152260>
- Park, S., & Maurer, F.** (2009). The role of blogging in generating a software product vision. *2009 ICSE Workshop on Cooperative and Human Aspects on Software Engineering*, 74–77. <https://doi.org/10.1109/CHASE.2009.5071414>
- Sketch.** (o. J.). It all starts with Sketch. Sketch. Abgerufen 20. Mai 2021, von <https://www.sketch.com/>
- Was ist ein Impressum?** - Einfach erklärt. (2018, Februar 20). <https://blog.zeta-producer.com/impressum-glossar/>

## Literaturverzeichnis / Jacob

**Azuma, R. T.** (1997). A Survey of Augmented Reality. *Presence: Teleoperators and Virtual Environments*, 6(4), 355–385. <https://doi.org/10.1162/pres.1997.6.4.355>  
**CONNECTED REALITY.** (2020, 29. Oktober). Augmented Reality, AR, WebAR. **CONNECTED REALITY** seit 2011. <https://connected-reality.com/augmented-reality/>

**Feiner, S., MacIntyre, B., Höllerer, T. & Webster, A.** (1997). A touring machine: Prototyping 3D mobile augmented reality systems for exploring the urban environment. *Personal Technologies*, 1(4), 208–217. <https://doi.org/10.1007/bf01682023>

**Kopp, A.** (2018, 1. Februar). Virtual Reality im Museum: Kultur erleben. Axel Kopp. <https://www.axelkopp.com/2018/01/virtual-reality-im-museum-kultur-erleben/>

**Markgraf, D.** (2018, 16. Februar). Augmented Reality. *Gabler Wirtschaftslexikon*. <https://wirtschaftslexikon.gabler.de/definition/augmented-reality-53628>

**Mircosoft-Autoren.** (2021, 28. Januar). A Tour of C# - C# Guide. Microsoft Docs. <https://docs.microsoft.com/en-us/dotnet/csharp/tour-of-csharp/>

Object Tracking | Fraunhofer IGD. (o. D.). Fraunhofer IGD | Fraunhofer-Institut für Graphische Datenverarbeitung IGD. Abgerufen am 14. Mai 2021, von <https://www.igd.fraunhofer.de/kompetenzen/technologien/object-tracking>

**Paradiso, J. A. & Landay, J. A.** (2009). Guest Editors' Introduction: Cross-Reality Environments. *IEEE Pervasive Computing*, 8(3), 14–15. <https://doi.org/10.1109/mprv.2009.47>

**Ryan Hipple.** (2017, 20. November). Unite Austin 2017 - Game Architecture with Scriptable Objects [Video]. YouTube. [https://www.youtube.com/watch?v=raQ3iH-hE\\_Kk&t=136s](https://www.youtube.com/watch?v=raQ3iH-hE_Kk&t=136s)

Unity Technologies. (o. D.-a). AR- und VR-Spiele. Unity. Abgerufen am 8. Mai 2021, von <https://unity.com/de/solutions/ar-and-vr-games>

**Unity Technologies.** (o. D.-b). Das AR Foundation-Framework von. Unity. Abgerufen am 8. Mai 2021, von <https://unity.com/de/unity/features/arfoundation>

**Unity Technologies.** (o. D.-c). Editor Scripting. Unity Learn. Abgerufen am 18. Mai 2021, von <https://learn.unity.com/tutorial/editor-scripting>

**Unity Technologies.** (o. D.-d). Mobile games. Unity. Abgerufen am 8. Mai 2021, von <https://unity.com/solutions/mobile>

**Unity Technologies.** (o. D.-e). Multiplattform. Unity. Abgerufen am 17. Mai 2021, von <https://unity.com/de/features/multiplattform>

**Unity Technologies.** (o. D.-f). Three ways to architect your game with ScriptableObjects. Unity. Abgerufen am 11. Mai 2021, von <https://unity.com/how-to/architect-game-code-scriptable-objects#three-pillars-game-engineering>

**Unity Technologies.** (o. D.-g). Unity - Manual: Important Classes - MonoBehaviour. Unity Documentation. Abgerufen am 11. Mai 2021, von <https://docs.unity3d.com>

com/Manual/class-MonoBehaviour.html

**Unity Technologies.** (2017a, Juli 12). Unity - Manual: Designing UI for Multiple Resolutions. Unity Documentation. <https://docs.unity3d.com/560/Documentation/Manual/HOWTO-UIMultiResolution.html>

**Unity Technologies.** (2017b, August 1). Unity - Manual: GameObjects. Unity Documentation. <https://docs.unity3d.com/Manual/GameObjects.html>

**Unity Technologies.** (2018, 15. Oktober). Unity - Manual: ScriptableObject. Unity Documentation. <https://docs.unity3d.com/Manual/class-ScriptableObject.html>

**Unity Technologies.** (2020a, Oktober 6). Canvas | Unity UI | 1.0.0. Unity Documentation. <https://docs.unity3d.com/Packages/com.unity.ugui@1.0/manual/class-Canvas.html>

**Unity Technologies.** (2020b, Oktober 6). Canvas Scaler | Unity UI | 1.0.0. Unity Documentation. <https://docs.unity3d.com/Packages/com.unity.ugui@1.0/manual/script-CanvasScaler.html>

**Unity Technologies.** (2021a, Mai 14). Unity - Manual: GameObjects. Unity Documentation. <https://docs.unity3d.com/2021.2/Documentation/Manual/GameObjects.html>

**Unity Technologies.** (2021b, Mai 15). Unity - Manual: The Game view. Unity Documentation. <https://docs.unity3d.com/Manual/GameView.html>

**Wikipedia-Autoren.** (2011, 14. Mai). Unity (Spiel-Engine). Wikipedia. [https://de.wikipedia.org/wiki/Unity\\_\(Spiel-Engine\)](https://de.wikipedia.org/wiki/Unity_(Spiel-Engine))

## Danksagung

Unsere gemeinsame Reise begann im Dezember 2020. Damals trafen wir auf Elke Munderloh und Nadine Scheffler in einem Café in der Überseestadt. Dort erzählten sie uns zum ersten Mal, was es mit dieser Idee für eine App auf sich hatte. Wir waren gefangen davon und wollten das Projekt unter einer Bedingung umsetzen - und zwar als Bachelorarbeit.

Nun ist seitdem mehr als ein halbes Jahr vergangen und wir blicken zurück auf eine sehr ereignisreiche Zeit. Obwohl die Pandemie die Welt fest im Griff hielt, ließen wir uns nicht davon abhalten die große Idee in ein Pilotprojekt zu verwandeln. Es war vor allem die moralische Stütze Elke, die uns immer wieder gut zusprach, ermutigte und mit Ihrer Begeisterung alle im Team ansteckte. Es waren auch die Worte von Nadine, die uns immer wieder daran erinnerten, dass wir gebraucht werden. Es waren aber auch die Hände vieler anderer im BGO, allen voran Dieter, die uns mutvoll unterstützten und Artefakte aus den Tiefen des Archives bargen.

Doch auch von unserer persönlichen Seite bekamen wir von Freunden und Familie den nötigen Rückhalt, der uns nicht nachgeben lies.

Auch unseren Prüfern haben wir viel zu verdanken. Verlässlichkeit, Flexibilität und guter Rat. Ohne Euch wären wir niemals so weit gekommen und hätten nicht so viel geschafft. Wir dachten nicht eine Sekunde daran, dass wir es nicht schaffen würden - nur durch Euch!

Dafür wollen wir Euch von Herzen danken und Euch diese Seite in unserer Arbeit widmen, denn Ihr seid ein Teil davon.



# Anhang

Die Inhalte der beiliegenden **CD** sind wie folgt:

- **APK:** APK der App
- **Bachelorarbeit:** Bachelorarbeit als PDF Datei
- **Code:** Quellcode der App
- **Screendesign:** Interaktiver Klick-Dummy und Screendesign als PDF Datei



