FH AACHEN
UNIVERSITY OF APPLIED SCIENCES

FH Aachen

Fachbereich 09
Medizintechnik und Technomathematik
Studiengang Angewandte Mathematik und Informatik

Seminararbeit

Eine prototypische Implementierung für Responsive Design und Touchbedienung im Webtool Miori Boards

Yannic Sieger
Matr.-Nr.: 3630695

Betreuer: **Prof. Dr. Alexander Voß**

Betreuerin: **Olga Wolf, Dipl.-Math.**

15. Dezember 2025

Eidesstattliche Erklärung

Ich versichere hiermit, dass ich die vorliegende Arbeit selbstständig verfasst und keine anderen als die im Literaturverzeichnis angegebenen Quellen benutzt habe.

Stellen, die wörtlich oder sinngemäß aus veröffentlichten oder noch nicht veröffentlichten Quellen entnommen sind, sind als solche kenntlich gemacht.

Die Zeichnungen oder Abbildungen in dieser Arbeit sind von mir selbst erstellt worden oder mit einem entsprechenden Quellennachweis versehen.

Diese Arbeit ist in gleicher oder ähnlicher Form noch bei keiner anderen Prüfungsbehörde eingereicht worden.

Im Rahmen der Erstellung dieser Arbeit wurde das KI-System "KI.connect.nrw" unterstützend zur sprachlichen Überarbeitung sowie zur fachlichen Reflexion und Präzisierung eigenständig entwickelter Argumente genutzt. Eine Übernahme von KI-generierten Texten oder inhaltlichen Lösungsvorschlägen erfolgte nicht. Sämtliche fachlichen Aussagen, Bewertungen und Schlussfolgerungen wurden eigenständig erarbeitet und verantwortet. Die Nutzung erfolgte im Einklang mit der Zweckbestimmung des Systems sowie unter Beachtungdatenschutz- und urheberrechtlicher Vorgaben.

Name: Yannic Sieger
Aachen, 15. Dezember 2025

Abstract

Im Zuge des Trends zu „New Work“ und der fortschreitenden Digitalisierung verlagert sich die Arbeit zunehmend auf mobile Endgeräte wie Smartphones und Tablets. Das Webtool Miori Boards, welches als zentrale Informationsquelle für das Projektmanagement dient, ist derzeit primär auf die Nutzung an Desktop-Systemen ausgelegt und verfügt weder über eine responsive Darstellung noch über eine Unterstützung für Touchgesten.
Ziel dieser Arbeit ist die Erarbeitung einer Implementierungsstrategie, um Miori Boards an die Anforderungen mobiler Endgeräte anzupassen. Hierfür werden zunächst grundlegende Technologien des Responsive Webdesigns, wie das CSS Box-Model, Flexbox und Grid, sowie Konzepte der Touchgestensteuerung mittels JavaScript PointerEvents analysiert. Auf Basis des Frameworks Vue.js und der Bibliothek VueUse wird eine Strategie entwickelt, die einen „Mobile-First“-Ansatz verfolgt.
Die praktische Umsetzung erfolgt exemplarisch durch einen Prototypen in Miori Boards. Dabei wird die Übersichtseite durch dynamische Layout-Anpassungen responsive gestaltet und der Präsentationsmodus um eine Wischgeste (Swipe) zur Navigation erweitert. Die Evaluation anhand einer eigens erstellten Benchmarkseite bestätigt, dass die entwickelten Konzepte die Anforderungen an Performance, Usability und technische Umsetzbarkeit erfüllen. Die Ergebnisse bilden somit eine fundierte Grundlage für die vollständige mobile Optimierung der Webanwendung.

Inhaltsverzeichnis

1 Einleitung

Dieses Kapitel führt in die Grundthematik der Seminararbeit ein und bildet den Rahmen für die weiteren Kapitel.

1.1 Motivation

Durch den stetig wachsenden Trend zu New Work[1] ist es möglich, von überall aus zu arbeiten. Im Zuge der fortschreitenden Digitalisierung ist es wichtig, eine zentrale Quelle zu haben, welche den aktuellen Stand eines Projekts anzeigen kann. Dabei können besonders Webapplikationen helfen, da diese auf internetfähigen Geräten aufrufbar sind und der Nutzer nicht abhängig vom Nutzungsgerät ist. Durch das universale Aufrufen von Webapplikationen verschiedener Systeme, muss beachtet werden, wie die Informationen und Daten darzustellen sind, da sich die digitalen Endgeräte deutlich unterscheiden können und es keine allgemeine Lösung gibt zur Darstellung auf allen digitalen Endgeräten.

1.2 Problemstellung

Das Webtool Miori Boards ermöglicht es Projekte zu planen und als zentrale Informationsquelle zu dienen. Miori Boards ist dabei auf die Nutzung von Desktop-Systemen und Laptops ausgelegt. Durch die fehlende Responsivität der Webseite, ist die Nutzung auf mobilen Endgeräten wie Tablets und Smartphones nur eingeschränkt möglich. Zudem fehlt es an einer Unterstützung für mobile Endgeräte mit Touchscreens, die primär über Touchgesten bedient werden. Dadurch ist die Bedienung der Webseite auf mobilen Endgeräten erschwert und es können nicht alle Funktionalitäten genutzt werden.

1.3 Ziel der Arbeit

Vor diesem Hintergrund soll in dieser Arbeit eine Implementierungsstrategie erarbeitet werden, wie Miori Boards gerecht den Anforderungen mobiler Endgeräte angepasst werden kann. Dabei soll die Webseite so erweitert werden, dass sie auch auf Touchgesten reagiert und die Darstellung auf mehreren Bildschirmgrößen funktioniert.

1.4 Aufbau der Arbeit

Zu Beginn werden Miori Boards vorgestellt und grundlegende Technologien für responsives Webdesign sowie Touchgesten erläutert. Daraufhin werden Anforderungen an die Implementierungsstrategie definiert, die bei der Umsetzung berücksichtigt werden müssen. Anschließend wird die Implementierungsstrategie ausgearbeitet und als Strategie zur Umsetzung vorgestellt. In Kapitel 5 wird die Strategie prototypisch an Miori Boards umgesetzt und mithilfe einer Benchmarkapplikation evaluiert. Abschließend wird ein Fazit gezogen und ein Ausblick auf mögliche weitere Erweiterungen gegeben.

[1] New Work beschreibt die Idee von modernen und flexiblen Formen der Büroarbeit und der Arbeitsorganisation, darunter zählt auch die situative mobile Arbeit [6]

2 Grundlagen und Stand der Technik

Im diesen Kapitel wird das Tool Miori Boards vorgestellt, sowie grundlegende Techniken und Konzepte, die für die Entwicklung einer responive Webanwendung mit Touchgestensteuerung notwendig sind.

2.1 Miori Boards

Miori Boards ist eine Webanwendung, entwickelt vom Lehrstuhl Produktionssystematik des WZL[1] der RWTH Aachen. Mithilfe von individuell erstellbaren Boards, ermöglicht Miori Boards ein **„effizientes Team- und Projektmanagement“** [5].
Ein Board besteht dabei aus mehreren Kacheln, die Widgets enthalten, zur Darstellung von **„eigenen Inhalten, Medien und Datenvisualisierungen“** [5], siehe Abbildung 2.1. Es gibt dabei das Personen-, Todo-, Tool-, Umfrage-, Tabellen-, Media- und Chart-Widget. Die Widgets im Board können individuell angepasst und angeordnet werden.
An einem Board können mehrere Personen beteiligt sein, die entweder die Owner- oder Member-Rolle besitzen. Member können die Widgets des Boards benutzen, währenddessen der Owner zusätzlich Personen zum Board einladen[2] kann und Widgets hinzufügen sowie entfernen kann. Ebenfalls kann der Owner zudem auch weitere Owner ernennen.

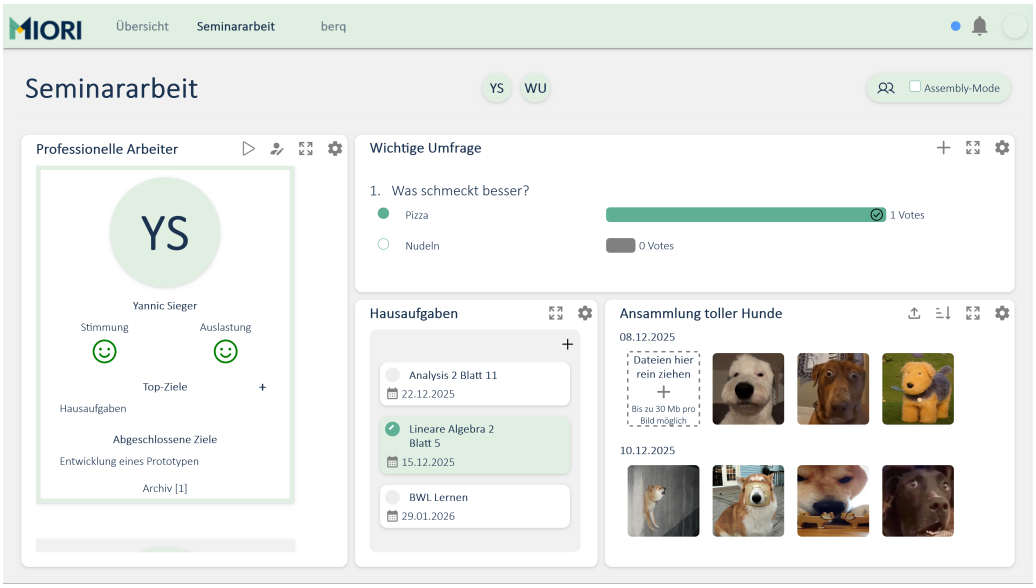


Abbildung 2.1: Miori Boards Board

2.2 Responsive (Web-) Design

Durch die Vielzahl an verschiedenen Bildschirmgrößen und Auflösungen moderner Endgeräte ist es notwendig, dass Webanwendungen sich flexibel an diese anpassen können, um den selben Inhalt darzustellen. Dazu gibt es die Idee von Responsive (Web-) Design, welches mehrere Technologien

[1] Werkzeugmaschinenlabor
[2] Dies geschieht über einen Einladungslink

und Ansätze kombiniert, um dies zu ermöglichen. Der Begriff selbst wurde von Ethan Marcotte im Jahr 2010 geprägt [7][25]. Im folgenden werden die wichtigsten Technologien und Prinzipien vorgestellt, die für das Responsive Design relevant sind.

2.2.1 CSS Box-Model

Die strukturelle Grundlage einer Webseite wird durch das HTML[3]-Dokument und dessen DOM[4] gebildet. Dabei fungiert ein einzelnes HTML-Element als Knotenpunkt im DOM-Baum. HTML-Elemente können weitere Elemente umschließen[5], wodurch eine hierarchische Struktur aus Eltern- und Kind-Elementen entsteht. Der DOM-Baum beginnt beim `<html>`-Wurzelelement, welches das gesamte Dokument repräsentiert [12].
HTML-Elemente sind nach dem Box-Model aufgebaut, welches aus vier Bereichen besteht, siehe Abbildung 2.2. Der Content beinhaltet den Inhalt des Elements, etwa in Form von weiteren Elementen oder reinen Texten sowie Grafiken. Die Border dient als visuelle Grenze des Elements in Form eines Rahmens. Das Padding definiert den Abstand zwischen dem Content und der Border, das Margin dagegen definiert den Außenabstand des Elements zu umliegenden Elementen. Die Größe des Elements wird durch die jeweiligen Größen der Seiten von Content, Padding und Border bestimmt [12].

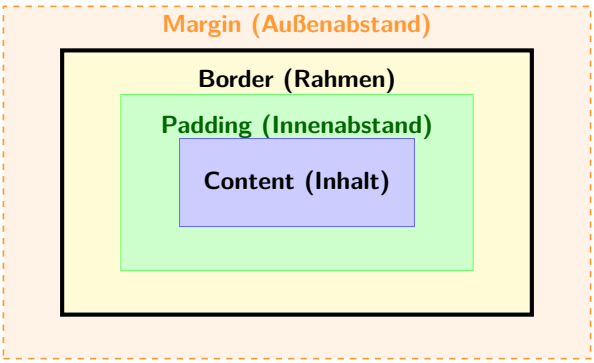


Abbildung 2.2: CSS Box Model

Im Kontext des Box-Models ist die CSS-Eigenschaft `box-sizing` relevant, da diese Eigenschaft bestimmt, wie zum Beispiel `width`, `height` sowie deren `min-` und `max-` Eigenschaften interpretiert werden. Bei dem Wert `box-sizing: content-box` werden die Eigenschaften direkt auf den Content angewendet, wodurch Padding und Border diesem Wert hinzuaddiert werden. Dagegen bezieht sich `box-sizing: border-box` auf das ganze Element und nicht nur auf den Content, wodurch der Wert die ganze Größe des Elements bestimmt und gegebenfalls den Content verkleinert, um in die Größe zu passen [8].
Um die Größe eines Elements zu bestimmen, gibt es verschiedene CSS-Einheiten, die genutzt werden können. Diese sind in der Tabelle 2.1 dargestellt.

[3] HyperText Markup Language
[4] Document Object Model
[5] Als eine Art Verschatelung

Tabelle 2.1: Relevante Einheiten für responsive Design [14][17]

Einheit	Beschreibung
`px`	feste CSS-Standardeinheit.
`em`	relative Einheit basierend auf der Schriftgröße des Elternelements.
`rem`	relative Einheit basierend auf der Schriftgröße des Root-Elements.
`fr`*	Flexible Einheit, die den verfügbaren Platz im Verhältnis zu anderen Elementen aufteilt.
`%`	relative Einheit, die basierend auf dem Elternelement die Größe bestimmt.

Eigentschaften, die mit * markiert sind, können nur in CSS Grids verwendet werden.

Neben den Einheiten gibt es noch die Funktion `calc()`, mit der mathematische Operationen durchgeführt werden können. Dadurch können verschiedene Einheiten kombiniert werden, um eine flexible Größe zu definieren [9]. Mit der Funktion `clamp()` kann eine Größe definiert werden, die sich flexibel anpasst, aber innerhalb eines bestimmten Bereichs bleibt. Dabei werden ein Minimalwert, ein bevorzugter Wert und ein Maximalwert angegeben [10].
Trotz der definierten Größe eines Elements kann es vorkommen, dass der Inhalt nicht in das Element passt. Dabei entsteht ein Überlauf beziehungweise ein Overflow. Das Verhalten des überlaufenden Teils kann mit der CSS-Eigenschaft `overflow` gesteuert werden [21]. Die verschiedenen Werte für die Eigenschaft `overflow` sind in der Tabelle 2.2 dargestellt.

Tabelle 2.2: Overflow-Eigenschaften im Responsive Design [21]

Wert	Darstellung	Besonderheit
`visible`	overflow bleibt sichtbar.	Ist der Initialwert.
`hidden`	overflow wird abgeschnitten.	Der abgeschnittene Inhalt ist nicht mehr zugänglich.
`scroll`	overflow wird scrollable mit Scrollbalken.	Es wird ein Scrollbalken für die x- als auch y-Achse angezeigt
`auto`	erst bei overflow wird es scrollable mit Scrollbalken.	Scrollbalken nehmen erst Platz im Layout ein, wenn diese sichtbar sind.

Bei der Nutzung von `overflow: auto` ist noch zu beachten, dass die Scrollbar eingefügt wird und es dadurch zu einer Layout-Verschiebung kommt. Dies kann mit der Eigenschaft `scrollbar-gutter: stable` verhindert werden, welche permanent Platz für die Scrollbar reserviert, unabhängig davon ob diese angezeigt wird oder nicht [27]. Mit der Erweiterung `overflow-x` und `overflow-y` kann das Overflow-Verhalten für die jeweilige Achse separat definiert werden [21].

2.2.2 Media Queries

Damit gewisse CSS[6]-Eigenschaften nur unter bestimmten Bedingungen angewendet werden, gibt es Media Queries. Media Queries ermöglichen es, dass Eigenschaften nur aktiviert werden, wenn beispielsweise die Viewport-Breite oder -Höhe einen bestimmten Wert überschreitet [31]. Diese Informationen werden vom User-Agent bereitgestellt, welcher eine digitale Identität anhand der verwendeten Hard- und Software darstellt. Media Queries gleichen diese Informationen ab und bestimmen so, ob die Bedingungen erfüllt sind [30]. Anhand dieser Bedingungen und logischen Operatoren wie beispielsweise `and`, `not` oder `only` können passend zur Nutzerumgebung Stylings aktiviert oder deaktiviert werden [31].

2.2.3 Viewport-Meta-Tag

Der Viewport ist der sichtbare Bereich einer Webseite auf dem Bildschirm eines Endgerätes. So ist der Viewport bei einem Desktop-Browser in der Regel das Browserfenster, während es bei mobilen Geräten der Bildschirmbereich ist, der die Webseite anzeigt. Der Viewport-Meta-Tag ist ein HTML-Tag im Header, der dem mobilen Browser mitteilt, dass dieser die Website auf 100% der

[6] Cascading Style Sheets

Bildschirmbreite setzen soll. Dies ist notwendig, da viele mobile Browser standardmäßig die Website auf eine Breite von `980px` gerendert haben, weil viele Websites nicht für mobile Geräte optimiert waren [25]. Ohne den Viewport-Meta-Tag würden Media-Queries nicht funktionieren, da die Bildschirmbreite immer `980px` wäre.

2.2.4 Responsive Layout-Technologien

Für die Anordnung von HTML-Elementen auf einer Webseite gibt es Layouts, die bestimmen, wie die Elemente positioniert werden. Responsive Layout-Technologien ermöglichen es, dass diese Anordnung flexibel ist und sich an verschiedene Bildschirmgrößen anpasst [25]. Die zwei wichtigsten Layout-Technologien im Kontext von Responsive Design sind **Flexbox** und **Grid**.

Allgemein

Beide Layout-Technologien sind nach einem Achsenmodell aufgebaut. Flexbox ist für eindimensionale Layouts gedacht, während Grid für zweidimensionale Layouts verwendet wird [16][13]. So besitzt Flexbox eine Hauptachse und eine Querachse, während ein Grid mit Zeilen und Spalten arbeitet. Beide Layout-Technologien besitzen CSS-Eigenschaften, die es ermöglichen, die Anordnung der Kindelemente entlang der Achsen zu steuern. Diese Eigenschaften sind in der Tabelle 2.3 dargestellt.

Tabelle 2.3: Anordnungen für responsive Layouts [11]

	`justify-*` (Hauptachse / Zeilen)	**`align-*` (Querachse / Spalten)**
`content`	Verteilt den freien Raum zwischen Kindelementen.*	Verteilt den freien Raum zwischen Kindelementen
`items`	Positioniert alle Kindelemente entlang der Hauptachse.	Richtet alle Kindelemente entlang der Querachse aus.*
`self`	Positioniert ein einzelnes Kindelement entlang der Hauptachse.	Richtet ein einzelnes Kindelement entlang der Querachse aus.*

Im Grid-Layout können alle Eigenschaften verwendet werden. Die Eigenschaften, die mit * markiert sind, funktionieren auch im Flexbox-Layout.

Einige CSS-Eigenschaften aus der Tabelle 2.3 können aufgrund der eindimensionalen Struktur von Flexbox nicht darauf angewendet werden. Eine weitere Eigenschaft, die beide Layouts nutzen, ist `gap`, welche einen Abstand zwischen den Kindelementen definiert [18].

Flexbox

Damit ein HTML-Element als Flex-Container fungiert, muss die CSS-Eigenschaft `display: flex` gesetzt sein. Dadurch werden alle direkten Kindelemente zu Flex-Elementen, die entlang der Haupt-

achse angeordnet sind [16]. Mit der Eigenschaft `flex-direction` kann die Richtung der Hauptachse angepasst werden, sodass die Flex-Elemente entweder horizontal oder vertikal sowie umgedreht[7] angeordnet werden [16]. Rechtwinklig zur Hauptachse befindet sich die Querachse, entlang derer die Flex-Elemente ebenfalls ausgerichtet werden können.
Durch den flexiblen Aufbau von Flexbox können die Flex-Elemente ihre Größe anpassen, um den verfügbaren Platz optimal auszunutzen [16]. Um dies zu steuern, gibt es Eigenschaften, die den Flex-Elementen übergeben werden können, siehe Tabelle 2.4.

Tabelle 2.4: Flex-Skalierungs-Eigenschaften [15]

Eigenschaft	Beschreibung
`flex-shrink`	Bestimmt einen Faktor, um wie viel ein Flex-Element schrumpft, wenn nicht genügend Platz vorhanden ist.
`flex-grow`	Bestimmt einen Faktor, um wie viel ein Flex-Element wächst, um den verfügbaren Platz auszufüllen.
`flex-basis`	Legt die Ausgangsgröße eines Flex-Elements fest, bevor der verfügbare Platz verteilt wird.

`flex-basis` überschreibt bei der Flexberechnung `width` und `height`, wenn der Initialwert `auto` überschrieben wird.

Die Eigenschaften `flex-shrink` und `flex-grow` können eingeschränkt werden, indem die Größe des Flex-Elementes mit `min-` und `max-` limitiert wird. Dadurch wird ein Größenbereich bestimmt, in dem das Flex-Element wächst oder schrumpft [16]. Die Eigenschaft `flex-basis` überschreibt bei der Flexberechnung `width` und `height` des Flex-Elementes, wenn der Initialwert `auto` nicht verändert wird.
Durch die einzelne Hauptachse ist es möglich, dass diese zu voll wird und die Flex-Elemente nicht mehr in den Container passen. In diesem Fall kann mit der Eigenschaft `flex-wrap: wrap` definiert werden, dass die Flex-Elemente in die nächste Zeile umbrechen [16]. Dadurch entsteht eine weitere **Flex Line**, die unter der ersten liegt, wo die Hauptachse weitergeführt wird.

Grid

Neben Flexboxen gibt es auch Grids als responsive Layout-Technologie. Um ein Element als Grid-Container zu definieren, muss die Eigenschaft `display: grid` gesetzt werden [13]. Dadurch entsteht ein Raster mit einer Spalten- und Zeilenachse. Initial besitzt das Grid nur eine Spalte und erstellt implizit so viele Zeilen wie es Kindelemente gibt. Zur Bestimmung von Spalten und Zeilen

[7] dabei wird an die Richtung noch `-reverse` drangehangen

gibt es folgende Eigenschaften in der Tabelle 2.5.

Tabelle 2.5: Grideigenschaften zur Deklarierung

Eigenschaft	Beschreibung
`grid-template-columns`	Definiert die Anzahl und Größe der Spalten in einem Grid-Layout.
`grid-template-rows`	Definiert die Anzahl und Größe der Zeilen in einem Grid-Layout.
`grid-column`	Legt fest, wie viele Spalten ein Grid-Element einnimmt und wo es beginnt.
`grid-row`	Legt fest, wie viele Zeilen ein Grid-Element einnimmt und wo es beginnt.
`grid-template-areas`	Definiert benannte Bereiche innerhalb des Grids, die zur einfacheren Platzierung von Elementen verwendet werden können.
`grid-area`	Weist einem Grid-Element einen benannten Bereich zu, der in `grid-template-areas` definiert ist.

Vordefinierte Spalten und Zeilen werden als explizite Spalten und Zeilen bezeichnet, während automatisch erstellte Spalten und Zeilen als implizite Spalten und Zeilen bezeichnet werden [13]. Falls es mehr Kindelemente gibt als explizite Zeilen oder Spalten, werden automatisch implizite Zeilen oder Spalten erstellt, um die Kindelemente aufzunehmen. Die Eigenschaften `grid-auto-rows` und `grid-auto-columns` definieren die Größe der impliziten Zeilen beziehungsweise Spalten [13]. Um eine variable Anzahl an Spalten oder Zeilen mit fester Größe zu erstellen, kann die Funktion `repeat()` genutzt werden, mit dem Wert `auto-fit`, wodurch so viele Spalten oder Zeilen entstehen, die in das Grid passen [13]. Die Funktion `repeat()` kann auch mit einer festen Anzahl genutzt werden, um die Lesbarkeit zu verbessern, wenn viele Spalten oder Zeilen mit derselben Größe definiert werden sollen [13].
Zur Bestimmung von Spalten- und Zeilengrößen können dieselben Einheiten benutzt werden, wie die aus der Tabelle 2.1. Im Grid-Kontext wird die Einheit `fr` benutzt, welche einen Bruchteil des verfügbaren Platzes im Grid darstellt, wodurch flexibel zur Containergröße die Zellengröße bestimmt werden kann [17]. Neben den Einheiten, gibt es noch die `minmax()` Funktion, mit der

ein Größenbereich definiert werden kann [19].
Kindelemente eines Grids können mit den Eigenschaften `grid-column` und `grid-row` auch definieren wie viele Zellen eingenommen werden mit dem Schlüsselwort `span`. Bei dem Wert `-1` wird die ganze Spalte oder Zeile eingenommen [13].

2.2.5 Responsive Bilder und Medien

Responsive Bilder und Medien umfasst die Idee, dass Bilder und Medien die verfügbare horizontale Breite des Containers beziehungsweise des Bildschirms einnehmen. Das Medium sollte dabei sein Verhältnis[8] beibehalten, um Verzerrungen zu vermeiden. Dies wird mit der CSS-Eigenschaft `max-width: 100%` sowie `height: auto` erreicht [25]. Bei responsiven Bildern kann noch zusätzlich noch das Ursprungsbild in verschiedenen Auflösungen bereitgestellt werden, um passend zum Nutzungsgerät die Bandbreite zu sparen sowie dem Layout gerecht zu werden [26].

2.2.6 Responsive Typografie

Responsive Typografie beschäftigt sich mit der Anpassung von Schriftgrößen. Dabei sind die CSS-Einheiten `em` und `rem` relevant.
Die Einheit `em` bezieht sich auf die Schriftgröße des Elternelements, während `rem` sich auf die Schriftgröße des Root-Elements (HTML-Tag) bezieht. Dadurch kann im Root-Element die Schriftgröße angepasst werden und alle `rem`-Werte passen sich daran an [14]. Die Root-Element Schriftgröße wird standardmäßig auf 16 Pixel gesetzt, kann aber mit der CSS-Eigenschaft `font-size` angepasst werden. Der Browser skaliert die Schriftgröße auch, wenn der Nutzer die Zoom-Funktion verwendet [25]. Wenn Texte zu lang werden für ihr Element, gibt es mehrere Möglichkeiten damit umzugehen. So kann der Text umgebrochen werden mit `overflow-wrap: break-word`, wodurch der Text in die nächste Zeile springt, wenn kein Platz mehr da ist. Falls es nicht gewünscht ist den ganzen Text zu zeigen, kann der Text mit `text-overflow` abgeschnitten werden.

2.3 Touchgestensteuerung

Durch die Einführung von grafischen Nutzeroberflächen in Betriebssystemen etablierte sich die Benutzung von Pointing Devices. Diese steuern einen Pointer auf der grafischen Nutzeroberfläche, um Interaktionen auszuführen, wie das Klicken von Buttons oder das Bewegen von Objekten. [2] Lange Zeit waren Computermäuse und Trackpads die gängigsten Pointing Devices. Mit dem Aufkommen von Touchscreens wurde auch der eigene Finger zu einem Pointing Device. Dabei wird die Position, Anzahl, Geschwindigkeit und Bewegung der Finger, Berührungsdauer und Anzahl der Berührungen ausgewertet, um verschiedene Aktionen auszuführen. Wenn mehrere vorbestimmte Bedingungen erfüllt sind, wird von einer Touchgeste gesprochen. Diese können sich ganz nach Betriebssystem und Anwendung unterscheiden.

[8] aspect-ratio

2.3.1 JavaScript EventListener

Damit Touchgesten in einer Webanwendung implementiert werden können, müssen diese mit JavaScript umgesetzt werden, da JavaScript die Funktionalität der Webseite bestimmt. In JavaScript werden verschiedene Aktionen oder Zustandsänderungen mit einem `Event` beschrieben. Da es verschiedene Arten von Events gibt, gibt es spezialisierte Events, die von anderen Events abgeleitet sind. Im Beispiel von der Abbildung 2.3 ist die Vererbungshierarchie der DOM-Events dargestellt, die für Touchgesten relevant sind.

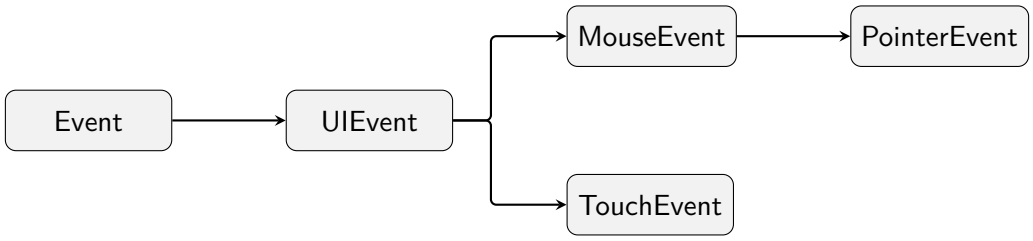


Abbildung 2.3: Horizontale Vererbungshierarchie der DOM-Events, speziell für Touchgesten [29][22]

Basierend auf dem `UIEvent`[9] gibt es das `MouseEvent` und das `TouchEvent`, die für Maus- beziehungsweise Touchereignisse zuständig sind. Es existiert noch basierend auf dem `MouseEvent` das `PointerEvent`, welches sowohl Maus- als auch Touchereignisse unterstützt und somit eine einheitliche Schnittstelle für beide Eingabemethoden bietet [22]. `PointerEvent` besitzt erweiterte Informationen wie `pointerType` und reagiert auf Stifteingaben.
Für `PointerEvents` gibt es verschiedene Aktionen, die als EventListener genutzt werden können, um auf Touchgesten zu reagieren. Folgende Liste 2.4 zeigt alle relevanten `PointerEvents` auf, die existieren.

Abbildung 2.4: Liste der Pointer Events in Web-APIs [22]

1. `pointerdown`
2. `pointerleave`
3. `gotpointercapture`
4. `pointercancel`
5. `pointermove`
6. `pointerup`
7. `pointerenter`
8. `lostpointercapture`
9. `pointerover`
10. `pointerout`

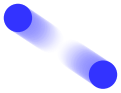
2.3.2 Arten von Touchgesten

Dadurch dass es kein einheitliches System für Touchgesten gibt, existieren viele verschiedene Arten von Touchgesten. Aus diesem Grund ist es wichtig die gängigsten und wichtigsten Touchgesten zu bestimmen. Damit ist garantiert, dass die meisten Nutzer die Touchgesten kennen und verwenden können.

[9] User Interface Event

Die Designdokumentationen von Microsoft [33], Apple [1] und Google [3] geben verschiedene Touchgesten an, die ein Entwickler für Android, Windows sowie MacOS beachten soll. Die Schnittmenge aller Touchgesten der Designdokumentationen ist in der Tabelle 2.6 aufgeführt. Die Dokumentationen von Microsoft, Apple und Google sind repräsentativ, da im Jahr 2025 71,88% der Desktopsysteme Windows und 8,7% MacOS verwendeten [36]. Als mobile Betriebssysteme wurden 2025 72,03% Android und 27,59% iOS genutzt [37].

Tabelle 2.6: Übersicht der unterstützten Touch-Gesten

Name	Touchgeste	Beschreibung	Use-Case
Tap		kurze Berührung	Auswählen oder aktivieren eines Elements
Double-Tap		zwei kurze Taps mit geringen Zeitabstand	Alternative Aktion zum Tap oder Zoom-In/-Out
Hold		lange Berührung	Alternative Aktion zum Tap
Scroll		Hold mit einer Wischbewegung	Bewegt den Inhalt
Swipe/Flick		schneller ruckartiger Scroll	Bewegt ein Element weg oder ermöglicht alternative Aktionen
Drag (and Drop)		Hold mit einer Navigationsbewegung	Bewegt ein Element an eine andere Position
Zoom		Wegziehen von zwei Berührungspunkten	Vergrößert den Inhalt
Pinch		Zusammenziehen von zwei Berührungspunkten	Verkleinert den Inhalt

3 Anforderungen

Damit eine ansprechende Implementierungsstrategie entwickelt werden kann, werden Anforderungen benötigt, die wichtige Aspekte und Limitationen hervorheben.

3.1 Allgemeine Anforderungen

Bevor die spezifischen Anforderungen definiert werden, muss zuvor genannt werden, wie die Qualitätsanforderungen und Rahmenbedingungen für die Implementierung aussehen sollen.
Die Implementierung sollte mit geringem Aufwand realisierbar sein. Auch die Wartbarkeit des Codes sollte gewährleistet sein, da diese Implementierung eine optionale Erweiterung von Miori Boards ist und der Quellcode nicht unnötig komplex werden darf. Wichtig ist auch, dass es erweiterbar ist, falls weitere Touchgesten oder neue Ansichten hinzukommen sollten. Hinsichtlich der Performance sollte diese konsistent sein, damit die Benutzung der Touchgestensteuerung und die verschiedenen Ansichten flüssig und ohne Verzögerungen funktionieren. Ein wichtiger Aspekt bei der Implementierung ist die Unterstützung verschiedener Rendering Engines. Eine Rendering Engine ist die Software in einem Browser, die das HTML und CSS einer Webseite visuell darstellt und Schnittstellen bereitstellt, die dann mit JavaScript angesprochen werden können. Rendering Engines decken die Grundfunktionalitäten von HTML und CSS ab, jedoch gibt es Features von CSS oder HTML die nicht von allen Engines unterstützt werden. Relevant dabei sind die Rendering Engines **Blink (Google)**, **Gecko (Mozilla)** und **WebKit (Apple)**, die die gängigsten Engines sind und unterstützt werden sollten bei der Implementierung [24]. In unserem Kontext wird die Rendering Engine Trident (Microsoft) nicht mehr berücksichtigt, da diese von Microsoft nicht mehr weiterentwickelt wird und stattdessen Microsoft Edge die Blink Engine nutzt. Der letzte wichtige Punkt der allgemeinen Anforderungen ist die Testbarkeit der Implementierung. Dadurch, dass nun mehrere Eingabegeräte sowie digitale Endgeräte zur Verwendung unterstützt werden sollen, ist es schwer bei jedem neuen Feature alle Möglichkeiten zu prüfen, ob es einwandfrei funktioniert. Die Tabelle 3.1 fasst die allgemeinen Anforderungen nochmals übersichtlich zusammen.

Tabelle 3.1: Allgemeine Anforderung an der Implementierung

Qualitätsanforderungen	Rahmenbedingungen
Erweiterung leicht möglich	Implementierungsaufwand gering
leichte Wartbarkeit möglich	Support verschiedener Rendering Engines
Testbarkeit der Implementierung	konsistente Performance

3.2 Anforderungen an das Responsive Design

Responsive Design ist eine grundlegende Voraussetzung, um Webseiten wie Miori Boards geräteübergreifend nutzbar zu machen. Miori Boards unterstützt derzeit nur Laptops und Desktop-Systeme. Bei der Betrachtung der Internetnutzung zeigt sich ein anderes Bild. Im zweiten Quartal 2025 nutzten mehr als 93% der Internetnutzer das Internet mit einem Smartphone. Dagegen nutzten nur etwa 59% der Internetnutzer Desktop-Systeme und Laptops, um das Internet zu besuchen [34]. Dies zeigt, dass die Mehrheit der Nutzer Webseiten mit dem Handy besucht. Damit eine Webseite auf verschiedenen Geräten ansprechend nutzbar ist, muss sie sich an die verschiedenen Bildschirmgrößen kontinuierlich anpassen können, weshalb dynamisches Layout notwendig ist. So ist es wichtig, dass es sich reaktiv verhält und auch jedem Element den Platz gibt, den es benötigt. Das Layout sollte auch den Platz optimal nutzen, indem es Elemente umsortiert oder in andere Darstellungen wechselt. Auch Elemente sollten sich anhand ihres vorhandenen Platzes anpassen. So sollten bei Platzmangel Unterelemente versteckt werden oder in eine alternative Darstellung wechseln. Text und Interaktionselemente sollten trotz begrenztem Platz eine Mindestgröße besitzen. Damit ist der Text angenehm zu lesen und die Interaktionselemente sind, je nach Eingabegerät, weiterhin nutzbar. Jedoch sollten Text und die Interaktionselemente nicht zu groß sein, da sonst der Text schwer lesbar wird oder die Interaktionselemente eine falsche Wichtigkeit bekommen sowie ungewollt ausgelöst werden können. Für die Implementierung des Responsive Designs ergeben sich somit die in der Tabelle 3.2 dargestellten Anforderungen.

Tabelle 3.2: Anforderungen an die Implementierung des Responsive Design

Funktionale Anforderungen	**Nicht-funktionale Anforderungen**
Dynamische Layout-Anpassung	Kontinuierliche Anpassung
Dynamische Element-Anpassung	Hohe Lesbarkeit und Bedienbarkeit
Funktionalität beibehalten bei Platzmangel	Optimale Platznutzung der Elemente

3.3 Anforderungen an die Touchgestensteuerung

Damit Miori Boards auch auf digitalen Endgeräten funktioniert, die keine Maus und Tastatur besitzen, ist eine Touchgestensteuerung notwendig, da diese zu einer der häufigsten Bedienungsarten zählt, mit denen das Internet bedient wird [34]. Bei der Implementierung der Touchgestensteuerung ist es wichtig Gesten zu verwenden, die die meisten Nutzer kennen, damit die Nutzung für den Nutzer intuitiv bleibt und nicht die Nutzererfahrung negativ beeinflusst. Ein weiterer wichtiger Punkt ist, dass die verschiedenen Touchgesten oder ihre Events nicht kollidieren, da sonst ein falsches Verhalten ausgelöst werden kann. So könnte zum Beispiel beim Wischen eines Elements auch ein Scroll-Event ausgelöst werden, wodurch das Wisch-Event abbrechen könnte. Es ist wichtig, dass ähnliche Touchgesten nicht gleichzeitig ein Event auslösen. Für die Nutzung einer Touchgestensteuerung ist es hilfreich mit Indikatoren zu visualisieren, dass etwas benutzbar ist. So können subtile Animationen, Elemente mit Schatten oder auch Symbole helfen, dem Nutzer zu zeigen, dass etwas bedienbar ist [4]. Zur Steigerung der Nutzererfahrung hilft es, wenn die Elemente, die ein Touch-Event haben, dynamisch auf die Touchgeste reagieren und nicht während der Aktion die Kontrolle entziehen, wegen einer Animation oder einem Breakpoint. Dadurch können Nutzer noch die Aktion abbrechen, falls es die nicht gewünschte Aktion war. Ebenfalls fühlt es sich für den Nutzer natürlicher an, wenn das Element dynamisch und direkt auf die Berührung reagiert [4]. Für die Implementierung der Touchgestensteuerung ergeben sich somit die in der Tabelle 3.3 dargestellten Anforderungen.

Tabelle 3.3: Anforderungen an die Implementierung der Touchgestensteuerung

Funktionale Anforderungen	**Nicht-funktionale Anforderungen**
Touchgestensteuerung bereitstellen	Intuitive Nutzung / Nutzererfahrung
Verwendung bekannter Gesten	Visuelle Indikatoren für Bedienbarkeit
Vermeidung von Konflikten zwischen Gesten	Natürliche, flüssige Reaktion auf Touchgesten

4 Implementierungsstrategie

Bevor die technische Umsetzung an Miori Boards stattfinden kann, ist es wichtig eine Strategie zu entwickeln, wie die Implementierung ablaufen soll. Dabei kann die Verwendung von Frameworks und Bibliotheken helfen, um die Implementierung zu erleichtern und zu beschleunigen. Zudem müssen anforderungsgerechte Lösungen für die Implementierung von Touchgesten und Responsive Design gefunden werden. Aus diesen Ergebnissen wird eine allgemeine Implementierungsstrategie abgeleitet, die bei der Umsetzung genutzt werden kann.

4.1 Verwendete Frameworks und Bibliotheken

Um eine einfache Implementierung zu ermöglichen, sind bereits existierende Lösungen des Problems in Form von Frameworks und Bibliotheken sehr hilfreich. Im folgenden Unterkapitel wird erläutert wieso **Vue.js**, die Utilities-Bibliothek **VueUse** und die WZL eigene Bibliothek **WZL Essentials** verwendet werden und wie sie die Implementierung der Touchgesten sowie des Responsive Designs unterstützen.

4.1.1 Vue.js

Vue.js[1] ist ein communitygestütztes, progressives JavaScript Webframework, ursprünglich von Evan You initiiert [40]. Mit Vue ist es möglich, SPA[2]s zu entwickeln, also Webseiten, die aus nur einem HTML-Dokument bestehen und mithilfe von JavaScript und CSS dynamisch angepasst werden. Dabei verwendet Vue ein Reaktivitätssystem mit `reactive-proxy`, welches Veränderungen erkennt und diese dynamisch überall anpasst. Dadurch werden Zustandsveränderungen automatisch erkannt und verarbeitet. Vue arbeitet mit SFC[3]s, welche aus HTML-, JS- und CSS-Code im jeweiligen `<template>`-, `<script>`- und `<style>`Tag der Datei bestehen. Bei den Komponenten können bereits definierte Komponenten wiederbenutzt werden. Dadurch lässt sich viel Code wiederverwenden und die Architektur des Projekts wird modularer. Zu der Modularität trägt auch die Idee der Composables bei. Composables sind Funktionen, die wiederverwendbaren Code in Form des reaktiven Vue-Systems bereitstellen. Dadurch lassen sich Funktionalitäten in einzelne Composables auslagern und in verschiedenen Komponenten wiederverwenden. Eine weitere Technologie von Vue sind die Directives. Directives sind spezielle Funktionen, die an ein HTML-Element gebunden werden können, um das Verhalten des Elementes zu verändern oder auf bestimmte Ereignisse zu reagieren. So gibt es beispielsweise die Directive `v-if`, die ein Element nur rendert, wenn eine bestimmte Bedingung erfüllt ist. Ebenso gibt es die Directive `v-on`, die es ermöglicht auf Events zu reagieren, wie beispielsweise einem Klick-Event. Directives sind erkennbar an dem Präfix `v-*`.
Für das Implementierungsvorhaben wird Vue benutzt, da es bereits das verwendete Frontend Framework von Miori Boards sowie aller anderen Miori Produkte ist. Darüber hinaus eignet sich Vue als Implementierungsgrundlage, da Komponenten mit einer Touchgestehe versehen werden können.

[1] Gesprochen wie das englische Wort **view**
[2] Single Page Application
[3] Single File Component

Dadurch haben alle anderen Komponenten, die von dieser Komponente Gebrauch machen, auch das neue Feature. Dies gilt auch für Responsive Design Entscheidungen, die das Element beziehungsweise die Komponente betreffen.

4.1.2 VueUse Core, Gesture und Motion

VueUse ist eine umfangreiche Sammlung von Utility-Funktionen für das Framework Vue.js. Die Bibliothek stellt viele nützliche Composables bereit, die häufige Anwendungsfälle abdecken und dadurch den Entwicklungsaufwand reduzieren. Ein großer Teil der Bibliothek beinhaltet EventListener, die das Event kapseln und dessen Werte reaktiv bereitgestellt werden. So gibt es beispielsweise die Composable `usePointer`, welche ein Objekt zurückgibt, das die wichtigen Informationen für ein Pointer-Event beinhaltet, wie die Position des Pointers, den Typ des Pointers und ob der Pointer gedrückt ist oder nicht. Dabei liefert `usePointer` immer den aktuellen Zustand des Pointers, wodurch reaktiv darauf reagiert werden kann. Die Bibliothek VueUse kann durch Add-ons modular erweitert werden, um noch mehr Utilities bereizustellen. Interessant für das Implementierungsvorhaben sind die Plugins **Gestures** und **Motion**.
VueUse Gestures bietet Composables und Directives an, die generische Touchgesten implementieren, wie beispielsweise `useSwipe`, `usePinch` oder `v-drag`. Da diese nicht nativ als Events in JavaScript existieren, reduziert die Nutzung dieser Composables und Directives viel Programmieraufwand, da die Gesten bereits abstrahiert vorliegen und nur noch in der Komponente verwendet werden müssen.
VueUse Motion bietet auch Composables an, die Animationen und Bewegungen von Elementen vereinfachen. So können Animationen erstellt werden, die auf bestimmte Zustandsänderungen reagieren. Dadurch lassen sich visuelle Indikatoren für Touchgesten einfach umsetzen, um dem Nutzer ein Feedback zu geben, dass die Geste erkannt wurde und ausgeführt wird.
Für die Implementierung soll VueUse sowie seine Erweiterungen verwendet werden, da eine Vielzahl von nützlichen Composables und Directives bereitgestellt wird, die den Programmieraufwand deutlich verringern. Dadurch ist die Implementierung von Touchgesten im Vue.js-Kontext effizient möglich.

4.1.3 WZL Essentials

Die WZL Essentials sind eine vom Lehrstuhl Produktionssystematik eigen entwickelte Bibliothek zur Bereitstellung von essenziellen Komponenten. Dadurch kann ein einheitliches Design und Verhalten der Komponenten gewährleistet werden. Die Bibliothek ist geschrieben in Vue.js. Zu den Grundkomponenten gehören beispielsweise Buttons, Dropdowns, Tooltips, Navigationselemente oder auch Boxen. Die vom WZL entwickelten Tools nutzen zwei verschiedene Style-Themes, die in den Essentials eingebunden sind: Miori und WZL. Die Essentials sind in den meisten Webanwendungen des Lehrstuhls Produktionssystematik integriert, dazu gehört auch Miori Boards.

4.2 Responsive Design

Damit die Implementierung des Responsive Design erfolgreich ist, müssen die Anforderungen aus Kapitel 3.2 erfüllt werden. Dazu zählt primär die Gestaltung der einzelnen Elemente sowie das Layout der gesamten Webseite als auch des einzelnen Elementes.

4.2.1 Element Gestaltung

Allgemein

Bei der Gestaltung eines Elementes ist es wichtig mit dem Boxmodel aus 2.2.1 zu arbeiten, da dies die Grundlage für jedes HTML-Element bildet. Um jedes Element einheitlich und alleinstehend für sich zu sein, sollte die Eigenschaft `box-sizing: border-box` gesetzt werden. Dadurch wird sichergestellt, dass die Größe des Elementes immer gleich bleibt, unabhängig von dem Inhalt des Elementes. Dadurch kommt es nicht zu unerwarteten Layout-Verschiebungen, wenn der Inhalt des Elementes größer oder kleiner wird. Bei `content-box` wiederum kann dies passieren, da das Element durch den Inhalt wächst und somit andere Elemente verschiebt oder überlappt.

Overflow

Die Nutzung von `overflow` kann helfen, um unerwartete Layout-Verschiebungen zu verhindern, wenn der Inhalt größer wird als das Element. Es ist ebenfalls auch gut geeignet, um sehr große Inhalte anzeigen zu können, welche nicht unbedingt vollständig sichtbar sein müssen. So können beispielsweise lange Tabellen, Texte, Listen oder Ansammlungen von Elementen gleicher Art angezeigt werden. Damit bleiben alle Informationen erhalten, sind aber nicht direkt sichbar. Dadurch ist der Sichtbereich des Elementes limitiert, jedoch kann der ganze Inhalt praktisch unendlich groß sein. Trotz dieser Freiheit sollte bevorzugt nur die Höhe des Inhalts größer sein als das Sichtbereich. Diese Limitierung ist notwendig, da das vertikale Scrollen für den Nutzer einfacher und intuitiver ist als das horizontale Scrollen, bedingt durch das Scrollen mit dem Mausrad oder das Wischen mit dem Finger. Ein horizontaler Überlauf kann jedoch genutzt werden, um Elemente anzuzeigen, die in beiden Dimensionen größer sind als das Sichtbereich und eine andere Darstellung schwer möglich ist, wie zum Beispiel bei Tabellen.
Die Nutzung von `overflow: auto` in Kombination mit `scrollbar-gutter: stable` ist eine gute Möglichkeit Elemente mit viel Inhalt anzuzeigen, ohne das Layout zu verändern. Besonders bei Elementen mit variabler Anzahl von Inhalten ist dies sehr hilfreich, da unabhängig von der Anzahl das Layout stabil bleibt, was sehr gewünscht ist bei einer responsive Webseite.

Größen

Wichtig bei der Gestaltung eines Elementes ist auch die Elementgröße selbst. Diese bestimmt wie das sichtbare Fenster aussieht, in dem der Inhalt oder `overflow`-Inhalt zu sehen ist. Dabei sind die CSS-Eigenschaften `width`, `height` sowie deren `min-` und `max-` Varianten relevant. Für die Bestimmung dieser Größen eignen sich die Einheiten `%`, `rem` und `px` sehr gut. Mit der Einheit `%` lassen sich gut Elemente gestalten, die sich an die Größe des Elternelementes anpassen. Dadurch

kann das Element flexibel auf verschiedene Bildschirmgrößen reagieren. Die Einheit `rem` eignet sich gut, um Elemente mit der Größe des Textes wachsen zu lassen. So wächst beispielsweise ein Button mit seinem Textinhalt, wenn im Browser gezoomt wird. Dadurch wird verhindert, dass der Text über den Button hinausläuft oder zu klein für den Button ist. Die Einheit `px` eignet sich gut, um eine feste Größe für ein Element zu definieren, welches unabhängig von der Bildschirmgröße oder dem Textinhalt ist. So können feste Abstände oder Größen definiert werden, die immer gleich bleiben sollen.
Damit Elemente auch auf den kleinsten gängigsten Bildschirmen nutzbar bleiben, sollte eine maximale Breite definiert werden, wo das Element noch nutzbar ist. Hierbei reicht es nur eine Breite zu definieren, da bei Elementen mit variabler Höhe ein Overflow mit Scroll genutzt werden kann, um den Inhalt anzuzeigen. Der Richtwert für die maximale Darstellungsbreite eines Elementes sollte bei `360px` liegen, da dies die kleinste gängigste Bildschirmbreite bei Smartphones ist [35]. Dadurch wird sichergestellt, dass das Element auch auf kleinen Bildschirmen gut nutzbar bleibt. Trotz diesem Richtwert sollten Elemente auch für größere Breiten designt werden, damit diese den Platz auf größeren Bildschirmen auch optimal nutzen können. Um dies zu erreichen, können die vorgestellten Layout-Technologien aus Kapitel 4.2.2 verwendet werden oder es werden mehrere Versionen des Elementes erstellt, die sich anhand des vorhandenen Platzes anpassen und gegebenfalls austauschen mithilfe von Media Queries und JavaScript.
Zu der Gestaltung eines Elementes gehört es auch wie der Text aussieht. So sollten Texte immer mindestens eine `em` Größe haben, bestenfalls eine `rem` Größe, da so alle Texte eine abhängige Größen zum Root-Element besitzen und gegebenfalls zentral dort angesteuert werden können, falls sich die Bildschirmgröße ändert oder ein Zoom verwendet wird. Wenn der Text zu lang sein sollte, kann es helfen diesen umbrechen zu lassen mit `word-wrap: break-word` und `overflow-wrap: break-word` oder der Text wird abgeschnitten mit `text-overflow`, falls der Text in nur einer Zeile sein darf. Da kann dann mithilfe eines Tooltips der ganze Text angezeigt werden.
Genauso sollten Bilder und Medien ihr Größenverhältnis beibehalten, aber niemals überlaufen, damit das Medium vollkommen anschaubar ist und keine Verzerrungen besitzt. Deshalb sollten Bilder limitiert werden mit `max-width: 100%`, damit diese nicht über den Bildschirm- oder Elementrand hinübergehen. Dazu sollten Bilder die Eigenschaft `height: auto` haben, da so das Bild in seinem Seitenverhältnis bleibt. Es ist ebenfalls möglich `max-height: 100%` zu nehmen, jedoch ist dies nicht empfehlenswert, da es dann eher zu einem horizontalen Scroll kommen kann, welches entgegen der Nutzerfreundlichkeit spricht.
Bei der Gestaltung von Icons, Buttons oder anderen interaktiven Elementen ist es wichtig, dass diese eine Mindestgröße besitzen, damit sie auch auf kleinen Bildschirmen gut bedienbar sind. Dabei sollte eine Mindestgröße von `24px` x `24px` genommen werden, da dies die empfohlene Mindestgröße für Touch-Ziele ist laut WCAG[4] [39]. Empfohlen wird aber eher die Größe `44px` x `44px` [38]. Dadurch wird sichergestellt, dass die Elemente auch auf kleinen Bildschirmen gut bedienbar sind und keine

[4] Web Content Accessibility Guidelines

Probleme bei der Bedienung auftreten.

4.2.2 Layout Gestaltung

Neben der Elementgestaltung ist die Gestaltung des Layouts der gesamten Webseite sowie des einzelnen Elementes wichtig für ein erfolgreiches Responsive Design.

Allgemein

Eines der wichtigsten Layouts ist das Layout des `<body>`-Element, weil dort Veränderungen am stärksten wahrgenommen werden, schließlich enthält dieses Element alle angezeigte Elemente und besitzt somit die größte `height` sowie *width*, optimalerweise mit `height: 100%` und `width: 100%`. Deshalb sollte besonders dieses Layout responsive gestaltet werden, da sonst die ganze Webseite darunter leidet. Dadurch, dass dieses Layout die gesamte Bildschirmgröße einnimmt, ist es wichtig, wie sich dieses Layout an verschiedene Bildschirmgrößen anpasst.
Die Bildschirmgrößen vieler Geräte sind unterschiedlich von ihrer Pixel-Breite und -Höhe. Trotz der verschiedenen Bildschirmgrößen, lassen sich diese in drei Gruppen einteilen. Die erste Gruppe sind Bildschirme im **Hochformat (Portrait Mode)**. Die zweite Gruppe sind Bildschirme im **Querformat (Landscape Mode)**. Die letzte Gruppe bilden quadratische Bildschirme, diese werden jedoch nur selten genutzt und sind eher eine Nische, weshalb diese Gruppe nicht weiter betrachtet wird. Tendenziell sind kleine Bildschirme eher im Hochformat ausgerichtet, während große Bildschirme meist im Querformat genutzt werden, dies ist aber keine feste Regel, da eine Drehung des Bildschirms möglich ist und somit beide Formate in allen Bildschirmgrößen vorhanden sind. Durch die Einordnung in diese zwei Gruppen ist es möglich für beide Gruppen ein Layout zu erstellen, welches sich an das jeweilige Format anpasst.
Damit das Layout auch responsive ist, sollten Layout-Technologien genutzt werden, die sich gut an verschiedene Bildschirmgrößen anpassen können. Dabei sollten die Technologien aus dem Kapitel 2.2 genutzt werden.

Flexbox

Eine Flexbox eignet sich sehr gut für eindimensionale Layouts, wo es nicht wichtig ist eine feste Anordnung zu besitzen. Seine Darstellung ist optimal für eine Ansammlung von ähnlichen beziehungsweisen gleichen Elementen, wie zum Beispiel bei einer Bildergalerie oder einer Liste von Einträgen. Durch den flexiblen Aufbau sind Flexboxen mit `flex-wrap: wrap` sehr einfach responsive zu gestalten, da diese Layout-Technologie automatisch die Elemente in die nächste Zeile umbrechen lässt, wenn nicht mehr genug Platz in der aktuellen Zeile ist. Dadurch passen sich Flexboxen sehr gut an verschiedene Bildschirmgrößen an. Durch die Nutzung von `flex-shrink` und `flex-grow` kann zusammen mit `min-` und `max-` Eigenschaften die Größe der Flex-Elemente gut gesteuert werden, wodurch sich die Elemente an die verfügbare Fläche anpassen können. Die Eigenschaft `flex-basis` sollte nicht benutzt werden, da diese die Größensteuerung des Elementes übernimmt und somit die Flexibilität des Elementes einschränkt. Die Nutzung von `gap` ist ebenfalls empfeh-

lenswert, um Abstände zwischen den Flex-Elementen zu schaffen, ohne dass zusätzliche Margins genutzt werden müssen. Bei der Anordnung der Flex-Elemente kann mit `justify-content` und `align-items` gearbeitet werden, um die Elemente optimal im Flex-Container zu positionieren. Falls ein Flex-Element auf der Querachse anders positioniert werden soll als die anderen Elemente, kann mit `align-self` gearbeitet werden, um dieses Element individuell zu positionieren.

Grid

Grid ist eine weitere Layout-Technologie, die sich gut für Responsive Design eignet. Grids arbeiten mit Zeilen und Spalten, die ein Raster bilden, in dem die Kindelemente positioniert werden können. Dadurch ist es möglich, dass Kindelemente mehrere Zeilen und Spalten einnehmen können, wodurch zweidimensionale Layouts möglich sind. Dabei sollten Zeilen und Spalten etwa feste Einheiten nutzen wie `rem` sowie `px` oder die flexible Einheit `fr` benutzen, um den verfügbaren Platz aufzuteilen. Die Einheit `%` eignet sich weniger, da der Entwickler manuell auf die 100% kommen muss und die Eigenschaft `min-*` der Grid-Elemente nicht beachtet wird, wodurch die Eigenschaften der Grid-Elemente ignoriert werden. mit `fr` werden die Eigenschaften der Grid-Elemente beachtet. Ebenfalls beachtet `fr` auch die `gap`-Eigenschaft des Grids. Durch diese festere Struktur, ist es unüblich sich an die Elementgröße anzupassen, weshalb es sich eignet mehrere Grid-Layouts für verschiedene Bildschirmgrößen beziehungsweise Elementgrößen zu erstellen und diese dann mit Media-Queries auszutauschen. Als Media-Queries eignen sich besonders die Eigenschaften `@media` und `@container`. Wenn ein Gridlayout sich verändert, ist es empfehlenswert mit `grid-template-areas` zu arbeiten, da so die Anordnung der Kindelemente sehr einfach angepasst werden kann, ohne dass die Kindelemente selbst angepasst werden müssen. Dadurch wird der Entwicklungsaufwand reduziert und die Lesbarkeit des Codes verbessert. Falls sich das Grid eher nach den Elementen richten soll, kann mit dem Wert `repeat(auto-fit, minmax(...))` gearbeitet werden. Dadurch entstehen so viele Spalten oder Zeilen, die ein Größenbereich haben und sich an die Größe der Gridelemente anpassen. Damit ist das Grid deutlich flexibler und passt sich eher an die Elternelementgröße an als an die Bildschirmgröße.

4.3 Touchgesten

Um eine erfolgreiche Implementierung zu gewährleisten, müssen passend zu den Anforderungen aus Kapitel 3.3 Lösungen gefunden werden, die für eine hohe Nutzerzufriedenheit sorgen und zeitgleich den Implementierungsaufwand gering halten.

4.3.1 Implementierung von Touchgesten

Bei der Implementierung von Touchgesten ist es fundamental, dass alle Aktionen der Webanwendung auch mit Touchgesten ausgeführt werden können. Einschränkungen für Touchnutzer führen zu einer schlechten Nutzererfahrung und limitieren die Zugänglichkeit der Webanwendung auf Touchgeräten. Bei einer komplexen Webanwendung ist es jedoch schwierig alle Aktionen mit einzigartigen Touchgesten abzubilden, da die Anzahl je nach Funktionsumfang stark variieren kann. Deshalb soll-

ten generische Aktionen mit allgemeinen Touchgesten abgebildet werden, die auf viele Aktionen angewendet werden können.
Für die Implementierung sollten die Javascript PointerEvents genutzt werden, da diese sowohl Maus- als auch Touchereignisse unterstützen und somit eine einheitliche Schnittstelle für beide Eingabemethoden bieten. Dadurch wird der Programmieraufwand reduziert, da nicht für jede Eingabemethode eigene EventListener implementiert werden müssen. `TouchEvents` sind auch für Touchgesten zuständig, jedoch werden diese weder vom Safari-Browser noch vom Firefox-Browser unterstützt [29], wodurch eine allgemeine Anforderung verletzt wird. `PointerEvents` werden von allen gängigen Browsern unterstützt und sind somit die bessere Wahl für die Implementierung von Touchgesten.
Mithilfe der `PointerEvents` ist es möglich eigene Touchgesten zu implementieren, indem Eigenschaften des Events mit Bedingungen verknüpft werden. Einige Touchgesten sind bereits vom Browser definiert und können ohne eigene Implementierung genutzt werden. Das Event `click` bildet ein Tap ab und `dblclick` ist ein Double-Tap. Die Touchgesten Scroll, Zoom und Pinch werden durch das Standardverhalten des Browsers abgedeckt. Elemente, die einen Scrollbalken besitzen, können mit einer Scrollgeste bedient werden. Zoom und Pinch sind auf der ganzen Webanwendung möglich und erlauben es dem Nutzer die Webanwendung zu vergrößern oder zu verkleinern. Dabei ist zu beachten, dass diese Geste eher einer Lupe gleichkommt, als der üblichen Zoom-Funktion des Browsers, die den Inhalt skaliert.
Dieses Verhalten kann mit der CSS-Eigenschaft `touch-action` angepasst werden, um das Browserverhalten auf dem Element beispielsweise zu deaktivieren. Ebenfalls kann beim `PointerEvent` die Methode `event.preventDefault()` aufgerufen werden, wodurch das Standardverhalten des Browsers abbricht.
Das Scrollen sollte nicht deaktiviert werden, da dies eine wichtige Funktion ist für die Webanwendung und die Nutzererfahrung stark beeinträchtigen würde. Für Zoom und Pinch kann es jedoch sinnvoll sein, diese zu deaktivieren, wenn die Webanwendung eigene Implementierungen für diese Gesten besitzt, um Konflikte zu vermeiden.
Ein weiterer wichtiger Aspekt ist, dass es keine native Lösung gibt, um `:hover`-Zustände mit Touchgeräten abzubilden. Dadurch könnte Styling oder auch Funktionalität verloren gehen, da Touchgeräte keinen Hover-Zustand besitzen. Eine Möglichkeit diese Funktionalität anzubieten ist es, eine Hold-Geste zu verwenden, da diese keine Tap-Aktion ausführt, aber dennoch eine Interaktion mit dem Element darstellt. Dadurch kann der Nutzer beispielsweise ein Tooltip öffnen, welches bei Hover mit der Maus geöffnet werden würde. Die restlichen definierten Touchgesten, siehe Tabelle 2.6, benötigen eine eigene Implementierung, die auf den `PointerEvents` basieren sollte.
Bei der Implementierung einer Touchgeste sollte der EventListener nur gültig sein für das Element, welches die Geste unterstützen soll. Dadurch wird verhindert, dass die Geste auf anderen Elementen ausgelöst wird, was zu unerwartetem Verhalten führen kann. Ebenfalls sollten Touchgesten auf sehr großen Elementen mit Bedacht eingesetzt werden, wenn die Kindelemente des Elementes ebenfalls Touchgesten besitzen, die ähnlich zur Touchgeste des Elternelementes sind. Dies gilt auch für mehrere Touchgesten auf einem Element. So kann ein Wischen auch als ein Scroll interpretiert werden

oder ein Hold wird ausgelöst, obwohl ein Drag and Drop ausgeführt werden sollte. Dies mindert deutlich die Nutzererfahrung und sollte vermieden werden.
Wenn komplexere Gesten implementiert werden sollen, ist es empfehlenswert mit VueUse sowie den Erweiterungen Gestures und Motion zu arbeiten. VueUse bietet die Composable `usePointer` an, welche alle wichtigen Informationen eines Pointers bereitstellt, wie die Position, den Typ und ob der Pointer gedrückt ist oder nicht. Dadurch lässt sich sehr einfach eine eigene Touchgeste implementieren. Für komplexere Gesten wie Swipe, Pinch oder Drag and Drop bietet VueUse Gestures bereits Composables an, die diese Gesten implementieren und nur noch in der Komponente verwendet werden müssen. Dadurch lässt sich der Programmieraufwand deutlich verringern und die Implementierung wird vereinfacht. Bei der Benutzung von `usePointer` oder den VueUse Gestures Composables ist es empfehlenswert noch VueUse Motion zu verwenden, da damit Animationen und auch visuelle Indikatoren für die Touchgeste möglich sind, um dem Nutzer eine Rückmeldung zu geben, dass seine Geste erkannt und ausgeführt wird. Dabei ist es wichtig, dass das Element dem Finger folgt, und der Nutzer nicht in seinem Vorhaben unterbrochen wird. Erst wenn der Finger losgelassen wird, darf eine Aktion stattfinden, basierend auf seiner Geste. Dadurch ist es möglich, dass der Nutzer beispielsweise ein Swipe abbricht, indem er den Finger nicht loslässt, sondern in die entgegengesetzte Richtung bewegt. Würde das Element ab einem bestimmten Punkt die Aktion ausführen, wäre es nicht mehr möglich die Geste abzubrechen, was die Nutzererfahrung stark beeinträchtigen würde.

4.3.2 Nutzung und visuelle Indikatoren

Bei der Gestaltung einer Touchgeste sollte nicht nur bedacht werden, wie die Touchgeste funktioniert. Es ist auch wichtig, dass der Nutzer weiß, dass dort eine Touchgeste möglich ist. Dabei können visuelle Indikatoren helfen, die dem Nutzer unterbewusst zeigen, dass dort eine Aktion möglich ist. Eine Möglichkeit wäre es eine Animation auf das Element anzuwenden. So könnte ein Element leicht pulsieren, sich bewegen oder die Geste andeuten, wodurch der Nutzer aufmerksam gemacht wird, dass dort eine Touchgeste möglich ist und gegebenenfalls gezeigt wird, was mit diesem Element zur Verfügung steht. Eine weitere Möglichkeit ist es ein Icon einzubinden, welches die Geste andeutet. So könnten Pfeile eine Richtung andeuten, in die ein Element bewegt werden kann oder Touch-Icons benutzt werden, um eine allgemeine Aktion anzuzeigen. Bei der Nutzung von Touchgesten, die ein Element bewegen ist es wichtig, dass das Element einen Schatten besitzt und sich so vom Hintergrund abhebt. Dadurch wird dem Nutzer signalisiert, dass das Element bewegt werden kann und es nicht fest mit dem Hintergrund verbunden ist. Allgemein sollten visuelle Indikatoren dezent eingesetzt werden, um den Nutzer nicht zu überfordern und die Webanwendung unübersichtlich wirken zu lassen. Dabei sind Animationen sehr aufdringlich, währends Icons dezenter sind und Schatten ein gängiges Gestaltungsmittel für interaktive Elemente sind. Aus diesem Grund sollten Animationen nur für die wichtigen Elemente eingesetzt werden. Icons können für alle interaktiven Elemente eingesetzt werden, sollten jedoch nicht zu dominant sein. Schatten können soweit immer eingesetzt werden, jedoch ist zu beachten, dass Schatten auch aus Styling-Gründen eingesetzt werden und nicht immer

ein interaktives Element bedeuten, weshalb es im richtigen Kontext erst als ein interaktives Element wahrgenommen wird.
Auch ohne visuelle Indikatoren können Touchgesten angedeutet werden, anhand des Aufbaus der Webanwendung. So lädt eine einzelne Karte dazu ein, diese zu wischen. Ebenso kann eine Liste von Elementen dazu einladen, diese zu scrollen. Dadurch wird dem Nutzer klar, dass er mit diesen Elementen interagieren kann, ohne das es explizit angezeigt wird. Wenn dies nicht eindeutig sichtbar ist, sollte ein visueller Indikator hinzugefügt werden, um die Nutzererfahrung zu verbessern. Elemente sollten innerhalb der Webanwendung auch einer Designlinie folgen, damit der Nutzer beim Benutzen der Seite lernt, welche Elemente interaktiv sind und welche nicht. Dadurch wird die Webanwendung konsistenter und die Nutzererfahrung verbessert sich. Das was der Nutzer gelernt hat, kann er auf andere Elemente der Webanwendung übertragen, wodurch er schneller und einfacher mit der Webanwendung interagieren kann.

4.4 Strategie

Nachdem nun alle Konzepte und Anforderungen zusammengekommen sind, wird in diesem Kapitel beschrieben wie die Implementierung des Prototyps in Miori Boards aussehen sollte.
Zu Beginn wird sich um das Responsive Design gekümmert, da dies eine Überarbeitung der gesamten grafischen Nutzeroberfläche bedeutet. Würde der Entwickler zuerst mit den Touchgesten anfangen, könnte es passieren, dass Elemente, die zuvor wichtig waren für die Touchgesten, durch das Responsive Design verändert oder entfernt werden. Dies würde zu unnötigen Mehraufwand führen.

Responsive Design

Für den Beginn der Implementierung des Responsive Designs bietet es sich an, zuerst ganz oben im HTML-Dokument zu starten und sich dann Stück für Stück nach unten vorzuarbeiten. Dadurch können schon Limitationen früh erkannt werden und diese vorbeugend zu lösen beziehungsweise mit zu Bedenken im weiteren Design. Dabei sollte der Entwickler sich zuerst überlegen welches Kindelement das Hauptelement ist. Da dieses Hauptelement einen Stellenwert hat, sollte es bevorzugt behandelt werden und soviel Platz einnehmen wie es braucht, da dies das Kernelement des Elternelementes ist. Wenn das Hauptelement noch nicht designt ist, ist es sinnvoll, sich zuerst Gedanken über dieses zu machen, um die Größe und den Platzbedarf abschätzen zu können. Es kann je nach Element auch Sinn ergeben von unten anzufangen, wenn tiefer im Elementbaum bereits Limitationen vorhanden oder bekannt sind, die beim Design des Elternelementes berücksichtigt werden müssen. Wenn das erste Element bestimmt ist, geht es darauffolgend mit dem zweitwichtigsten Element weiter. Dies geht solange bis alle Kindelemente des Elternelementes bestimmt sind.
Bei der Bestimmung der Elemente sollte der Entwickler zuerst anfangen für kleine Bildschirme zu designen (Mobile-First Ansatz) [20]. Dadurch wird sichergestellt, dass die wichtigsten Elemente auch auf kleinen Bildschirmen Platz finden und nicht von unwichtigen Elementen verdrängt werden. Dazu empfiehlt es sich mit einer Breite von `360px` zu starten, da dies die kleinste gängige Bildschirmbreite

von Smartphones ist. Die gängigste kleinste Bildschirmhöhe beträgt `640px`. Interaktive Elemente sollten dabei eine Mindestgröße von `24px` x `24px` besitzen, bevorzugt aber `44px` x `44px` groß sein.
Wenn alle Elemente bestimmt sind, kann sich nun entscheiden werden welches Layout benutzt wird. So bieten sich Grids gut an für feste Anordnungen von Elementen, während Flexboxen gut für flexible Anordnungen geeignet sind, wo die exakte Anordnung der Elemente nicht so wichtig ist. Besonders bei Grids ist zu beachten, dass je nach Bildschirmgröße ein anderes Grid-Template benutzt werden sollte, um die Elemente bestmöglich anzuordnen.
Nach diesem Schema arbeitet sich der Entwickler rekursiv durch den DOM-Baum, bis alle Elemente auf der Ansicht beziehungweise Seite angepasst sind. Danach können Anpassungen gemacht werden, um die Ansicht für größere Bildschirme zu optimieren. Bei der Benutzung von Flexboxen und Grids ist dies meist nur eine Anpassung der Anordnung der Elemente, da die Größen sich meist automatisch anpassen. Elemente selbst können jedoch in eine komplett alternative Darstellung wechseln, wenn dies für größere Bildschirme sinnvoll ist, wie beispielsweise eine Navigationsleiste, die von einem Dropdown mit einzelnen Einträgen zu einer Auflistung der Menüpunkte wechselt.

Touchgesten

Wenn die Ansicht nun responsive ist, kann mit der Implementierung der Touchgesten begonnen werden. So sollten primär nur EventListener benutzt werden, die PointerEvents unterstützen. Falls doch ein MouseEvent oder KeyboardEvent benutzt wird, muss es dazu passend ein Fallback für Touchgeräte geben. Dabei ist es wichtig zu beachten was die Aktion macht, um damit passend eine Geste zuzordnen. Wenn die Implementierung erfolgt ist, sollte noch ein Indikator eingebaut werden, wenn die Geste nicht direkt ersichtlich ist aus dem Elementenkontext heraus. Je komplexer die Geste ist, desto wichtiger ist es einen Indikator zu haben, damit der Nutzer die Geste auch wirklich entdeckt und versteht.

Miori Boards

Im Kontext von Miori Boards sollten bei der Implementierung zuerst alle Komponenten angepasst werden, die aus den WZL-Essentials stammen, da diese in der gesamten Webanwendung verwendet werden und von sich aus Limitationen besitzen können. Hier ist die Benutzung von simplen Touchgesten und einem guten Responsive Design besonders wichtig, da diese Komponenten sehr oft verwendet werden. Danach sollten alle wiederverwendbaren Komponenten von Miori Boards angepasst werden, da diese ebenfalls an mehreren Stellen verwendet werden. Dort können komplexere Touchgesten und spezialisiertere Responsive Designs verwendet werden, da diese Komponenten nur in Miori Boards genutzt werden und nicht in anderen Projekten, wodurch ein generisches Design nicht notwendig ist. Zum Schluss sollten die einzigartigen Elemente der einzelnen Ansichten beziehungsweise Seiten angepasst werden. Wenn nach dieser Strategie vorgegangen wird, sollte eine gute Grundlage für die Implementierung des Prototyps in Miori Boards geschaffen sein und folglich auch für alle weiteren Webanwendungen, die nach dieser Strategie arbeiten.

5 Prototyp und Evaluation

Nachdem in Kapitel 4 die Implementierungsstrategie für den Prototypen beschrieben wurde, wird in diesem Kapitel beschrieben, wie der Prototyp in Miori Boards umgesetzt wurde. Dazu folgt eine Evaluation des Konzepts

5.1 Prototyp in Miori Boards

Für die prototypische Implementation in Miori Boards wurde exemplarisch die Übersichtseite im Kontext des Responsive Designs überarbeitet. Dazu wurde im Präsentationsmodus des Kachelwidgets eine Swipetouchgeste implementiert. Dazu wurden die in Kapitel 4.4 beschriebenen Strategien angewandt.

5.1.1 Responsive Übersichtseite

Die Übersichtseite von Miori Boards dient als Startpunkt der Webseite und bietet eine Personenkarte an mit Profilbild, Name und E-Mail der Person, sowie ihr Anwesenheitsstatus und mögliche Abwesendheitszeiträume. Des Weiteren gibt es kleine Boardkacheln, die alle Boards repräsentieren, auf die die Person Zugriff hat. Über die Ansammlung der Boardkachel, gibt es eine Suchleiste, mit der nach Boards gesucht werden kann. Unter der Boardkachelauflistung gibt es noch einen Button, um Boards zu erstellen. Die Übersichtseite wurde primär für Laptop- und Desktopbildschirme entwickelt.

Abbildung 5.1: Miori Boards Übersichtseite ohne Responsive Design

Abbildung 5.2: Miori Boards Übersichtseite mit Responsive Design

Wie in der Abbildung 5.1 zu sehen, ist die Übersichtseite nicht geeignet für kleinere Bildschirme, da die Personenkarte überläuft in die Horizontale sowie die Boardkachelauflistung ebenfalls keinen festen Rahmen hat. Ein weiterer Negativpunkt ist, dass der Button zur Boarderstellung unter der

Boardsauflistung ist, wodurch bei einer großen Anzahl an Boards lange gescrollt werden muss, um den Button zu erreichen.
Zu Beginn der Überarbeitung wurde der Mobile-First Ansatz verfolgt, in dem die Bildschirmgröße auf `360px` x `640px` eingestellt wurde, um die Ansicht für ein typisches Smartphone zu simulieren, siehe 5.1. Da die Elemente bereits vorgegeben sind, muss kein neuer HTML-Code geschrieben werden, sondern es muss nur das CSS angepasst werden, um die Ansicht responsive zu bekommen. Dazu wurde zu Beginn die Personenkarte überarbeitet. Die Personenkarte nutzt bereits das Flexbox Layout, jedoch gibt es nur zwei Kindelemente, nämlich das Profilbild und die Informationen. Das Profilbild ist neben dem Informationselement positioniert, wodurch es zu einem Overflow kommt. Mit der Eigenschaft `flex-wrap: wrap` kann dies behoben werden, wodurch bei zu geringen Platz das Informationselement unter das Profilbild rutscht. Dadurch existiert kein Overflow in der Horizontalen mehr. Ebenfalls wurde der Personenkarte ein Padding von `20px` gegeben, damit der Inhalt nicht direkt am Rand ist. Da das Informationselement der Personenkarte bereits kleiner als 360px groß ist, muss am Layout des Elementes nicht angepasst werden. Die Kindelemente von dem Informationselement nutzen keine responsive Layout-Technologie sondern folgen der Standard-Block-Darstellung. Damit es nun zu keinem Overflow kommt, wird der Name mit der Eigenschaft `word-break: break-word` versehen, wodurch der Name in die nächste Zeile umbricht, wenn kein Platz mehr ist. Die E-Mail Adresse wird mit der Eigenschaft `text-overflow: ellipsis` versehen, wodurch die E-Mail Adresse abgeschnitten wird, wenn der Platz nicht ausreicht. Die Anwesenheitselemente folgen einer festen Größe, wodurch diese nicht angepasst werden müssen.
Da nun die Personenkarte responsive ist, wird als nächstes die Boardkachelauflistung angepasst. Diese nutzt Flexbox und auch die Eigenschaft `flex-wrap: wrap`. Der Grund wieso es zu einem Overflow kommt, liegt daran, dass die Boardkachel keine Limitierung der Breite hat, wodurch es sich über den Bildschirm ausdehnt. Dies wurde behoben, indem die Boardkachel eine Breite von `100%` bekommen hat, wodurch die Boardkachel nur den Platz einnimmt, den es gibt. Durch diese Eigenschaft, passiert es jedoch, dass alle Boardkacheln immer eine ganze Reihe in der Flexbox einnehmen, da sie ja den ganzen horizontalen Platz für sich beanspruchen. Damit dies nicht mehr passiert, wird eine maximale Breite von `340px` für die Boardkachel gesetzt, wodurch das Element nur eine limitierte Breite einnimmt. Die Größe von `340px` wurde rein aus stylischen Gründen gewählt. Durch diese Limitierung sieht die Boardkachel bei jeder Bildschirmgröße größer als `360px` gleich aus, da die Boardkachel immer die maximale Breite einnimmt und nur bei noch kleineren Bildschirmen schrumpft.
Als letzte Änderung wurde der Button zur Borderstellung neu positioniert, damit dieser leichter erreichbar ist. Dazu wurde der Button neben der Boardsuchleiste positioniert, wodurch eine Toolbar passend zur Boardkachelauflistung entsteht. Der Button ist dabei `48px` x `48px` groß, wodurch er auch auf Touchgeräten gut bedienbar ist. Die Toolbar selbst nutzt ebenfalls Flexbox, um die Elemente nebeneinander anzuordnen. Falls der Platz nicht ausreicht, schrumpft nur die Suchleiste, da der Button ansonsten zu klein werden würde und die Bedienbarkeit darunter leidet.
Durch diese Anpassungen ist die Übersichtseite nun responsive und sieht sowohl auf kleinen Bild-

schirmen als auch auf großen Bildschirmen gut aus, siehe Abbildung 5.2.

5.1.2 Touchgesten im Präsentationsmodus

Einer der wichtigsten Features von Miori Boards ist das Kachelwidget, bei dem jeder Nutzer des Boards eine Kachel besitzt, um dort verschiedene Inhalte zu präsentieren. Dabei gibt es einen Präsentationsmodus, bei dem die Kachel zentral auf dem Bildschirm liegt und der Nutzer seine Inhalte präsentieren kann. Der Präsentationsmodus wurde primär für Desktop- und Laptopbildschirme entwickelt, bei denen der Nutzer eine Tastatur zur Verfügung hat, um mit den Pfeiltasten die Kacheln zu wechseln. Durch diese Implementierung ist es nicht möglich, dass Nutzer ohne Tastatur den Präsentationsmodus bedienen können. Aus diesem Grund soll eine Touchgestensteuerung implementiert werden, um den Präsentationsmodus auch auf Touchgeräten (und Mäusen) bedienbar zu machen.

Abbildung 5.3: Miori Boards Präsentationsmodus

Wie in Abbilung 5.3 zu sehen, liegt die Kachel zentral auf dem Bildschirm und der Rest des Bildschirm ist mit einem Overlay abgedunkelt, wodurch die Kachel das einzige eindeutig sichtbare Element ist. Aufgrund der Kachelform bietet es sich an eine Swipegeste zu implementieren, um die Kachel zu wechseln. Dabei stellt ein Swipe nach Links die linke Pfeiltaste dar und ein Swipe nach Rechts die rechte Pfeiltaste. Durch den Aufbau des Präsentationsmodus ist bereits klar, dass die Kachel das einzige interaktive Element ist, wodurch keine weiteren Hinweise für die Bedienung notwendig sind. Für die Touchgestenimplementierung wird die Directive `v-drag` verwendet und dazu wird die Funktion `apply()` aus dem Motionplugin benutzt, um die Kachelbewegung zu animieren. Mit der Directive `v-drag` kann eine Funktion übergeben werden, die bestimmt wie die Logik auszusehen hat bei einem Drag an dem Kachelelement. Die Directive übergibt dabei ein Objekt, welches verschiedene Attribute besitzt, die den Zustand des Elementes beschreiben. Dabei wird zuerst beobachtet, ob es eine Bewegung in die X-Richtung oder Y-Richtung ist. Da die Kachel nur horizontal geswipt werden soll, werden Bewegungen in die Y-Richtung ignoriert und verändern das Element nicht. Bei Bewegungen in die X-Richtung wird die Kachel mit der Funktion `set()` verschoben, um eine flüssige Animation zu erzeugen, die dem Finger folgt. Danach wird aus dem Objekt geprüft, ob ein Swipe

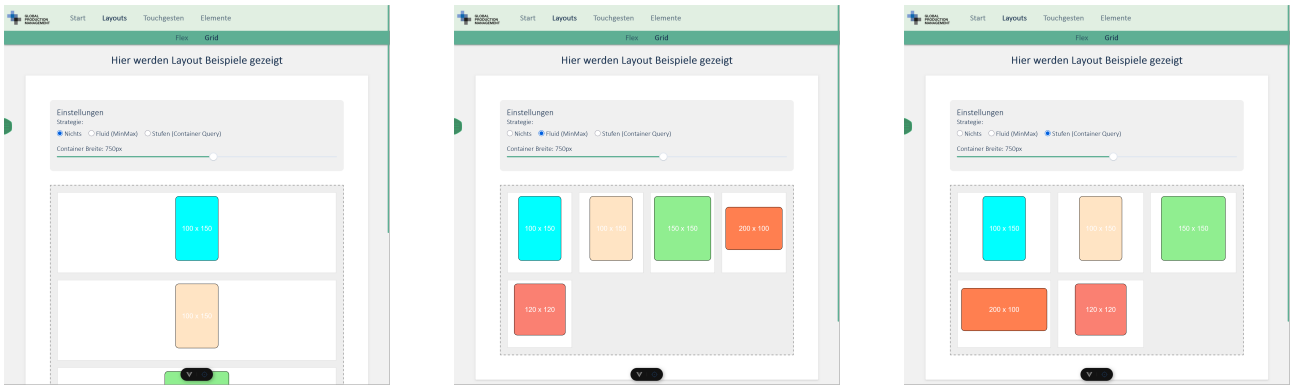
stattgefunden hat. Dabei wird der Swipezustand mit `-1`, `0` und `1` definiert. Bei `-1` wurde ein Swipe nach Links erkannt, bei `1` ein Swipe nach Rechts und bei `0` wurde kein Swipe erkannt. Bei der Prüfung des Zustandes können nun die Funktionen zum Kachelwechsel aufgerufen werden, wenn ein Swipe erkannt wurde. Anschließend wird noch überprüft, ob noch ein Dragging stattfindet. Falls nicht, wird die Kachel mit der Funktion `apply()` wieder in die Ursprungsposition animiert, um die Kachel zurückzusetzen.
Durch diese Implementierung ist es nun möglich, den Präsentationsmodus mithilfe einer Touchgeste zu bedienen, wodurch Nutzer ohne Tastatur ebenfalls den Präsentationsmodus nutzen können. Ebenfalls können Nutzer mit Maus die Kachel wechseln, indem sie die Kachel anklicken und ziehen, um sie zu wechseln, da die Directive `v-drag` mit `PointerEvents` arbeitet.

5.2 Evaluation des Konzepts

Um das Konzept zu evaluieren wurde neben dem Prototypen in Miori Boards noch eine Benchmarkseite erstellt, die einzelne Touchgesten und deren Funktionsweise demonstrieren. Folglich wurden auch Responsive Design Konzepte beispielhaft gezeigt, um diese ebenfalls zu demonstrieren. Ebenso wurden die in Kapitel 3 beschriebenen Anforderungen auf ihre Umsetzbarkeit im Prototypen und der Benchmarkseite überprüft.

5.2.1 Benchmarkseite

Die prototypische Implementierung an Boards zeigt bereits, dass in einzelnen Anwendungsfällen die Implementierungsstrategie erfolgreich funktioniert. Dies evaluiert jedoch nicht, dass das Konzept allgemein funktioniert. Aus diesem Grund wurde eine Benchmarkseite erstellt, die verschiedene Touchgesten und Responsive Design Konzepte demonstriert.



(a) ohne Responsive Design (b) mit minmax-Funktion (c) mit Media Queries

Abbildung 5.4: Vergleich verschiedener Grid-Layouts auf der Benchmarkseite

Die Abbildung 5.4 zeigt einen Ausschnitt der Benchmarkseite. Dort wurden alle definierten Touchgesten aus der Tabelle 2.6 implementiert und können ausprobiert werden. Die Responsive Layout-Technologien Flexbox und Grid wurden ebenfalls implementiert und können auf der Benchmarkseite getestet werden mit verschiedenen Ansätzen diese zu benutzen. Auch wurden einzelne Strategi-

en für responsive Elemente eingebaut, wie das Verhalten von Bildern oder Texten ist, wenn das Elternelement verschiedene Größen annimmt oder wenn eine Liste in den Overflow geht.

5.2.2 Umsetzbarkeit der Anforderungen

Zuletzt wird überprüft, ob die in Kapitel 3 beschriebenen Anforderungen im Prototypen und der Benchmarkseite umgesetzt werden konnten.
Dabei wird mit den allgemeinen Anforderungen aus der Tabelle 3.1 angefangen. Durch die Nutzung von Vue.js und dem Ökosystem ist der Implementierungsaufwand gering, da viele Funktionalitäten bereits durch Bibliotheken abgedeckt werden können. Der modulare Aufbau der Komponenten sorgt für eine leichte Wartbarkeit und Erweiterbarkeit der Implementierung. Ebenfalls ist die Testbarkeit durch den modularen Aufbau möglich mit Benchmarks der Komponenten. Die benutzten CSS-Eigenschaften und Javascript-Events sowie das Vue.js Ökosystem werden von allen gängigen Rendering Engines unterstützt. Mit der Verwendung von grundlegenden CSS-Eigenschaften und elementaren Javascript-Events kann eine konsistente Performance erreicht werden. Bei der übermäßigen Nutzung von den VueUse Bibliotheken besteht jedoch die Gefahr, dass die Performance leidet, wenn zu viele EventListener gleichzeitig aktiv sind. Damit wurden die allgemeinen Anforderungen weitesgehend erfüllt.
Als nächstes werden die Anforderungen der Touchgesten behandelt, die in der Tabelle 3.3 beschrieben wurden. Bei der Implementierungsstrategie wurde darauf geachtet, nur Touchgesten zu benutzen, die eine große Nutzerzahl kennen. Mithilfe von `PointerEvents` und VueUse Gestures Directives sowie Composables kann die Touchgestensteuerung bereitgestellt werden. Die Bibliothek VueUse Motion hilft dabei, dass die Touchgesten eine flüssige und natürliche Reaktion zeigen, sowie visuelle Indikatoren ermöglichen, um die Nutzbarkeit zu verbessern. Der Konflikt von ähnlichen Gesten kann vermieden werden durch die Implementierung in einem Elementenscope sowie dem Deaktivieren von Standardgesten des Browsers. Mit der Empfehlung von großen Icons und Button steigt die Nutzererfahrung auf Touchgeräten, sowie die Zugänglichkeit der Anwendung.Somit wurden auch die Anforderungen der Touchgesten erfüllt.
Abschließend werden die Anforderungen des Responsive Designs aus der Tabelle 3.2 überprüft. Mit der Vorstellung von Flexbox und Grid wurden zwei dynamische Layout-Systeme angeboten, die eine flexible Anordnung von Elementen ermöglichen. Mithilfe von der Overflow-Eigenschaft können Elemente ihre Funktionalität behalten bei Platzmangel, sowie sich dynamisch an den Platz anpassen. Die Verwendung von Media Queries sowie relativer Einheiten erlauben eine reaktive Anpassung der Elemente an verschiedenen Bildschirmgrößen. Durch die Empfehlung des Mobile-First Ansatzes wird sichergestellt, dass die wichtigsten Elemente auch auf kleinen Bildschirmen Platz finden. Dank `rem` und `word-break` ist es möglich die Lesbarkeit von Texten auf verschiedenen Bildschirmgrößen zu gewährleisten. Somit wurden auch die Anforderungen des Responsive Designs erfüllt.
Insgesamt konnte das Konzept erfolgreich in der Benchmarkseite und dem Prototypen in Miori Boards umgesetzt werden, wodurch die Umsetzbarkeit der Anforderungen bestätigt werden konnte.

6 Fazit und Ausblick

In diesem finalen Kapitel werden die wichtigsten Erkenntnisse und Ergebnisse der Seminararbeit zusammengefasst. Zudem wird ein Ausblick auf mögliche zukünftige Entwicklungen und Erweiterungen gegeben, die auf den in dieser Arbeit behandelten Themen aufbauen können.

6.1 Zusammenfassung der Ergebnisse

Ziel war es eine Implementierungsstrategie für die Umsetzung von Responsive Design und Touchgesten auf Miori Boards zu entwickeln. Mithilfe des Einsatzes von Vue.js und VueUse sowie der Strategie für Touchgesten und Responsive Design konnte eine solide Grundlage geschaffen werden, die eine effiziente und benutzerfreundliche Implementierung ermöglicht. Mit dem Mobile-First Ansatz sowie der komponentenbasierten Implementierung werden bereits potenzielle Probleme verhindert, bevor diese auftreten können. Es ist jedoch wichtig zu beachten, dass bei zu vielen EventListenern in einer Ansicht die Performance beeinträchtigt werden kann. Die prototypische Umsetzung der Strategie zeigt, dass diese an Miori Boards erfolgreich funktioniert und somit der Grundstein für die Implementierung gelegt ist. Die entwickelte Strategie bietet damit eine passende Basis für zukünftige Projekte und kann als Leitfaden für die Umsetzung ähnlicher Anforderungen dienen.

6.2 Ausblick

Nun muss nur noch die tatsächliche Implementierung auf den Miori Boards erfolgen. Nach der Implementierung gibt es noch weitere Möglichkeiten zur Erweiterung von Miori Boards.

6.2.1 Progressive Web App (PWA)

Eine mögliche Erweiterung wäre die Entwicklung von Miori Boards als PWA[1]. PWAs ermöglichen es den Nutzern, die Anwendung wie eine native App zu installieren und gegebenefalls auch offline zu öffnen. Dies könnte die Benutzererfahrung weiter verbessern und die Zugänglichkeit der Anwendung erhöhen, da nicht mehr eine Webseite sondern eine Anwendung benutzt wird [23].

6.2.2 Web- und Service-Worker

Infolge der PWA-Implementierung könnten Web- und Service-Worker genutzt werden, um Hintergrundprozesse zu ermöglichen. Dadurch könnten beispielsweise Push-Benachrichtigungen implementiert werden, die den Nutzer über wichtige Ereignisse informieren, auch wenn die Anwendung nicht aktiv genutzt wird [32]. Service-Worker könnten zudem genutzt werden, um Ressourcen zu cachen und so die Ladezeiten der Anwendung zu verbessern oder die Offlinenutzung der Anwendung ermöglichen [28].

[1] Progressive Web App

Quellenverzeichnis

[1] Apple. „Gestures,“ besucht am 15. Dez. 2025. Adresse: https://developer.apple.com/design/human-interface-guidelines/gestures#Specifications

[2] D. C. ENGELBART, „X-Y POSITION INDICATOR FOR A DISPLAY SYSTEM,“ US 3541541 A, Patented Nov. 17, 1970, 1970. Adresse: https://worldwide.espacenet.com/publicationDetails/biblio?locale=de_EP&CC=US&NR=3541541

[3] Google. „Gestures,“ besucht am 15. Dez. 2025. Adresse: https://m3.material.io/foundations/interaction/gestures

[4] Google. „Gestures,“ besucht am 15. Dez. 2025. Adresse: https://m2.material.io/design/interaction/gestures.html#properties

[5] GPMC. „Miori Boards,“ besucht am 15. Dez. 2025. Adresse: https://www.miori.tools/shopfloor-board/

[6] B.-J. Krings, „New Work und die Zukunft der Arbeit,“ *Aus Politik und Zeitgeschichte*, Nr. 46/2023, S. 04–09, 8. Nov. 2023, ISSN: 0479-611X. besucht am 15. Dez. 2025. Adresse: https://www.bpb.de/shop/zeitschriften/apuz/new-work-2023/542500/new-work-und-die-zukunft-der-arbeit/

[7] E. Marcotte. „Responsive Web Design,“ A List Apart, besucht am 15. Dez. 2025. Adresse: https://alistapart.com/article/responsive-web-design/

[8] MDN. „box-sizing,“ besucht am 15. Dez. 2025. Adresse: https://developer.mozilla.org/en-US/docs/Web/CSS/box-sizing

[9] MDN. „calc(),“ besucht am 15. Dez. 2025. Adresse: https://developer.mozilla.org/en-US/docs/Web/CSS/calc()

[10] MDN. „clamp(),“ besucht am 15. Dez. 2025. Adresse: https://developer.mozilla.org/en-US/docs/Web/CSS/clamp()

[11] MDN. „CSS box alignment overview,“ besucht am 15. Dez. 2025. Adresse: https://developer.mozilla.org/en-US/docs/Web/CSS/Guides/Box_alignment/Overview

[12] MDN. „CSS Box Model,“ besucht am 15. Dez. 2025. Adresse: https://developer.mozilla.org/en-US/docs/Learn/CSS/Building_blocks/The_box_model

[13] MDN. „CSS Grid Layout,“ besucht am 15. Dez. 2025. Adresse: https://developer.mozilla.org/en-US/docs/Learn_web_development/Core/CSS_layout/Grids

[14] MDN. „CSS Values and Units,“ besucht am 15. Dez. 2025. Adresse: https://developer.mozilla.org/en-US/docs/Learn_web_development/Core/Styling_basics/Values_and_units

[15] MDN. „flex,“ besucht am 15. Dez. 2025. Adresse: https://developer.mozilla.org/en-US/docs/Web/CSS/Reference/Properties/flex

[16] MDN. „Flexbox,“ besucht am 15. Dez. 2025. Adresse: https://developer.mozilla.org/en-US/docs/Learn_web_development/Core/CSS_layout/Flexbox

[17] MDN. „fr,“ besucht am 15. Dez. 2025. Adresse: https://developer.mozilla.org/en-US/docs/Web/CSS/fr

[18] MDN. „gap,“ besucht am 15. Dez. 2025. Adresse: https://developer.mozilla.org/en-US/docs/Web/CSS/gap

[19] MDN. „minmax(),“ besucht am 15. Dez. 2025. Adresse: https://developer.mozilla.org/en-US/docs/Web/CSS/Reference/Values/minmax

[20] MDN. „Mobile First,“ besucht am 15. Dez. 2025. Adresse: https://developer.mozilla.org/en-US/docs/Glossary/Mobile_First

[21] MDN. „overflow,“ besucht am 15. Dez. 2025. Adresse: https://developer.mozilla.org/en-US/docs/Web/CSS/overflow

[22] MDN. „PointerEvent,“ besucht am 15. Dez. 2025. Adresse: https://developer.mozilla.org/en-US/docs/Web/API/PointerEvent

[23] MDN. „Progressive Web Apps,“ besucht am 15. Dez. 2025. Adresse: https://developer.mozilla.org/en-US/docs/Web/Progressive_web_apps

[24] MDN. „Rendering engine,“ besucht am 15. Dez. 2025. Adresse: https://developer.mozilla.org/en-US/docs/Glossary/Engine/Rendering

[25] MDN. „Responsive Design,“ besucht am 15. Dez. 2025. Adresse: https://developer.mozilla.org/en-US/docs/Learn_web_development/Core/CSS_layout/Responsive_Design

[26] MDN. „Responsive Images,“ besucht am 15. Dez. 2025. Adresse: https://developer.mozilla.org/en-US/docs/Web/HTML/Guides/Responsive_images

[27] MDN. „scrollbar-gutter,“ besucht am 15. Dez. 2025. Adresse: https://developer.mozilla.org/en-US/docs/Web/CSS/scrollbar-gutter

[28] MDN. „Service Workers,“ besucht am 15. Dez. 2025. Adresse: https://developer.mozilla.org/en-US/docs/Web/API/Service_Worker_API

[29] MDN. „Touch Events,“ besucht am 15. Dez. 2025. Adresse: https://developer.mozilla.org/en-US/docs/Web/API/Touch_events/Using_Touch_events

[30] MDN. „User Agent,“ besucht am 15. Dez. 2025. Adresse: https://developer.mozilla.org/en-US/docs/Glossary/User_agent

[31] MDN. „Using Media Queries,“ besucht am 15. Dez. 2025. Adresse: https://developer.mozilla.org/en-US/docs/Web/CSS/CSS_media_queries/Using_media_queries

[32] MDN. „Using Web Workers,“ besucht am 15. Dez. 2025. Adresse: https://developer.mozilla.org/en-US/docs/Web/API/Web_Workers_API/Using_web_workers

[33] Microsoft. „Übersicht über Windows Touchgesten,“ besucht am 15. Dez. 2025. Adresse: https://learn.microsoft.com/de-de/windows/win32/wintouch/windows-touch-gestures-overview

[34] A. Petrosyan. „Share of users worldwide accessing the internet in 2nd quarter 2025, by device,“ besucht am 15. Dez. 2025. Adresse: https://www.statista.com/statistics/1289755/internet-access-by-device-worldwide

[35] StatCounter. „Distribution of mobile screen resolutions used worldwide in 2024,“ besucht am 15. Dez. 2025. Adresse: https://www.statista.com/statistics/1445438/leading-mobile-screen-resolutions-worldwide/

[36] StatCounter. „Marktanteile der führenden Betriebssysteme weltweit von Januar 2009 bis Juli 2025,“ besucht am 15. Dez. 2025. Adresse: https://de.statista.com/statistik/daten/studie/157902/umfrage/marktanteil-der-genutzten-betriebssysteme-weltweit-seit-2009/

[37] StatCounter. „Marktanteile der führenden mobilen Betriebssysteme an der Internetnutzung mit Mobiltelefonen weltweit von Januar 2011 bis Juli 2025,“ besucht am 15. Dez. 2025. Adresse: https://de.statista.com/statistik/daten/studie/184335/umfrage/marktanteil-der-mobilen-betriebssysteme-weltweit-seit-2009/

[38] W3C. „Target Size (Enhanced),“ besucht am 15. Dez. 2025. Adresse: https://www.w3.org/WAI/WCAG22/Understanding/target-size-enhanced

[39] W3C. „Target Size (Minimum),“ besucht am 15. Dez. 2025. Adresse: https://www.w3.org/WAI/WCAG22/Understanding/target-size-minimum

[40] E. You. „Vuejs/core,“ besucht am 15. Dez. 2025. Adresse: https://github.com/vuejs/core

WZL Werkzeugmaschinenlabor

HTML HyperText Markup Language

DOM Document Object Model

CSS Cascading Style Sheets

rem root em

px pixel

fr fractional unit

API Application Programming Interface

UIEvent User Interface Event

SPA Single Page Application

SFC Single File Component

WCAG Web Content Accessibility Guidelines

PWA Progressive Web App

Abbildungsverzeichnis

Tabellenverzeichnis