Ostbayerische Technische Hochschule Amberg-Weiden
Fakultät Elektrotechnik, Medien und Informatik

Studiengang Medieninformatik

Bachelorarbeit

von

Tim **Hoffmann**

Entwurf und Implementierung einer Progressive Web App am Beispiel der technischen Transition einer Cloud-Lösung für webbasierte Datenbearbeitung und Reporting

Design and implementation of a progressive web app using the example of the technical transition of a cloud solution for webbased data processing and reporting

Ostbayerische Technische Hochschule Amberg-Weiden
Fakultät Elektrotechnik, Medien und Informatik

Studiengang Medieninformatik

Bachelorarbeit

von

Tim **Hoffmann**

Entwurf und Implementierung einer Progressive Web App am Beispiel der technischen Transition einer Cloud-Lösung für webbasierte Datenbearbeitung und Reporting

Design and implementation of a progressive web app using the example of the technical transition of a cloud solution for webbased data processing and reporting

Bearbeitungszeitraum: von 8. November 2021
bis 7. April 2022

1. Prüfer: Prof. Dr.-Ing. Christoph P. Neumann

2. Prüfer: Prof. Dr.-Ing. Ulrich Schäfer

Ostbayerische Technische Hochschule Amberg-Weiden
Fakultät Elektrotechnik, Medien und Informatik

Ostbayerische Technische Hochschule
Amberg-Weiden

Bestätigung gemäß § 12 APO

Name und Vorname
der Studentin/des Studenten: **Hoffmann, Tim**

Studiengang: **Medieninformatik**

Ich bestätige, dass ich die Bachelorarbeit mit dem Titel:

Entwurf und Implementierung einer Progressive Web App am Beispiel der technischen Transition einer Cloud-Lösung für webbasierte Datenbearbeitung und Reporting

selbständig verfasst, noch nicht anderweitig für Prüfungszwecke vorgelegt, keine anderen als die angegebenen Quellen oder Hilfsmittel benützt sowie wörtliche und sinngemäße Zitate als solche gekennzeichnet habe.

Datum: 5. April 2022

Unterschrift:

Ostbayerische Technische Hochschule Amberg-Weiden
Fakultät Elektrotechnik, Medien und Informatik

Ostbayerische Technische Hochschule
Amberg-Weiden

Bachelorarbeit Zusammenfassung

Studentin/Student (Name, Vorname):	**Hoffmann, Tim**
Studiengang:	Medieninformatik
Aufgabensteller, Professor:	Prof. Dr.-Ing. Christoph P. Neumann
Durchgeführt in (Firma/Behörde/Hochschule):	INSIGMA IT Engineering GmbH, Frechen
Betreuer in Firma/Behörde:	Dr. Günter Goetz
Ausgabedatum: 8. November 2021	Abgabedatum: 7. April 2022

Titel:

Entwurf und Implementierung einer Progressive Web App am Beispiel der technischen Transition einer Cloud-Lösung für webbasierte Datenbearbeitung und Reporting

---

Zusammenfassung:

Progressive Web Apps sind eine neue Form von Webanwendung, die Eigenschaften nativer Anwendungen aufweisen. Im Rahmen dieser Arbeit wird untersucht, inwieweit es technisch möglich ist eine MVC-Webanwendung in eine Progressive Web App zu transformieren.

Schlüsselwörter: MIA®, Progressive Web Apps, MVC-Webanwendung

Inhaltsverzeichnis

Abkürzungs- und Begriffsverzeichnis

.cshtml	Dabei handelt es sich um eine HTML-Datei, die auch ASP.net-Module rendert und serverseitig dynamischen Code erstellt
Client	Dieser Begriff bezeichnet ein Programm oder einen Computer, der mit einen Server kommuniziert.
C#	Eine objektorientierte Programmiersprache, die von Microsoft entwickelt wurde und 2001 erschienen ist.
CFS	Die Abkürzung für Client-Fund-Stewardship und die englische Übersetzung für Fremdgeldverwaltung.
CMS	Ist die Kurzform für Content-Management-System und ist eine Software zur gemeinschaftlichen Erstellung, Bearbeitung und Darstellung digitaler Inhalte.
CSR	CSR steht für Client-Side-Rendering. Dabei wird eine Webseite erst auf der Clientseite gerendert.
DI	Diese Abkürzung steht für Dependency Injection. Dabei handelt es sich um eine Technik, mit der ein Objekt die Abhängigkeiten eines anderen Objekts zur Verfügung stellt.
FGV	Diese Abkürzung steht für Fremdgeldverwaltung und ist der deutsche Name der zu entwickelnden PWA.
JSON	Die Abkürzung steht für JavaScript Object Notation und ist ein Datenformat in einer einfach lesbaren Textform.
MPA	Das ist die Kurzform von Multi-Page-Application. Damit sind Webseiten gemeint, die jedes mal neu geladen werden müssen, wenn sich der Inhalt ändert.
MVC	Die Abkürzung steht für Model-View-Controller und beschreibt ein Muster zur Unterteilung einer Software.
Proxy	Komponente eines Systems, die als Schnittstelle zwischen weiteren Komponenten vermittelt.

PWA	Steht für Progressive Web App und meint eine Webseite mit einigen Merkmalen einer nativen Anwendung.
Request	Anfrage eines Clients an einen Server.
Response	Antwort des Servers an einen Client.
Sass	Eine Stylesheetsprache, welche die Erstellung von CSS vereinfacht.
SPA	Ist die Kurzform von Single-Page-Application und steht für eine Webseite, welche Daten vom Server nachlädt, ohne die komplette Seite neu zu laden.
SSR	Ist die Kurzform von Server-Side-Rendering. Dabei wird die Webseite auf dem Server gerendert und anschließend an den Client gesendet.
Stored Procedure	Eine Stored Procedure ist ein vorbereitete SQL- Anweisung in Datenbankmanagementsystemen die immer wieder ausgeführt werden kann.
Thread	Stellt eine Aktivtität (Programmausführung) innerhalb eines Prozesses dar.
URL	Steht für Uniform Resource Locator und ermöglicht die eindeutige Identifizierung einer Ressource im Internet.
W3C	Das World Wide Web Consortium entwickelt Standards für das Internet.
Wildcard-Zertifikat	Darunter versteht man ein SSL-Zertifikat, das alle Subdomains einer Domain mit SSL-Verschlüsselung versieht.

Abbildungsverzeichnis

Tabellenverzeichnis

Listings

Kapitel 1

Einleitung

Dieses Kapitel dient der thematischen Einführung in die Arbeit. Es werden die Ziele der Arbeit definiert und das methodische Vorgehen festgelegt.

1.1 Thematische Einführung

Die Bachelorarbeit wird in Zusammenarbeit mit dem Unternehmen INSIGMA IT Engineering GmbH verfasst. Das Unternehmen hat seine Zentrale in Frechen, westlich von Köln. Die Abteilung für Softwareentwicklung ist in den Geschäftsfeldern Automotive, Cloud Services, Krankenversicherungen und MIA® tätig.

Die Bachelorarbeit ist im Geschäftsfeld MIA® angesiedelt. MIA® bezeichnet ein von der INSIGMA entwickeltes Konzept, um individuelle Anforderungen auf Basis der zugrunde liegenden Daten und Prozesse schnell und unkompliziert mit Hilfe eines technischen Frameworks webbasiert umzusetzen. Es besteht aus kundenspezifischen und allgemein verwendbaren Modulen wie zum Beispiel Projektmanagement, Arbeitszeiterfassung und CMS-Funktionalitäten. Die Seiten werden bei jedem Aufruf unter Berücksichtigung eingegebener Filter und der aktuellen Daten aus der Datenbank erzeugt. Voraussetzung für die ordentliche Darstellung der Daten auf einer Seite ist eine stetige und konstante Internetverbindung.

Da die mobile Internetnutzung innerhalb der letzten sechs Jahre stark anstieg, nähert sie sich der Gesamtinternetnutzung weiter an. So lag die Gesamtinternetnutzung im Jahr 2015 bei 78%, wohingegen die mobile Internetnutzung bei 54% lag. Im Jahr 2020 lag die Gesamtnutzung bei 88% und die mobile Internetnutzung bereits bei 80% (Initiative D21 e. V., 2021). Das bedeutet, immer mehr mobile Endgeräte wie Laptops, Smartphones, Smartwatches und Tablets benötigen eine Internetverbindung, während sie in Bewegung sind. Dadurch entstehen neue Herausforderungen für den Webentwickler, da er die Qualität der Internetverbindung aufgrund von niedriger Netzqualität oder begrenztem Datenvolumen nicht beeinflussen kann. Mit den PWAs ist ein relativ junges Konzept entstanden, das Abhilfe schaffen soll. Dieses Konzept zielt darauf ab, die Reichweite einer Webanwendung mit den Funktionalitäten einer

nativen App zu verknüpfen (Richard und LePage, 2020). Allgemein betrachtet handelt es sich bei PWA's um Web-Apps, die durch Eigenschaften einer nativen App wie zum Beispiel Offlinefähigkeit und Installierbarkeit erweitert werden.

1.2 Ziel der Arbeit

Die Bachelorarbeit wird sich mit der Frage auseinandersetzen, inwieweit die Webanwendung MIA® in eine PWA überführt werden kann. Dazu werde ich anhand einer ausgewählten Teilfunktionalität einen Prototyp entwickeln. Mithilfe dieses Prototyps sollen die Anforderungen an eine PWA, sowie die Einsatzmöglichkeiten analysiert und evaluiert werden. Im Rahmen der Arbeit gilt es, folgende Fragen zu beantworten:

- Eignet sich das Konzept PWA, um MIA® in eine benutzerfreundliche, offlinefähige App überführen zu können?
- Wie lässt sich eine MVC-Webanwendung in eine Progressive Web App überführen?

1.3 Methodisches Vorgehen

Die Fragestellung dieser Bachelorarbeit wird über die Entwicklung einer Anwendung beantwortet werden. Die Arbeit wird in acht Kapitel eingeteilt. Das erste Kapitel dient der Einleitung in die Bachelorarbeit. Das zweite und das dritte Kapitel der Arbeit dienen der Einführung in die Thematik und beschreiben die Grundlagen, Begriffserklärungen und den aktuellen Stand der Technik. Im vierten Kapitel werden die Anforderungen an die zu realisierende Anwendung definiert. Auf Grundlage der Anforderung wird im fünften Kapitel ein Konzept zur Transition einer MVC-Webanwendung in eine Progressive Web App erarbeitet. Im sechsten Kapitel wird auf die Implementierung der Anwendung eingegangen. In den letzten beiden Kapiteln wird die Anwendung evaluiert, ein Fazit gezogen sowie ein Ausblick gegeben.

Kapitel 2

Grundlagen

In diesem Kapitel werden zunächst die wichtigsten Grundlagen und Technologien beschrieben, welche in dieser Arbeit zum Einsatz kommen werden. Des Weiteren werden die zugrunde liegenden Rahmenbedingungen erläutert. Dazu werden im ersten Schritt die Grundlagen von Web Apps, Native Apps und Progressive Web Apps erläutert. Im Anschluss wird das MVC-Pattern beschrieben, nach welchem die Cloud-Lösung MIA® entwickelt wurde. Abschließend wird auf MIA® und die technischen Merkmale eingegangen.

2.1 Applikation

Der Begriff App steht für Applikation, ist eine Anwendungssoftware, welche für einen bestimmten Zweck programmiert wurde und sich durch eine Reihe an Funktionen auszeichnet. Da es sich bei einer PWA um eine Webapp mit Funktionen einer nativen App handelt, wird im Folgenden auf die drei Apptypen eingegangen und ihre Merkmale und Eigenschaften beschrieben.

2.1.1 Native Applikation

Unter einer nativen Applikation oder auch nativen Anwendung wird ein Programm bezeichnet, welches für den Einsatz auf einer bestimmten Plattform oder einem spezifischen Gerät entwickelt wurde. Mit dem Begriff Plattform sind Betriebssysteme wie zum Beispiel macOS, Windows, iOS oder auch Android gemeint. Bei der Entwicklung nativer Anwendungen werden plattformspezifische Programmiersprachen verwendet. Für Android wird dabei auf Java und Kotlin zurückgegriffen, wohingegen die Entwicklung für Apples iOS mit den Programmiersprachen Swift oder Objective-C realisiert wird (Jobe, 2013).

Durch die Plattformabhängigkeit kann die Anwendung optimal auf das Betriebssystem und dessen Eigenschaften abgestimmt werden. Das führt zu einer besonders guten Performance und kann dem Benutzer eine hohe Bedienungsfreundlichkeit bieten. Ebenfalls ist es möglich die Funktionen des Betriebssystems zu nutzen, so kann die

Hardware des Geräts genutzt werden, um die Funktionalitäten der Anwendung zu erweitern, zum Beispiel die Verwendung von Bluetooth als Form der Datenübertragung oder der Einsatz des Beschleunigungssensors, um Bewegungen zu erkennen. Zu den Nachteilen zählt der hohe Entwicklungs- und Wartungsaufwand. Der Grund dafür liegt darin, dass native Anwendungen gegebenenfalls für mehrere Betriebssysteme entwickelt werden müssen, ebenso die erforderlichen Updates. Native Anwendungen werden bevorzugt über einen App-Store vertrieben und müssen auf einem Gerät installiert werden. Durch die Installation kann die App selbst dann verwendet werden, wenn keine oder eine langsame Internetverbindung besteht (Augsten, 2019b).

2.1.2 Web Applikation

Eine Web Applikation oder auch Webanwendung genannt wird über einen Webserver angefragt und im Webbrowser eines Endgeräts ausgeführt. Dadurch benötigt sie keine Installation. Sie ist plattformunabhängig und kann auf nahezu allen Geräten verwendet werden, die einen Browser besitzen und eine entsprechende Internetverbindung aufweisen (IONOS Digitalguide, 2021).

Diese plattformunabhängigen Anwendungen werden gemäß dem Prinzip der Client-Server-Architektur entwickelt. Der Client übermittelt Anfragen an den Server und dient als Schnittstelle zum Benutzer, indem er ihm die Benutzeroberfläche (Frontend) präsentiert. Der Server stellt die Anwendung bereit, nimmt alle Anfragen des Clients entgegen, verarbeitet sie und sendet eine Antwort an den Client zurück (Niemann, 1995). Dieses Verfahren wird in Abbildung 2.1 dargestellt.

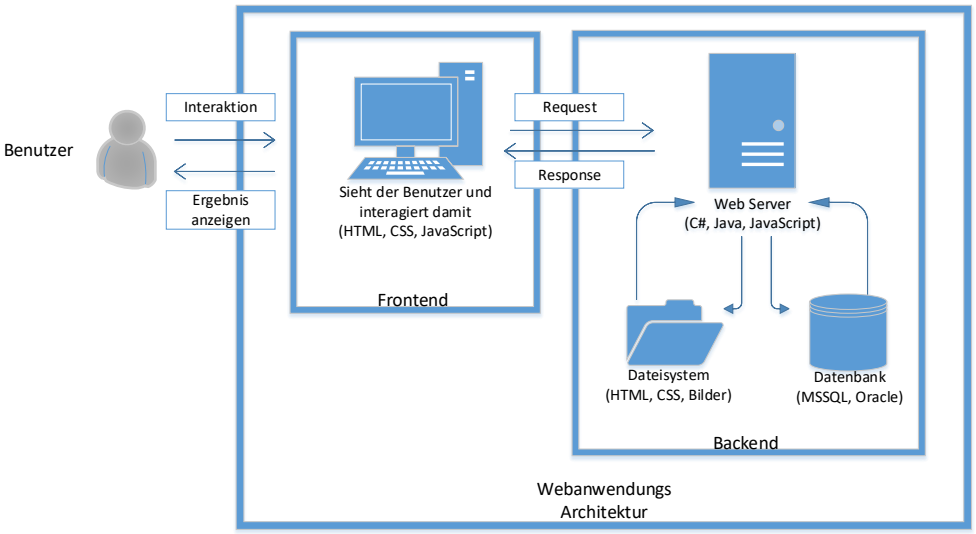

Abbildung 2.1: Architektur einer Webanwendung
(Quelle: Eigene Darstellung)

Der Benutzer kann über das vom Webbrowser (Client) dargestellte Frontend Anfragen (Requests) an einen Webserver (Backend) senden. Dabei werden Eingaben und Parameter des Nutzers an den Webserver übertragen. Nach dem der Webserver den Request erhalten hat, wird dieser verarbeitet, zum Beispiel durch einen Anmeldevorgang oder eine Datenbankabfrage. Im Anschluss sendet der Server eine Response mit dem Ergebnis an den Client zurück. Dieser visualisiert das Ergebnis für den Benutzer.

2.1.3 Progressive Web Apps

Der Begriff Progressive Web App bezeichnet eine neue Methode der App-Entwicklung. Sie kann als Hybrid zwischen einer Webseite und einer nativen App gesehen werden. Die PWA kann über eine URL im Internet aufgerufen werden und wird in einem Browser ausgeführt, dadurch ist sie plattformunabhängig. Dazu kommt, dass sie keine Installation benötigt und somit unabhängig von einzelnen plattformabhängigen App-Stores benutzt werden kann. Der Unterschied zur normalen Webanwendung besteht darin, dass die Progressive Web App durch einen Service Worker und ein Web App Manifest erweitert wird und dadurch Eigenschaften einer nativen App erhält. Diese Erweiterung führt dazu, dass sie ohne Installationsprozess auf dem Home-Bildschirm gespeichert werden kann und wenn passend implementiert, auch offline ausgeführt werden kann. Dafür ist ein Offline-Cache im Browser notwendig, der bei schlechter oder nicht vorhandener Internetverbindung die PWA mit den benötigten Dateien und Daten versorgt (Osmani, 2015).

2.2 Model-View-Controller

Das Model-View-Controller-Pattern ist ein Architektur- bzw. Entwurfsmuster zur Unterteilung einer Software in drei verschiedene Komponenten. Es findet vor allem bei Anwendungen mit grafischer Benutzeroberfläche Anwendung und teilt das Programm in Model (Datenmodel), View (Präsentation) und Controller (Programmsteuerung) auf (Starke, 2020). Da MIA® eine abgewandelte Form des MVC-Patterns verwendet, soll dieses hier anhand der Abbildung 2.2 erläutert werden.

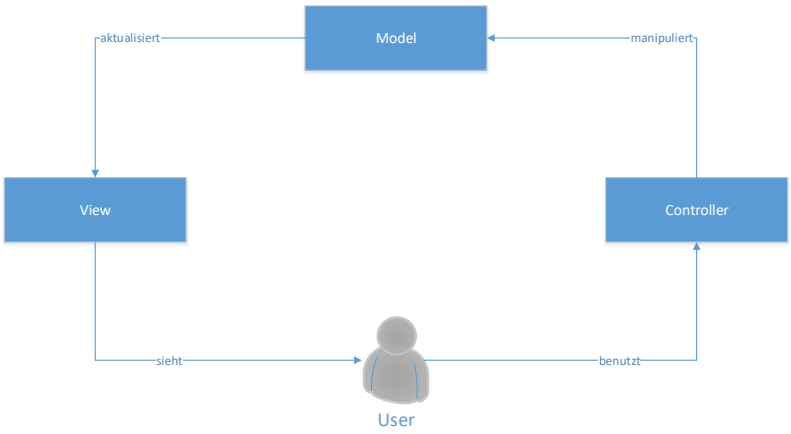

Abbildung 2.2: Architektur des MVC-Patterns mit Benutzerinteraktion (Quelle: Eigene Darstellung)

Das Model verwaltet die Datenstruktur der Anwendung. Es hat die Aufgabe, Daten zu speichern und stellt Funktionen für Datenoperationen bereit. Diese Operationen können das Hinzufügen, Bearbeiten oder Löschen von Daten sein sowie die Suche nach gewissen Datensätzen. Die im Model implementierten Methoden dürfen nicht mit der Benutzerinteraktion oder der Visualisierung assoziiert sein.

Die View stellt die grafische Benutzeroberfläche bereit, auf welcher die Daten des Models visualisiert werden. Dadurch entsteht eine Abhängigkeit der View zum Model, da Änderungen am Model sich automatisch auf die dazugehörigen Views auswirken. Neben der Darstellung der Daten besitzt eine View auch Kontrollelemente, wie zum Beispiel Buttons oder Eingabefelder, mit welchen der Benutzer interagieren kann. Das bei der Interaktion erzeugte Ereignis wird an den Controller weitergeleitet.

Die Aufgabe des Controllers besteht darin, den Zustand der View und des Models anhand von Benutzereingaben zu steuern. Dazu nimmt er die Eingaben der View entgegen und bestimmt die dazugehörigen Methoden im Model. Er übergibt die Daten der View an das Model und stößt somit eine Änderung an den Daten bzw. am Zustand des Models an (Goll, 2011).

2.3 MIA®

MIA® bezeichnet ein von INSIGMA entwickeltes Konzept, um individuelle Anforderungen auf Basis der zugrunde liegenden Daten und Prozesse schnell und unkompliziert mit Hilfe eines technischen Frameworks webbasiert umzusetzen. In Abbildung 2.3 ist die Startseite von MIA dargestellt.

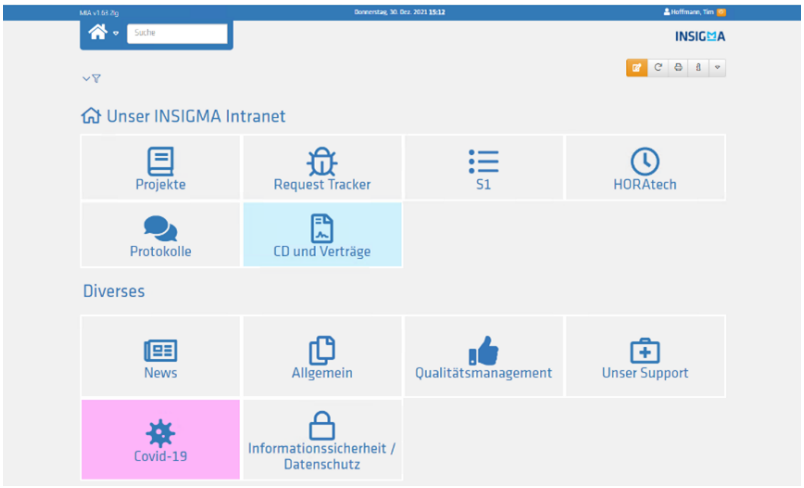

Abbildung 2.3: MIA® Startseite mit einer Übersicht über vorhandene Module (Quelle: Eigener Screenshot aus MIA®)

Es besteht aus individuellen und allgemeinen inhaltlich, funktionalen Modulen wie zum Beispiel Projektmanagement (Siehe Abbildung 2.4), Arbeitszeiterfassung und CMS-Funktionalitäten. Die Module sind branchenunabhängig und können für verschiedenste Zwecke genutzt werden. Die verfügbaren Daten und Funktionen werden über Modulseiten im Browser dargestellt. Die Erzeugung der Seiten erfolgt über die aktuellen Daten aus der Datenbank sowie unter Berücksichtigung eingegebener Filter. Voraussetzung für die ordentliche Darstellung der Daten auf einer Seite ist eine stetige und konstante Internetverbindung.

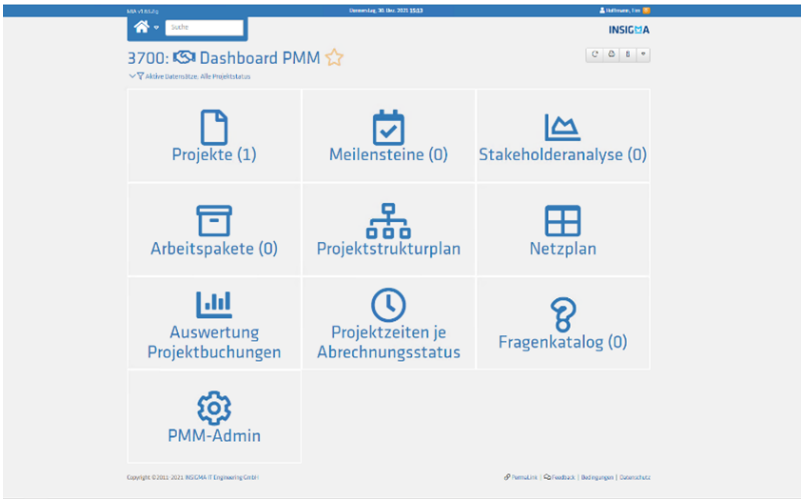

Abbildung 2.4: MIA® Projektmangagement Modul (Quelle: Eigener Screenshot aus MIA®)

Die MIA® Webanwendung wird mit dem Model View Controller Pattern realisiert. Dabei ist zu beachten, dass das Model nicht wie im Lehrbuch als statisches Model in einer Datenbank persistiert wird, sondern erst zur Laufzeit durch die Ausgabe der Datenbank erstellt wird. Genauer gesagt wird das Model durch ein SQL Select Statement generiert, welches dann durch die View im Browser dargestellt wird.

Kapitel 3

Progressive Web Apps

Der Terminus „Progressive Web App“ wurde im Jahr 2015 durch das Unternehmen Google geprägt. Damit sind fortschrittliche Webanwendungen gemeint, welche bestimmte Eigenschaften besitzen und dafür die damals neuesten Browser-Technologien wie HTML5, CSS3, JavaScript, Service Worker und Web App Manifest nutzen (Google Developers, 2015). Im Folgenden soll näher auf die Eigenschaften einer PWA eingegangen werden sowie auf die technischen Hintergründe.

3.1 Eigenschaften

Progressive Web App ist im Grunde nur ein Begriff für einen bestimmten Typ Webanwendung, welche um Funktionsmerkmale einer nativen App erweitert wird. Dabei ist auffallend, dass je nach Quelle die Anzahl der Eigenschaften variiert. So verwendet die Mozilla Foundation lediglich acht Eigenschaften, um eine Progressive Web App zu beschreiben (MDN, 2020a). Osmani (2015) von Google verwendet dagegen zehn Eigenschaften, um eine PWA zu definieren. In der Arbeit verwende ich die von Osmani beschriebenen Eigenschaften, um eine PWA zu definieren :

- Progressive
- Responsive
- Connectivity independent
- App-like
- Fresh
- Safe
- Discoverable
- Re-engageable
- Installable

- Linkable

3.1.1 Progressive

„Die Eigenschaft Progressive bedeutet, dass ältere Webbrowser nicht von der Nutzung einer Progressive Web App ausgeschlossen werden sollen. Setzt der Anwender einen starken Webbrowser ein, erhält er umgekehrt einen erweiterten Funktionsumfang.“ (Liebel, 2019, S. 100)

Dies impliziert die Gewährleistung einer plattformübergreifenden Grundfunktionalität für alle Browser. Da ältere Webbrowser oder Browserversionen gewisse PWA-Schnittstellen nicht zur Verfügung stellen, muss bei der Implementierung durch sogenannte Feature Detection überprüft werden, ob die erforderlichen Schnittstellen zur Verfügung stehen. Falls nichtzutreffend soll die Anwendung nur die ihr möglichen Funktionalitäten bereitstellen, um einen Programmabsturz zu vermeiden. Anhand des Microsoft Internet Explorers lässt es sich folgendermaßen erklären. Aufgrund der fehlenden Unterstützung für Service Worker lassen sich keine Offlinefunktionalitäten bereitstellen. Dennoch sollte es möglich sein, alle Funktionalitäten zu nutzen, die Online stattfinden können und ohne Service Worker auskommen. Starke und moderne Webbrowser wie Google Chrome, der Service Worker unterstützt, können dahingegen auf einen höheren Funktionsumfang zurückgreifen und bieten somit eine höhere Anwendererfahrung (Liebel, 2019).

3.1.2 Responsive

Passt sich das Layout einer Progressive Web App automatisch an die vorhandene Bildschirmauflösung an, so wird diese Eigenschaft responsiv genannt. Sie gewährleistet auf Computern, Tablets und Smartphones ein angepasstes und wiedererkennbares Design und bietet dadurch eine gleichbleibend hohe Benutzerfreundlichkeit (Frain, 2012).

3.1.3 Connectivity independent

Die Verfügbarkeit mobiler Internetverbindungen ist bedingt durch äußere Einflüsse stark schwankend bis gar nicht vorhanden. Dieser Umstand wird durch die Eigenschaft Connectivity independent (Verbindungsunabhängig) adressiert. Eine Progressive Web App hat auch bei nicht existenter oder schlechter Internetverbindung zu funktionieren. Dabei müssen nicht alle Funktionalitäten, wie zum Beispiel Durchführung einer Transaktion oder Versand einer E-Mail möglich sein. Es geht darum, dass der Benutzer weiterhin auf die Anwendung zugreifen kann und bei der Durchführung seiner Kernaufgabe nicht gestört wird (Liebel, 2019).

Um eine Progressive Web App offlinefähig zu gestalten, wird ein Service Worker verwendet. Dieser wird in JavaScript implementiert. In diesem Service Worker können durch Ereignisse HTTPS-Requests abgefangen, bearbeitet oder speziell behandelt werden. Ebenfalls ist es mit dem Service Worker möglich, HTTPS-Responses in einem

festgelegten Cache im Browser zu speichern. Ist eine Progressive Web App offline, können HTTPS-Requests mit den im Cache gespeicherten Daten beantwortet werden (Siegrist, 2021).

3.1.4 App-like

Die Eigenschaft App-like illustriert, dass eine Progressive Web App sowohl im Aussehen als auch in der Bedienung einer nativen App ähneln soll. Damit sind gestalterische Aspekte wie Navigationsstrukturen als auch die Verwendung nativer Schnittstellen, wie zum Beispiel Positionserkennung oder Kamerazugriff gemeint (Liebel, 2019).

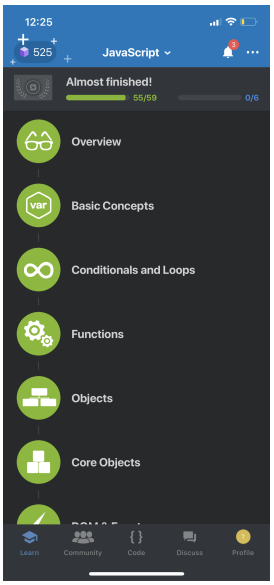

Abbildung 3.1: Native App mit geschlossener Liste (Quelle: Eigener Screenshot aus Sololearn)

Abbildung 3.2: Native App mit offener Liste (Quelle: Eigener Screenshot aus Sololearn)

In Abbildung 3.1 ist eine native App mit ihren Navigationselementen sowohl am oberen als auch am unteren Bildschirmrand erkennbar. Diese dienen dazu, innerhalb der App zwischen verschiedenen Oberflächen zu navigieren. Im mittleren Teil befindet sich eine scrollbare Liste, die sich bei Bedarf ausklappen lässt, um weitere Informationen anzuzeigen, siehe Abbildung 3.2 . Eine Progressive Web App soll sich in ihrer Bedienung genauso anfühlen wie eine native App und auf ähnliche oder gleiche Navigationsstrukturen zurückgreifen.

3.1.5 Fresh

Die Eigenschaft Fresh muss in Abhängigkeit zur Eigenschaft Connectivity independent betrachtet werden. Dabei wird der Service Worker verwendet, um eine bestimmte Version der Webseite zu speichern und somit offline verfügbar zu machen. Nach Springer (2019) ist es bei Webanwendungen selbstverständlich, dass diese stets auf

dem aktuellen Stand sind. Das folgt aus der Tatsache, dass die Anwendung nicht installiert werden muss und die verwendeten Daten bei jedem Aufruf neu vom Server geladen werden. Deshalb muss bei der Konzeption einer Progressive Web App darauf geachtet werden, dass neue Versionen der Anwendung trotz veralteter Offlinekopie im Cache, dem Benutzer schnellstmöglich zur Verfügung gestellt werden können.

3.1.6 Safe

Das Merkmal Safe beschreibt ein Muss-Kriterium bei der Entwicklung einer Progressive Web App. Das Ziel ist es eine sichere Verbindung zwischen Client und Server durch das Kommunikationsprotokoll HTTPS herstellen zu können, damit der Service Worker registriert werden kann. Dies ist ohne HTTPS nicht möglich. Durch die Verwendung von HTTPS kann gewährleistet werden, dass sowohl Quelldateien der Progressive Web App als auch Benutzerdaten sicher übertragen werden und vor Datendiebstahl geschützt sind (Liebel, 2019).

Ein weiterer Grund für die Verwendung von HTTPS liegt in der Möglichkeit auf eine große Anzahl an Webschnittstellen zugreifen können, die zum Teil sensible Nutzerdaten verwenden. Das können sowohl klassische Webschnittstellen wie die „geolocation“-API sein, welche es Nutzern ermöglicht, einer Webanwendung die eigenen Position mitzuteilen (MDN, 2021), als auch Progressive Web App spezifische Schnittstellen wie die Service Worker API. Mit dieser ist es dem Service Worker möglich Netzwerkanfragen abzufangen, zu bearbeiten und passende Maßnahmen zu ergreifen, wenn kein Netzwerk verfügbar ist oder neue Ressourcen auf dem Server vorhanden sind. Des Weiteren ist es dem Service Worker möglich, unabhängig vom Lebenszyklus der Anwendung Hintergrundaktualisierungen auszuführen, um den Cache zu aktualisieren als auch Push-Benachrichtigungen an den Nutzer zu senden, um darauf hinzuweisen, dass neuer Inhalt verfügbar ist (MDN, 2020b).

3.1.7 Discoverable

Die Definition lautet: „Die Eigenschaft Discoverable besagt, dass eine Progressive Web App mithilfe des Web App Manifests von regulären Webseiten unterschieden werden muss.“ (Liebel, 2019, S. 118). Dadurch soll es Suchmaschinen möglich sein, Progressive Web Apps als solche zu erkennen und extra indexieren zu können. Die Identifizierung wird durch die Registrierung eines Service Workers und das Hinzufügen eines Web App Manifests ermöglicht (Springer, 2019).

3.1.8 Re-engageable

Die Eigenschaft Re-engageable setzt sich mit der Frage auseinander, wie der Benutzer angesprochen werden kann, um ihn zur Wiederverwendung der App zu bewegen. Das geschieht in nativen Anwendungen über Push-Benachrichtigungen, die den Nutzer auf etwas hinweisen sollen wie zum Beispiel Aktualisierungen, neue Inhalte oder Kalendererinnerungen (Liebel, 2019).

Progressive Web Apps sind ebenfalls in der Lage Push-Benachrichtigungen zu verwenden. Dafür werden die Funktionen der Push API und die der Notification API kombiniert. Die Push-API wird verwendet, um den Service Worker mit einem Server zu verbinden, dadurch kann dieser ohne Nutzerinteraktion neue Daten vom Server empfangen. Die Notification API kann nach Freigabe des Benutzers vom Service Worker verwendet werden, um den Benutzer neue Informationen anzuzeigen oder ihn darauf hinweisen, dass ein bestimmter Prozess abgeschlossen wurde (MDN, 2022a).

3.1.9 Installable

Die Eigenschaft Installable impliziert, dass eine Progressive Web App, ähnlich wie eine native App, auf dem System installiert werden kann. Dafür benötigt die Anwendung zwingend ein Web App Manifest in welchem Icons, Start-Url und weitere Eigenschaften definiert sind. Die Installation einer Progressive Web App ist jedoch nicht vergleichbar mit der einer nativen App, da sie weiterhin im Browser ausgeführt wird. Es wird lediglich eine Verknüpfung auf dem Homebildschirm angelegt, mit welcher die Anwendung im Browser wieder geöffnet werden kann. Für Progressive Web Apps sind auch keine Installationspakete vorgesehen, da die Anwendung weiterhin im Browser läuft. Je nach verwendetem Gerät und Konfiguration läuft die Anwendung in einem eigenen Fenster oder deckt den gesamten Bildschirm ab (Liebel, 2019).

3.1.10 Linkable

„Die Eigenschaft Linkable gibt an, dass auf die Progressive Web App mithilfe einer URL verwiesen werden kann, im Idealfall bis zu einem bestimmten Zielzustand oder einer Zielsicht.“ (Liebel, 2019, S. 126)

Dieses Merkmal stellt sicher, dass die Anwendung sich per URL leicht verbreiten lässt. Dadurch entfällt die Notwendigkeit eines App Stores oder die eines Setup-Pakets zur Verbreitung der Anwendung. Durch die Verwendung einer URL ist es ebenfalls möglich, auf eine bestimmte Seite, auch Zielsicht oder Zielzustand genannt, innerhalb der Anwendung zu verweisen.

3.2 Service Worker

Nachdem die Eigenschaften einer Progressive Web App erläutert wurden, werden in diesem Abschnitt die Service Worker näher betrachtet. Sie bilden die technische Basis von Progressive Web Apps und sind eine relativ neue W3C-Standard-Webtechnik. Durch Service Worker können Webanwendungen um Funktionalitäten nativer Anwendungen wie zum Beispiel Offline-Fähigkeit, Push- Benachrichtigungen, Hintergrundaktualisierungen oder Performanceverbesserungen erweitert werden. Dafür werden in den nächsten Schritten die technischen Hintergründe erläutert, der Lebenszyklus eines Service Workers vorgestellt sowie verschiedene Caching-Strategien definiert.

3.2.1 Technische Hintergründe

Web Worker ermöglichen es, JavaScript Code unabhängig vom Haupt-Thread der Webanwendung in einem eigenen Thread im Browser auszuführen. Das ist eine notwendige Voraussetzung, da JavaScript eine Single Threaded Programmiersprache ist. Daraus folgt, dass JavaScript Operationen wie Berechnungen oder DOM-Updates nur synchron, das heißt sequentiell ausgeführt werden können. Bei besonders aufwändigen Operationen kann dies die Benutzerinteraktion mit der Webanwendung beeinträchtigen oder komplett unterbinden (Häßler, 2019).

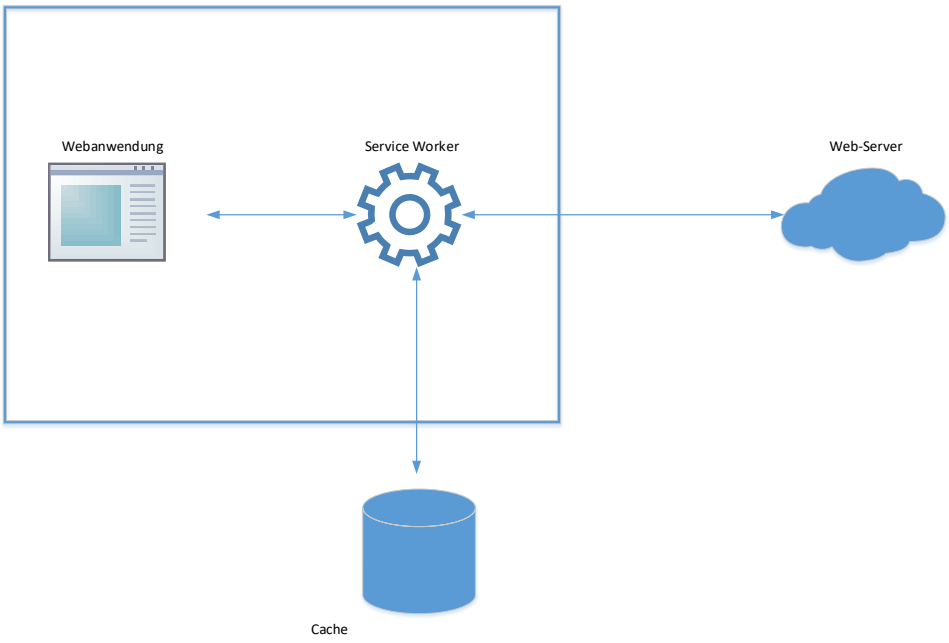

Abbildung 3.3: Funktionsweise des Service Workers
(Quelle: Eigene Darstellung)

Ein Service Worker ist eine spezielle Art des Web Workers. Er organisiert, dass JavaScript Code unabhängig vom Haupt-Thread der Webanwendung in einem eigenen Thread ausgeführt werden kann. Durch das Arbeiten in einem von der Anwendung losgelösten Thread hat er jedoch keine direkte Zugriffsmöglichkeit auf die Webanwendung und ihre DOM-Elemente. Er fungiert als eine Art Proxy zwischen der Anwendung und dem Web-Server oder dem Cache. Dies verdeutlicht Abbildung 3.3, in welcher eingehende Netzwerkanfragen je nach verwendeter Caching-Strategie vom Service Worker abgefangen und abgearbeitet werden (Rojas, 2019).

3.2.2 Lebenszyklus und Ereignisse

Der Lebenszyklus eines Service Workers ist unabhängig von dem der Webanwendung. Im Folgenden sollen die wichtigsten Abschnitte des Lebenszyklus eines Servcie Workers erläutert sowie die dazugehörigen Ereignisse/Funktionen gezeigt werden (Gaunt, 2021):

1. Registration:
   Bevor ein Service Worker installiert werden kann, muss er in der Webanwendung registriert werden. Dazu wird im JavaScript der Webanwendung die *register*-Funktion verwendet, siehe Listing 3.1. War die Registrierung erfolgreich, wird die Service Worker Datei heruntergeladen und der Browser beginnt die Installation im Hintergrund.

```
1    navigator.serviceWorker.register('/sw.js')
```

Listing 3.1: Registrierung des Service Workers

2. Installation:
   Das Installations-Ereignis in Listing 3.2 wird nur einmal im Lebenszyklus eines Service Workers ausgelöst. Es hat die Funktion, den Worker zu installieren und kann erste Assets wie zum Beispiel HTML-, CSS- und JavaScript-Dateien cachen.

```
1    self.addEventListener('install', function(event) {
2      // Installationschritte ausfuehren
3    });
```

Listing 3.2: Eventlistener für das Service Worker install-Event

3. Activated:
   Nach der Installation erfolgt die Aktivierung (Siehe Listing 3.3) des Service Workers. Dieses Ereignis eignet sich dazu, veraltete Caches zu aktualisieren oder zu löschen.

```
1    self.addEventListener('activate', function(event) {
2    // Caches bearbeiten
3    });
```

Listing 3.3: Eventlistener für das Service Worker activate-Event

4. Idle:
   Abschließend wechselt der Service Worker in den Zustand Idle. Er ist nun vollständig einsatzbereit und wartet auf HTTP-Requests. Diese Requests lösen das in Listing 3.4 gezeigte fetch-Ereignis innerhalb des Service Workers aus. Das fängt den Request ab und entscheidet je nach Implementierung, ob neue Inhalte vom Server geladen oder bestehende Inhalte aus dem Cache verwendet werden.

```
1    self.addEventListener('fetch', function(event) {
2    // HTTP-Requests manipuliern
3    });
```

Listing 3.4: Eventlistener für das Service Worker fetch-Event

3.2.3 Caching-Strategien

Die Hauptaufgabe von Service Workern besteht darin, bestimmte Seiten und Inhalte offline zur Verfügung zu stellen. Dafür gibt es eine Reihe von Caching-Strategien, die je nach Anwendungsfall zum Einsatz kommen können, um Inhalte bereitzustellen. Die Strategien, die in nachfolgender Tabelle 3.1 beschrieben werden, lassen sich für jeden Request einzeln festlegen.

Strategie	Beschreibung	Anwendungsfall
Network only	Die Inhalte müssen zu jederzeit aktuell sein und werden deshalb immer neu vom Server angefordert.	Zahlungsverkehr
Network falling back to cache	Die Inhalte sollten aktuell sein, es kann aber bei schlechter oder fehlender Internetverbindung auf die gecachten Inhalte zurückgegriffen werden.	Preise
Stale-while-revalidate	Die angeforderten Inhalte werden sofort aus dem Cache bereitgestellt. Im Anschluss werden die Inhalte im Cache aktualisiert, um bei der nächsten Anforderung aktuellere Inhalte bereitstellen zu können.	Nachrichten
Cache first, fall back to network	Der Inhalt wird zur Performance-Verbesserung direkt durch den Cache bereitgestellt. Ist der Inhalt im Cache nicht verfügbar, wird er neu angefordert und im Cache gespeichert. Ebenfalls sollen die Inhalte gelegentlich aktualisiert werden.	App Shell
Cache only	Der Inhalt wird nur aus dem Cache zur Verfügung gestellt, da er kaum aktualisiert werden muss.	Statischer Inhalt wie Bilder, Icons und Bibliotheken

Tabelle 3.1: Erklärung einzelner Caching-Strategien (Chen, 2020).

3.3 Web-App Manifest

Das Web-App-Manifest ist eine JSON-Datei, die von einer Progressive Web App zwingend benötigt wird. Sie enthält eine Liste von Ressourcen und Eigenschaften, die den Browser darüber informiert, wie sich eine Progressive Web App zu verhalten hat, sollte sie vom Benutzer auf dem Desktop oder einem Mobilgerät installiert werden. Dafür werden in der Manifestdatei unter anderem der Kurzname und der Name der Anwendung, Icons und Farbschema sowie die Start-URL festgelegt.

Das Manifest wird mittels eines link-Elements in das head-Element des HTML-Dokuments eingebunden, siehe Listing 3.5.

```
<head>
  <link rel="manifest" href="manifest.json">
</head>
```

Listing 3.5: Web-App Manifest einbinden

Der Inhalt einer typischen Manifest-Datei wird in Listing 3.6 beschrieben.

```
{
    "name": "My Progressive Web App",
    "short_name": "MyPWA",
    "description": "A prototyp of a progressive web app."
    "icons": [
        {
            "src": "/android-chrome-192x192.png",
            "sizes": "192x192",
            "type": "image/png",
            "purpose": "any maskable"
        },
        {
            "src": "/android-chrome-512x512.png",
            "sizes": "512x512",
            "type": "image/png"
        }
    ],
    "theme_color": "orange",
    "background_color": "red",
    "display": "fullscreen",
    "orientation": "landscape",
    "start_url": "/index.html"
}
```

Listing 3.6: Aussehen einer Web-App-Manifest-Datei

Nachfolgenden werden die wichtigsten Eigenschaften beschrieben. Die Eigenschaften *name* und *short_name* werden verwendet, um der Anwendung einen lesbaren Namen zu geben. Der *short_name* wird an Stellen verwendet, an denen nicht genügend Platz ist,

um den vollständigen Namen anzuzeigen. *icons* wird verwendet, um eine Reihe von Bildobjekten zu definieren, die als Anwendungssymbole in verschiedenen Kontexten dienen. Die Standardfarbe *theme_color* der Anwendung beeinflusst, wie die Anwendung von Betriebssystemen angezeigt wird. Die Eigenschaft *background_color* gibt die erwartete Hintergrundfarbe der Webanwendung an. Dieser Wert wird verwendet, wenn das Manifest vor dem Stylesheet der Webanwendung geladen wurde. Die *display*-Eigenschaft definiert den Anzeigemodus der Anwendung. Gültige Werte dafür sind z.B. *fullscreen*, welche den gesamten Anzeigebereich verwendet oder *minimal-ui*, welches die Anwendung in einem Browser mit einer minimalen Anzahl an UI-Elementen zur Steuerung und Navigation darstellt. Die *orientation* gibt die Standardausrichtung der Anwendung an. Die Eigenschaft *start_url* definiert die URL, die nach der Installation zum Starten der Anwendung verwendet wird (W3C, 2022).

3.4 Trennung von Application Shell und Inhalt

Die Application Shell stellt für eine Progressive Web App eine Art Gerüst dar und beschreibt ein Minimum an HTML, CSS und JavaScript. Dieses wird benötigt, um die Ladezeiten der Anwendung zu verringern und diese offline darstellen zu können. Die Inhalte der App Shell werden nicht bei jedem Seitenaufruf vom Server neu angefordert, sondern nur die benötigten Inhalte (Osmani, 2019).

3.5 Architektur und Entwicklung von Progressive Web Apps

Es gibt zwei verschiedene Ansätze für das Rendern einer Webseite:

1. Server-Side-Rendering (SSR):
   Wie der Name schon andeutet, wird eine Webseite auf dem Server gerendert, bevor sie an den Client gesendet wird. Dies impliziert zwar einen schnelleren ersten Ladevorgang, führt jedoch zu einer langsameren Navigation zwischen den einzelnen Seiten, da neue HTML-Inhalte erst heruntergeladen werden müssen.
2. Client-Side-Rendering (CSR):
   Dieser Mechanismus führt zu einem länger dauernden ersten Seitenaufruf, da zuerst alle Inhalte und Daten heruntergeladen und dann gerendert werden. Allerdings ist die Webseite im weiteren Verlauf performanter, da nicht mehr alle Inhalte neu geladen werden, sondern nur die anzuzeigenden Daten.

Bei der Entwicklung einer Progressive Web App kann auch auf SSR und CSR gleichzeitig zurückgegriffen werden, um eine bestmögliche Benutzererfahrung zu gewährleisten. Die Webseite wird zuerst auf dem Server gerendert, die dazugehörige App Shell im Cache des Clients gespeichert und bei Veränderung auf der Clientseite angepasst. In Folge verläuft sowohl das erste Laden der Webseite als auch die Navigation zwischen den Seiten reibungslos (MDN, 2022c).

3.6 Analyse des aktuellen Stands

Das Ziel dieser Arbeit ist die Beantwortung der Frage, inwiefern sich die Webanwendung MIA®, welche auf dem MVC-Pattern beruht, in eine Progressive Web App überführen lässt. Nachdem im vorherigen Kapitel der Begriff Progressive Web App erläutert und auf ihre Merkmale und technische Eigenschaften eingegangen wurde, analysiert dieser Abschnitt, wie Progressive Web Apps entwickelt werden und welche Relevanz dies für die Überführung einer MVC-Webanwendung in eine Progressive Web App hat. Dafür bin ich bei meiner Untersuchung folgenden Fragen nachgegangen:

1. Welche Frameworks werden verwendet um Progressive Web Apps zu erstellen?
2. Lassen sich mit ASP.NET Progressive Web Apps erstellen?

3.6.1 Aktueller Stand

Progressive Web Apps beruhen auf den Standard-Webtechnologien HTML, CSS und JavaScript. Deshalb werden häufig JavaScript Frameworks eingesetzt, um Progressive Web Apps zu entwickeln. Laut Patel (2021) eignen sich die Frameworks Vue, React, Angular, PWA Builder und Ionic am besten, um eine Progressive Web App zu entwickeln. Es existieren einige Forschungsarbeiten und Bücher zur Entwicklung von Progressive Web Apps auf der Basis dieser Frameworks. Dabei wird gelegentlich, wie zum Beispiel von Rojas: „Progressive applications are, in general terms, a regular Single Page Application“ (Rojas, 2019), die Annahme getroffen, bei Progressive Web Apps würde es sich um reine Single Page Applications handeln. Single Page Applications beschreiben Webanwendungen, die aus einer einzigen HTML-Seite bestehen und Client-Side-Rendering benutzen. Diese Annahme ist jedoch nichtzutreffend, Progressive Web Apps können sich sowohl des Client-Side-Renderings als auch des Server-Side-Renderings bedienen (Osmani, 2015). Das ist ein wichtiger Punkt für das Überführen einer .NET MVC Webanwendung in eine Progressive Web App, da diese typischerweise über Server-Side-Rendering bereitgestellt werden. Mit der ASP.NET Core Blazor WebAssembly ist es möglich, Progressive Web Apps zu erstellen (Microsoft Docs, 2022). Dennoch gibt es keinen offiziellen Leitfaden wie ASP.NET MVC Webanwendung in Progressive Web Apps überführt werden können. Ebenfalls gibt es Stand heute kaum Forschungsarbeiten, die sich mit dieser Fragestellung beschäftigen. Lediglich in Foren und Blogs findet man Diskussionen zu dieser Frage und auch Ansätze dazu, wie sich eine MVC-Webanwendung umbauen lässt, um die minimalen Anforderungen an eine Progressive Web App zu erfüllen.

3.6.2 Fazit

Zusammenfassend lässt sich sagen, dass weiterer Forschungsbedarf bezüglich der Fragestellung besteht, wie eine MVC-Webanwendung in eine Progressive Web App überführt werden kann. Es wird zwar darauf eingegangen, wie sich neue Systeme als Progressive Web Apps entwickeln lassen, jedoch nicht wie sich bereits bestehende Systeme bestmöglich in eine Progressive Web App überführen lassen, um deren volles Potenzial ausschöpfen zu können.

Kapitel 4

Fachliche Anforderungen an das System

In diesem Kapitel werden die fachlichen Anforderungen festgelegt, welche den Umfang der zu entwickelnden Anwendung abstecken und der Beantwortung der Forschungsfragen dienen. Zur Identifizierung dieser wird die Abkürzung FA (fachliche Anforderung) sowie eine fortlaufende Nummer verwendet. Bei der Anwendung handelt es sich um eine Kassenanwendung der SKM Rhein-Sieg, welche als Modul für MIA® entwickelt wird. Sie soll den Betreuern der SKM den Umgang mit ihren Klienten vereinfachen, indem sie mit deren Hilfe die Konten ihrer Klienten einsehen und bei Bedarf Ein- und Auszahlungen an diese vornehmen können. Die Anforderungen werden nach der Entwicklung der Anwendung verwendet, um zu evaluieren und mögliche Verbesserungen zu benennen. Des Weiteren soll damit sichergestellt werden, dass die Forschungsfrage: „Eignet sich das Konzept PWA, um MIA® in eine benutzerfreundliche, offlinefähige App überführen zu können?“ beantwortet werden kann. Zur Konkretisierung der Problematiken werden drei weiterführende Fragestellungen untersucht:

- Wie lässt sich eine MVC-Webanwendung in eine PWA überführen?
- Wie kann die Anwendung an MIA®-Datenbanken angebunden werden?
- Welche Daten können problemlos offline genutzt werden?

4.1 FA-1

Identifikator	FA-1
Beschreibung	Die SKM-Betreuer haben keinen festen Arbeitsplatz, daher muss es möglich sein, dass sie im Außendienst Transaktionen mit ihren Klienten mittels Mobilgerät durchführen können.
Zielsetzung	Das System soll eine grafische Benutzeroberfläche für Mobilgeräte besitzen. Deshalb muss ein passendes Design der einzelnen Oberflächen gefunden werden, damit der Betreuer die Funktionalitäten der Anwendung mit seinem Mobilgerät optimal nutzen kann.
Priorität	Hoch
Anmerkung	

4.2 FA-2

Identifikator	FA-2
Beschreibung	Die SKM-Kasse soll Ein- und Auszahlung an Klienten vornehmen können.
Zielsetzung	Der Betreuer soll über einen Zahlungsdialog Bargeld an seine Klienten ein und auszahlen können. Eine Transaktion besteht aus der Art (Ein- oder Auszahlung), einem Betrag, einem Kommentar zur Transaktion, einer Unterschrift des Klienten sowie bei Bedarf aus einem zusätzlichen Anhang. Nach Durchführung der Transaktion soll dem Betreuer mitgeteilt werden, ob diese erfolgreich war oder ob ein Fehler aufgetreten ist.
Priorität	Hoch
Anmerkung	Das Ein- und Auszahlen von Geld soll aus Sicherheitsgründen nur möglich sein, wenn die Anwendung über eine Internetverbindung verfügt. Der Klient darf innerhalb des Zahlungsdialogs keine direkte Möglichkeit haben, die Übersicht der Kontoinformationen zu sehen bzw. sie durch Drücken eines Buttons zu erreichen. Der Benutzer soll zwischen seinen Klienten sowie einer Liste mit allen Klienten wechseln können, um ggf. auch Transaktionen mit anderen Klienten durchführen zu können. Handelt es sich bei der Kasse um eine Standortkasse, muss ein PIN zur Autorisierung bzw. zum Abbruch der Transaktion eingegeben werden.

4.3 FA-3

Identifikator	FA-3
Beschreibung	Die SKM-Kasse muss es dem Betreuer ermöglichen, Bargeldeinlagen auf sein Konto zu akzeptieren.
Zielsetzung	Die Anwendung soll dem Betreuer der SKM eine Übersichtsseite mit seinen offenen Bargeldeinlagen anzeigen. Innerhalb dieser Übersichtsseite soll er diese akzeptieren und somit dem Betrag seinem Konto hinzufügen können.
Priorität	Hoch
Anmerkung	Da eine Bargeldeinlage in den MIA®-Datenbanken persistiert wird, darf sie nur durchgeführt werden, wenn das Endgerät eine Internetverbindung besitzt.

4.4 FA-4

Identifikator	FA-4
Beschreibung	Die SKM-Kasse soll die Anforderungen an eine Progressive Web App erfüllen.
Zielsetzung	Das Ziel ist die Offlinefähigkeit und die Installierbarkeit der Anwendung auf Mobilgeräten. Um die Benutzererfahrung zu verbessern, wird die Anwendung als Progressive Web App entwickelt werden. Dazu muss innerhalb der Anwendung ein Service Worker und ein Web App Manifest eingebunden werden.
Priorität	Mittel
Anmerkung	Die Anwendung soll auch in Browsern laufen, die Progressive Web Apps nicht unterstützen. Dazu muss darauf geachtet werden, dass der Service Worker und das Web App Manifest die Verwendbarkeit grundlegender Funktionen nicht einschränken, sollten diese von einem bestimmten Browser nicht unterstützt werden.

4.5 FA-5

Identifikator	FA-5
Beschreibung	Wiederverwendung der Authentifizierung, des Benutzermanagements und der MIA®-Persistenz Mechanismen aus dem MIA®-Framework.
Zielsetzung	Dadurch sollen Entwicklungsaufwände geschont werden, indem MIA®-Basisfunktionalitäten integriert werden, statt diese im PWA-Technologiestack neu zu entwickeln.
Priorität	Mittel
Anmerkung	Die Anmeldung wird über das bereits bestehenden MIA®-Anmeldesystem durchgeführt.

Kapitel 5

Entwurf

Dieses Kapitel dient der Vorstellung eines Entwurfs für die zu entwickelnde Progressive Web App. Die Anforderungen aus Kapitel 4 sollen im Entwurf erfüllt werden und später als Grundlage für die Entwicklung der Anwendung dienen. Hinführend zu diesem Entwurf werden im ersten Abschnitt allgemeine Überlegungen vorgestellt, wie sich eine Progressive Web App anhand des MVC-Patterns entwickeln lässt. Im folgenden Abschnitt wird auf die weitere Vorgehensweise in der Arbeit eingegangen sowie Gründe für die Entscheidung für einen spezifischen Lösungsansatz genannt.

5.1 Allgemeine Überlegungen

Innerhalb dieses Abschnitts werden Überlegungen zur optimalen Progressiven Web App, die das MVC-Pattern verwendet, angestellt. In diesem Zusammenhang muss sowohl auf die aktuell geltenden Eigenschaften und technischen Merkmale einer Progressive Web App eingegangen werden als auch auf den aktuellen Stand der Technik.

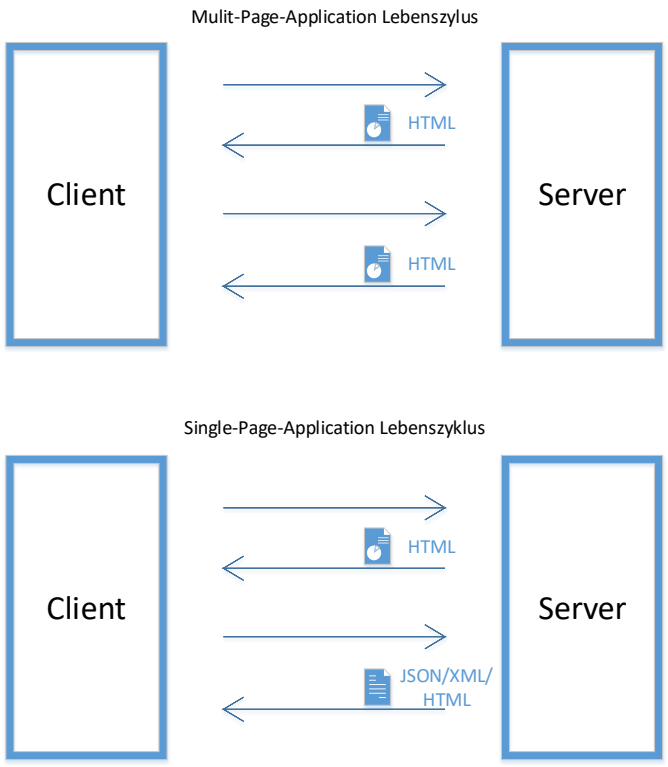

Abbildung 5.1: Gegenüberstellung Multi-Page-Application und Single-Page-Application (Quelle: Eigene Darstellung)

Wie in Abschnitt 3.5 beschrieben, gibt es verschiedene architektonische Ansätze für die Entwicklung einer Progressive Web App. Welcher dieser Ansätze bei der Entwicklung einer Progressive Web App gewählt wird, hängt von den zu erwartenden Benutzern als auch von dem Einsatzzweck der Anwendung ab. Bei den Benutzern ist zu beachten, welchen Browser sie verwenden, um die Anwendung aufzurufen. Da im Browser der JavaScript Code der Anwendung ausgeführt wird, kann es bei älteren Webbrowsern durch fehlende Performance, mangelnde Kompatibilität und mangelnder Benutzererfahrung zum Verlust potenzieller Benutzer kommen. Deshalb sprechen aktuelle Browser für CSR und veraltete Browser für SSR. Ebenfalls nicht zu vernachlässigen ist der Einsatzzweck der Anwendung. CSR sollte für Anwendungen genutzt werden, welche eine hohe Benutzerinteraktion fordern und mit vielen dynamischen Inhalten arbeiten, die einer laufenden Veränderung unterliegen. Dabei dauert das anfängliche Laden einmal länger, da nach Laden der Basisinhalte wie HTML oder CSS nochmal ein Request für dynamische Inhalte gesendet werden muss. Jedoch bietet CSR im weiteren Verlauf eine bessere Performance, da nicht jede Seite neu angefordert wird, sondern nur die sich ändernden Daten, siehe Abbildung 5.1 „Single-Page-Application Lebenszyklus“. SSR hingegen wird sinnvollerweise für textbasierte Anwendungen verwendet, die wenig Benutzerinteraktion und kaum dynamische Inhalte anbieten. Der Vorteil drückt sich darin aus, dass die statischen Inhalte schnell auf dem Server gerendert werden können und die vollständige Seite an den Client gesendet wird,

siehe Abbildung 5.1 Multi-Page-Application Lebenszyklus. Um die Vorteile der beiden architektonischen Ansätze sinnvoll kombinieren zu können, lässt sich auch ein hybrider Ansatz aus SSR und CSR verwenden. So können die Server durch CSR in bestimmten Bereichen entlastet werden, wohingegen bei textuellen Inhalten auf SSR zurückgegriffen wird, um den Benutzer eine bestmögliche Performance zu bieten.

Im Rahmen meiner Abschlussarbeit habe ich mich entschieden, den reinen Single-Page Ansatz für interaktive Anwendungen mit dynamischem Inhalt zu verwerfen und mich auf einen hybriden Ansatz zu konzentrieren. Das ist unter anderem der Tatsache geschuldet, dass hybride Ansätze flexibler auf bestimmte Anforderungen reagieren können als auch dem Umstand, dass MIA® das MVC-Pattern verwendet und somit ein reiner Single-Page Ansatz schwer realisierbar ist.

5.2 Entwurf von Mockups und Prozessen anhand der Anforderungen

In diesem Abschnitt wird ein Entwurf zur Entwicklung einer Progressive Web App für eine Kassenanwendung vorgestellt, welche an das MIA®-Framework angebunden ist. Dafür wird für jede in Kapitel 5 beschriebene funktionale Anforderung anhand der Beschreibung, Zielsetzung und Anmerkung ein Konzept entwickelt. Die dazugehörigen Programmablaufpläne werden nach DIN 66 001 erstellt.

5.2.1 Konzept FA-1

In diesem Abschnitt werde ich auf die Umsetzung der Anforderung FA-1: „Das System soll eine grafische Benutzeroberfläche für Mobilgeräte besitzen.“ eingehen. FA-1 beschreibt die Benutzeroberfläche der Hauptseite für die zu entwickelnde Anwendung.

Mobile-First-Ansatz

Aufgrund der Anforderung „grafische Benutzeroberfläche für Mobilgeräte“ wird ein Mobile-First Ansatz gewählt. Diese Entscheidung muss jedoch kritisch hinterfragt werden, da dabei das Design für größere Geräte und normale Desktopansichten im ersten Schritt vernachlässigt wird. Da die Anwendung aufgrund des Außendienstes der Betreuer fast ausschließlich auf Smartphones und Tablets genutzt wird, hat die Entwicklung einer passenden Oberfläche für diese Geräte Priorität und rechtfertigt damit die Verwendung dieser Strategie. Für das Konzept bedeutet das vor allem Beschränkung auf das Wesentliche, Design-Entwürfe werden auf das Smartphone-Display angepasst sowie keine großen Bilder und unnötigen Funktionen.

Programmverhalten bei Benutzerinteraktion

Da bei der Entwicklung dieser Progressive Web App das MVC-Pattern zugrunde gelegt wird, ist bei der Konzeption der Benutzeroberfläche darauf zu achten, wie mit der Interaktion des Nutzers umgegangen wird. Interaktionen, die nur die Darstellung von

Daten anpassen, können problemlos über einen Request realisiert werden, der nur die benötigten Daten anfordert. Diese werden dann clientseitig in die Benutzeroberfläche gerendert. Dadurch wird anstatt der kompletten Seite nur der benötigte Inhalt neu angefordert. Das fördert die User Experience und erfüllt die Eigenschaft App-like, siehe Abschnitt 3.1.4, einer Progressive Web App, da die Steuerung innerhalb der Anwendung der einer nativen App ähnelt. Die Funktionalitäten, die jedoch eine Operation in einer MIA®-Datenbank durchführen und dadurch zu einer Veränderung am Model führen haben zur Folge, dass die View mit dem dargestellten Model aktualisiert werden muss. Dies wird mit einem erneuten Request der Seite realisiert, welche serverseitig gerendert wird. Dieses Vorgehen erfüllt die Progressive Web App Anforderung Fresh (siehe 3.1.5), laut der stets aktuelle Daten anzuzeigen sind. Dabei ist darauf zu achten, dass die Menge der angeforderten Daten gering ist, um die Ladezeiten der Seite zu minimieren und dem Benutzer wieder eine schnelle Interaktion mit der aktualisierten Seite zu ermöglichen.

Mockup

Folgende Darstellung Abbildung 5.2 illustriert ein Mockup, welches nach dem Mobile-First-Prinzip erstellt wurde und damit ein minimalistisches Design sowie nur die notwendigen Funktionalitäten beinhaltet. Ebenfalls wurde beim Erstellen darauf geachtet, dass der Benutzer einen Großteil der Funktionalitäten innerhalb einer View erledigen kann, um ein Neuladen der Seite zu verhindern.

Da das MVC-Pattern zugrunde gelegt wird, muss bei der Konzeption der Benutzeroberfläche darauf geachtet werden, wie mit der Interaktion des Nutzers umgegangen wird. Die Startseite Abbildung 5.2 besteht aus drei Sektionen, einer Kopfzeile, einem Hauptteil sowie einer Fußzeile. In der Kopfzeile befindet sich ein Icon zum Öffnen eines Dialogs für ausstehende Bargeldeinlagen, Informationen zum Benutzer und der aktuelle Kontostand. Innerhalb des Hauptteils der Startseite werden dem Benutzer seine zugewiesenen Klienten in einer Liste angezeigt. Über dieser Liste befinden sich ein Edit-Button sowie ein Suchen-Feld. Über den Edit-Button lässt sich ein Dropdown-Menü öffnen, verbunden mit der Option, sich alle in der Datenbank vorhandenen Klienten anzeigen lassen zu können. Innerhalb des Suchen-Felds soll der Betreuer clientseitig nach einzelnen Klienten filtern können. Durch das Klicken des Pfeil-Icons öffnet sich zum ausgewählten Klienten ein Feld mit Informationen sowie einem Button, um Ein- und Auszahlungen vornehmen zu können. In der Fußzeile der Startseite befindet sich der Name und das Logo des Kunden sowie ein Copyright-Vermerk.

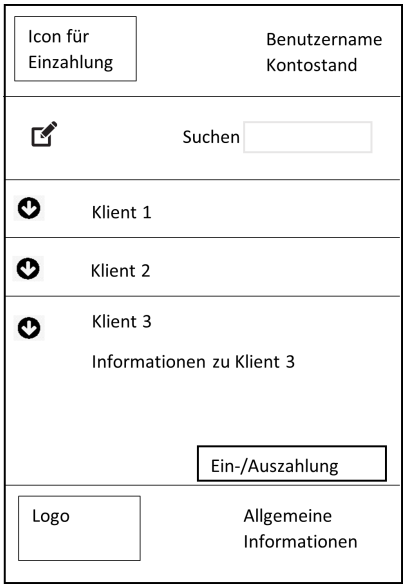

Abbildung 5.2: Mockup der Startseite
(Quelle: Eigene Darstellung)

5.2.2 Konzept FA-2

Ziel der FA-2 ist es, dem Betreuer die Möglichkeit zu geben, innerhalb der Anwendung Ein- und Auszahlungen an Klienten vorzunehmen. In diesem Abschnitt wird zuerst der Prozess einer Transaktion entworfen und anschließend die dazugehörige Benutzeroberfläche als Mockup designt.

Prozess Transaktion

Der Prozess einer Transaktion besteht aus verschiedenen Schritten. Ist der Benutzer dem System gegenüber authentifiziert, kann er aus einer Liste mit Klienten den entsprechenden Zahlungsempfänger oder -leistenden auswählen. Dadurch öffnet sich ein Eingabefeld, in welches der Benutzer die Details einer Transaktion einzutragen hat. Eine normale Transaktion besteht aus der Art der Zahlung, einem Betrag, einem Kommentar zur Transaktion, einer Unterschrift des Klienten sowie bei Bedarf aus einem zusätzlichen Anhang. Handelt es sich bei der Kasse um eine Standortkasse, besteht die Notwendigkeit einer weiteren Autorisierung. Eine Standortkasse läuft auf einem Mobilgerät in einem SKM-Büro. In diesem Büro arbeiten mehrere Betreuer und haben Zugriff auf die gleiche Kasse. Damit innerhalb dieser Kasse einzelne Transaktionen bestimmten Betreuern zugeordnet werden können, müssen diese sich durch eine PIN gegenüber dem System identifizieren. Dazu muss das System in der Lage sein, unterschiedliche Kassentypen zu identifizieren, um bei Bedarf ein anderes Programmverhalten zu zeigen. Bevor die Transaktion durchgeführt werden kann, sind die Eingaben des Benutzers zu validieren. Sind sie valide, wird die Buchung an den Server übergeben und in der Datenbank gebucht. Im Anschluss erhält der Benutzer eine Transaktionsbestätigung. Sind die Eingaben fehlerhaft, wird der Benutzer darauf hingewiesen. Er hat die Möglichkeit diese auszubessern und die Buchung erneut

durchzuführen oder den gesamten Zahlungsprozess abzubrechen. Im Falle eines Abbruchs wird der Benutzer auf eine neue Seite geführt, die auf den Abbruch hinweist. Dieser Ablauf dient dem Datenschutz und hat zur Folge, dass Klienten nicht direkt Informationen anderer Klienten einsehen können (Siehe Abbildung 5.3).

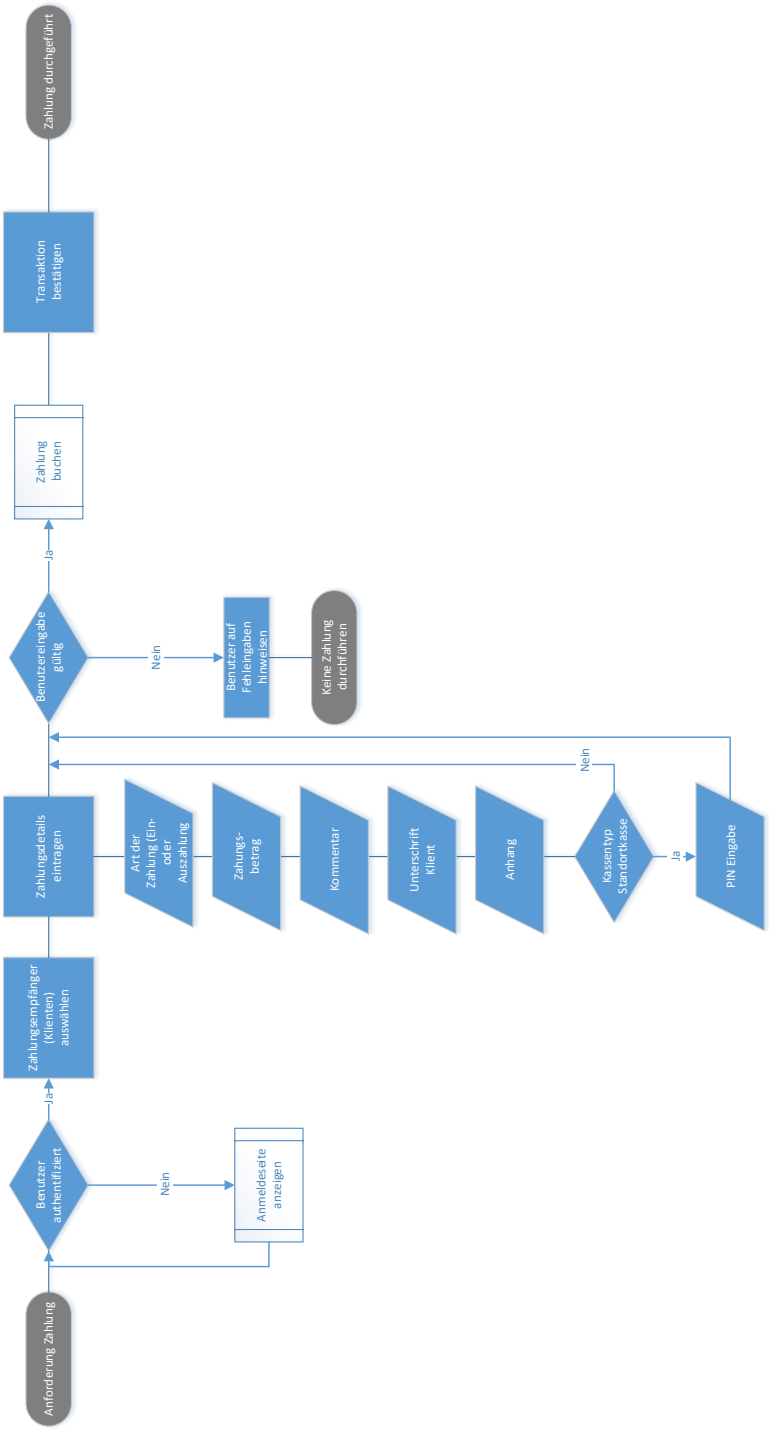

Abbildung 5.3: Prozess zur Durchführung einer Transaktion
(Quelle: Eigene Darstellung)

Mockups

Die in diesem Abschnitt entworfenen Mockups sollen in der SKM-Kasse den Prozess einer Transaktion darstellen, die von den Kassentypen Handkasse (Siehe Abbildung 5.4) und Standortkasse (Siehe Abbildung 5.5) durchgeführt werden. Aufgrund der in FA-1 beschriebenen Notwendigkeit eines Mobile-First-Ansatzes muss auch bei der Konzeption dieser Mockups darauf geachtet werden, diesen Ansatz umzusetzen. Durch das Drücken des Buttons Ein- und Auszahlung (Abbildung 5.2) wird je nach Kassenart einer der folgenden Zahlungsdialoge innerhalb der Übersichtsseite geöffnet:

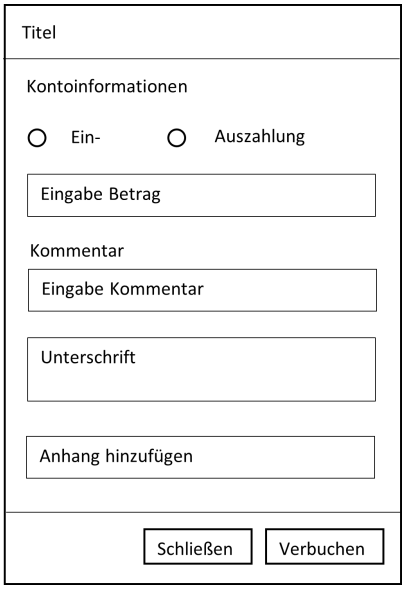

Abbildung 5.4: Mockup Transaktion Handkasse ohne PIN (Quelle: Eigene Darstellung)

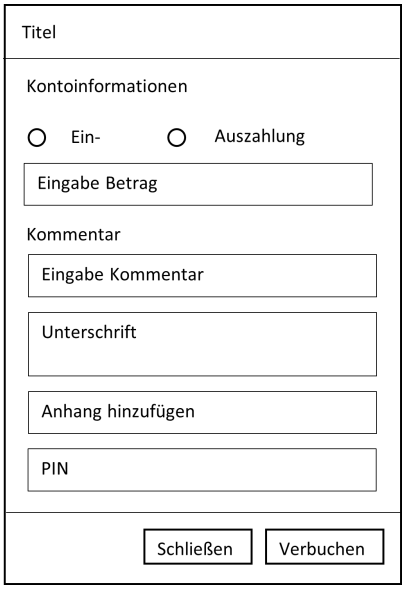

Abbildung 5.5: Mockup Transaktion Standortkasse mit PIN (Quelle: Eigene Darstellung)

Der Zahlungsdialog zeigt an erster Stelle Kontoinformationen des ausgewählten Klienten an. Danach wird in verschiedenen Eingabefeldern die Art der Zahlung, der Betrag und ein Kommentar eingegeben. Anschließend bestätigt der Klient mit seiner Unterschrift diese Zahlungsinformationen. Im Anhang hat der Klient die Möglichkeit Belege oder Fotos hinzuzufügen, um die Notwendigkeit einer Transaktion zu untermauern.

5.2.3 Konzept FA-3

In diesem Abschnitt werde ich kurz auf die geplante Umsetzung der Anforderung FA-3 eingehen. FA-3 beschreibt das Akzeptieren von Bargeldeinlage seitens des Betreuers, um Geld auf sein Konto einzuzahlen. Im ersten Schritt wird für den Prozessablauf ein Konzept erstellt, und im Anschluss daran erfolgt die Entwicklung der Mockups.

Prozess Umbuchung

Der Prozess einer Umbuchung beginnt damit, dass der Benutzer gegenüber dem System authentifiziert ist. Ist dies der Fall werden offene Bargeldeinlagen in der Daten-

bank abgefragt. Sind welche vorhanden, wird dies dem Benutzer angezeigt, und er kann sie im nächsten Schritt akzeptieren. Das führt dazu, dass der in der Buchung genannte Geldbetrag dem Konto des Betreuers hinzugefügt wird und die dazugehörigen Daten in der Datenbank aktualisiert werden. Sind keine Bargeldeinlagen vorhanden, soll das dem Betreuer angezeigt und der Zugang zur Oberfläche für Umbuchungen gesperrt werden (Siehe Abbildung 5.6).

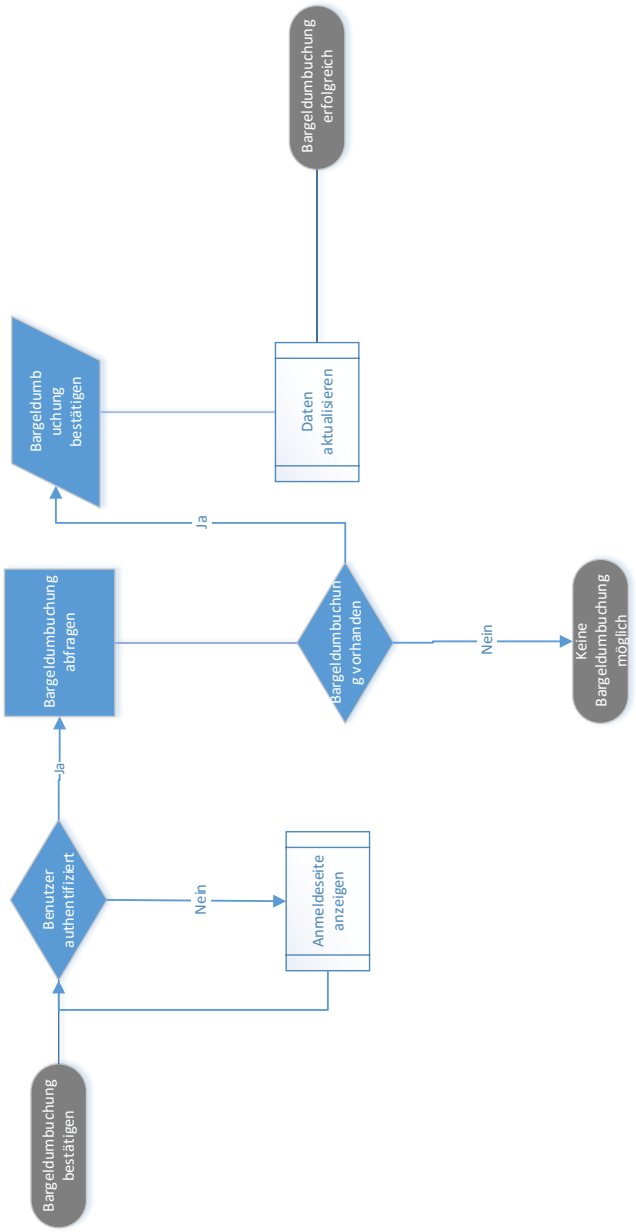

Abbildung 5.6: Prozess zur Durchführung einer Bargeldeinlage (Quelle: Eigene Darstellung)

Mockup

Das Mockup zeigt die geplante Oberfläche für die Anforderung FA-3. Dazu soll innerhalb der View ein Modal clientseitig gerendert werden, um das Laden einer neuen Seite vom Server zu verhindern. Innerhalb des Dialogs gibt es eine Liste mit offenen Buchungen. Zu jedem Element dieser Liste werden unter anderem der Überweiser, der Betrag sowie das Datum angezeigt. Durch das Klicken des Häkchens der jeweiligen Bargeldeinlagen kann der Betreuer die Entgegennahme des Geldes bestätigen, und es wird seinem Konto hinzugefügt, siehe Abbildung 5.7.

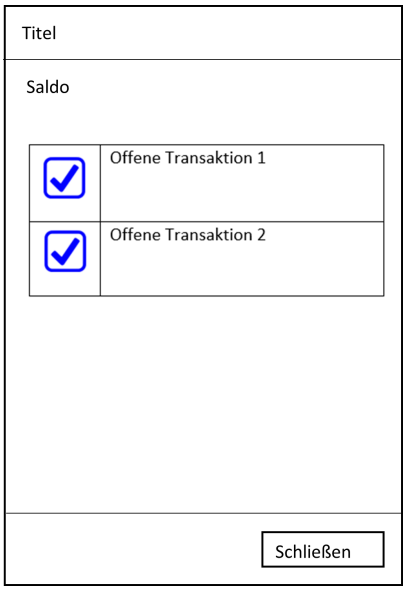

Abbildung 5.7: Mockup zum Anzeigen und Bestätigen von Umbuchungen (Quelle: Eigene Darstellung)

5.2.4 Konzept FA-4

In diesem Abschnitt werde ich auf die geplante Umsetzung der Anforderung FA-4: „Die SKM-Handkasse soll die Anforderungen an eine Progressive Web App erfüllen.“ eingehen. Zwingend benötigte technische Merkmale sind ein Service Worker und ein Web App Manifest sowie die Übertragung durch das HTTPS-Protokoll. Das Ziel ist die Installierbarkeit der Anwendung sowie die Offlinefähigkeit. Der Fokus liegt vor allem auf der Offlinefähigkeit, da SKM-Betreuer meist im Außendienst tätig sind und deshalb eine beständige Internetverbindung nicht gewährleistet werden kann. Dafür wird in den nächsten Schritten ein Konzept ausgearbeitet, um die anderen Anforderungen mit dieser in Einklang zu bringen und verschiedene Caching-Strategien in Bezug auf die benötigten Inhalte zu evaluieren.

Anpassung der Prozesse Zahlung und Umbuchung

Nachfolgender Abschnitt beschreibt die Durchführung von Transaktionen im Falle einer fehlenden Internetverbindung. Grundsätzlich ist es möglich Daten in einem Browser durch eine IndexedDB zu speichern und zu bearbeiten. Nach Rojas (2019) ist

eine IndexedDB ein Datenbanksystem, welches in Browsern zur Verfügung steht und sich dazu eignet große Datenmengen zu speichern. Somit wäre es technisch gesehen möglich, offline Zahlungen und Bargeldbuchungen durchzuführen und in einer IndexedDB zwischenzuspeichern. Sobald eine Verbindung zum Internet besteht, könnten die Daten aus der IndexedDB an eine MIA®-Datenbank übertragen werden. Allerdings wirft dieses Vorgehen einige Probleme auf. Erstens ist der Kontostand des Klienten nicht aktuell, was zu Mehrfachauszahlungen durch verschiedene SKM-Betreuer führen könnte. Zweitens ist der Missbrauch durch den Betreuer nicht auszuschließen, da die Datenbank innerhalb des Browsers bearbeitet und gelöscht werden kann. Folglich ist die Datenintegrität nicht gewährleistet.

Zusätzlich existiert je nach Browser eine Limitierung hinsichtlich des Speicherplatzes. Der Entscheidungsalgorithmus was nach Erreichen des Speicherplatzlimits zu löschen ist, unterscheidet sich von Browser von Browser (MDN, 2022b). Dadurch können Daten zu getätigten Zahlungen und Bargeldeinlagen unbemerkt verloren gehen.

Aufgrund der oben genannten Probleme dürfen Zahlungen und Bargeldeinlagen nur durchgeführt werden, wenn eine Internetverbindung vorhanden ist, um sie direkt in einer MIA®-Datenbank zu persistieren. Dafür müssen die Prozesse für Bargeldeinlagen (Siehe Abbildung 5.8) und Zahlungen (Siehe Abbildung 5.9) dahingehend angepasst werden, dass diese nur bei einer bestehenden Internetverbindung möglich sind.

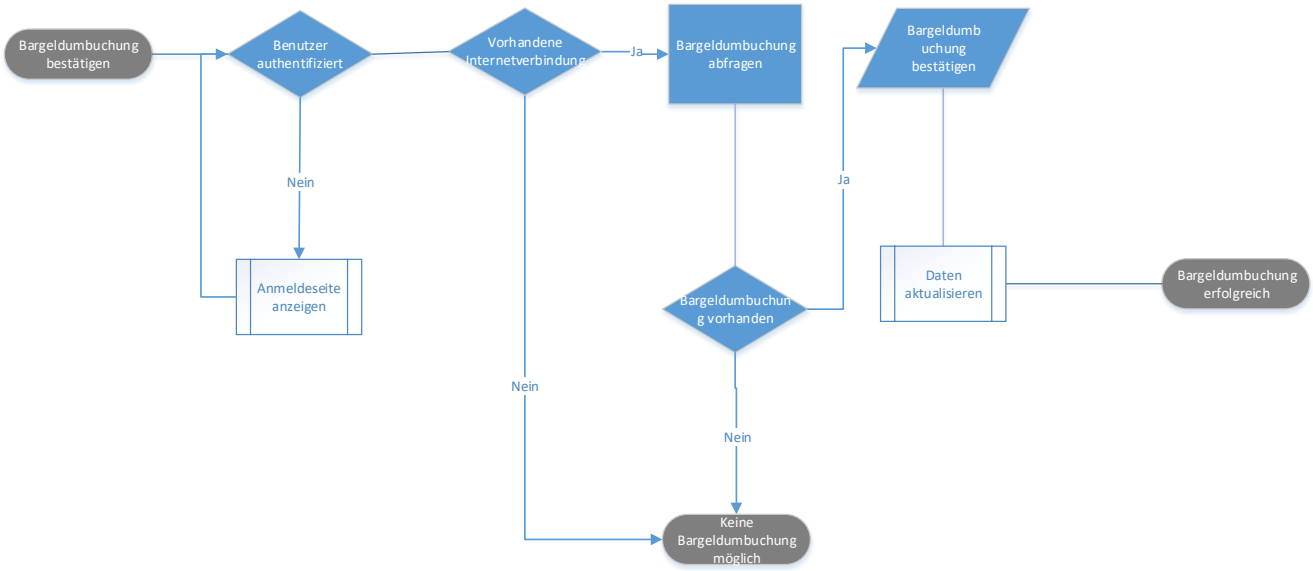

Abbildung 5.8: Anpassung des Prozess Bargeldeinlage
(Quelle: Eigene Darstellung)

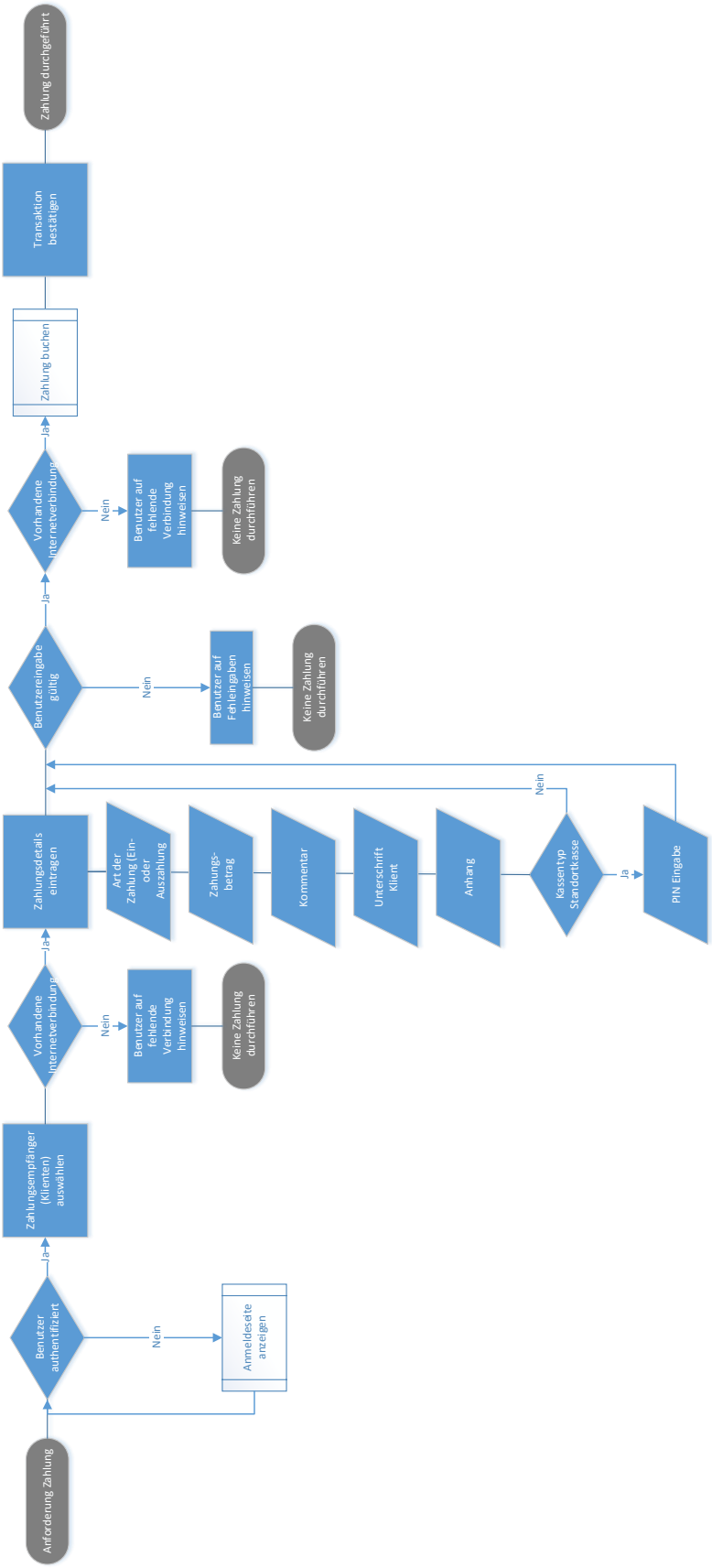

Abbildung 5.9: Anpassung des Prozess Transaktion
(Quelle: Eigene Darstellung)

Offlinefähigkeit

Bei der Überführung einer MVC-Webanwendung in eine Progressive Web App können verschiedene Caching Strategien verwendet werden, um die Offlinefähigkeit der Anwendung sicherzustellen. Nachdem im vorherigen Kapitel beschrieben wurde, warum Zahlungen und Bargeldeinlagen nur online durchgeführt werden dürfen, erläutert dieser Abschnitt, wie sich die Offlinefähigkeit der Anwendung darstellt und warum für den jeweiligen Fall eine spezifische Caching-Strategie verwendet wird. Befindet sich die Anwendung im Offlinemodus, hat der SKM-Betreuer die Möglichkeit, sich seine Klienten sowie alle Klienten der SKM anzeigen zu lassen. Das bedeutet eine Oberfläche mit sich ändernden anzuzeigenden Daten. Nach der Installation des Service Workers wird eine Reihe von Dateien automatisch im Cache gespeichert. Dadurch ist die Anwendung nach erstmaligen Aufrufen sofort offlinefähig. Im weiteren Verlauf wird für statische Dateien wie HTML, CSS, JavaScript, Schriftarten, oder Bilder auf die Strategie: „Cache first, fall back to Network“ zurückgegriffen. Das hat den Vorteil, dass sie bei Bedarf sofort aus dem Cache zurückgegeben werden können ohne, dass eine Anfrage an einen Server gesendet werden muss. Sollte der Inhalt nicht im Cache vorhanden sein, wird die Ressource am Server angefragt und im Cache gespeichert. Das führt zu schnelleren Ladezeiten der Seite und erhöht die Performance. Für dynamische Inhalte und Dateien wird die Strategie: „Network falling back to cache“ verwendet. Dabei werden Dateien und Inhalte primär beim Server angefragt, um die jeweils aktuelle Version zu erhalten. Sie werden im Cache abgelegt und lassen sich verwenden, wenn die Anwendung keine Internetverbindung hat. Dadurch benutzt der SKM-Betreuer immer die aktuellen Inhalte und sieht bei fehlender Verbindung zumindest den letzten Stand.

Installierbarkeit

Die Installierbarkeit der Anwendung wird durch ein Web App Manifest sichergestellt. Innerhalb dieser Datei wird die Start-URL festgelegt, um beim Öffnen der App auf die Anmeldeseite oder direkt auf die Hauptseite der App zu gelangen, sofern die Benutzerauthentifizierung erfolgreich war. Des Weiteren werden Icons definiert, um die App auf dem Endgerät darzustellen.

5.2.5 Konzept FA-5

Die Konzeption der Anforderung zur Wiederverwendung der Authentifizierung, des Benutzermanagements und der MIA®-Persistenz Mechanismen aus dem MIA®-Framework wird im folgenden Abschnitt thematisiert. Bevor dem Nutzer der Zugriff zur App gewährt werden kann, muss er sich anmelden. Das geschieht, indem sich der Nutzer innerhalb eines externen Prozesses durch die Eingabe seiner Anmeldeinformationen in MIA® authentifiziert. Ist die Anmeldung erfolgreich, wird dem MIA®-Nutzer seine entsprechende Kasse zugeordnet und der Benutzer wird in die Anwendung weitergeleitet. Während der Verwendung der Anwendung benötigt der Benutzer lesenden und schreibenden Zugriff auf die MIA®-Datenbanken, um seine Daten bearbeiten zu können. Der Zugriff auf die Datenbank erfolgt über Stored Procedures für welche der

Benutzer eine Ausführberechtigung braucht. Schließt der Benutzer die Anwendung oder läuft die Gültigkeit des MIA®-Anmeldetokens ab, wird die Verbindung getrennt und der Benutzer muss sich erneut anmelden (Siehe Abbildung 5.10).

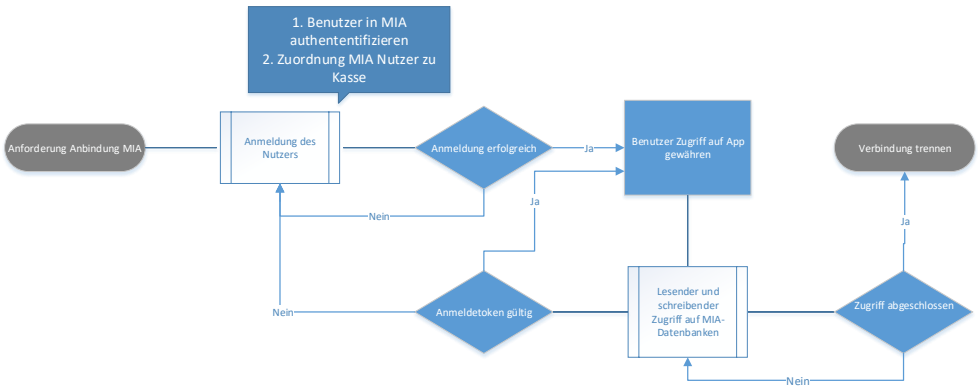

Abbildung 5.10: Einbettung der Anwendung in den Kontext MIA®
(Quelle: Eigene Darstellung)

5.3 Entwurf der MVC-Architektur

In diesem Abschnitt wird ein Entwurf für die Views, Models und den Controller vorgestellt. Dabei werden die oben vorgestellten Mockups und Prozesse verwendet und in eine MVC-Architektur überführt.

5.3.1 Entwurf der View

Die Anwendung besteht aus einer Reihe verschiedener Oberflächen zur Visualisierung von Informationen. Um den Benutzer ein einheitliches Design zu bieten, werden alle Fenster innerhalb der *ModuleLayout View* angezeigt. Hier wird das Grundgerüst der Oberfläche definiert. Durch das einheitliche Design, in welchem die verschiedenen Views angezeigt werden, erhält die Anwendung die Eigenschaft App-Like. Innerhalb dieses Grundgerüsts werden folgende Views dargestellt:

- *Index View:*
  Ist die Übersichtseite und wurde in Abschnitt 5.2.1 erstellt.

- *PaymentConfirm View:*
  Nach Übermittlung einer Transaktion wird der Benutzer auf dieser neuen Seite darauf hingewiesen, ob diese erfolgreich oder nicht erfolgreich war. Durch das Aufrufen einer neuen Seite wird sichergestellt, dass eine Internetverbindung vorhanden ist und die Transkation in der Datenbank gespeichert wurde, da die Daten der Seite vom Server geladen werden.
- *Error View:*
  Sollte innerhalb der Anwendung ein funktionaler Fehler auftreten, wird der Benutzer auf diese Seite geleitet ohne Absturz der Anwendung.
- *PaymentQuit View:*
  Diese View dient dem Datenschutz. Da Klienten von SKM-Betreuern nicht die Möglichkeit haben sollen, Daten anderer Klienten direkt einzusehen, wird bei Abbruch eines Zahlungsdialogs auf eine neue Seite verwiesen.
- *Offline View:*
  Sollte es aus aufgrund eines Fehler nicht möglich sein, Offline die angeforderte Seite darzustellen, wird der Benutzer auf die Offline-Seite verwiesen. Diese informiert ihn darüber, dass aktuell keine Internetverbindung besteht und die geforderten Informationen nicht angezeigt werden können.

Des Weiteren wurde aus Usability-Gründen die Entscheidung getroffen, die *OpenTransactions View* für offene Bargeldeinlagen sowie die *Payment View* als Modal innerhalb der *Index View*, siehe Abbildung 5.11, darzustellen.

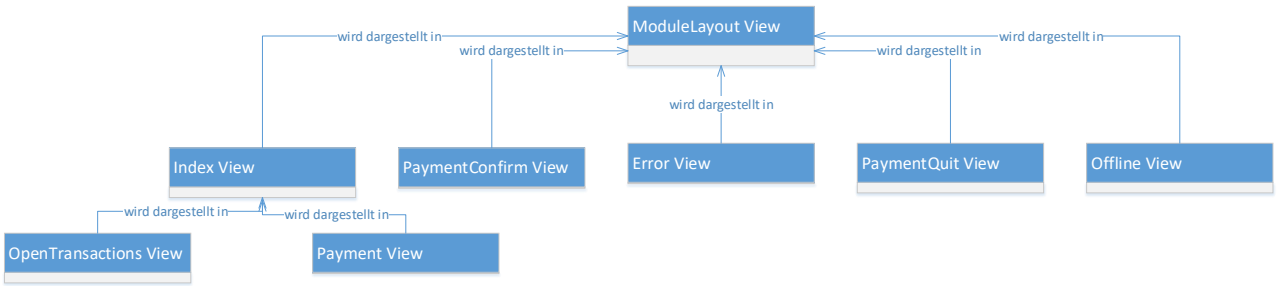

Abbildung 5.11: Entwurf der View
(Quelle: Eigene Darstellung)

Durch das Öffnen innerhalb der Index View (Startseite) wird es vermieden, den Zahlungsdialog für jeden Klienten sowie offene Bargeldeinlagen vom Server einzeln anzufordern. Hiermit werden die Eigenschaften einer Progressive Web App mit folgenden Merkmalen realisiert. Sie ist App-like, da sich die Anwendung wie eine native App anfüllt, sie ist Responsive, da die vorhanden Bildschirmabmessungen bestmöglich genutzt werden und sie ist Connectivity independent, da sich der Zahlungsdialog auch bei schlechter Internetverbindung schnell öffnen lässt.

5.3.2 Entwurf des Models

Das Model der Anwendung besteht aus fünf verschiedenen Klassen (Siehe Abbildung 5.12).

- *PaymentViewModel:*
  Dieses Model beinhaltet die Daten einer Zahlung. Dazu gehören z.B. Betreuer, Betreuter, Konto, Betrag, Kommentar und Art der Zahlung. Jeder Zahlung ist ein Betreuer zugeordnet.
- *ClientViewModel:*
  Innerhalb dieses Models werden die Daten eines Betreuten gespeichert. Dazu gehört sein Account, Name, Kontostand usw.
- *WalletViewModel:*
  Das Model dient der Speicherung und Bearbeitung des Wallets. Ein Wallet besteht aus einem Namen, Art, offenen Bargeldeinlagen sowie dem Kontostand des SKM-Betreuers.
- *WalletTransactionViewModel:*
  Das Model beinhaltet alle notwendigen Informationen einer Bargeldeinlagen. Dazu zählen Datum, Betrag, Durchführbarkeit usw.
- *ErrorViewModel:*
  Dieses Model enthält Informationen zu möglichen Fehlern, zum Beispiel einen Fehlercode, eine Fehlernachricht sowie eine dazugehörige URL.

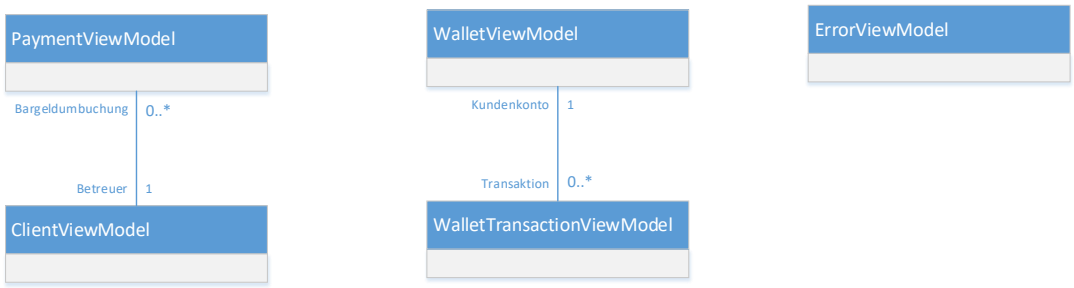

Abbildung 5.12: Entwurf des Models
(Quelle: Eigene Darstellung)

5.3.3 Entwurf Controller

Der Controller muss aufgrund der Entscheidung für einen hybriden Ansatz auf Anfrage sowohl server-seitig gerenderte Webseiten als auch durch ajax-Requests angeforderte Datensätze zurückgeben können. Dazu müssen innerhalb des Controllers Methoden bereitgestellt werden, die

- Daten weiterleiten, um sie in der Datenbank zu speichern,
- Daten aus der Datenbank anfordern,
- einzelne Seiten anhand der Daten aus der Datenbank rendern.

Um die Komplexität der Anwendung zu verringern, werden die benötigten Methoden alle innerhalb eines Controllers definiert und implementiert.

5.4 Fazit

Innerhalb dieses Kapitels wurde im Abschnitt 5.1 festgelegt, dass ein reiner Single-Page-Ansatz, aufgrund der vorhandenen Anforderungen und Gegebenheiten nicht sinnvoll ist. Vor diesem Hintergrund wird in Abschnitt 5.2 ein Konzept entworfen, das mithilfe eines hybriden Ansatzes eine MVC-Webanwendung in eine Progressive Web App überführt. Abschnitt 5.3 stellt die Architektur gemäß den Anforderungen des zu entwickelnden Systems dar. Die existierende MVC-Webanwendung wird je nach Bedarf Client-Side-Rendering oder Server-Side-Rendering einsetzen, um die Anforderungen an eine Progressive Web App erfüllen zu können. In Kapitel 6 wird dieses Konzept implementiert und auf mögliche Probleme und Lösungen eingegangen.

Kapitel 6

Implementierung

Das im vorangegangenen Kapitel vorgestellte Konzept wird in den nächsten Abschnitten dazu genutzt, eine konkrete Lösung vorzustellen. Dafür wird zunächst definiert, welche Technologien, Bibliotheken und Frameworks bei der Erstellung der Anwendung zum Einsatz kommen. Im zweiten Teil erfolgt die Entwicklung einer MVC-Webanwendung, welche die Anforderungen FA-1, FA-2, FA-3 und FA-5 zu erfüllen hat. Anschließend wird im darauffolgenden Abschnitt die Anforderung FA-4, die zur Vollständigkeit des entwickelten Konzepts benötigt wird, realisiert, dabei wird die Anwendung in eine Progressive Web App überführt.

6.1 Verwendete Technologien, Bibliotheken und Frameworks

Für die Implementierung der Anwendung wird das für MIA® verwendete und von Microsoft entwickelte .NET Core 3.1 Framework eingesetzt. Dabei kommen die in Tabelle 6.1 beschriebenen Technologien zum Einsatz.

Technologie	Einsatzzweck
TypeScript	Sie dient zur Entwicklung des Service Workers sowie der Skripte für die Webseite. TypeScript-Dateien werden durch einen vorhandenen Gulp-Prozess in JavaScript-Dateien kompiliert und laufen im Browser des Benutzers.
CSHTML	Es ist ein Dateiformat, das von Razor für die Erstellung von Webseiten genutzt wird. Dabei lassen sich durch eine spezielle Syntax HTML und C# zusammen verwenden (FileTypes, 2022). Es wird im Projekt zur serverseitigen Erstellung der Benutzeroberflächen eingesetzt.
Sass	Sass ermöglicht es Variablen, Regeln und Funktionen in einer CSS-kompatiblen Syntax, zur Gestaltung der Oberfläche zu verwenden (Sass, 2022). Sass wird wieder in CSS kompiliert.
C#	C# dient zur Implementierung der Models, des Controllers, des Services und des Data Access Layers

Tabelle 6.1: Verwendete Technologien

Für die Entwicklung der Benutzeroberfläche werden die beiden Bibliotheken jQuery und drawingboard.js verwendet. jQuery ist eine freie JavaScript-Bibliothek, die Funktionen zur DOM-Manipulation und Navigation bereitstellt. Für jQuery wird zudem das Plugin Datatables verwendet. Dabei handelt es sich um ein Tool, das auf Grundlage der progressiven Erweiterung aufgebaut ist und erweiterte Funktionalitäten für HTML-Tabellen anbietet („DataTables: Table plug-in for jQuery“, 2022). drawingboard.js ist ebenfalls eine freie JavaScript-Bibliothek, welche jQuery benötigt. Mit ihr lassen sich Zeichenflächen in eine Oberfläche integrieren sowie Funktionen zum Auslesen der Zeichenflächen nutzen (Pelletier, 2013). Des Weiteren wird für die Umsetzung der Benutzeroberfläche das Frontend-CSS-Framework Bootstrap eingesetzt. Dadurch lässt sich ein responsives Design gewährleisten, womit sichergestellt ist, dass die Anwendung auf allen Endgeräten passend dargestellt wird.

6.2 Implementierung der Kassenanwendung

In diesem Abschnitt wird zuerst der Feinentwurf der Anwendung vorgestellt. Anschließend wird erklärt, wie die einzelnen Komponenten umgesetzt werden und auf welcher Grundlage diese Entscheidungen getroffen wurden.

6.2.1 Feinentwurf der Architektur

Zunächst erfolgt die Darstellung der Architektur für die zu entwickelnde Anwendung auf Grundlage der bereits beschriebenen Anforderungen. Dazu wird das System zuerst in unterschiedliche Pakete gegliedert. Folgende Pakete werden innerhalb der Architektur umgesetzt:

- View
- Controller
- Model
- Service
- Data Access Layer

Jedes dieser Pakete dient einer bestimmten Aufgabe und besitzt eine Reihe von Klassen mit Attributen und Methoden, siehe Abbildung 6.1. Die Entscheidung fiel zugunsten eines Services, der innerhalb der Anwendung immer und überall verfügbar ist, um Daten, die in der Anwendung öfter benötigt werden, einheitlich durch diesen bereitstellen zu können. Das Paket Data Access Layer steuert die Verbindung der Datenbank mit der Anwendung. Dadurch wird der Controller entlastet und die Architektur des Systems bleibt übersichtlicher durch die Trennung von Zuständigkeiten. Die Eigenschaften und Aufgaben von Views, Controller und Models wurden bereits im Grundlagenkapitel Abschnitt 2.2 erläutert.

Abbildung 6.1: Feinentwurf der Architektur
(Quelle: Eigene Darstellung)

6.2.2 View

Die Implementierung der einzelnen Views der Benutzeroberfläche findet unter Beachtung der aufgestellten Anforderungen sowie der im Entwurf erstellen Mockups statt. Die Entwicklung erfolgt mit Hilfe von C#, HTML, TypeScript und Sass. Um die im Entwurf erstellte Struktur der Views beizubehalten, werden die einzelnen CSHTML-Dateien im Projekt folgendermaßen gegliedert (Siehe Abbildung 6.2). Im Ordner *CFS/* werden alle Oberflächen abgelegt, die innerhalb des Grundgerüst der Anwendung gezeigt werden sollen. Die Dateien, die innerhalb dieses Ordners mit einen Unterstrich beginnen, sind Partial Views, die innerhalb der *Index.cshtml* aufgerufen werden. Im Ordner *Shared/* befindet sich das allgemeine Layout der Anwendung, sprich das Grundgerüst in welchem die Oberflächen des *CFS/* Ordners gezeigt werden.

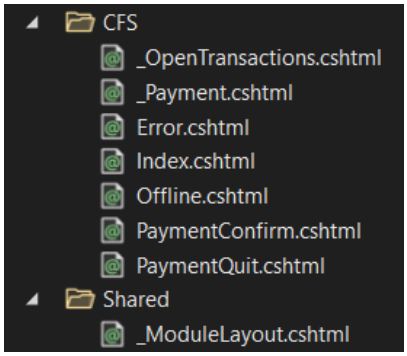

Abbildung 6.2: Ordnerstruktur der View
(Quelle: Eigener Screenshot aus der Entwicklungsumgebung)

Die zuvor erstellten Mockups werden mit Hilfe von C#, HTML, TypeScript und Sass realisiert. Um den Benutzern ein einheitliches Design zu bieten und für die PWA eine App-Shell bereitzustellen sowie die Eigenschaft „App-like“ zu erfüllen, wird wie im Entwurf beschrieben in der Datei *_ModuleLayout* ein allgemeines Layout für die Anwendung implementiert. Dieses besteht aus einem *header*-Tag, einem *main*-Tag und einem *footer*-Tag, siehe Listing 6.1.

```
<header>
  <nav class="navbar navbar-expand-sm navbar-toggleable-sm
     navbar-light bg-white border-bottom box-shadow mb-3">
    <div class="container">
      <a id="transaction-modal" class="navbar-brand" data-toggle="
         modal" data-target="#TransactionModal"><i class="fal
         fa-wallet display-4"><sup><span id="
         supervisor-number-transactions" class="badge badge-secondary"
          style="font-size:24px">@w.OpenTransactions.Count</span></sup
         ></i></a>
      <div class="float-right text-right">@w.UserName <br /><span id="
         supervisor-wallet-amount" class="badge badge-secondary">@w.
         CashCalcDisplay &euro;</span></div>
    </div>
  </nav>
</header>
<div class="container">
  <main role="main" class="pb-3">
    @RenderBody()
  </main>
</div>
<footer class="border-top footer text-muted">
  <div class="container">
    <div class="row">
      <div class="col-auto">
        <img src="/logo/" alt="Logo" style="width:60px" />
      </div>
      <div class="col ">
        <small>
          Eine Handkasse der @pos?.MandantShort <br />
          &copy;
          <script language="JavaScript">
            var aktuellesDatum = new Date();
            var Jahr = aktuellesDatum.getFullYear();
            document.write(Jahr);
          </script>
          - INSIGMA IT Engineering GmbH
        </small>
      </div>
    </div>
  </div>
</footer>
```

Listing 6.1: ModuleLayout-Datei

Für die Anzeige der Benutzerinformationen wird der *header*-Tag verwendet. In diesem stehen Name sowie Kontostand, gegebenenfalls lassen sich auch offene Bargeldumbuchungen einsehen. Der *main*-Tag wird verwendet, um die verschiedenen Seiten innerhalb der Anwendung darzustellen. Dies geschieht durch den Aufruf von *RenderBody()*. Dadurch werden die verschiedenen Inhaltsseiten wie zum Beispiel die Liste mit den Klienten sowie Transaktionsbestätigungen und -abbrüche innerhalb des definierten Layouts angezeigt. Der *footer*-Tag wird benutzt, um allgemeine Informationen und das Kundenlogo anzuzeigen, siehe Abbildung 6.3.

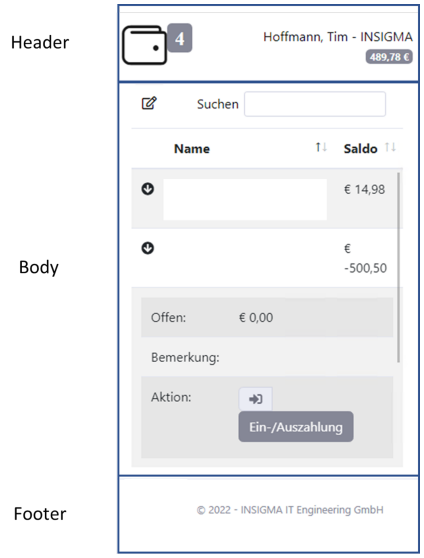

Abbildung 6.3: Aufbau der Benutzeroberfläche
(Quelle: Screenshot aus der SKM-Kasse)

Zur Darstellung der Klientenliste wird das jQuery-Plugin DataTables verwendet. Damit werden HTML-Tabellen um viele Funktionalitäten erweitert, wie z.B. Suchfunktionen oder Sortierfunktionen. Die in einer DataTable dargestellte Information wird über einen ajax-Request nach dem Laden der Seite vom Server angefordert und anschließend clientseitig gerendert. Dadurch ist es möglich, Daten innerhalb der Seite nachzuladen, ohne die Seite zu wechseln oder neu laden zu müssen. Das Initialisieren einer DataTable ist in Listing 6.2 zu sehen.

```
function getTable() {
  return $('#ClientTable').DataTable({
    "language": {
      "url": "/assets/localizations/datatables.net/de.json"
    },
    "scrollY": "55vh",
    "scrollCollapse": true,
    "info": false,
    "paging": false,
    "ajax": getData("CFS.Clients.Range"),
    "columns": [
      {
        "className": 'details-control',
        "orderable": false,
        "data": null,
        "defaultContent": '<i class="fas fa-arrow-circle-down"></i>'
      },
      { "data": "accountDisplay" },
      { "data": "balanceDisplay" }
    ],
    "order": [[1, 'asc']]
  });
}
```

Listing 6.2: Initialisierung der DataTable

Das ist insofern relevant, da der Nutzer innerhalb der Anwendung zwischen verschiedenen Klientenlisten wechseln will. Die Auswahl der entsprechenden Liste kann der Anwender über eine Schaltfläche innerhalb der Anwendung treffen. Mithilfe der in Listing 6.3 durchgeführten Implementierung ist es möglich, auf die Auswahl des Benutzers zu reagieren. Zuerst wird die Auswahl als Cookie im Browser gespeichert, um bei Wiederaufrufen der Seite die Nutzerpräferenz laden zu können. Danach wird je nach Wahl des Nutzers ein ajax-Request an die entsprechende URL gesendet, um die geforderten Daten in der Tabelle visualisieren zu können.

```
1   /**
2     *  Daten anfordern beim Umschalten des Switches
3     */
4   $(function () {
5     $('#clientSwitch').change(function () {
6       if ($(this).prop('checked') == true) {
7         setCookie("CFS.Clients.Range", 2);
8         table.ajax.url('/cfs/objects?Range=2').load();
9       }
10      else {
11        setCookie("CFS.Clients.Range", 1);
12        table.ajax.url('/cfs/objects?Range=1').load()
13      }
14    })
15  })
16 });
```

Listing 6.3: Auswahl der anzuzeigenden Daten

Diese Vorgehensweise hat den Vorteil, dass beim Wechseln der Klienten-Listen nur die jeweilige Liste vom Server angefragt werden muss. Da nur eine Datei angefordert wird, reduziert sich der Netzwerkverkehr auf ein Minimum. Bei einer MVC-Webanwendung wäre die komplette Seite serverseitig neu zu rendern.

6.2.3 Controller

In dieser Anwendung hat der Controller die Aufgabe, den Zustand der View und des Models anhand von Benutzereingaben und Ereignissen zu steuern. In der Tabelle 6.2 werden die Methoden spezifiziert, die benötigt werden, um den Anforderungen und dem Entwurf zu entsprechen.

Methode (ohne Parameter)	Anmerkung
Index()	Diese Methode wird beim Starten der Anwendung aufgerufen und gibt eine HTML-Seite zurück. Diese beinhaltet die Startseite mit der Klienten-Liste sowie Möglichkeiten Transaktionen und Bargeldumbuchungen vorzunehmen.
Objects()	Diese Methode wird nach dem Laden der Startseite, Benutzerinteraktion oder bei Herstellung einer Internetverbindung aufgerufen und liefert ein JSON-Objekt zurück. Dieses JSON-Objekt enthält je nach Benutzerauswahl eine Liste mit Klienten.
CommitTransaction()	Diese Methode wird bei der Bestätigung einer Bargeldumbuchung aufgerufen. Sie ruft in die DAL die entsprechenden Methoden auf, um die Buchung in der Datenbank zu persistieren
PaymentConfirm()	Diese Methode dient der Durchführung einer Transaktion. Ihr werden die benötigten Parameter übergeben, welche sie dann an die DAL weitergibt, welches die Transaktion in der Datenbank persistiert und das Model anpasst. Anschließend gibt diese Methode eine neue Seite zurück, die über den Transaktionsstatus informiert.
PaymentQuit()	Diese Methode wird beim Abbruch einer Transaktion ausgeführt und gibt eine HTML-Seite zurück, die den Anwender über den Abbruch informiert.
Offline()	Diese Methode gibt eine HTML-Seite zurück, die anzeigt, dass keine Internetverbindung besteht und die angeforderten Seiten nicht gecacht wurden bzw. offline nicht verfügbar ist.
GetWalletData()	Diese Methode wird bei der erneuten Herstellung einer Internetverbindung aufgerufen und gibt ein JSON-Objekt zurück. Dieses JSON-Objekt enthält offene Bargeldumbuchungen und dient der Überprüfung inwieweit während der fehlenden Internetverbindung Daten auf dem Server aktualisiert wurden.

Tabelle 6.2: Im Controller verwendete Methoden

6.2.4 Model

Das Model stellt die Datenstruktur der Anwendung und enthält die Daten, die in der View angezeigt werden. Es besteht wie im Entwurf beschrieben aus folgenden Komponenten:

- PaymentViewModel
- ClientViewModel
- WalletViewModel
- WalletTransactionViewModel
- ErrorViewModel
- PosTypeEnum

Das Model wurde um Enum *PosTypeEnum* erweitert. Es enthält eine Aufzählung der verschiedenen Kassentypen in der Anwendung.

In der Regel hat das Model die Aufgabe, Daten zu speichern und stellt Funktionen für Datenoperationen bereit. Innerhalb dieser Anwendung enthält das Model jedoch keine Geschäftslogik zur Bearbeitung der Daten. Diese Logik wird aus Designgründen in eine extra Data Access Layer(DAL) ausgelagert. Der Vorteil dieser Vorgehensweise liegt zum einen in der Verbesserung der Codequalität durch Wiederverwendbarkeit und zum anderen in der Kapselung des Datenzugriffs innerhalb einer Klasse.

6.2.5 Service

Der Service wird durch Dependency Injection (DI) der Anwendung bereitgestellt. Dabei handelt es sich um einen Begriff aus der objektorientierten Programmierung. Dazu werden Methoden einer externen Instanz aufgerufen, um Objekten oder Klassen ihre Abhängigkeiten zuzuweisen (Augsten, 2019a). Durch Verwendung der DI können die Daten des Wallet einheitlich durch diesen bereitgestellt werden. Dies ist sinnvoll, da die Daten des Wallets in der Anwendung öfter benötigt werden. Die Entscheidung fiel zugunsten des Sevices, da das *_ModuleLayout* immer nur auf das Model der angezeigten View zugreifen kann. Um das *WalletViewModel* jedoch innerhalb der ganzen Anwendung zugänglich zu machen, wurde auf ein Service zurückgegriffen, auf den per DI verwiesen wird. Dadurch muss das *WalletViewModel* nicht in jede View extra eingebaut werden. Das erhöht die Codequalität dahingehend, dass der Code einfacher und übersichtlicher ist.

6.2.6 Data Access Layer

Durch die Verwendung der Data Access Layer, wird der Zugriff auf die MIA®-Datenbank vereinheitlicht und standardisiert. Der Vorteil ist darin zu sehen, dass Änderungen und Neuerungen an Datenbankoperationen nicht verstreut in den einzelnen Models implementiert werden müssen, sondern zentral im DAL-Code angepasst bzw. hinzugefügt werden können. Dadurch sinkt die Zahl der Abhängigkeiten, und

das Risiko eines Fehlers sinkt. Um der Anwendung die Möglichkeit zu bieten, Daten in MIA®-Datenbanken zu persistieren, ist zuerst die Etablierung einer Verbindung notwendig. Dazu dient die Funktion in Listing 6.4. Sie gibt ein Objekt der Klasse *DbConnection* zurück, welches benötigt wird, um eine Verbindung herzustellen.

```
1 protected override DbConnection CreateConnection(Guid? contextInfo =
      null, bool ignoreLowSeverities = false) {
2       return base.CreateConnection(contextInfo, true);
3    }
```

Listing 6.4: Datenbank-Verbindung herstellen

Nach Herstellung einer Verbindung zur Datenbank lässt sich mit der Funktion *CreateMIAProcCommand* im Listing 6.5 ein Datenbank-Befehl erstellen. Dieser besteht aus einer Datenbankverbindung (con), einem Prozedurnamen (procName), einem Mandanten (mandantGID) sowie einem Benutzer (userGID). Der Prozedurname legt fest, welche Prozedur in MIA® ausgeführt werden soll, um die Daten zu persistieren. Mandant und Benutzer werden benötigt, um die Operation zuordnen zu können.

```
1  private DbCommand CreateMIAProcCommand(DbConnection con, string procName
       , Guid mandantGID, Guid userGID) {
2       var cmd = CreateCommand(con);
3
4       cmd.CommandText = procName;
5       cmd.CommandType = CommandType.StoredProcedure;
6
7       cmd.Parameters.Add(new SqlParameter("@U", userGID));
8       cmd.Parameters.Add(new SqlParameter("@M", mandantGID));
9       cmd.Parameters.Add(new SqlParameter("@POS", null));
10      cmd.Parameters.Add(new SqlParameter("@L", null));
11      return cmd;
12   }
```

Listing 6.5: Erstellen eines MIA®-Prozedur-Befehls

Nachdem eine *DbConnection* besteht und ein *DbCommand* erstellt wurde, können die Daten an die Datenbank übertragen werden. Dazu werden dem jeweiligen *DbCommand* (Siehe Listing 6.6) die entsprechenden Parameter übergeben, die von der angesprochenen MIA®-Prozedur benötigt werden.

```
1  using (var cmd = CreateMIAProcCommand(con, "insCFSPayment", mandant,
       user)) {
2            cmd.Parameters.Add(new SqlParameter("@PersonalAccount",
               personalAccount));
3            cmd.Parameters.Add(new SqlParameter("@Receipt",
               uniqueidentifier));
4            cmd.Parameters.Add(new SqlParameter("@Amount", amount));
5            cmd.Parameters.Add(new SqlParameter("@BookingText",
               bookingText));
6            cmd.Parameters.Add(new SqlParameter("@GeoData", geoData));
7            cmd.Parameters.Add(new SqlParameter("@Signature", signature));
8            cmd.Parameters.Add(new SqlParameter("@Attachment", attachement
               ));
9            cmd.Parameters.Add(new SqlParameter("@F", flag));
10           cmd.Parameters.Add(new SqlParameter("@PIN", PIN));
11           using (var reader = cmd.ExecuteReader()) {
12             if (reader.Read()) {
13               Result.Flag = SqlHelpers.GetSqlValue<int>(reader, "F", 0);
14               Result.Display = SqlHelpers.GetSqlValue<string>(reader, "
                   Display", "");
15               if (SqlHelpers.ColumnExists(reader, "Signature")) {
16                 Result.Signature = SqlHelpers.GetSqlValue<string>(reader
                     , "Signature", "");
17               }
18               if (SqlHelpers.ColumnExists(reader, "Receipt")) {
19                 Result.Receipt = SqlHelpers.GetSqlValue<Guid>(reader, "
                     Receipt");
20               }
21
22             };
23           }
24         }
```

Listing 6.6: Eintrag der Daten in die Datenbank

6.3 Transition der Kassenanwendung in eine Progressive Web App

In diesem Unterkapitel wird die Implementierung der Transition der Kassenanwendung in eine Progressive Web App beschrieben. Zunächst müssen ein Service Worker und ein Webmanifest hinzugefügt werden. Anschließen werden die Funktionalitäten der Kassenanwendung dahingehend angepasst, dass sie auch offline ein korrektes Verhalten zeigen.

6.3.1 Service Worker

Der Service Worker wird benötigt, um eine Progressive Web App offlinefähig zu machen. Dadurch kann sie unabhängig von der Netzwerkverbindung verwendet werden, was sie in der Handhabung näher an eine native App führt und gleichzeitig die beiden Eigenschaften App-Like und Connectivity Independet einer PWA erfüllt. Da auf den Lebenszyklus des Service Workers bereits in den Grundlagen schon eingegangen wurde, werden im Folgenden nur die Einzelheiten der jeweiligen Implementierung beleuchtet.

Die **Registrierung** (Siehe Listing 6.7) eines Service Workers erfolgt beim ersten Aufruf einer Seite oder sobald eine neue Version des registrierten Service Workers vorhanden ist. Dafür wird im ersten Schritt darauf gewartet, dass das *window*-Objekt, also die Oberfläche der Anwendung, das *load*-Ereignis auslöst. Dieses wird erzeugt, sobald die angeforderte Ressource (*window*) und die von ihr abhängigen Ressourcen geladen wurden. Dadurch wird die Ladezeit zur Darstellung der Anwendung im Browser des Benutzers stark verkürzt. Nach Eintritt des Ereignisses wird durch die *if*-Abfrage geprüft, ob das *serviceWorker*-Attribut im *navigator*-Objekt vorhanden ist. Es soll sicherstellen, dass der Service Worker nur in Browsern registriert wird, die Service Worker unterstützen. Anschließend wird die *navigator.serviceWorker.register()* Funktion aufgerufen, welche den Service Worker über den als Parameter übergebenen Pfad bezieht und anschließend registriert.

```
1 $(window).on('load', function () {
2   if ('serviceWorker' in navigator) {
3     navigator.serviceWorker.register('/serviceworker').then(function (
        reg) {
4     }).catch(function (error) {
5       portal.info('Registration failed with ' + error);
6     });
7   }
8 }
```

Listing 6.7: Registrierung des Service Worker

Während der **Installation**, siehe Listing 6.8, des Service Workers wird zuerst die *self.skipWaiting()* Funktion aufgerufen. Diese erzwingt die Aktivierung einer neuen Version des Service Workers sofern eine solche vorliegt, die dann die Kontrolle übernimmt. Anschließend werden zwei vordefinierte Listen für dynamische und statische Ressourcen wie zum Beispiel HTML-Seiten, JS, CSS und Bilder im Cache gespeichert.

```
const DYNAMIC_FILES = [
  // HTML
  // Data

];
const STATIC_FILES = [
  // HTML
  // JavaScript
  // CSS
  // Pictures
  // Fonts
  // Datatable
  // Manifest
]

// Installation of the Serviceworker
self.addEventListener('install', function (event) {
  self.skipWaiting()
  // Precaching
  event.waitUntil(
    caches.open(CACHE_NAME + SW_VERSION)
      .then(function (cache) {
        return cache.addAll(STATIC_FILES.concat(DYNAMIC_FILES));
      })
  );
});
```

Listing 6.8: Installation des Service Workers

Der nachfolgende Schritt ist die **Aktivierung** (Siehe Listing 6.9). Nach Abschluss der Installation, wird das *activate*-Ereignis ausgelöst. Im Zuge dessen wird mit der *self.clients.claim()* Funktion sichergestellt, dass der Service Worker sofort die Kontrolle über die Seite erhält, ansonsten müsste die Seite neu geladen werden, damit der Service Worker diese übernimmt. Anschließend wird geprüft, inwieweit eine neue Version des Service Workers vorliegt und gegebenenfalls der veraltete Cache gelöscht.

```
self.addEventListener('activate', event => {
  // Remove old caches
  event.waitUntil(
    (async () => {
      self.clients.claim();
      const keys = await caches.keys();
      return keys.map(async (cache) => {
        if (cache !== CACHE_NAME + SW_VERSION) {
          return await caches.delete(cache);
        }
      })
    })()
  )
})
```

Listing 6.9: Aktivierung des Service Workers

Wurden alle vorherigen Schritte durchlaufen, achtet der Service Worker auf das fetch-Ereignis. Innerhalb dieses Ereignisses lässt sich festlegen, wie mit den verschiedenen

Anfragen umgegangen wird. Für dynamische Inhalte wie Kontostände und Bargeldumbuchungen wird der in Abbildung 6.4 gezeigte Ansatz „Network falling back to cache“ verwendet. Für statische Inhalte wie CSS-Dateien oder Logos wird „Cache first, fall back to Network“ benutzt, siehe Abbildung 6.5. Für alle anderen Anfragen ist zwingend eine Internetverbindung notwendig.

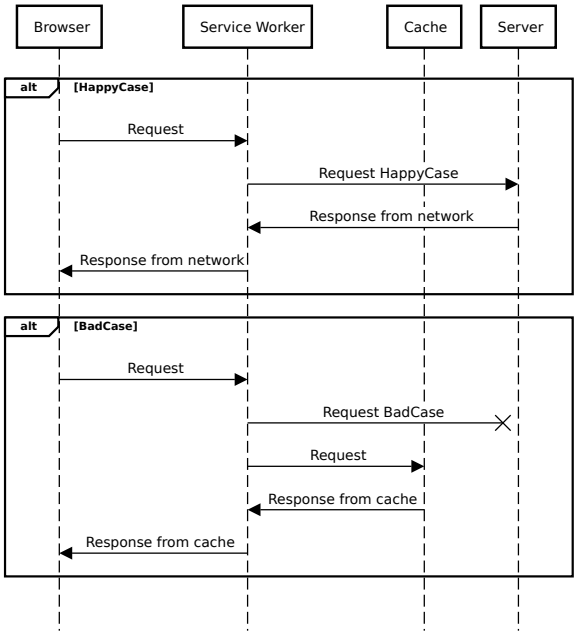

Abbildung 6.4: Caching-Strategie für dynamische Inhalte (Quelle: Eigene Darstellung)

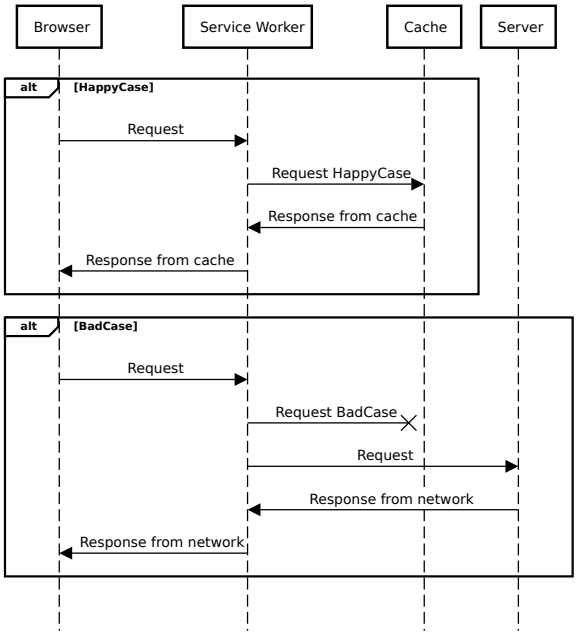

Abbildung 6.5: Caching-Strategie für statische Inhalte (Quelle: Eigene Darstellung)

6.3.2 Web-App-Manifest

Um die Progressive Web App auf einem Endgerät installieren zu können, wird das in Listing 6.10 gezeigte Webmanifest verwendet.

```
{
  "name": "MIA Fremdgeldverwaltung",
  "short_name": "FGV",
  "icons": [
    {
      "src": "/assets/themes/miaappdemo/android-chrome-192
        x192.png",
      "sizes": "192x192",
      "type": "image/png",
      "purpose":  "maskable"
    },
    {
      "src": "/assets/themes/miaappdemo/android-chrome-512
        x512.png",
      "sizes": "512x512",
      "type": "image/png"
    }
  ],
  "start_url": "/CFS/",
  "theme_color": "#ffffff",
  "background_color": "#ffffff",
  "display": "fullscreen"
}
```

Listing 6.10: Aussehen der in der SKM-Kasse verwendeten Web-App-Manifest-Datei

Das Manifest ist auch auf iOS-Geräten gültig, jedoch ist zu vermerken, dass die in dem Manifest definierten Icons von Geräten mit dem Betriebssystem iOS nicht beachtet werden. Aus diesem Grund muss ein gültiges *apple-touch-icon* im *head*-Tag der *_moduleLayout*-Datei hinzugefügt werden. Nach dem Einbinden des Webmanifests in die Anwendung kann diese auf dem jeweiligen Gerät installiert werden.

6.3.3 HTTPS Unterstützung

Die Anwendung wird auf den Servern der INSIGMA gehostet, auf welchen ein gültiges Wildcard-Zertifikat installiert ist und der Domain mia-cloud zugeordnet wurde. Da die URL der Anwendung diese Domain beinhaltet, entspricht sie der Wildcard und erhält ein gültiges SSL-Zertifikat. Beim Aufrufen der URL prüft der Webbrowser das ausgestellte Zertifikat auf seine Gültigkeit. Wird das Zertifikat anerkannt, startet die Installation des Service Workers (Liebel, 2019).

6.3.4 Anpassung der Kassenanwendung

Die Überführung der Webanwendung in eine PWA erfolgte durch die Bereitstellung mittels HTTPS sowie dem Hinzufügen eines Service Workers und eines Web-App-Manifests. In diesem Abschnitt werden die Prozesse der Kassenanwendung gemäß dem in Abschnitt 5.2.4 erstellten Konzept angepasst. Darin ist beschrieben, wie mit den Prozessen Zahlung und Bargeldbuchungen im Falle einer fehlenden Internetverbindung umgegangen werden soll. Zur Erkennung der Netzwerkverbindung werden das *online*- und *offline*-Ereignis der *window*-Schnittstelle verwendet. Um beim Wechsel zwischen Online und Offline entsprechend reagieren zu können, muss ein EventListener implementiert werden, wie in Listing 6.11 dargestellt.

```
1  if (navigator.onLine) {
2    onlineHandler();
3  } else {
4    offlineHandler();
5  }
6
7  window.addEventListener('online', onlineHandler);
8  window.addEventListener('offline', offlineHandler);
```

Listing 6.11: EventListener der Internetverbindung

Dieser reagiert auf den *OnlineHandler* und *OfflineHandler* und löst je nach Verbindungsstatus die in Listing 6.12 gezeigten Funktionen aus. Diese Funktionen dienen unter anderem dem Sperren und Entsperren von Buttons, um den Benutzer vom Öffnen und Durchführen von Zahlungen abzuhalten sowie Funktionsaufrufe, die bei der Herstellung einer Internetverbindung die Daten der Anwendung aktualisieren.

```
1  function onlineHandler() {
2    if (isOnline === false) {
3      updateTableData();
4      updateWalletData();
5    }
6    isOnline = true;
7    $(".commit-transaction").css("pointer-events", "");
8    $("#transaction-modal").attr("data-toggle", "modal");
9    $(".open-modal").removeClass('disabled');
10   $(".submit-modal").prop({ disabled: false });
11 }
12
13 function offlineHandler() {
14   isOnline = false;
15   $(".commit-transaction").css("pointer-events", "none");
16   $("#transaction-modal").removeAttr('data-toggle');
17   $(".open-modal").addClass('disabled');
18   $(".submit-modal").prop({ disabled: true });
19 }
```

Listing 6.12: Funktionen zum Steuern des Funktionsumfangs abhängig von der Internetverbindung

Kapitel 7

Evaluierung

In diesem Kapitel wird die im Kapitel 6 entwickelte Progressive Web App evaluiert. Dazu wird die Anwendung mit den in Kapitel 4 definierten Anforderungen abgeglichen und es wird überprüft, ob diese erfüllt werden. Abschließend wird ein Fazit gezogen.

7.1 Auswertung der implementierten Anforderungen

Innerhalb dieses Abschnitts sollen die in Kapitel 4 festgelegten Anforderungen mit dem Ist-Zustand der Kassenanwendung abgeglichen werden. FA-1 beschreibt die Notwendigkeit einer grafischen Benutzeroberfläche für Mobilgeräte. Die Oberflächen der Anwendung konnten ohne Probleme anhand der im Konzept erstellten Mockups implementiert werden. Bei der Implementierung wurde jedoch darauf geachtet, dass auch eine benutzerfreundliche Darstellung auf anderen Endgeräten wie z.B. Desktops möglich ist. Die Anforderungen FA-2 und FA-3 beschreiben Funktionalitäten innerhalb der Anwendung, nämlich Transaktionen mit Klienten durchführen und Bargeldumbuchungen bestätigen. Die Funktionalitäten wurden anhand der im Entwurf erstellten Prozesse implementiert und werden ebenfalls in einer Benutzeroberfläche für Mobilgeräte angezeigt. Es gilt jedoch zu beachten, dass die Prozesse bei der Transition in eine Progressive Web App nochmal überarbeitet werden mussten, um ein konsistentes Verhalten gewährleisten zu können. Die Anforderung FA-4 besagt, dass die Kassenanwendung die Anforderungen an eine Progressive Web App erfüllen soll. Zur Evaluierung dieser Anforderung wird Google Lighthouse verwendet. Dabei handelt es sich um ein Analysetool zur Untersuchung von Webseiten und Webanwendung hinsichtlich Performance, Progressive Web Apps, Barrierefreiheit, Vorgehensweisen und vielem mehr (Google Developers, 2021).

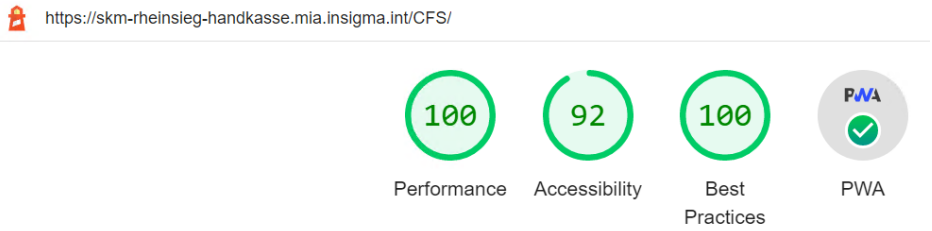

Abbildung 7.1: Ergebnis Google Lighthouse Report
(Quelle: Eigener Screenshot des Lighthouse Reports)

Wie in Abbildung 7.1 zu sehen, erreicht die Anwendung in den getesteten Bereichen fasst alle geforderten Kriterien. Der Bereich SEO wurde nicht getestet, da er für die Anwendung nicht relevant ist. Das Ergebnis der Lighthouse Analyse bestätigt, dass es sich bei der Anwendung um eine Progressive Web App handelt. In Abbildung 7.2 sind die Kriterien aufgelistet, welche laut Google Lighthouse erfüllt sein müssen, damit eine Anwendung als Progressive Web App zählt.

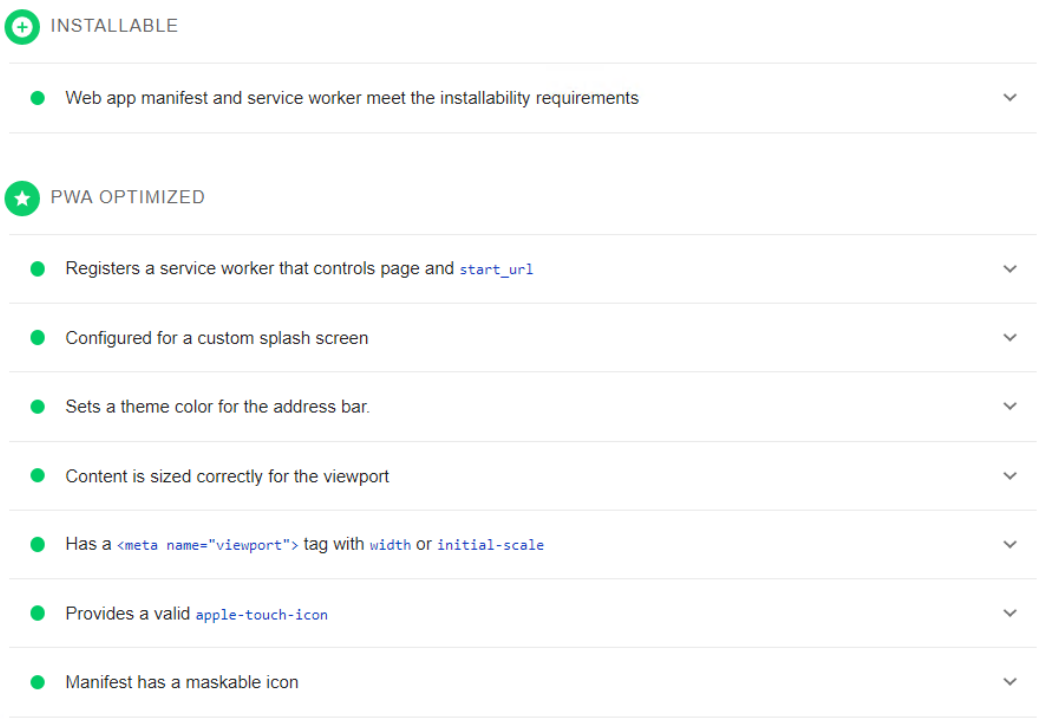

Abbildung 7.2: Ergebnis Google Lighthouse PWA Kriterienkatalog
(Quelle: Eigener Screenshot des Lighthouse Reports)

Als letzte Anforderung FA-5 wurde die Wiederverwendung der Authentifizierung, des Benutzermanagements und der MIA®-Persistenz Mechanismen aus dem MIA®-Framework definiert. Die Wiederverwendung der Authentifizierung und des Benutzermanagements geschieht durch die Platzierung der Anwendung als Modul im Kontext MIA®. Dadurch erfolgen die Anmeldung und Benutzerverwaltung automatisch durch MIA® (Siehe Abbildung 7.3). Das Persistieren der Daten erfolgt ebenfalls durch MIA®. Dazu werden MIA®-Schnittstellen verwendet, um die Daten in der entsprechenden

Datenbank zu persistieren. Ist ein Eintrag erfolgreich, erhält man eine entsprechende Information von MIA®, siehe Abbildung 7.4.

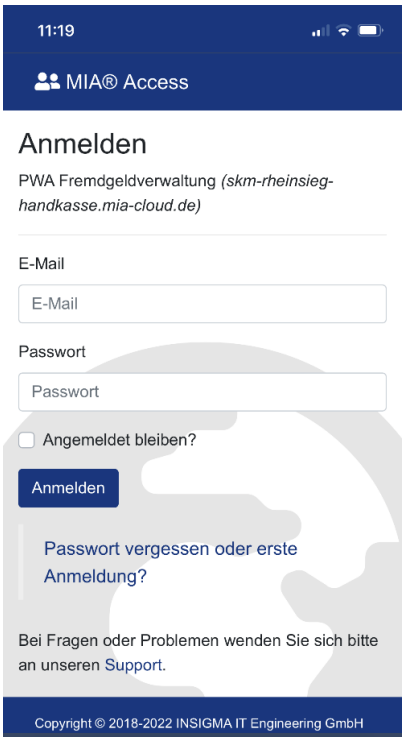

Abbildung 7.3: Anmeldung MIA® beim Aufruf der Anwendungs-URL (Quelle: Eigener Screenshot aus der SKM-Kasse)

Abbildung 7.4: Buchungsbestätigung seitens der Datenbank (Quelle: Eigener Screenshot aus der SKM-Kasse)

7.2 Übertragbarkeit auf MIA®

Die Anwendung wurde als eigenständiges Modul für MIA® entworfen. Auf Grundlage des ausgearbeiteten Konzepts lassen sich bereits bestehende Module in Progressive Web Apps überführen. Bei der Transition eines Modules in eine Progressive Web App ist jedoch zu beachten, dass jeder Prozess der Anwendung dahingehend untersucht werden muss, inwiefern er PWA kompatibel ist und wie er anzupassen ist, um sowohl eine funktionierende Anwendung als auch ein gute Benutzererfahrung zu bieten. Des Weiteren hat sich der Entwickler zu vergegenwärtigen, dass durch das Speichern von Daten im Browser die Persistenz der Daten nicht gewährleistet werden kann. Dies könnte dazu genutzt werden, Benutzereingaben oder Daten zwischenzuspeichern, um sie dann bei der Wiederherstellung mit dem Server zu synchronisieren. In diesem Zusammenhang gilt es den Anwendungsfall genau zu analysieren und abzuwägen, welche Daten dazu geeignet sind.

7.3 Fazit

Mithilfe des in Kapitel 5 ausgearbeiteten Entwurfs sind alle im Kapitel 4 definierten Anforderungen an die Anwendung umsetzbar. Innerhalb des Kapitel 6 werden die gestellten Anforderungen anhand des erstellten Konzepts implementiert. Dadurch konnte die Überführbarkeit von MIA® in eine Progressive Web App bestätigt werden. Dem Kunden SKM wurde die Anwendung FGV als Modul über MIA® Anfang Januar 2022 zum Testen bereitgestellt und Stand 30.03.2022 gab es nur positives Feedback.

Kapitel 8

Zusammenfassung und Ausblick

Diese Arbeit analysiert und untersucht die Möglichkeit, die auf dem MVC-Pattern beruhende Webanwendung MIA® in eine Progressive Web App zu überführen zu können. Diese Frage wird an Hand einer Teilfunktionalität bzw. eines Moduls geklärt.

Das für die Untersuchung ausgewählte Modul dient der Verwaltung von Fremdgeldkonten durch Betreuer der SKM für einen entsprechenden Personenkreis, der zur Durchführung dieser Aufgaben selbst nicht in der Lage ist.

Dazu werden zunächst in den Kapiteln 2 und 3 der Stand der Technik sowie die Grundlagen von Progressiven Web Apps erläutert. Im Anschluss wurden im vierten Kapitel die Anforderungen an die zu entwickelnde Anwendung festgelegt.

In Kapitel 5 wurde anhand der definierten Anforderungen ein Entwurf ausgearbeitet zur Entwicklung der Anwendung und der Überführung in eine Progressive Web App. Zunächst galt es zu klären, welcher architektonische Ansatz vor dem Hintergrund der Anforderungen für die Überführung sinnvoller Weise zu wählen ist. Im Folgenden wird erläutert, welche Gründe dafür maßgebend waren, dass bei der Entwicklung der Anwendung ein Mobile-First-Ansatz verwendet wird und wie die Anwendung auf Benutzerinteraktionen zu reagieren hat. Anschließend wurden Mockups der Oberflächen als auch die benötigten Prozesse modelliert.

Nach der Untersuchung welche Prozessmodifikationen vollzogen werden müssen, um die Anwendung PWA tauglich zu gestalten, erfolgte die Ausarbeitung der Überführungsstrategie. In diesem Zusammenhang wurde auch der Einsatz diverser Caching-Strategien für spezifische Aufgabenstellungen erläutert. Abschließend wird mit Hilfe des MVC-Patterns die Architektur der Anwendung festgelegt. Kapitel 6 beschreibt die Implementierung der Anwendung, indem zunächst der Feinentwurf der Architektur dargestellt wird. Im weiteren Verlauf werden die getroffenen Entwurfsentscheidungen erläutert und anhand spezifischer Quellcode Ausschnitte veranschaulicht. Im Zuge der Evaluierung, die in Kapitel 7 stattfindet, werden die in Kapitel 4 dargestellten Anforderungen an die App durch das Google Tool Lighthouse überprüft und dokumentiert. Als Ergebnis konnten die PWA Eigenschaften der Anwendung belegt werden. Zur Frage inwieweit MIA® in eine Progressive Web App überführt werden kann,

lässt sich zusammenfassend sagen, dass die hier vorgelegte Analyse die Möglichkeit demonstriert, einzelne MIA® Module überführen zu können. Bei der Transition in eine PWA ist jedoch zu untersuchen, welche Funktionalitäten anzupassen sind und inwiefern die verwendeten Daten gegebenenfalls offlinefähig sind. Obschon potenziell die Möglichkeit der Transition besteht, ist es sinnvoller bei der Entwicklung neuer Module auf PWA Tauglichkeit zu achten und bereits bestehende Module nur nach eingehender Prüfung der vorhandenen Prozesse und einer Bewertung des Mehrwertes, für das zu überführende Modul in eine PWA zu überführen.

Literatur

Augsten, S. (2019a). Was ist Dependency Injection? *Dev-Insider*. Zugriff 2. März 2022 unter https://www.dev-insider.de/was-ist-dependency-injection-a-814452/

Augsten, S. (2019b). Was ist eine Native App? *Dev-Insider*. Zugriff 25. November 2021 unter https://www.dev-insider.de/was-ist-eine-native-app-a-827532/

Chen, J. (2020). Service worker caching and HTTP caching. Zugriff 1. Februar 2022 unter https://web.dev/service-worker-caching-and-http-caching/

DataTables: Table plug-in for jQuery. (2022). Zugriff 22. Februar 2022 unter https://datatables.net/

FileTypes. (2022). Dateiendung .CSHTML: Datenbank der Dateiendungen und Dateitypen. Zugriff 22. Februar 2022 unter https://www.filetypes.de/extension/cshtml

Frain, B. (2012). *Responsive web design with HTML and CSS3* (Online-Ausg). Packt Pub.

Gaunt, M. (2021). Service Workers: an Introduction: Web Fundamentals. Zugriff 1. Februar 2022 unter https://developers.google.com/web/fundamentals/primers/service-workers

Goll, J. (2011). *Methoden und Architekturen der Softwaretechnik* (1. Aufl.). Vieweg + Teubner. https://doi.org/10.1007/978-3-8348-8164-9

Google Developers. (2015). Progressive Web Apps (Chrome Dev Summit 2015). Zugriff 11. Februar 2021 unter https://developers.google.com/web/shows/cds/2015/progressive-web-apps-chrome-dev-summit-2015

Google Developers. (2021). Lighthouse. Zugriff 4. März 2022 unter https://developers.google.com/web/tools/lighthouse

Häßler, U. (2019). Javascript Web Worker. Zugriff 1. Februar 2022 unter https://www.mediaevent.de/javascript/web-worker.html

Initiative D21 e. V. (2021). D21-Digital-Index 2020/2021 - Jährliches Lagebild zur Digitalen Gesellschaft. https://initiatived21.de/app/uploads/2021/02/d21-digital-index-2020_2021.pdf

IONOS Digitalguide. (2021). Was ist eine Web-App? Definition und Web-App-Beispiele. Zugriff 25. November 2021 unter https://www.ionos.de/digitalguide/websites/web-entwicklung/verschiedene-app-formate-was-ist-eine-web-app/

Jobe, W. (2013). Native Apps Vs. Mobile Web Apps. *International Journal of Interactive Mobile Technologies (iJIM)*, *7*(4), 27. https://doi.org/10.3991/ijim.v7i4.3226

Liebel, C. (2019). *Progressive Web Apps: Das Praxisbuch* (1. Auflage). Rheinwerk Verlag.

MDN. (2020a). Progressive Web-Apps. Zugriff 11. Januar 2022 unter https://developer.mozilla.org/de/docs/Web/Progressive_web_apps

MDN. (2020b). Service Worker API - Web API Referenz. Zugriff 11. Januar 2022 unter https://developer.mozilla.org/de/docs/Web/API/Service_Worker_API

MDN. (2021). Verwenden von geolocation. Zugriff 11. Januar 2022 unter https://developer.mozilla.org/de/docs/Web/API/Geolocation_API

MDN. (2022a). How to make PWAs re-engageable using Notifications and Push: Progressive web apps (PWAs). Zugriff 16. März 2022 unter https://developer.mozilla.org/en-US/docs/Web/Progressive_web_apps/Re-engageable_Notifications_Push

MDN. (2022b). IndexedDB API - Web APIs. Zugriff 16. Februar 2022 unter https://developer.mozilla.org/en-US/docs/Web/API/IndexedDB_API

MDN. (2022c). Progressive web app structure: Progressive web apps (PWAs). Zugriff 2. Februar 2022 unter https://developer.mozilla.org/en-US/docs/Web/Progressive_web_apps/App_structure

Microsoft Docs. (2022). Erstellen progressiver Webanwendungen mit Blazor WebAssembly für ASP.NET Core. Zugriff 16. März 2022 unter https://docs.microsoft.com/de-de/aspnet/core/blazor/progressive-web-app?view=aspnetcore-6.0&tabs=visual-studio

Niemann, K. D. (Hrsg.). (1995). *Client/Server-Architektur: Organisation und Methodik der Anwendungsentwicklung*. Vieweg.

Osmani, A. (2015). Getting Started with Progressive Web Apps. Zugriff 16. März 2022 unter https://developers.google.com/web/updates/2015/12/getting-started-pwa

Osmani, A. (2019). The App Shell Model. Zugriff 16. März 2022 unter https://developers.google.com/web/fundamentals/architecture/app-shell

Patel, J. (2021). Top 6 Progressive Web App Frameworks To Consider For 2022. *Monocubed*. Zugriff 2. Februar 2022 unter https://www.monocubed.com/progressive-web-app-frameworks/

Pelletier, E. (2013). drawingboard.js. Zugriff 16. März 2022 unter https://github.com/Leimi/drawingboard.js#drawingboardjs

Richard, S. & LePage, P. (2020). What are PWAs. Zugriff 16. März 2022 unter https://web.dev/what-are-pwas/

Rojas, C. (2019). *Building Progressive Web Applications with Vue.js: Reliable, Fast, and Engaging Apps with Vue.js* (1st ed.). Apress; Safari. https://learning.oreilly.com/library/view/-/9781484253342/?ar

Sass. (2022). Documentation. Zugriff 22. Februar 2022 unter https://sass-lang.com/documentation

Siegrist, K. (2021). Service Worker in Web-Anwendungen. Zugriff 11. Januar 2022 unter https://www.informatik-aktuell.de/entwicklung/methoden/service-worker-in-web-anwendungen.html

Springer, S. (2019). Offlinemodus: Die Achillesferse der Progressive Web Apps? Zugriff 11. Januar 2022 unter https://entwickler.de/programmierung/offlinemodus-die-achillesferse-der-progressive-web-apps/

Starke, G. (2020). *Effektive Softwarearchitekturen: Ein praktischer Leitfaden* (9., überarbeitete Auflage). Hanser.

W3C. (2022). Web Application Manifest. Zugriff 1. Februar 2022 unter https://www.w3.org/TR/appmanifest/#dfn-lang

Anhang

Übersicht über elektronische Anhänge:

- Quellcode des implementierten Systems innerhalb der Abschlussarbeit
- Abschlussarbeit in elektronischer Form