Berliner Hochschule für Technik

Entwurf und Entwicklung eines digitalen Impfpasses für iOS Geräte

Julian Veerkamp (903049)

Betreuer: Prof. Dr. Graupner

Gutachter: Prof. Dr. Macos

14 Juli 2022

Inhaltsverzeichnis

1 Einleitung

1.1 Motivation

Die andauernde COVID-19[1] Pandemie hat ein klares Defizit in der Digitalisierung und Verwaltung von Impfbestätigungen aufgezeigt. Es gibt keine verlässliche Lösung, um Impfungen digital zu verwalten, einfach über den Stand von Auffrischungsimpfungen informiert zu werden und Dritten eine sichere Bestätigung über den Impfzustand zu geben.
Anwendungen wie die CovPass-App[2] und die Corona Warn App[3] stellen erste Versuche der Digitalisierung des Impfpasses dar, jedoch fokussieren sich diese lediglich auf die Corona Impfung. Die Corona Warn App fokussiert sich dabei mehr auf die Kontaktverfolgung und das Verwalten von Testergebnissen. Die CovPass-App hat zwar einen klaren Fokus auf das Präsentieren der Zertifikate, unterstützt jedoch keine Darstellung anderen Impfnachweise.

1.2 Zielsetzung

Das Ziel dieser Arbeit ist der Entwurf eines digitalen Impfpasses mit Impfempfehlungen, sowie die Entwicklung eines iOS Frontend Clients (im weiteren “die App”) und eines simplen skalierbaren Backends. Die App soll intuitiv benutzbar sein und der nutzenden Person erlauben, ihre Impfungen und Impfzertifikate einfach zu verwalten. Dabei sollen alle personenbezogenen Daten lokal auf dem Gerät verwaltet werden. Das Backend soll Informationen zu den unterstützten Impfstoffen und den Empfehlungen bereitstellen.
Es soll weiterhin möglich sein, Impfpässe für andere Personen oder Haustiere zu erstellen und diese bei Bedarf verschlüsselt zu ex- und importieren.

[1] Im weiteren auch Corona oder Covid benannt
[2] https://digitaler-impfnachweis-app.de
[3] https://www.coronawarn.app/

Da die Zertifikate allgemein gehalten sind und theoretisch auch für andere Impfungen verwendet werden können, soll die App so entworfen werden, dass Impfungen jeder Art ein Zertifikat bekommen können.

1.3 Aufbau der Arbeit

Das erste Kapitel *Einleitung* beschreibt die Motivation und die Zielsetzung der Arbeit. Anschließend werden im zweiten Kapitel wichtige Grundlagen vorgestellt und verwendete Technologien und Frameworks gezeigt.

Im dritten Kapitel, der *Anforderungsanalyse*, wird die Zielgruppe beschrieben und daraus die Use Cases und die Anforderungen an die App abgeleitet.

Im vierten Kapitel wird das Projekt strukturiert und ein Design Mockup, sowie ein Prototyp für das Frontend entworfen. Für das Backend wird die API und die Transfer Objekte definiert.

Dieser Entwurf wird daraufhin im fünften Kapitel umgesetzt. Dabei wird auf Besonderheiten der Implementierung eingegangen und Code Auschnitte, sowie Diagramme gezeigt.

Das Kapitel 6 *Validierung der Anforderungen* befasst sich mit dem Präsentieren der Ergebnisse und nimmt Bezug auf die ausgearbeiteten Anforderungen aus Kapitel 3.

Zum Abschluss werden im letzten Kapitel *Fazit und Ausblick* die Ergebnisse der Arbeit zusammengetragen und ein Ausblick auf noch anstehende Aufgaben gegeben.

2 Grundlagen

Dieses Kapitel beschreibt die theoretischen und technischen Grundlagen, die für den Entwurf und die Entwicklung einer Impfpass App notwendig sind. Es werden der Aufbau des Corona Impfzertifikates gezeigt, Fragen zum Datenschutz geklärt und eine Reihe an wichtigen modernen Frameworks vorgestellt.

2.1 COVID-Zertifikat

> Das digitale COVID-Zertifikat der EU soll gemäß der Verordnung (EU) 2021/953 als Nachweis dafür dienen, dass eine Person gegen COVID-19 geimpft wurde, ein negatives Testergebnis erhalten hat oder von einer Infektion genesen ist.
> (European Commission 2021, S. 230/32)

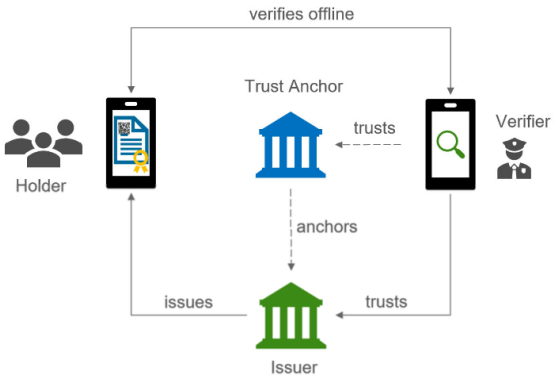


Abbildung 1: DCCG Trust Anchoring, Darstellung aus (eHealth Network 2022a, S.8)

Das COVID-Zertifikat wurde von der EU als einheitliches Modell entwickelt, um länderübergreifend den COVID-19 Impfstatus, ein negatives Testergebnisse oder eine Genesung zu testieren. Dabei sollen Zertifikate von einer Landesinstanz (Issuer) ausgestellt und von Nutzer:innen verifiziert werden können. Dafür wird eine EU-weite Instanz eingerichtet, das Digital COVID Certificate Gateway. Dieses Gateway fungiert als Trust Anchor, sammelt also alle vertrauenswürdigen Issuer und stellt diese zu Verifizierungszwecken zur Verfügung.
Abbildung 1 zeigt den Ablauf vom Ausstellen und Verifizieren eines Zertifikates (eHealth

Network 2022a, S. 8).

2.1.1 *Aufbau des COVID-Zertifikats*

Das COVID-Zertifikat besteht aus mehrere ineinander verpackten Datenstrukturen, die dem Endverbraucher als QR-Code bereit gestellt werden.

QR-Code

> The QR (Quick Response) Code is a two-dimensional (2-D) matrix code that belongs to a larger set of machine-readable codes, all of which are often referred to as barcodes, regardless of whether they are made up of bars, squares or other-shaped elements.
> (ADC 2012, S. 1)

QR-Codes wurden 1994 von dem japanischen Unternehmen Denso Wave eingeführt, um Autoteile durch die Produktions- und Lieferkette zu verfolgen. Durch seine Flexibilität hat sich der QR-Code schnell auf viele andere Bereiche ausgebreitet, nicht nur auf Bereiche in denen herkömmliche 1D-Codes (Barcodes) verwendet wurden, sondern auch auf völlig neue Bereiche, wie Werbung, Coupons oder Tickets (ADC 2012, S. 8).

Um die Daten resistent gegen Umwelteinflüsse zu codieren, bieten QR-Codes Fehlerkorrektur mithilfe der Reed-Solomon Codes (RS-Codes) an. Diese Fehlerkorrektur kann von 7% bis 30% der Daten rekonstruieren, basierend auf dem RS-Code Level, mit dem die Daten encodiert wurden (ADC 2012, S. 5–6).

CBOR Web Token

> Die Nutzdaten werden als CBOR mit einer digitalen COSE-Signatur strukturiert und verschlüsselt. Dies wird gemeinhin als „CBOR Web Token“ (CWT) bezeichnet und ist in RFC 8392 (1) definiert.
> (European Commission 2021, S. 230/35)

CBOR[1] oder Concise Binary Object Representation basiert auf dem bekannten JSON Datenformat und kann schemalos Objekte in ein binäres Format encodieren. Dieses Format ist

[1] https://cbor.io

besonders durch seine geringe Größe für das Codieren in einem QR-Code geeignet.
COSE[2], oder CBOR Object Signing and Encryption, ist eine Datenstruktur, um ein CBOR Objekt zu verschlüsseln und optional zu signieren.

Der CBOR Web Token soll mit zlib[3] komprimiert und als Base45 Encodierter Text in einen QR-Code verpackt werden (European Commission 2021, S. 230/38). Um zu erkennen um was für eine Datenstruktur es sich bei den verschlüsselten Daten handelt, sollen die Base45-codierten Daten den Präfix “HC1” erhalten. Um die Rückwärtskompatibilität sicherzustellen beschreibt dieses Schlüsselwort zum einen die Daten mit den ersten beiden Zeichen, sowie die Version mit dem letzten Zeiche (European Commission 2021, S. 230/38).

Abbildung 2 zeigt den im Durchführungsbeschluss (European Commission 2021, S. 230/37) definierten Serialisierungsablauf.

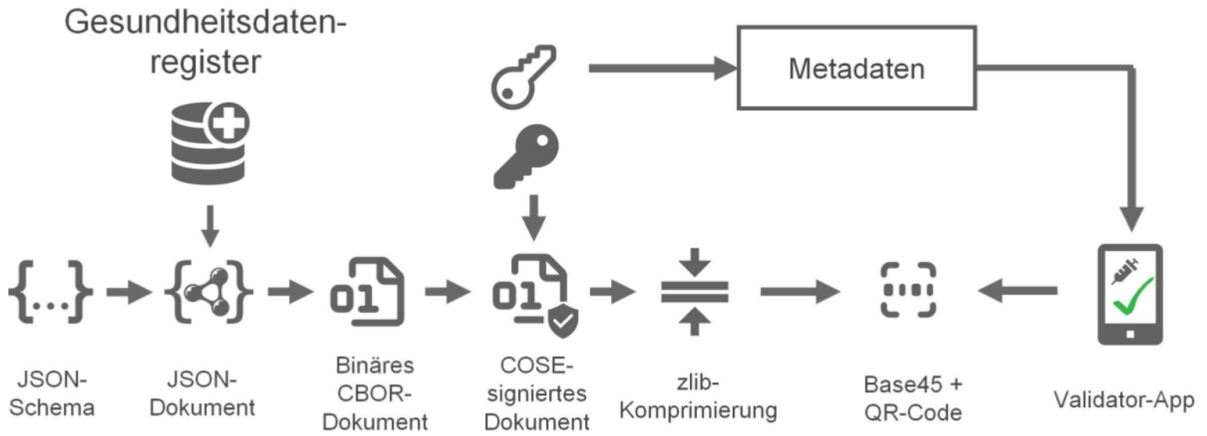


Abbildung 2: Zertifikat Serialisierungsablauf, Darstellung aus (European Commission 2021, S. 230/37)

Das Zertifikat wird nach einem JSON Schema mit den Gesundheitsdaten gefüllt, in ein CBOR Dokument konvertiert. Dieses wird anschließend mit COSE verschlüsselt, mit zlib komprimiert und als Base45-codierter QR-Code exportiert.

[2]https://tools.ietf.org/html/rfc8152
[3]https://zlib.net

2.1.2 *Corona Impfzertifikat*

Die Nutzdaten der COVID-Zertifikate unterstützen sowohl Impf- als auch Test- und Genesungszertifikate.
Für die App sind jedoch nur die Impfzertifikate relevant. Diese sollen hier genauer betrachtet werden.

Quellcode 1: Impfzertifikat JSON

```
[
  "nam":
  [
    "fn": "Schneider",
    "gn": "Andrea",
    "gnt": "ANDREA",
    "fnt": "SCHNEIDER"
  ],
  "ver": "1.0.0",
  "v":
  [
    [
      "co": "DE",
      "ma": "ORG-100031184",
      "mp": "EU/1/20/1507",
      "dn": 1,
      "tg": "840539006",
      "ci": "01DE/00001/1119349007/880NAQ26IFZN2NJ7MKP5ZDNNI#S",
      "sd": 2,
      "dt": "2021-04-12",
      "vp": "1119349007",
      "is": "Bundesministerium für Gesundheit"]
    ],
  "dob": "1943-04-18"
]
```

Quellcode 1 zeigt die Nutzdaten von einem Beispiel COVID-Zertifikat[4] mit einer Moderna Impfung. Das Objekt besteht aus einem allgemeinen Teil, dieser umfasst den Namen (`nam`) der betroffenen Person, das Geburtsdatum(`dob`) und ein Feld um die Versionsnummer (`ver`) des Zertifikats festzuhalten (eHealth Network 2022b, S. 5–6).

Der Name besteht aus dem Familiennamen (`fn`), dem Vornamen (`gn`) sowie den bereinigten

[4] https://github.com/corona-warn-app/cwa-app-ios/blob/release/2.25.x/src/xcode/ENA/ENA/Source/Services/HealthCertificate/Models/__tests__/HealthCertificate%2BMock.swift

Namen, um eventuelle Missverständnisse oder Codierungsfehler von komplexeren Namen mit Sonderzeichen vorzubeugen.

Neben dem allgemeinen Teil kann das Objekt noch einen Impfungs-, Test- oder Genesungseintrag enthalten. Per Spezifikation ist jedoch nur ein Eintrag pro Zertifikat erlaubt (eHealth Network 2022b, S. 7). Für die App ist der Impfeintrag (`v`) von besonderem Interesse und soll hier genauer betrachtet werden.

Der Impfeintrag enthält eine eindeutige ID der Impfstelle (`id`), eine eindeutige ID des Erregers (`tg`), und weitere Informationen über die Impfung (`vp`), den Impfstoff (`mp`), Hersteller (`ma`) und wie viele Impfungen schon stattgefunden haben (`dn`) und die maximale Anzahl empfohlener Impfungen (`sd`) (eHealth Network 2022b, S. 7–8).

Um das Objekt möglichst platzsparend mit einem QR-Code codieren zu können, sollen für alle Attributnamen Akronyme verwendet werden, z.B. `gn` anstatt `givenName` (eHealth Network 2022c, S. 5).

2.1.3 *Digital COVID Certificate Gateway*

> To validate this data structure in each country represented by different EU citizens, cryptographic public keys must be shared across the EU.
> (eHealth Network 2022a, S. 9)

Um die Echtheit eines Zertifikats zu garantieren, bedarf es einer zentralen Vertrauensinstanz. Diese Vertrauensinstanz sammelt eine Liste an vertrauenswürdigen Zertifikataustellern und ist zuständig für das Verwalten dieser vertrauenswürdigen Aussteller.

Da das Signieren der Zertifikate mit einem Public/Private Key Verfahren funktioniert, muss die Vertrauensinstanz lediglich eine Liste der Public Keys bereitstellen und kann alte oder widerrechtlich veröffentlichte Schlüssel widerrufen (eHealth Network 2022a, S. 18).

Abbildung 3 zeigt die vorgeschlagene Architektur für ein EU-weites Zertifikatverifizierungssystem, dem Digital COVID Certificate Gateway (eHealth Network 2022a, S. 8).

Neben dem DCCG soll es noch nationale Backends geben, die von den Ländern selber bereit gestellt werden um mit den lokalen Gesetze vor Ort kompatibel zu sein (eHealth Network 2022d, S. 8).

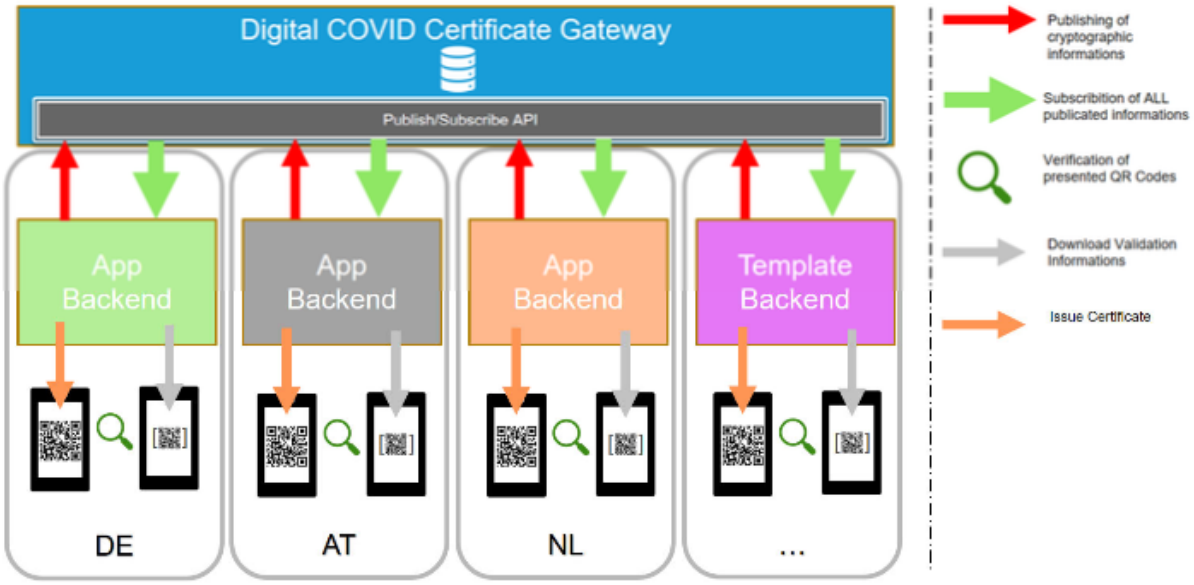


Abbildung 3: DCCG übersicht, Darstellung aus (eHealth Network 2022a, S. 8)

2.2 Datenschutz

Durch die schützenswerte Natur der zu verarbeitenden Daten ist der Datenschutz für die App ein grundlegendes Thema.
In Artikel 4 der Datenschutz-Grundverordnung werden die personenbezogenen Daten und die Gesundheitsdaten definiert.

> Gesundheitsdaten [sind] personenbezogene Daten, die sich auf die körperliche oder geistige Gesundheit einer natürlichen Person [...] beziehen und aus denen Informationen über deren Gesundheitszustand hervorgehen.
> (DSGVO 2018, Art. 4)

Da die App Impfungen einer, anhand von Namen und Geburtsdatum, identifizierbaren Person verarbeitet, bedarf es nach Artikel 6.1(a) der DSGVO einer Einwilligung der betroffenen Person. Eine Einwilligung, wie in Artikel 7 der DSGVO beschrieben, muss von der betroffenen Person explizit erteilt werden und kann jederzeit widerrufen werden.
Da es sich bei den zu verarbeitenden Daten auch um Gesundheitsdaten handelt, muss sichergestellt werden, dass die Verarbeitung nach Artikel 9 DSGVO rechtsmäßig ist. Dies ist gegeben, da Artikel 9.2 (a) die Verarbeitung von Gesundheitsdaten bei ausdrücklicher Einwilligung der betroffenen Person erlaubt.

2.3 Frameworks

2.3.1 *SwiftUI*

SwiftUI[5] ist ein deklaratives UI-Framework von Apple. Es vereint Storyboards und programmatische UI-Programmierung. Im Gegensatz zu imperativen Frameworks, wie UIKit[6], steht bei deklarativem Code die Beschreibung des Problems im Vordergrund.
SwiftUI bietet der nutzenden Person verschiedene Bausteine an, um das Layout und den Inhalt einer UI-Komponente zu bauen. Diese Bausteine können mit sogenannten Modifiern angepasst werden.

Quellcode 2: SwiftUI Modifier Beispiel

```
1  struct ExampleView: View {
2      var body: some View {
3          Text("Hello World")
4              .fontWeight(.bold)
5              .foregroundColor(.black)
6              .padding()
7              .cornerRadius(10)
8              .background(.gray)
9              .padding()
10     }
11 }
```

Modifier sind als Methoden der View Klasse implementiert und lassen sich aneinanderreihen. Dies hat zur Folge, das beim Entwickeln Schreibarbeit abgenommen wird und der Code leserlich bleibt, da redundante Symbole, wie die Instanz, die man modifizieren möchte, nicht ausgeschrieben werden (s. Quellcode 2). Dieses Aneinanderreihen hat zur Folge, dass die Modifier sequenziell abgearbeitet werden. Abbildung 4 zeigt, wie sich die UI-Komponente ändert, wenn in Quellcode 2 die siebten und achten Zeile getauscht wird. Vor dem Vertauschen, links, ist der Hintergrund hinter dem Text grau mit abgerundeten Ecken, nach dem Vertauschen sind die Ecken nicht abgerundet,

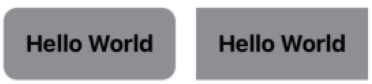
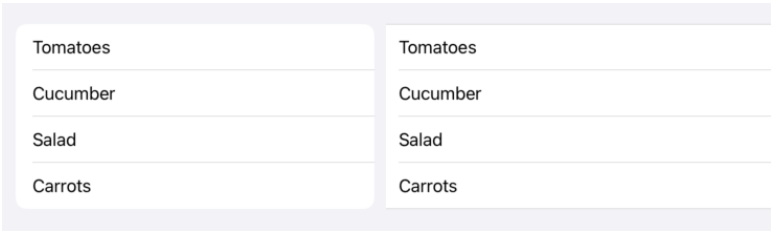


Abbildung 4: Vergleich der Modifier Reihenfolge in Quellcode 1

[5]https://developer.apple.com/documentation/swiftui/
[6]https://developer.apple.com/documentation/uikit

obwohl der `.cornerRadius()` Modifier immer noch aufgerufen wird. Dies ist eine Folge der sequenziellen Abarbeitung der Modifier. Der `.background()` Modifier färbt den gesamten Hintergrund ein, der ihm von den vorhergehenden Modifiern zur Verfügung gestellt wird. Wenn die Ecken mit dem `.cornerRadius()` Modifier abgerundet wurden bevor die Hintergrundfarbe gesetzt, dann wird der Hintergrund nur bis zu den neuen abgerundeten Grenzen gemalt.

Vergleich SwiftUI und UIKit



(a) SwiftUI **(b)** UIKit

Abbildung 5: Listen Implementierung

Um die Vorteile von SwiftUI für das schnelle Entwerfen von komplexen Layouts zu zeigen, ist hier ein Beispiel gezeigt, das in SwiftUI und UIKit eine simple Liste implementiert.

Quellcode 3 ist die Implementierung der Liste in SwiftUI(Apple [kein Datum] a). Diese Liste kommt standardmäßig mit einer visuell ansprechenden Konfiguration, die zur allgemeinen Apple Design Sprache passt.

Quellcode 4 ist die Implementierung der Liste in UIKit(Apple [kein Datum] b).

Quellcode 3: Beispiel Liste mit SwiftUI

```
1 struct ExampleView: View {
2     @State var items: [Item] = Item.sampleData
3 
4     var body: some View {
5         List(items, id: \.self) { item in
6             Text(item.name)
7         }
8     }
9 }
```

Quellcode 4: Beispiel Liste mit UIKit

```
1  class ExampleViewController: UICollectionViewController {
2      typealias DataSource = UICollectionViewDiffableDataSource<
           Int, String>
3      typealias Snapshot = NSDiffableDataSourceSnapshot<Int,
           String>
4
5      var dataSource: DataSource!
6
7      override func viewDidLoad() {
8          super.viewDidLoad()
9
10         let listLayout = listLayout()
11         collectionView.collectionViewLayout = listLayout
12
13         let cellRegistration = UICollectionView.CellRegistration
               { (cell: UICollectionViewListCell, indexPath:
              IndexPath, itemIdentifier: String) in
14             let item = Item.sampleData[indexPath.item]
15             var contentConfiguration = cell.
                   defaultContentConfiguration()
16             contentConfiguration.text = item.name
17             cell.contentConfiguration = contentConfiguration
18         }
19
20         dataSource = DataSource(collectionView: collectionView)
              { (collectionView: UICollectionView, indexPath:
              IndexPath, itemIdentifier: String) in
21             return collectionView.dequeueConfiguredReusableCell(
                   using: cellRegistration, for: indexPath, item:
                   itemIdentifier)
22         }
23
24         var snapshot = Snapshot()
25         snapshot.appendSections([0])
26         snapshot.appendItems(Item.sampleData.map { $0.name })
27         dataSource.apply(snapshot)
28
29         collectionView.dataSource = dataSource
30     }
31
32     private func listLayout() ->
           UICollectionViewCompositionalLayout {
33         let listConfiguration =
```

```
            UICollectionLayoutListConfiguration(appearance: .
            grouped)
34          return UICollectionViewCompositionalLayout.list(using:
            listConfiguration)
35      }
36  }
```

Aus den Code Beispielen ist klar ersichtlich wieviel Arbeit SwiftUI Entwickler:innen in den Grundbausteinen eines UI Frontends abnimmt. Funktionen wie das animierte Hinzufügen von neuen Einträgen und das automatische Aktualisieren alter Listeneinträge wird von SwiftUI als Standardkonfiguration mitgeliefert, damit Entwickler:innen mehr Zeit in die Business Logic investieren können.

Property Wrapper
Um die Kommunikation zwischen verschiedenen SwiftUI Views sicher zu stellen, werden verschiedene Property Wrapper zur Verfügung gestellt.

> A property wrapper adds a layer of separation between code that manages how a property is stored and the code that defines a property.
> (Apple [kein Datum] c, Properties)

Die gängigsten Property Wrapper sind:

`@State`: Eine State Variable beschreibt den Zustand der View, wenn sich die zu Grunde liegende Variable verändert werden alle SwiftUI Views, die diese Variable benutzen neu gerendert.
`@Binding`: Eine Binding Variable erlaubt die beidseitige Kommunikation zwischen Views. Während State Variablen nur hierarchisch tieferliegende Views neu malt, können mithilfe von Bindings aus tieferen Views die Variablen höherer Views geändert werden.
`@OberservedObject`: Ein ObservableObject ist eine Klasse, die es Subklassen erlaubt, komplexe Modelle mit SwiftUI kompatibel zu machen. Die Attribute, die sich wie State Variablen verhalten sollen, werden in der Klasse mit dem `@Published` Property Wrapper versehen. Um die Klasse in einer SwiftUI View zu benutzen wird der `@ObservedObject` Wrapper verwendet.

2.3.2 *Combine*

> The Combine framework provides a declarative approach for how your app processes events. Rather than potentially implementing multiple [...] completion handler closures,

> you can create a single processing chain for a given event source.
> (Apple [kein Datum] d)

Einige Funktionen in SwiftUI, wie das automatische Neumalen von aktualisierten Objekten, werden mit Combine[7] realisiert.
Combine verpackt das Observer Pattern (Gamma, Helm, Johnson, Vlissides 1994, S. 127) in ein für Entwickler:innen einfach zu benutzendes Framework.

Die drei Grundpfeiler von Combine sind die Publisher, die Subscriber und die Operator.

Publisher erlauben es Events an ein oder mehrere Subscriber zu senden (Shai Mishali, Todorov 2021, S. 38). Ein Subscriber kann diese Events mit der `.sink()` Methode verarbeiten ohne den Publisher zu blockieren.
Publisher laufen grundsätzlich ohne festes Ende und geben ihre Events solange an die Subscriber weiter, bis diese ihre Subscription beenden oder der Publisher entscheidet, dass keine Events mehr veröffentlicht werden müssen(Shai Mishali, Todorov 2021, S. S39–40).

Die Operatoren in Combine verhalten sich ähnlich zu dem Modifiern in SwiftUI. Sie sind auf dem Publisher implementiert und erlauben das sequenzielle Bearbeiten und Umwandeln dieses Publishers (Shai Mishali, Todorov 2021, S. 74).

Quellcode 5 zeigt einen simplen Publisher, der vier Events in Form von Integern ausgibt. Die `.sink()` Methode stellt das Ende der Kette dar und behandelt mit der `receiveValue` Closure das Erhalten eines Events und mit der `receiveCompletion` Closure den finalen Abschluss des Publishers.

Quellcode 5: Combine Beispiel

```swift
let myRange = (0...3)
_ = myRange.publisher
    .sink(receiveCompletion: { print ("completion: \($0)") },
          receiveValue: { print ("value: \($0)") })

// Prints:
//  value: 0
//  value: 1
//  value: 2
//  value: 3
//  completion: finished
```

[7] https://developer.apple.com/documentation/combine

Combine wird von SwiftUI für die `@State`, `@Binding` und `@ObservableObject` Property Wrapper genutzt, taucht aber auch in anderen Frameworks wie Core Data auf um diese mit SwiftUI kompatibel zu halten und Entwickler:innen mehr Flexibilität zu gewähren. Abbildung 6 zeigt einige Anwendungsgebiete von Core Data im Apple Ökosystem (Shai Mishali, Todorov 2021, S. 24).

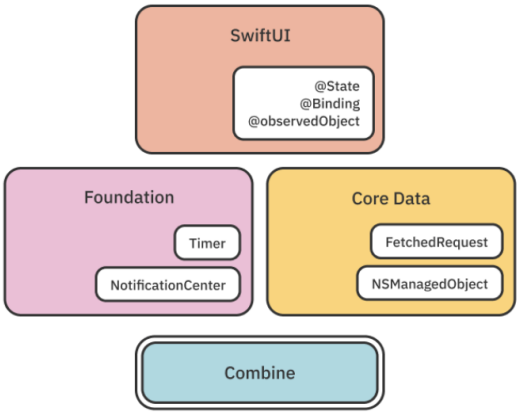


Abbildung 6: Anwendungsbereiche von Combine, Darstellung aus (Shai Mishali, Todorov 2021, S. 24)

2.3.3 *CoreData*

Neben UI Komponenten und reaktiver Kommunikation zwischen den Komponenten ist das persistente Speichern von Daten wichtig. Für iOS gibt es verschiedene etablierte Varianten.
Das naheliegende Framework Core Data[8] wird von Apple angeboten und ist ein Objektgraph und Persistenz Framework.

> In an object-oriented program, groups of objects form a network through their relationships with each other—either through a direct reference to another object or through a chain of intermediate references. These groups of objects are referred to as object graphs. (Apple 2018)

Alternativ wird SQLite[9] standardmäßig von iOS mitgeliefert(yapstudios 2022). Über die Jahre haben sich verschiedene Wrapper wie SQLiteSwift[10] oder die YapDatabase[11] etabliert, S welche die Benutzung von SQLite mit Swift angenehmer ermöglichen.

Obwohl Core Data auch auf SQLite basiert, bietet es als Object Relational Mapping (ORM) viele, für Entwickler:innen nützliche, Funktionen an, die über die Fähigkeiten von SQLite hinaus gehen.

[8] https://developer.apple.com/documentation/coredata
[9] https://www.sqlite.org/index.html
[10] https://github.com/stephencelis/SQLite.swift
[11] https://github.com/yapstudios/YapDatabase

Core Data kann für Entwickler:innen automatisch Migrationen zu neuen Objektversionen vornehmen und kann mithilfe von iCloud[12] über mehrere Geräte synchronisiert werden.

Modelle werden in Core Data über eine spezielle `.xcdatamodel` Datei erstellt. Die Swift Repräsentation der Modelle kann von XCode automatisch generiert oder vom Entwickler manuell verwaltet werden.

Über einen `NSManagedObjectContext` können Entwickler:innen mit diesen Modellen interagierten. Der Objekt Kontext ist ein separater Container unabhängig von der persistenten Datenbank. In diesem können Objekte hinzugefügt, bearbeitet oder gelöscht werden. Um die Änderungen persistent in die Datenbank zu schreiben, muss der Kontext explizit gespeichert werden (Wals 2022, S. 24–25).
Eine Objektinstanz gehört nur einem Context an, es können aber mehrere Instanzen in verschiedenen Kontexten verwendet werden (Apple [kein Datum] e, NSManagedObjectContext).

2.3.4 *AWS Cloud*

Für das Verwalten der Impfdaten und -empfehlungen soll ein Backend verantwortlich sein. Die Skalierbarkeit der Backendlösung ist dabei von grundlegender Wichtigkeit.
Die AWS[13] Cloud bietet Entwickler:innen verschiedene Dienste an um ein skalierbares Backend zu entwerfen.

Das Amazon API Gateway[14] erlaubt es skalierbare Backends zu verwalten. Dabei übernimmt das API Gateway alle Aufgaben, die mit der Annahme und Verarbeitung von bis zu Hunderttausenden von gleichzeitigen API-Aufrufen verbunden sind (Amazon Web Services 2021a, S. 3).
Die einzelnen Routen einer REST[15] API können mit dem API Gateway als einzigartige Lambda[16] Funktionen modelliert werden.

> AWS Lambda is an event-driven, serverless compute service that extends other AWS services with custom logic, or creates other backend services that operate with scale, performance, and security. (Amazon Web Services 2021b, S. 3)

[12] https://developer.apple.com/icloud/cloudkit/
[13] https://aws.amazon.com/de/
[14] https://aws.amazon.com/de/api-gateway/
[15] https://www.ics.uci.edu/~fielding/pubs/dissertation/fielding_dissertation_2up.pdf
[16] https://aws.amazon.com/de/lambda/

3 Anforderungsanalyse

Für die erfolgreiche Umsetzung eines Softwareprojekts müssen die Anforderungen klar definiert werden. Das Ziel der Anforderungsanalyse ist es, diese Anforderungen herauszuarbeiten. Dafür werden die Zielgruppe und ihre Anforderungen analysiert und die daraus resultierenden Use Cases visualisiert. Die Anforderungen werden dann aus den Use Cases abgeleitet und als Funktionale Anforderungen festgehalten.

3.1 Zielgruppe

Die App soll das digitale Verwalten von Impfungen erlauben und der nutzenden Person darauf basierende Empfehlungen aussprechen. Dafür wird den Nutzer:innen die Möglichkeit geboten, Impfzertifikate, wie das Corona Impfzertifikat, zu einem digitalen Impfpass hinzuzufügen oder Impfungen, die keine Zertifikate unterstützen, manuell einzutragen. Die App speichert zugelassene Impfstoffnamen, um vor Falscheintragungen zu schützen.
Nutzer:innen sollen weiterhin neue Impfpässe für andere Personen oder Haustiere hinzufügen können. Um die App flexibel zu halten und z.B. Eltern das Verwalten von Impfpässen ihrer Kinder zu erleichtern, können diese Impfpässe verschlüsselt exportiert werden. So können die Kinder ihren eigenen Impfpass übertragen bekommen, wenn sie alt genug dafür sind.
Die Empfehlungen werden von der App angefragt und lokal ausgewertet. Dabei werden das Alter, das Geschlecht und Risikoländer in Betracht gezogen, sowie ausgelaufene Zertifikate und Impfungen.

Die App soll in Englisch entwickelt werden. Dadurch können Nutzer:innen nicht nur international den Impfstatus nachweisen, sondern es erleichtert auch Migrierten den Zugang und die Kommunikation mit ärztlichem Fachpersonal.

3.2 Use Cases

Ein Use Case Diagramm, oder Anwendungsfalldiagramm, ist ein UML-Diagramm zur Darstellung der Akteure und ihrer erwarteten Anwendungsfälle. Das Diagramm zeigt mögliche Abhängigkeiten und Beziehungen zwischen den Akteuren und Anwendungsfällen (Laplante 2009, S. 78).

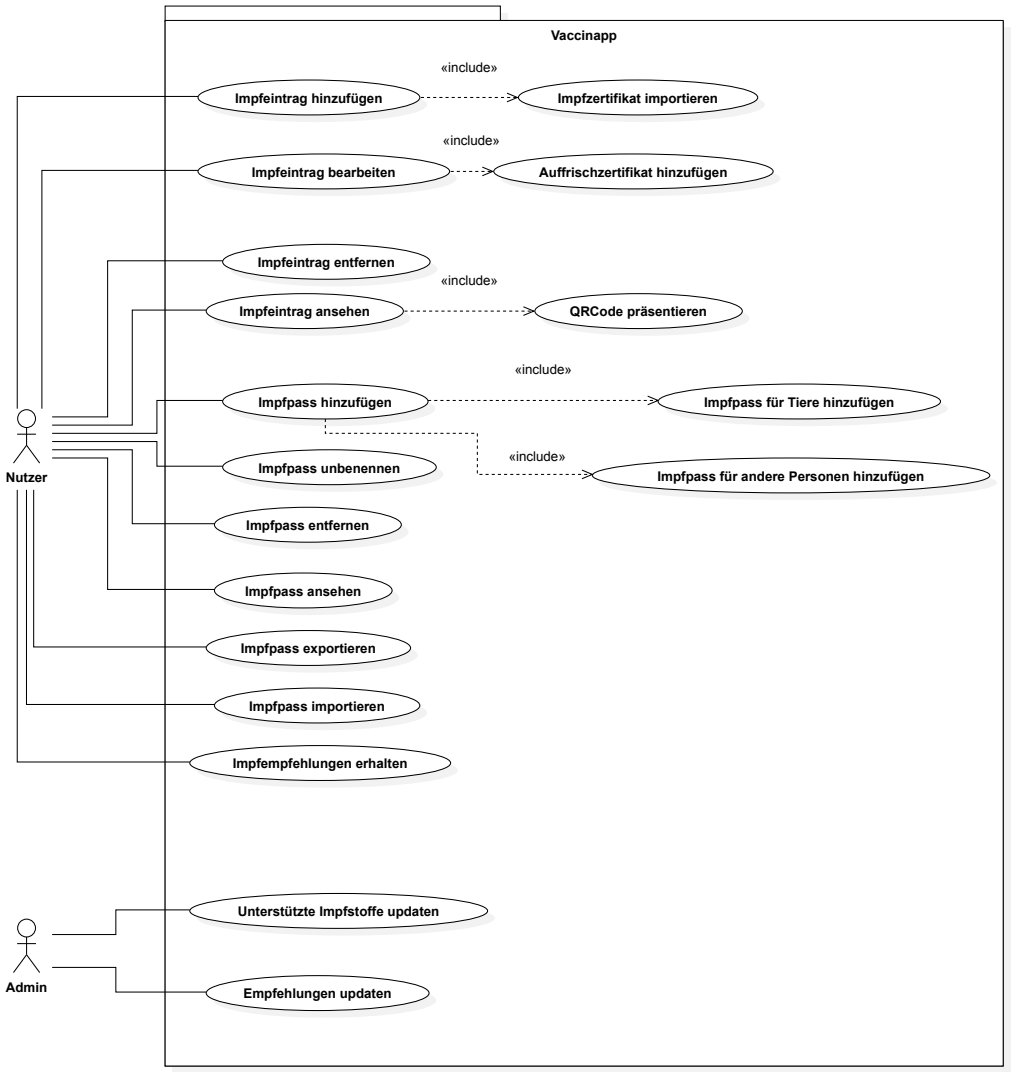


Abbildung 7: Use Case Diagramm

3.3 User Stories

User Stories sind simpel formulierte Anforderungen die ein Akteur an ein Softwaresystem hat. Sie sind eindeutig und lassen sich anhand von Akzeptanztests überprüfen (Laplante 2009, S. 58–59).

ID	*Beschreibung*
US #01	*Als Nutzer:in möchte ich meine existierenden Impfungen in die App eintragen können.*
US #02	*Als Nutzer:in möchte ich mein Impfzertifikat als QR-Code in die App importieren.*
US #03	*Als Nutzer:in möchte ich meine importierten Impfungen bearbeiten und updaten können.*
US #04	*Als Nutzer:in möchte ich neue Zertifikate für eine bestehende Impfung importieren können.*
US #05	*Als Nutzer:in möchte ich Impfungen entfernen können.*
US #06	*Als Nutzer:in möchte ich meine Impfungsdetails ansehen können.*
US #07	*Als Nutzer:in möchte in meinen QR-Code von zertifizierten Impfungen präsentieren können.*
US #08	*Als Nutzer:in möchte ich neue Pässe hinzufügen, um die Impfungen anderer Personen oder Tiere zu verwalten.*
US #09	*Als Nutzer:in möchte ich meine Pässe bearbeiten können.*
US #10	*Als Nutzer:in möchte ich meine Pässe entfernen können.*
US #11	*Als Nutzer:in möchte ich alle meine Impfungen in einem Pass ansehen können.*
US #12	*Als Nutzer:in möchte ich meine Pässe verschlüsselt exportieren können.*
US #13	*Als Nutzer:in möchte ich exportierte Impfpässe importieren können.*
US #14	*Als Nutzer:in möchte ich über anstehende Auffrischimpfungen informiert werden.*

ID	*Beschreibung*
US #15	*Als Nutzer:in möchte ich über empfohlende Impfungen in meinem Aufenthaltsland informiert werden.*
US #16	*Als Admin möchte ich neu unterstützte Impfungen ohne einen neuen Release verwalten können.*

3.4 Funktionale Anforderungen

Funktionale Anforderungen sind konkrete Anforderungen an ein Softwaresystem. Sie beschreiben die Dienste, die das System bereitstellen soll, und wie das System auf Eingaben reagieren wird (Laplante 2009, S. 6).

Die funktionalen Anforderungen lassen sich in 3 Gruppen aufteilen:

Muss-Anforderungen beschreiben Kriterien, die das Produkt als Minimal Viable Product (MVP) erfüllen muss.
Soll-Anforderungen sind Kriterien, die nicht grundlegend nötig sind und über den MVP hinausgehen, aber dennoch umgesetzt wurden.
Kann-Anforderungen sind Kriterien, die nicht im Rahmen dieser Arbeit implementiert sind, aber Ausblick auf die Weiterentwicklung geben.

ID	*Beschreibung*	folgt aus
FA#01	*Die App muss das Verwalten der Impfeinträge unterstützen.*	US #01, #03, #05, #06
FA#02	*Die App muss das Verwalten der Impfpässe unterstützen.*	US #08, #09, #10, #11
FA#03	*Die App muss das Importieren von Impfzertifikaten über QR-Codes unterstützen.*	US #02
FA#04	*Die App muss Zertifikate für eine Zweitimpfung automatisch einordnen.*	US #04

ID	*Beschreibung*	folgt aus
FA#05	*Die App muss den QR-Code des neusten Zertifikats anzeigen.*	US #07
FA#06	*Die App soll das verschlüsselte Exportieren eines Impfpasses unterstützen.*	US #12
FA#07	*Die App soll das Importieren eines verschlüsselten Impfpasses unterstützen.*	US #13
FA#08	*Die App soll Zertifikate auf ihre Gültigkeit überprüfen.*	US #14
FA#09	*Die App soll Empfehlungen für fehlende Impfungen anzeigen.*	US #14, #15
FA#10	*Die App soll die Empfehlungen nach Ländern filtern können.*	US #15
FA#11	*Die App kann Impfaufkleber aus dem physischen Impfpass einscannen.*	US #01
FA#12	*Das Backend soll Anfragen zu Impfempfehlungen beantworten können.*	US #15
FA#13	*Das Backend soll Anfragen zu unterstützten Impfstoffen beantworten können.*	US #16, #17
FA#14	*Das Backend kann den RSS-Feed der STIKO lesen und die tabellarisierten Empfehlungen verarbeiten.*	US #16, #17, #15

3.5 Nicht-Funktionale Anforderungen

Nicht-Funktionale Anforderung beschreiben externe Anforderungen an ein System (Laplante 2009, S. 7).

ID	*Beschreibung*
NFA#01	*Die App muss alle personenbezogenen Daten auf dem Endgerät behandeln.*
NFA#02	*Das Backend soll skalierbar sein.*
NFA#03	*Die App soll Zertifikate in zwei Klicks anzeigen können.*
NFA#04	*Die App soll sich an die Apple Design Richtlinien (Human Interface Guidelines) halten.*
NFA#05	*Die App soll einfach zu benutzen sein.*

4 Entwurf

Bevor mit der Entwicklung begonnen werden kann, müssen neben der Anforderungsanalyse noch weitere Spezifikationen entworfen werden. In diesem Kapitel werden zum einen für das Frontend ein Mockup Design entworfen, sowie einen Prototyp entwickelt, um das grundlegende Konzept früh testen zu können. Zum anderen muss die Backend API entworfen werden, sowie benötigte Data Transfer Objektmodelle definiert werden. Abschließend sollen die Entwicklungsumgebung und die zu benutzenden Dependencies festgehalten werden.

4.1 Frontend

4.1.1 *Design*

Der erste Entwurf der App wurde mit dem Design Tool Sketch[1] zusammen mit der offiziellen Apple UI Library[2] erstellt. Die Library enthält flexible Designelemente, die das aktuelle UI von iOS Geräten widerspiegeln. Durch diese bekannten Designelemente kann eine App entworfen werden, die sich an die empfohlenen Human Interface Guidelines[3] hält. Dies führt dazu, dass sich die App für die Nutzer:innen schon beim ersten Öffnen vertraut anfühlt, da bekannte Konzepte und Strukturen wiedererkannt werden.

Abbildung 8 (s. S. 23) zeigt die grundlegende Struktur der App. Die Knoten stellen separate Ansichten dar, die Kanten stehen für die Navigationswege zwischen den Ansichten.
Beim Öffnen der App wird zunächst die Impfübersicht (VaccinationList) angezeigt, die Impfungen sind die Basis der gesamten App und sollen den Nutzer:innen schnell zur Verfügung stehen. Auf der gleichen Ebene der Impfübersicht sind die Empfehlungen (Recommendations)

[1] https://www.sketch.com
[2] https://developer.apple.com/design/resources/
[3] https://developer.apple.com/design/human-interface-guidelines/guidelines/overview

sowie die Einstellungen (Settings) angeordnet. Diese Ansichten sollen mit der vertrauten Tab Bar[4], einer vertikalen Liste von Knöpfen am unteren Ende des Bildschirms, erreichbar sein.

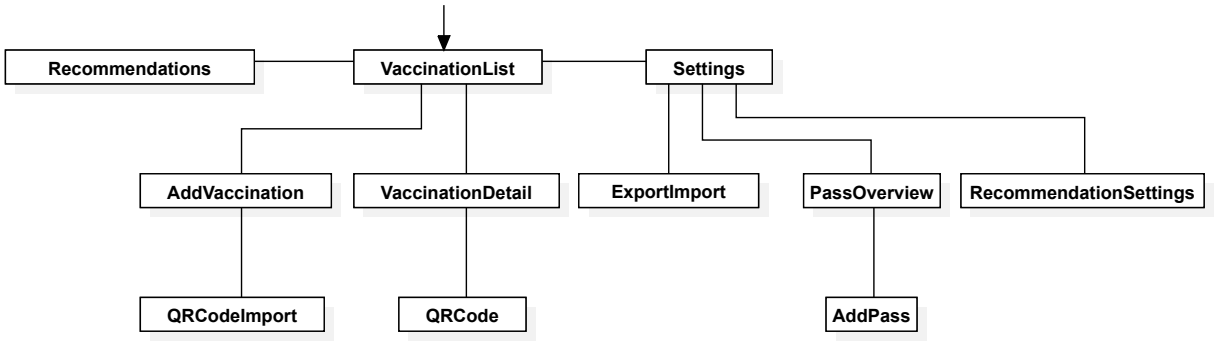


Abbildung 8: App View Struktur

Von der Impfübersicht aus gelangt der oder die Nutzer:in entweder zu einer Detailübersicht einer Impfung (VaccinationDetail) oder zu dem Formular zum Hinzufügen neuer Impfungen (AddVaccination). Die Detailübersicht zeigt weitere Informationen zu der Impfung und für Impfungen mit Zertifikaten auch den QR-Code (QR-Code) an. Die Einstellungen erlauben es neue Impfpässe hinzuzufügen (PassOverview), bestehende Pässe zu ex- oder importieren (Export-Import) und die Länderauswahl für Empfehlungen anzupassen.

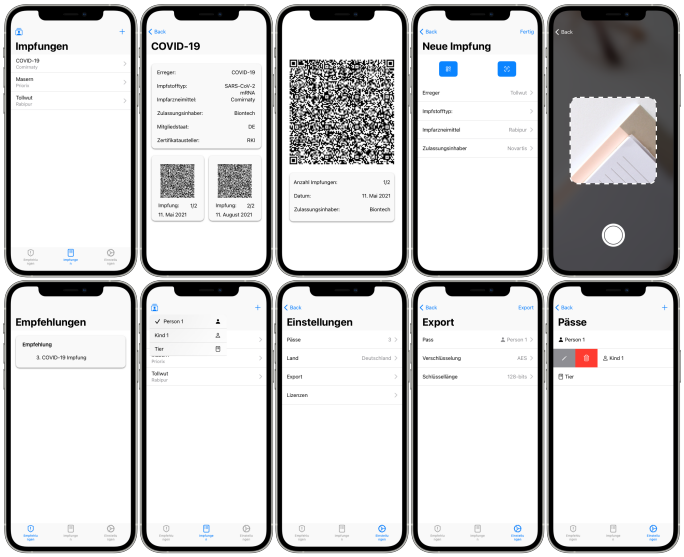


Abbildung 9: Design Mockups

[4]https://developer.apple.com/design/human-interface-guidelines/components/navigation-and-search/tab-bars

Die Design Mockups der einzelnen Ansichten sind in Abbildung 9 dargestellt. Sie dienen der Referenz und Verbildlichung der App Struktur, stellen jedoch keine festen Designvorschriften dar. Von links nach rechts werden gezeigt: Impfübersicht, Impfungsdetails, QR-Code, Impfung hinzufügen, QR-Code Import, Empfehlungen, Impfliste mit Pass Wechseln Menü, Einstellungen, Export/Import Ansicht und die Passübersicht.
Die Icons, die für UI Elemente benötigt werden, kommen aus der SFSymbols[5] Sammlung von Apple. Diese Icons werden nativ von iOS Geräten unterstützt und sind von Grund auf für verschiedene Display- und Textgrößen entworfen worden. Sie Skalieren automatisch mit den Bedienbarkeitseinstellungen im Endgerät und fördern so die Bedienbarkeit.

Prototype

Um das Design und den User Flow zu testen, wurde das Sketch Prototype Feature benutzt. Dieses erlaubt das Erstellen von klickbaren “Hotspots” um den oder die Nutzer:in durch die einzelnen Ansichten zu navigieren. Abbildung 10 zeigt den gesamten Prototyp, die gelben Pfeile stellen mögliche Navigationswege durch die App dar. Durch die Sketch Mirror App[6] kann der Prototyp direkt auf dem Endgerät getestet werden. So wird sicher gestellt, dass das Design nicht kontextfrei entworfen wird und Elemente auf dem Endgerät die erwarteten Maße haben.

[5] https://developer.apple.com/sf-symbols/
[6] https://apps.apple.com/us/app/sketch-mirror/id677296955

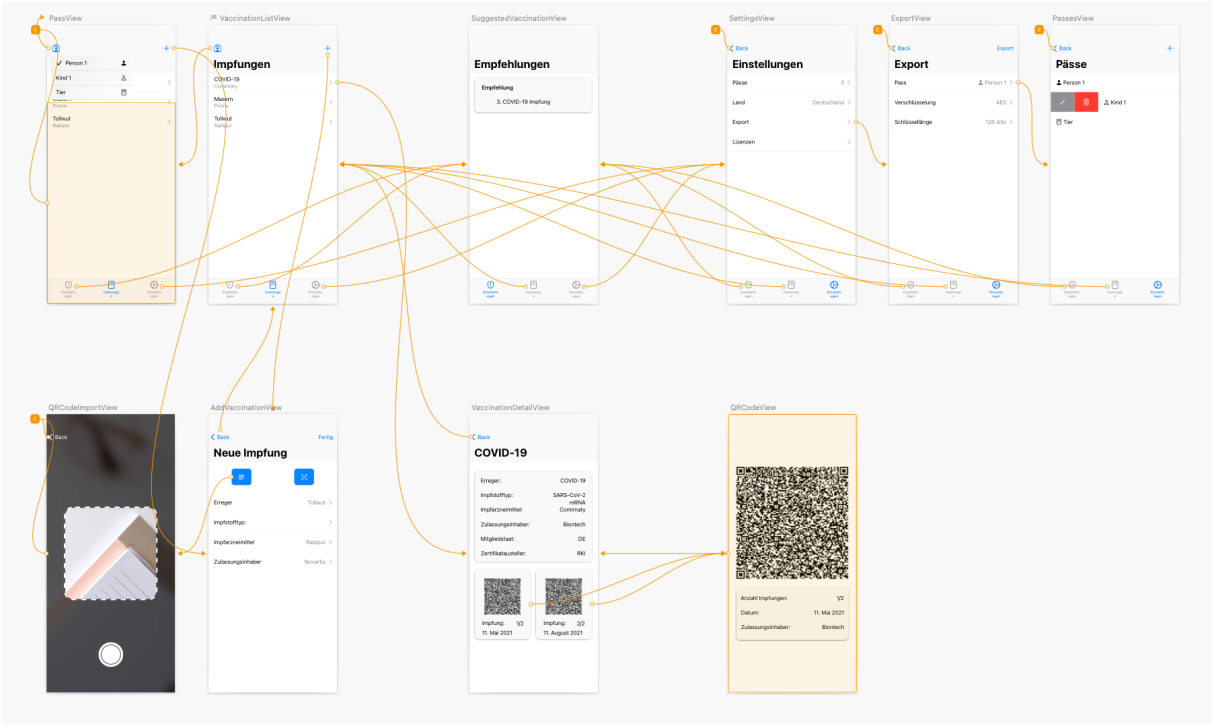


Abbildung 10: Prototype Flow

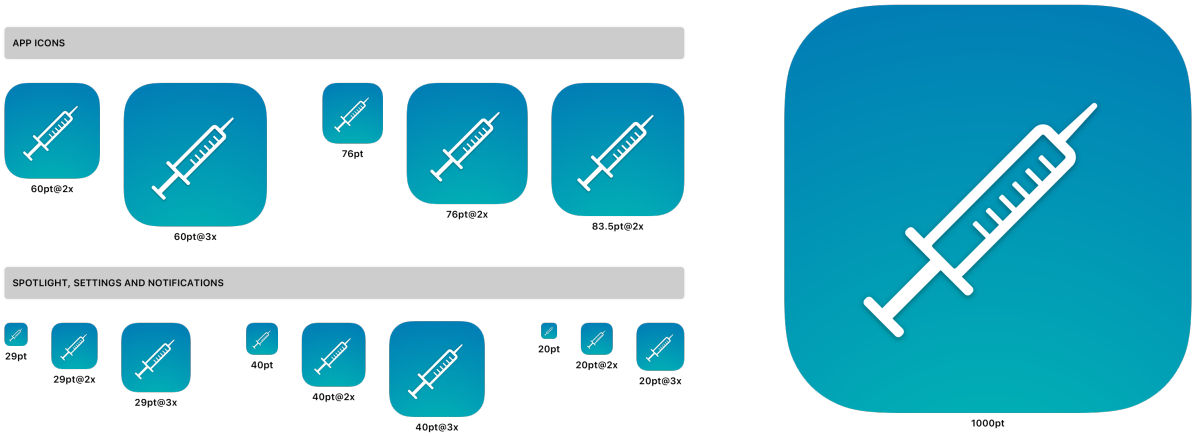


Abbildung 11: Vaccinapp Logo

4.1.2 *Branding*

Zu dem Entwurf einer App gehört auch ein klares Branding auszuarbeiten. Das Logo und der Name einer App sind die ersten Kontaktpunkte neuer Nutzer:innen und haben dadurch starke Auswirkungen auf den ersten Eindruck.

Um das Logo modern, aber auch freundlich zu gestalten, wurde ein simples weißes Piktogramm auf einem angenehm blauen Farbverlauf gewählt. Die Farbe blau wird von vielen Menschen als eine beruhigende Farbe gesehen. Positive Assoziation mit dem Himmel oder Gewässern entstehen (Farbtonkarte 2022).
Das Piktogramm ist eine Spritze, ein klares Zeichen für Impfungen. Ein leichter Schlagschatten differenziert das Logo von dem Hintergrund und verbessert so die Lesbarkeit.
Abbildung 11 zeigt das Logo in verschiedenen Größen, um die Wirkung des Logos in jedem Kontext zu zeigen. Bei den Größen handelt es sich um übliche von Apple vorgeschriebene Größen, für alle Displaygrößen, den App Store, sowie Icons neben Benachrichtigungen.
Bei dem Entwurf des Logos wurde Wert daraufgelegt, dass es sich zwischen herkömmlichen iOS Apps konsistent einordnet und nicht zu sehr hervorsticht, aber dennoch schnell zu erkennen ist um welche Anwendung es sich handelt.
Abbildung 12 zeigt das Logo auf einem Standard iPhone Homescreen neben anderen Apps.

Abbildung 12: Logo im Kontext

Als Name wurde das Portmanteau-Wort *Vaccinapp* gewählt. Es setzt sich aus den englischen Wörtern *Vaccination* und *App* zusammen. Vaccinapp beschreibt so die App in einem einprägsamen Wort.

4.2 Backend

Das Backend ist für das Verwalten der unterstützten Impfungen zuständig, sowie den dazugehörigen Impfstoffen und Empfehlungen. Es wird nach dem Pull Prinzip gearbeitet, das heißt die App ist dafür verantwortlich die neusten Änderungen periodisch vom Backend abzufragen und lokal zwischenzuspeichern. Diese Anfragen werden dabei vollkommen stateless verar-

beitet, die tatsächliche Auswertung und Filterung der Empfehlungen passiert lokal auf dem Endgerät.

Die zu implementierenden Routen sind:

- `/recommend/{country}`
- `/update`

/recommend
Die `/recommend` Route ist zuständig für die Empfehlungen für ein explizites Land, z.B. `/recommend/germany`, oder generelle Empfehlungen mit dem Parameter `/recommend/all`. Die Route gibt eine Liste an empfohlenen Impfungen zurück.

/update
Die `/update` Route ist dafür zuständig ein komplettes Set der unterstützten Impfungen zurückzugeben. Diese können anschließend auf dem Endgerät genutzt werden, um die lokalen Impfungen und Impfstoffe zu aktualisieren.

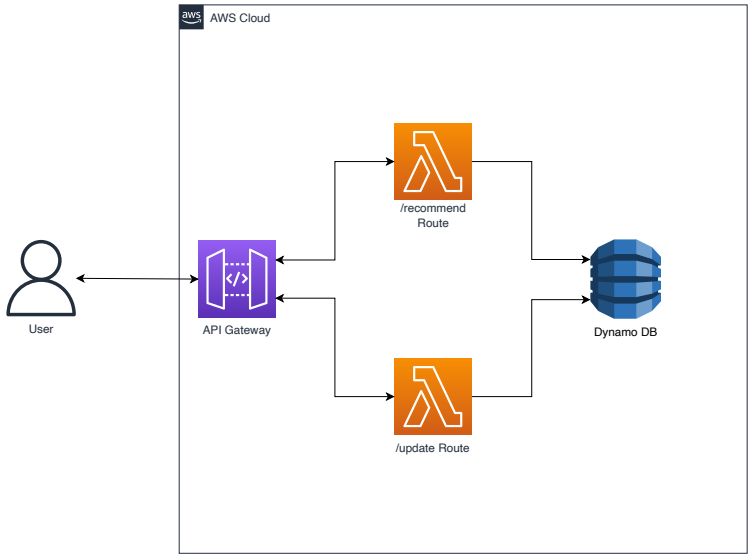


Abbildung 13: Backend Architekturdiagramm

4.2.1 *Data Transfer Object*

Um die erfolgreiche Kommunikation zwischen Backend und Frontend sicherzustellen, bedarf es klar definierter Modelle, die von beiden Seiten verarbeitet werden können. Die Data Transfer

Objekte (DTO) auf der Frontendseite sind Subklassen der `Codable` Klasse[7]. Diese erlaubt unter anderem das einfach Konvertieren von und zu JSON. Durch die klare Definition der API, der JSON Antwort, siehe Quellcode 6, und dem Swift DTO, siehe Quellcode 7, können Front- und Backend unabhängig voneinander entwickelt und getestet werden.

Quellcode 6: DTO JSON

```
{
  "name": "Tollwut",
  "targetID": "000000001",
  "type": "human",
  "vaccines": [
    {
      "id": "EU/0/15/0001",
      "name": "Rabipur",
      "manufacturer": "ACME Inc."
    }
  ],
  "age": 12,
  "countries": ["thailand"],
  "gender": "male"
}
```

Quellcode 7: DTO Swift

```
struct DiseaseTargetDTO: Codable {
    var targetID: String
    var type: String
    var vaccines: [VaccineDTO]
    var age: String?
    var gender: String?
    var name: String
    var countries: [String]?
}

struct VaccineDTO: Codable {
    var id: String
    var name: String
    var manufacturer: String
}
```

[7]https://developer.apple.com/documentation/foundation/archives_and_serialization/encoding_and_decoding_custom_types

Das `DiseaseTargetDTO` soll alle Informationen zu einem einzigartigen Erreger enthalten, es werden allgemeine Sachen wie der Name (`name`), eine Identifikationsnummer (`targetID`) und auf welches Lebewesen die Krankheit zutrifft (`type`), zum Beispiel `human`, `cat` oder `dog`, gespeichert. Das DTO enthält weiterhin eine Liste an unterstützten Impfstoffen (`vaccines`) und verschiedene optionale Felder, um Empfehlungen auszusprechen.
Die optionalen Felder müssen nur bei zutreffenden Empfehlungen ausgefüllt werden. Es werden altersabhängige (`age`), landesabhängige (`countries`) und geschlechtsabhängige (`gender`) Empfehlungen unterstützt.

4.3 Tech Stack

Die App soll nativ für iOS Geräte entwickelt werden. Als UI Framework soll SwiftUI verwendet werden, für die Persistenz soll Core Data verwendet werden. Netzwerkanfragen sollen asynchron mit dem Combine Framework behandelt werden, damit das UI nicht blockiert wird und reaktionsschnell bleibt. Neben den genannten Frameworks werden noch verschiedene Frameworks zum Decodieren des QR-Codes in einen CBORWebToken verwendet: CodeScanner[8], ASN1Decoder[9], base45-swift[10] und SwiftCBOR[11], sowie Source Code aus der CovPassApp[12]. Der Source Code aus der CovPassApp umfasst das CBORWebToken Modell sowie dafür benötigte Objekte und sollen in einem klar erkenntlichen `./External/`-Ordner abgelegt werden. Der Code soll mit dem XCTest[13] Framework getestet werden.

Das Backend wird in Python mit Hilfe des serverless[14] Frameworks geschrieben.

GitLab[15] soll zur Versionierung des Codes und der Sprintplanung verwendet werden. Zusätzlich zu den Issues wurde GanttLab[16] verwendet, um eine klare Übersicht über die Projektplanung zu erhalten.

[8] https://github.com/twostraws/CodeScanner
[9] https://github.com/filom/ASN1Decoder
[10] https://github.com/ehn-dcc-development/base45-swift
[11] https://github.com/unrelentingtech/SwiftCBOR
[12] https://github.com/Digitaler-Impfnachweis/covpass-ios
[13] https://developer.apple.com/documentation/xctest
[14] https://www.serverless.com
[15] https://about.gitlab.com
[16] https://www.ganttlab.com

5 Entwicklung

In diesem Kapitel sollen besondere Implementierungsdetails hervorgehoben und die allgemeine Architektur der App und des Backends beschrieben werden. Anschließend wird auf die Teststrategie und das Deployment eingegangen.

5.1 Frontend

Die Architektur des Frontends folgt dem Model-View-Controller Muster. Das MVC Muster in der reinsten Form teilt ein Software System in 3 Komponenten auf:
Das *Model* ist ein Objekt, das die Informationen über die Domäne verwaltet. Dieses Objekt ist ein nicht visuelles Objekt.
Die *View* präsentiert das *Model* einem oder eienr Nutzer:in und ist lediglich für dieses Präsentieren gedacht.
Der *Controller* ist zuständig für das Verarbeiten von User Interaktionen und das Manipulieren der Modelle. (Fowler 2015, S. 330)

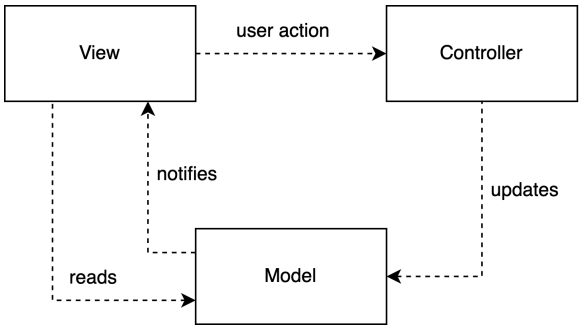


Abbildung 14: MVC Architekturdiagramm

> As I think about MVC I see two principal separations: separating the presentation from the model and separating the controller from the view. Of these the separation of presentation from model is one of the most fundamental heuristics of good software design. (Fowler 2015, S. 331)

Das MVC Muster spaltet also die Model- und die Kontrolllogik von der Präsentation. Dabei ist die Teilung von Model und Präsentation von besonderem Interesse. Die Präsentation ist

zwar abhängig vom Model, das Model jedoch nicht von der Präsentation. Durch eine strikte Trennung kann das Model in einem Vakuum entwickelt und auf seine Funktionsweise getestet werden (Fowler 2015, S. 331).

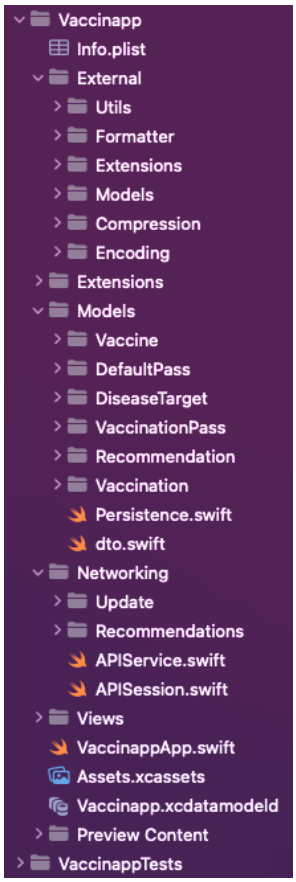


Abbildung 15: Ordner Übersicht

Die Trennung von Präsentation und Kontrolllogik ist dabei etwas komplexer. Die SwiftUI API mit den PropertyWrappern wie `@State` und `@Binding`, machen das Erstellen eines separaten Controllers redundant. Die NetworkingController als grundlegende Kommunikationspfade zwischen Backend und Endnutzer:in sollen auf ihre Korrektheit getestet werden können und wurden daher in separate Klassen ausgelagert.

Das Projekt ist logisch in einen `Model`, einen `Networking`- und einen `Views`-Teil gegliedert. Daneben gibt es noch den `External`-Ordner, in ihm ist die CBORWebToken Implementierung der CovPassCheck App enthalten.

Der Model-Ordner enthält alle Core Data Dateien, jedes Model erhält einen Ordner und ist jeweils aufgeteilt in mehrere Dateien:

`<name>+CoreDataClass.swift`: Die CoreDataClass Dateien enthalten die grundlegende Klasse und werden von XCode aus dem XCDataModel generiert und bei Bedarf angepasst.

`<name>+CoreDataProperties.swift`: Die CoreDataProperties enthalten die Attribute und Funktionen zum Hinzufügen von Elementen zu einer Many-To-Many oder Many-To-One Beziehung. Die Properties werden auch aus dem XCDataModel generiert.

`<name>+Wrappers.swift`: Die Wrapper Dateien enthalten Hilfsmethoden um einfach auf optionale Attribute zuzugreifen und statische Methoden die Core Data Anfragen (NSPredicate) zurückgeben. Diese NSPredicates können dann an anderen Stellen genutzt werden und helfen beim Verständnis, da sie mit einem Namen versehen sind.

5.1.1 *Startpunkt*

Quellcode 8 zeigt den Einstiegspunkt der App. Beim Starten der App wird in der `init()`-Methode zuerst ein Update vom Backend ausgeführt um die neusten Änderungen der Erreger

und der Impfstoffe zu erhalten. Anschließend wird der `DefaultPass` geladen oder, wenn keiner existiert, ein neuer erstellt.
Der `DefaultPass` ist ein simples Container-Objekt, das einen `VaccinationPass` referenziert. Er stellt den zuletzt ausgewählten bzw. gerade aktiven Pass dar. Die Computated Property[1] `body` stellt die Startszene der App dar. In diesem Fall ein einzelnes Fenster mit einer Tab Bar. Die Tab Bar enthält, wie in Kapitel 4.1.1 beschrieben, die Empfehlungs- und Impfliste, sowie die Einstellungen.
Zum Schluss werden noch globale Einstellungen über SwiftUI Modifier definiert. Der Modifier `.accentColor()` setzt die Akzent Farbe für alle Buttons und Highlights. `.environment()` erlaubt das zugreifen auf den Core Data Context aus jeder View.

Quellcode 8: Vaccinapp Einstiegspunkt

```
1  import SwiftUI
2  import CoreData
3  import Combine
4
5  @main
6  struct VaccinappApp: App {
7      @AppStorage("didLaunchBefore") var didLaunchBefore: Bool =
           false
8      let persistenceController = PersistenceController.shared
9
10     @StateObject var defaultPass: DefaultPass
11     @State private var selection = 2
12
13     init() {
14         persistenceController.update()
15
16         let context = persistenceController.container.
               viewContext
17         let fetchRequest = DefaultPass.fetchRequest()
18         fetchRequest.fetchLimit = 1
19
20         if let pass = try? context.fetch(fetchRequest).first {
21             self._defaultPass = StateObject(wrappedValue: pass)
22         } else {
23             self._defaultPass = StateObject(wrappedValue:
                   DefaultPass(context: context))
24             try? context.save()
25         }
```

[1] https://docs.swift.org/swift-book/LanguageGuide/Properties.html

```
26
27          UITabBar.appearance().scrollEdgeAppearance =
                UITabBarAppearance()
28      }
29
30      var body: some Scene {
31          WindowGroup {
32              ZStack {
33                  if !didLaunchBefore {
34                      Onboarding(defaultPass: defaultPass)
35                  } else {
36                      TabView(selection: $selection) {
37                          Recommendations(defaultPass: defaultPass
                                )
38                              .tabItem {
39                                  Image(systemName: "rectangle.
                                        stack")
40                                  Text("Recommendations")
41                              }
42                              .tag(1)
43                          PassView(defaultPass: defaultPass)
44                              .tabItem {
45                                  Image(systemName: "person.text.
                                        rectangle")
46                                  Text("Vaccinations")
47                              }
48                              .tag(2)
49                          Settings()
50                              .tabItem{
51                                  Image(systemName: "gear")
52                                  Text("Settings")
53                              }
54                              .tag(3)
55                      }
56                  }
57              }
58              .accentColor(.mint)
59              .navigationViewStyle(StackNavigationViewStyle())
60              .environment(\.managedObjectContext,
                    persistenceController.container.viewContext)
61          }
62      }
63  }
```

5.1.2 *Modelle*

Um CoreData in der gesamten App verwenden zu können, wurde eine Singleton Klasse namens `Persistence.swift` erstellt (s. Quellcode 9). Das Singleton Pattern stellt sicher, dass von dieser Klasse nur eine Instanz existiert und diese global erreichbar ist (Gamma, Helm, Johnson, Vlissides 1994, S. 127). Das `Persistence` struct bietet zwei statische Instanzen an, die `shared` Instanz ist für den normalen Gebrauch gedacht, die `preview` Instanz stellt einen Beispiel Test Container für die XCode Previews bereit und ist nur für Testzwecke zu benutzen, da die Objekte nur im Arbeitsspeicher existieren und nicht persistent gespeichert werden.

Quellcode 9: Persistence Singleton

```
1  import CoreData
2  import Combine
3
4  struct PersistenceController {
5      static let shared = PersistenceController()
6
7      static var preview: PersistenceController = {
8          let result = PersistenceController(inMemory: true)
9          let viewContext = result.container.viewContext
10
11         do {
12             try viewContext.save()
13         } catch {
14             let nsError = error as NSError
15             fatalError("Unresolved error \(nsError), \(nsError.
                   userInfo)")
16         }
17         return result
18     }()
19
20     let container: NSPersistentContainer
21
22     init(inMemory: Bool = false) {
23         container = NSPersistentContainer(name: "Vaccinapp")
24         if inMemory {
25             container.persistentStoreDescriptions.first!.url =
                   URL(fileURLWithPath: "/dev/null")
26         }
27         container.loadPersistentStores(completionHandler: { (
               storeDescription, error) in
28             if let error = error as NSError? {
```

```
29                 fatalError("Unresolved error \(error), \(error.
                      userInfo)")
30             }
31         })
32         container.viewContext.
              automaticallyMergesChangesFromParent = true
33     }
34 }
```

Neben diesen Attributen enthält die Persistence Klasse alle nötigen Funktionen, um neue Einträge der Vaccinations, Targets, Vaccines und Recommendations in den Core Data Context zu schreiben. Quellcode 10 zeigt die `addVaccination()` Methode, die aus einem Data Transfer Object ein neues Core Data Objekt erstellt und dem Core Data Context hinzufügt.

Quellcode 10: addVaccination() Methode

```
1  func addVaccination(_ item: AddVaccinationDTO, to pass:
      VaccinationPass, context: NSManagedObjectContext) throws {
2      let newItem = Vaccination(context: context)
3      newItem.timestamp = item.timestamp
4      newItem.target = item.diseaseSelection
5      newItem.vaccine = item.vaccineSelection
6      newItem.addPass(pass)
7      newItem.valid = item.valid
8
9      for c in item.certificates {
10         try newItem.addCertificate(c)
11     }
12 }
```

Abbildung 17 zeigt die Modelle und ihre Beziehungen.

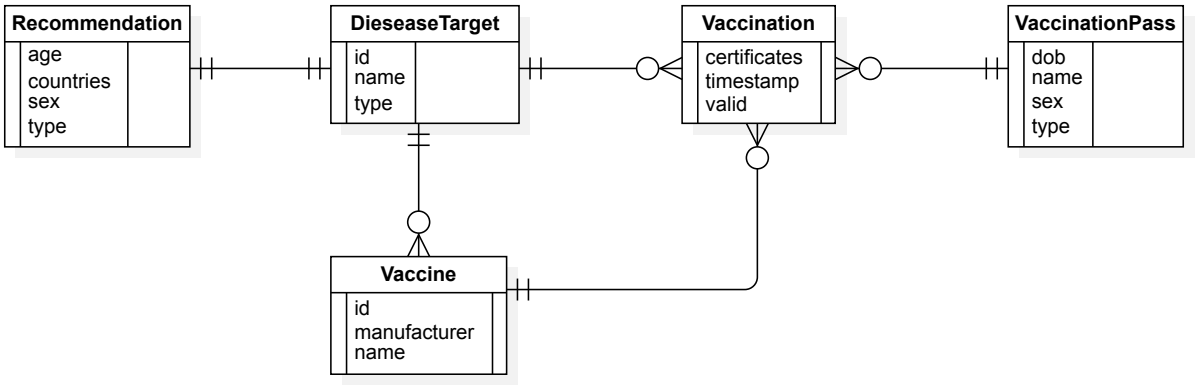


Abbildung 16: ER Diagramm für den Frontend Objekt Graph

5.1.3 *Views*

Die Views wurden in SwiftUI umgesetzt. Quellcode 11 zeigt die Übersicht über die Impfungen von einem Pass an.

Quellcode 11: PassView.swift

```
1  struct PassView: View {
2      @Environment(\.managedObjectContext) var context
3
4      @FetchRequest(
5          sortDescriptors: [NSSortDescriptor(keyPath: \
               VaccinationPass.name, ascending: true)],
6          animation: .default)
7      private var passes: FetchedResults<VaccinationPass>
8
9      @ObservedObject var defaultPass: DefaultPass
10     @State private var name = ""
11     @State private var isShowingAlert = false
12
13     var body: some View {
14         NavigationView {
15             VaccinationList(controller:
                   VaccinationListController(pass: defaultPass.pass)
                   )
16                 .navigationTitle(defaultPass.pass?.name ?? "No
                       Pass")
17                 .toolbar {
```

```
18                      ToolbarItem(placement: .navigationBarLeading
                            ) {
19                          Menu {
20                              ForEach(passes) { pass in
21                                  Button {
22                                      selectPass(pass)
23                                  } label: {
24                                      Text("\(pass.wrappedType.
                                            emoji)   \(pass.name)")
25                                  }
26                              }
27                          } label: {
28                              Label("Pass Menu", systemImage: "
                                    person.crop.rectangle.stack.fill"
                                    )
29                          }
30                      }
31                      ToolbarItem(placement: .
                            navigationBarTrailing) {
32                          NavigationLink {
33                              if let pass = defaultPass.pass {
34                                  AddVaccination(pass: pass,
                                        showAlert: $isShowingAlert)
35                              } else {
36                                  AddPassWrapper()
37                              }
38                          } label: {
39                              Label("Add Item", systemImage: "plus
                                    ")
40                          }
41                      }
42                  }
43                  .alert(isPresented: $isShowingAlert) {
44                      Alert(
45                          title: Text("Duplicate Vaccination"),
46                          message: Text("A Vaccination Entry for
                                the selected Disease Target exists
                                already.")
47                      )
48                  }
49          }
50      }
51
52      func selectPass(_ pass: VaccinationPass) {
53          print("Selected Pass: \(pass.name)")
```

```
54            defaultPass.selectNewPass(pass, in: context)
55        }
56    }
```

Die View zeigt die Liste der Impfungen unter einer Navigation Bar[2] (Nav Bar) an. Die Nav Bar kann mit dem `.toolbar()` Modifier angepasst werden. Es werden zwei `ToolBarItems` hinzugefügt. Auf der linken Seite das Menü zum Auswählen des aktiven Passes und auf der rechten Seite den Knopf zum Hinzufügen neuer Impfungen.
Um Fehler, die beim Hinzufügen neuer Impfungen auftreten, darstellen zu können, wird der `VaccinationList` die `isShowingalert` Variable als Binding übergeben. Dieses Binding bewirkt, dass die Variable in der `VaccinationList` geändert werden kann und diese Änderungen anschließend in der `PassView` reflektiert wird. In diesem konkreten Beispiel bedeutet dies, dass ein Fenster mit einem Fehler dargestellt wird, wenn die Variable `isShowingAltert` auf wahr gesetzt wird.

5.1.4 *Networking*

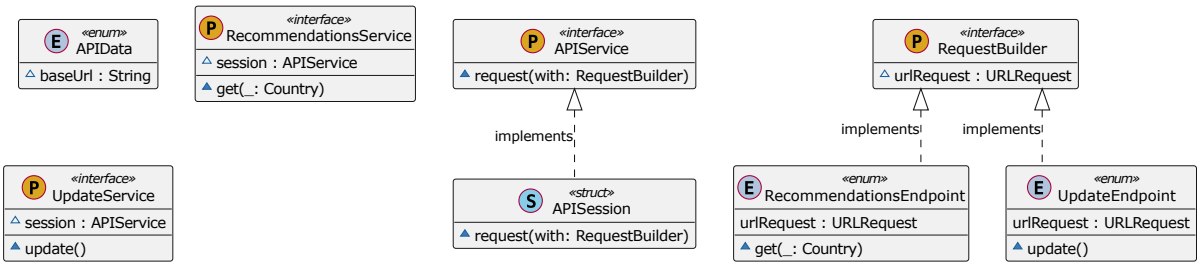


Abbildung 17: Networking Klassendiagramm

Abbildung 16 zeigt die wichtigen Objekte der Networking Architektur. Das APISession Objekt ist der Kern der Networking Architektur und beschreibt den grundlegenden Ablauf einer Anfrage und der Auswertung der Antwort. Die Architektur benutzt das Builder Pattern.

> [The Builder pattern separates] the construction of a complex object from its representation so that the same construction process can create different representations(Gamma, Helm, Johnson, Vlissides 1994, S. 97).

[2]https://developer.apple.com/design/human-interface-guidelines/components/navigation-and-search/navigation-bars/

Dabei fungieren die Endpoint Objekte als Builder, die von der `request()` -Methode verarbeitet werden können. Dadurch muss der Ablauf der Decodierung der JSON Antworten nur einmal geschrieben werden.
Es wurden verschiedene Interfaces erstellt, in Swift `protocols` genannt, um den Code leichter testen können. So kann z.B. für jedes Interface eine spezielle Test Version erstellt werden, um damit andere Teile der Networking Architektur isoliert zu testen.

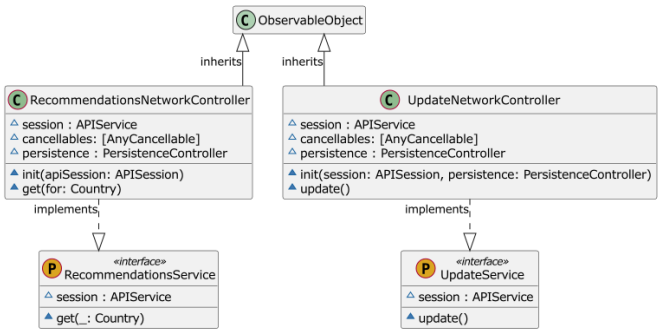


Abbildung 18: Networking Controller Klassendiagramm

Um einen Service im Frontend nutzen zu können, implementiert ein Controller das entsprechende Service Interface und behandelt dort das Verarbeiten der Ergebnisse. Abbildung 17 zeigt die implementierten Controller der Service Protokolle.

Die Networking Architektur basiert auf einer Implementierung von dem Blog Swift Compiled (Brown 2020) und wurde für die MVC Architektur angepasst.

5.1.5 *CovPass App CBORWebToken Implementierung*

Die CovPass App ist eine von IBM entwickelte App zum Speichern und Überprüfen von COVID Zertifikaten. Die App ist Open-Source unter der Apache 2.0 Lizenz veröffentlicht. Diese Lizenz erlaubt das Modifizieren und Verteilen des Source Codes für kommerziellen oder privaten Nutzen.
Explizit werden die Objekte `Base45Coder`, `Compression`, `CoseSign1Message`, `ExtendedCBORWebToken` und `CBORWebToken` genutzt, sowie alle Objekte, die von den genannten Objekten benötigt werden.

Quellcode 12 zeigt den Decodierablauf, die `Scanner.extractPayload()` Methode decodiert die Text Repräsentation des QR-Codes. Dafür wird, wie im Kapitel 2.1.1 beschrieben, der Base45 codierte Text decodiert und in eine CoseSign1Message umgewandelt. Der Payload dieser Nachricht wird anschließend mit der SwiftCBOR Library decodiert und in ein JSON umgewandelt. Diese JSON Datei wird wiederum mit Hilfe der Codable Klasse in einen CBORWebToken umgewandelt.

Der CBORWebToken enthält alle wichtigen Daten über die Impfung und kann so in der restlichen App weiterverwendet und im Core Data Store gespeichert werden.

Quellcode 12: Scanner.extractPayload() Methode

```
1  static func extractPayload(payload: String) throws ->
       ExtendedCBORWebToken {
2      if payload.hasPrefix("HC1:") {
3          let base45Decoded = try Base45Coder.decode(String(
               payload.dropFirst(4)))
4          guard let decompressedPayload = Compression.decompress(
               Data(base45Decoded)) else {
5              throw ScannerError.unsupportedPayload
6          }
7
8          let cosePayload = try CoseSign1Message(
               decompressedPayload: decompressedPayload)
9
10         let cborDecodedPayload = try CBOR.decode(cosePayload.
               payload)
11         let certificateJson = decode(cborObject:
               cborDecodedPayload)!
12
13         let serializedJSON = try JSONSerialization.data(
               withJSONObject: certificateJson as Any)
14         let certificate = try JSONDecoder().decode(CBORWebToken.
               self, from: serializedJSON)
15
16         return ExtendedCBORWebToken(vaccinationCertificate:
               certificate, vaccinationQRCodeData: payload)
17
18     } else {
19         throw ScannerError.unsupportedPayload
20     }
21 }
```

5.2 Backend

Durch die simple Natur von Lambda Funktionen besteht das Backend lediglich aus 2 Python Funktionen.

Quellcode 13 zeigt die /update Route. Hier wird lediglich der Inhalt der dynamoDB gescannt

und als JSON an das Endgerät geschickt.

Quellcode 13: /update Route

```
def update(event, context):
    scanRes = table.scan()
    items = scanRes['Items']

    body = {
        "items": items
    }

    response = {"statusCode": 200, "body": json.dumps(body, cls=
       DecimalEncoder)}

    return response
```

Quellcode 14 zeigt die /recommend Route. Die Route erwartet einen Parameter zur Auswahl des Landes oder `all` um Empfehlungen für alle Länder zu erhalten. Basierend auf dem Parameter wird die dynamoDB entweder nach einem Land gefiltert oder ungefiltert an das Endgerät gesandt.

Quellcode 14: /recommend Route

```
def recommend(event, context):
    parameters = event['pathParameters']
    country = parameters['country']

    if country != "all":
        filter_expr = (Attr('countries').contains(country) |
           Attr('countries').not_exists())
        scanRes = table.scan(FilterExpression=filter_expr)
    else:
        scanRes = table.scan()

    recommendations = scanRes['Items']

    body = {
        "recommendations": recommendations
    }

    response = {"statusCode": 200, "body": json.dumps(body, cls=
       DecimalEncoder)}
    return response
```

Die tatsächliche Auswertung findet, wie in 5.1.4 beschrieben, auf dem Endgerät statt.

5.3 Tests

Da der Fokus dieser Arbeit auf der Frontend Applikation liegt, wurde lediglich eine Teststrategie für diese entworfen.
Zuerst wurden Kern Elemente der Businesslogik definiert. Dazu zählen das Erreger Modell, die QR-Code Scanner Funktionalität und die APISession.

Um das Erreger Modell zu testen, wurden Unit Tests mit dem XCTest Framework geschrieben. Diese Unit Tests decken das Hinzufügen und Updaten bestehender Erreger, sowie die wichtigsten Wrapper Methoden ab.
Der QR-Code Scanner wird auf das erfolgreiche Extrahieren eines validen Payloads, sowie das Ausgeben eines Fehlers bei invalidem Payloads getestet.
Um die API Session zu testen, wurde von dem modularen Aufbau dieser Gebrauch gemacht. Es wurde ein MockService geschrieben, der den UpdateService implementiert. Mit Hilfe dieses MockServices kann dann die APISession einfach getestet werden.

Quellcode 15 zeigt einen Test des Erreger Models. Der Test verifiziert, das beim Hinzufügen neuer Erreger Duplikate erkannt und automatisch die bestehenden Erreger aktualisiert werden.

Quellcode 15: UpdateExistingTarget Test

```
1  func testUpdateExistingTarget() throws {
2      let newTarget = DiseaseTargetDTO(targetID: "000000001", type
           : "human", vaccines: [], name: "Updated Target 1")
3
4      let initialCount = try context.count(for: targetRequest)
5      XCTAssertEqual(initialCount, 1)
6
7      persistence.addNewTarget(newTarget, context: context)
8
9      let count = try context.count(for: targetRequest)
10     XCTAssertEqual(count, 1)
11
12     let r = targetRequest
```

```
13      r.predicate = NSPredicate(format: "name == %@", newTarget.
            name)
14
15      if let t = try? context.fetch(r).first {
16          XCTAssertEqual(t.name, newTarget.name)
17      } else {
18          XCTFail("Missing Target")
19      }
20  }
```

Mit diesen Tests kann eine Abdeckung von 24.8 % der App erreicht werden.

5.4 Deployment

Neben automatisierten User Tests wurde die App über App Store Connect und Testflight als Beta veröffentlicht und an Tester aus verschiedenen Altersgruppen verteilt.
Insgesamt haben 7 Testpersonen die App in 30 Sessions über 4 Beta Versionen hinweg getestet und mündlich Feedback gegeben.
Das Feedback war durchweg positiv. Es wurde die grundlegende Bedienbarkeit gelobt, jedoch waren nicht alle Funktionen sofort ersichtlich. Das Menü zum Wechseln des Impfpasses ist nicht intuitiv für alle Nutzer:innen verständlich und das Erstellen neuer Pässe ist in den Einstellungen versteckt.

Das Backend wird mit dem serverless Framework über die AWS Cloud bereitgestellt.

Die App kann über folgenden Link auf iOS 15 Geräten und M1 Macs über TestFlight getestet werden:

https://testflight.apple.com/join/pZxa2bRm

Die neuste Betaversion ist aufgrund von Apple Vorschriften nur 90 Tage gültig und kann danach nicht mehr installiert werden.

Für das Backend wurde ein kleiner Beispieldatensatz entworfen. Dieser enthält mehrere unterstützte Erreger mit einer Abdeckung verschiedener Typen und Empfehlungsarten. Dieser Datensatz erfüllt lediglich den Zweck der Präsentation, es wird keine Korrektheit der Impfungen oder Empfehlungen garantiert.

Der komplette Beispieldatensatz ist in Anhang 2 gelistet.

6 Validierung der Anforderungen

Dieses Kapitel zeigt die Ergebnisse der Frontend Implementierung. Es werden die umgesetzten Features der App beschrieben und auf die funktionale Anforderungen aus Kapitel 3 eingegangen.

6.1 Onboarding

> Onboarding is a virtual unboxing experience that helps users get started with an app. (Google 2022)

Die Onboarding Ansicht wird beim ersten Starten der App gezeigt (s. Abbildung 19). Sie fordert die Nutzer:innen dazu auf, einen neuen Impfpass zu erstellen. Dort kann den Namen, das Geschlecht und den Geburtstag eingegeben werden.

Es wird weiterhin darauf hingewiesen, dass in den Einstellungen später weitere Impfpässe erstellt werden können.

Der Onboarding Prozess vereinfacht die Benutzung der App und trägt damit einen Teil zum Erfüllen von NFA#05 bei.

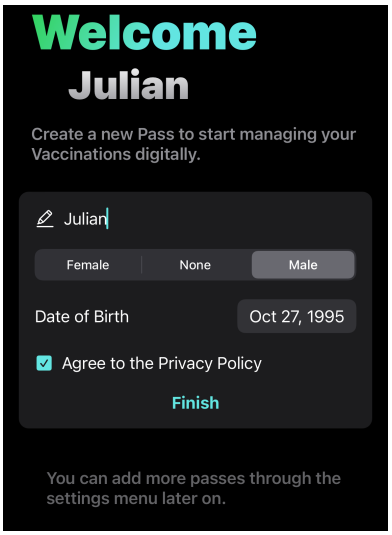


Abbildung 19: Onboarding Screenshot

6.2 Impfungen

Nach dem Onboarding Prozess wird die Impfübersicht angezeigt (s. Abbildung 20). Diese Übersicht listet alle Impfungen des aktiven Passes auf und erlaubt das Entfernen der Impfeinträge. In der Navbar stehen den Nutzer:innen zwei Optionen zur Verfügung. Auf der einen Seite kann der aktiven Pass gewechselt werden, auf der anderen Seite kann der Dialog zum Hinzufügen neuer Impfungen gestartet werden.

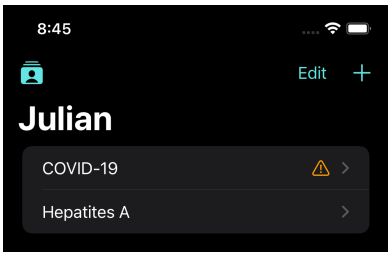


Abbildung 20: Impfliste mit ungültiger Impfung

Die Liste zeigt neben den Impfungen auch abgelaufene oder ungültige Impfungen an. Diese werden mit einem gelb-orangen Dreieck hervorgehoben. Das Symbol und die Farbe sind bekannte Warnsymbole und zeigen so schnell, dass bei dieser Impfung die Gültigkeit überprüft werden muss.

Wenn der oder die Nutzer:in einen Impfeintrag anklickt, wird er oder sie auf eine Detail Übersichtsseite weitergeleitet (s. Abbildung 21). Dort kann bei jeder Impfung eingesehen werden, welcher Impfstoff verwendet wurde und gegen welchen Erreger dieser wirkt. Auch wird das Datum der Impfung gezeigt. Bei Impfungen mit Zertifikaten, wird das jeweils neuste Zertifikat als QR-Code groß in der Mitte gezeigt. Unter dem QR-Code werden weitere Informationen zu dem Zertifikat gezeigt, z.B. ob die Impfung gültig ist, für welche Person das Zertifikat ausgestellt wurde und wann es abläuft. Die Detailseite erfüllt damit FA#05.
Um FA#08 zu erfüllen überprüft die App beim Darstellen der Liste jede Impfung auf ihre Gültigkeit basierenden auf den Informationen aus dem Zertifikat.

Abbildung 21: Detailübersicht einer Impfung

Das Menü zum Wechseln des aktiven Passes benutzt Emojis zum Darstellen des Passtyps und verbessert somit die Verständlichkeit und Eindeutigkeit der Passnamen (s. Abbildung 22).

Über den Button zum Hinzufügen wird der oder die Nutzer:in zu einem Formular gebracht, das es erlaubt, neue Impfungen hinzuzufügen (s. Abbildung 23). Dort kann aus einer Liste von unterstützten Erregern ausgewählt werden. Für den gewählten Erreger werden dann die Impfstoffe geladen

und zur Auswahl bereitgestellt.

Um das Formular intuitiv zu gestalten, bleibt die Liste der Impfstoffe deaktiviert, bis ein Erreger ausgewählt wurde und der Knopf zum Speichern bleibt deaktiviert, bis das Formular vollständig ausgefüllt wurde.

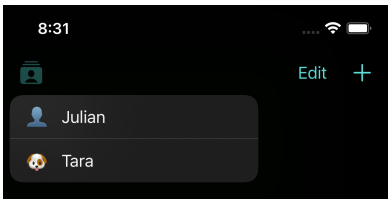


Abbildung 22: Menü zum Wechseln des aktiven Impfpasses

Neben den manuellen Einträgen kann durch das Einscannen eines Impfzertifikats das Formular automatisch ausgefüllt werden. Bei einem Klick auf den `Scan Certificate`-Button wird eine Kameraansicht geöffnet. Mit dieser können Nutzer:innen einen QR-Code einscannen. Es können mehrere Zertifikate hintereinander eingescannt werden. Wenn das gleiche Zertifikat mehrmals eingescannt wurde, wird im Formular ein Fehler präsentiert. Die Anforderungen FA#03 und FA#04 sind damit erfüllt.

Die bei der Impfübersicht und dem Hinzufügen-Formular beschriebenen Funktionen decken FA#01 ab.

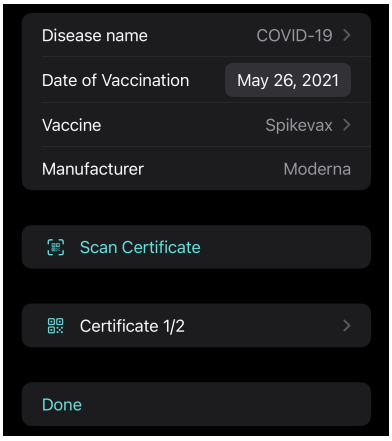
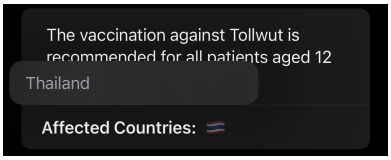


Abbildung 23: Formular zum Hinzufügen eines Impfeintrags

6.3 Empfehlungen

Von der Impfübersicht kann der oder die Nutzer:in über die Tab Bar zu den Empfehlungen navigieren. Dort werden aktuelle Empfehlungen für ein gewähltes Land angezeigt. Über die Fahne in der Navbar können die Empfehlungseinstellungen geöffnet werden, um das Land für die Empfehlungen zu ändern oder Empfehlungen für alle Länder zu erhalten.
Diese Ansicht erfüllt die Anforderungen FA#09 und FA#10.

(a) Eine Impfempfehlungen



(b) Der Name eines Landes wird bei Anklicken der Fahne angezeigt

Abbildung 24: Impfempfehlungen

Wie in Abbildung 24 (a) zu sehen, bestehen die Empfehlungen aus einem beschreibenden Text. Dieser Text zeigt den Namen des Erregers und, wenn in der Empfehlung enthalten, das Alter und das Geschlecht an. Zusätzlich zu dem Text gibt es eine Liste der betroffenen Länder. Die Länder werden als Flaggen nebeneinander aufgereiht, damit der oder die Nutzer:in einfach erkennen kann, auf welches Land sich die Empfehlungen beziehen.
Wenn eine Flagge unbekannt ist, kann durch ein Klicken auf diese ein Fenster mit dem Namen angezeigt werden (s. Abbildung 24 (b)).

Es werden nur Empfehlungen für den aktiven Pass angezeigt.

6.4 Einstellungen

Die letzte Option in der Tab Bar sind die Einstellungen. In dieser Ansicht werden den Nutzer:innen eine Liste an Optionen geboten:

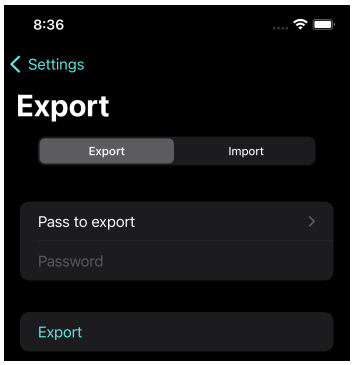


Abbildung 25: Die Export Ansicht

Pass Overview: Die Passübersicht erlaubt es existierende Pässe zu bearbeiten, Pässe zu löschen und neue Pässe zu erstellen. Die Pässe werden in einer Liste analog zu den Impfungen dargestellt. Das Bearbeiten der Pässe umfasst das Ändern des Namens, des Geschlechts und des Geburtsdatums. Beim Erstellen könnten Fehler auftreten und es ist einfacher, Pässe

bearbeiten zu können als einen komplett neuen Pass erstellen zu müssen.
Die Passübersicht deckt die funktionale Anforderung FA#02 ab.

Export: In der Exportübersicht (s. Abbildung 25), kann der oder die Nutzer:in einen bestehenden Pass verschlüsselt exportieren und einen bereits exportierten Pass wieder importieren. Diese Funktion deckt FA#06 und FA#07 ab.
Nach dem Verschlüsseln eines Impfpasses wird ein Share Sheet[1] geöffnet, dieses erlaubt es den Pass direkt über den bevorzugten Kommunikationsweg zu teilen oder in eine Cloud-Lösung der Wahl hochzuladen.

Recommendations: In der Empfehlungsansicht der Einstellungen können die Empfehlungen angepasst werden. Da nur die Länder als variable Empfehlungsfilter implementiert sind, tauchen nur diese in der Liste auf. Es ist aber denkbar, dass an diesem Punkt noch andere Einstellungen untergebracht werden. Abbildung 26 zeigt die Liste der unterstützten Länder.

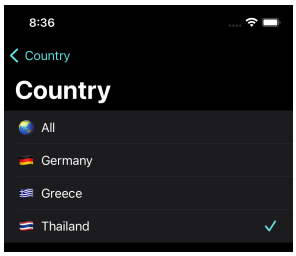


Abbildung 26: Die Liste der unterstützten Länder

Licenses: Unter der Licenses Ansicht wird eine Liste aller Lizenzen von verwendetem Quellcode und Frameworks angezeigt.

Privacy Policy: Unter der Privacy Policy Ansicht kann die Datenschutzerklärung auf englisch gelesen werden.

6.5 Zusammenfassung

Es konnten alle Muss- und Soll-Anforderungen umgesetzt werden.

Wie in Kapitel 4 und 5 beschrieben, kann das Backend die App mit neuen Empfehlungen und Impfstoffen versorgen und erfüllt somit FA#12 und FA#13.
Das Backend wurde außerdem in einer serverlosen Architektur entworfen um leicht skalierbar zu sein. Es erfüllt damit NFA#02.

[1] https://developer.apple.com/design/human-interface-guidelines/components/menus-and-actions/activity-views

Die App vearbeitet alle personenbezogenen Informationen auf dem Engerät und hält sich an die Apple Human Interface Guidelines. Dies erfüllt NFA#01 und NFA#04. Zertifikate lassen sich bei einem Neustart der App in 2 Klicks präsentieren, es wird damit BFA#03 erfüllt.

NFA#05 ist teilweise erfüllt, das Feedback der Nutzer:innen war grundlegend positiv, aber nicht jede Funktionen war für jede Altersgruppe sofort ersichtlich.

7 Fazit und Ausblick

Das Ziel dieser Arbeit war der Entwurf und die Entwicklung eines digitalen Impfpasses names Vaccinapp. Zu Beginn wurden die wichtigen Grundlagen des COVID-Zertifikats beschrieben sowie einen Einstieg in einen modernen und skalierbaren Tech Stack für iOS Entwicklung gegeben.

Anschließend wurden die Anforderungen an die App festgehalten und ein Design Prototyp angefertigt. Bei dem Design wurde besonderen Wert darauf gelegt etablierte Designelemente zu verwenden, um den Nutzer:innen eine bekannte Umgebung zu präsentieren.
Auf dieser Basis wurde eine App entworfen, die es erlaubt Impfpässe für verschiedene Personen und Haustiere zu verwalten. Der oder die Nutzer:in kann in einem digitalen Pass Impfungen verwalten, Zertifikate importieren und bekommt Empfehlungen für bestimmte Impfungen vorgeschlagen, basierend auf Personenbezogenen Daten, wie dem Alter und Geschlecht, sowie dem Aufenthaltsort.

Das Ergebnis der Arbeit erfüllt, wie in Kapitel 6 *Validierung der Anforderungen* gezeigt, alle gestellten funktionalen Anforderungen.

7.1 Ausblick

Obwohl die App allen Ansprüchen der Anforderungsanalyse gerecht wurde, bedarf es noch einiger Änderungen bevor sie veröffentlicht werden kann.
Der Fokus dieser Arbeit lag auf dem Frontend, das Backend ist also dementsprechend unausgereift. Die grundlegenden Funktionen wurden umgesetzt, aber die Datenbeschaffung ist nicht definiert worden.

Abbildung 27 zeigt eine mögliche Backend Architektur, die verschiedene Defizite der aktuellen Implementierung beseitigt.

Eine automatische Lösung wird benötigt, um schnell auf neue Empfehlungen einzugehen. Die STIKO veröffentlich regelmäßig neue Empfehlungen auf einem RSS Feed[1], aber diese Empfehlungen sind in Form einer PDF. Es bedarf also einer Anwendung, die aus einer PDF strukturierte Tabelleneinträge auslesen und auswerten kann. Es kann nicht garantiert werden, dass diese PDFs immer der gleichen Struktur entsprechen. Dies kann höheren Wartungsaufwand der Anwendung bedeuten. Alternativ bietet die WHO auf einer zentralen Webseite verschiedene Tabellen mit Empfehlungen an[2]. Diese sind jedoch auch in PDF-Form, es treten demnach die gleichen Probleme wie bei dem STIKO RSS Feed auf.

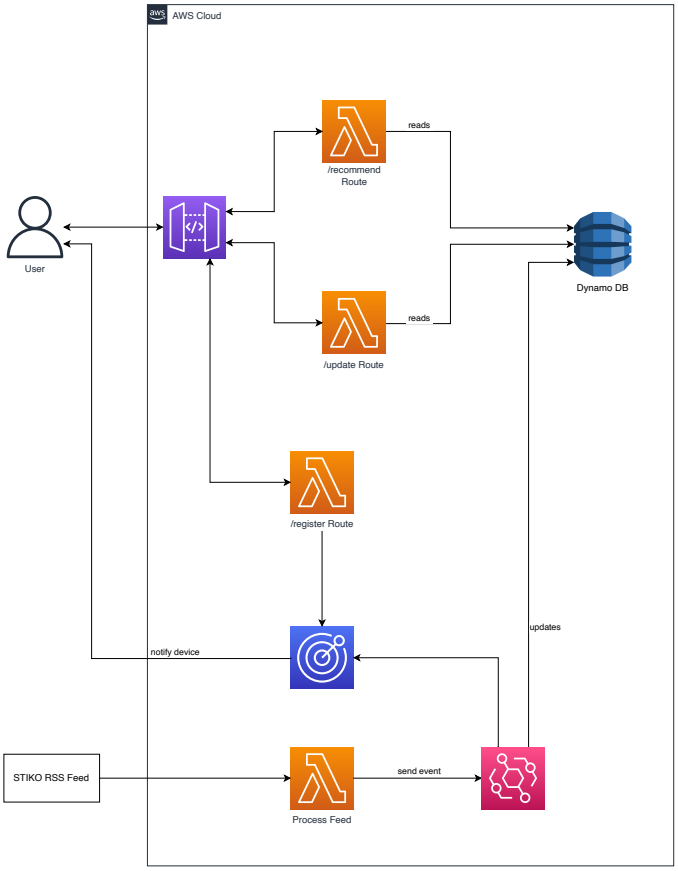


Abbildung 27: Ausblick Backend Architektur

Obwohl die Empfehlungen mit den Geschlechts-, Alters- und Landesfiltern recht flexibel wirken, gibt es noch weitere nicht implementierte Edge Cases. Babys und Kleinkinder fallen durch die grobe Rasterung des Altersfilter durch. Impfempfehlungen bei weniger als einem Jahr sind nicht möglich und bedürfen mehr Arbeit. Bei solch heiklen Zielgruppen ist es empfehlenswert, immer einen Arzt aufzusuchen, um professionell beraten zu werden.
Die Landesfilter sind noch sehr simpel gehalten und beziehen sich lediglich auf ein spezifisches Land. Es ist also zu prüfen, ob eine feinere Aufteilung sinnvoll ist, da nicht zwangsweise in jedem Teil eines Landes die gleichen Risiken gelten. Es wäre möglich die Empfehlungen mit Texten zu erweitern, die die Risikogebiete beschreiben, ähnlich wie es auf der Webseite des

[1] https://www.rki.de/DE/Service/RSS/RSS_node.html
[2] https://www.who.int/teams/immunization-vaccines-and-biologicals/policies/who-recommendations-for-routine-immunization—summary-tables

Tropeninstituts[3] der Charité getan wird.

Für das Frontend müssen noch anwenderfreundlichere Fehler implementiert werden. Bei Aktionen, die größere Änderungen im Hintergrund ausführen, wie z.B. das Hinzufügen neuer Zertifikate, sollen haptische Feedbacks genutzt werden, um die Nutzer:innen einfach über den Erfolg oder Misserfolg zu benachrichtigen. Die Taptic Engine eignet sich sehr gut, um verschiedene leichte Vibrationsmuster abzuspielen. Außerdem muss der Onboarding Prozess weiter ausgebaut werden, es werden zwar alle Einstellungen, die zum Benutzen der App notwendig sind, abgearbeitet. Jedoch ist es für neue oder weniger affine iOS Nutzer:innen nicht sofort ersichtlich, wofür verschiedene Icons zuständig sind und wie Impfungen hinzugefügt werden. Der Onboarding Prozess kann also um weitere Ansichten ergänzt werden, die den Nutzer:innen einmalig erklären wo neue Impfungen oder Pässe hinzufügt werden und wie zwischen den Pässen gewechselt wird.

[3] https://tropeninstitut.de/ihr-reiseziel/thailand#Malaria

Abbildungsverzeichnis

Quellcodeverzeichnis

Glossar

Begriff	Synonym	Erklärung
API	Application Programming Interface	Programmierschnittstelle
App	Vaccinapp	Frontend iOS Application
Backend		Der Server der die unterstützten Impfungen verwaltet und dem Frontend bereitstellt
Business Logic	Geschäftslogik	Der Teil des Codes der Anwendungspezifische Logik beschreibt
Completion Handler Closures		Eine anonyme Funktion, die beim Beenden einer Aufgabe ausgeführt wird
Computated Property		Ein komplexes Attribut, dass den Wert während der Laufzeit erst berechnet
Context	NSManagedContext	Ein Layer zwischen der SQLite Datenbank und dem Entwickler
Domäne		Themenbereich eines Software Systems
Erreger	DiseaseTarget	Ein Erreger gegen den geimpft werden kann
Impfempfehlung	Recommendation	Eine Empfehlung für eine Impfung
Impfpass	VaccinationPass	Ein digitaler Pass um Impfungen personenbezogen zu speichern.
Impfstoff	Vaccine	Der explizite Impfstoff der bei einer Impfung benutzt wurde

Begriff	Synonym	Erklärung
Impfung	Vaccination	Eine explizite Impfung einer Person
Impfzertifikat	Health Certificate, Corona QR-Code	Ein Nachweis für eine COVID-19 Impfung, Genesung oder ein Testergebnis
NSPredicate		Core Data Query
Object Relational Mapping	ORM	Erlaubt einfaches Abbilden von Attributen eines Objektes in einer relationalen Datenbank
Reed-Solomon Codes	RS Codes	Eine Sammlung von Fehlerberichtigungsalgorithmen von Irving S. Reed and Gustave Solomon
User Flow		Der Ablauf von Aktivitäten, die dem oder der Nutzer:in von einem System bereitgestellt werden
Verwalten		Das Hinzufügen, Lesen, Bearbeiten und Löschen einer Entität

Abkürzungen

Abkürzung	Erklärung
API	Application Programming Interface
CWA	Corona Warn App
DSGVO	Datenschutgrundverordnung
DTO	Data Transfer Object
MVC	Model-View-Controller
MVP	Minimal Viable Product
QR-Code	Quick Response Code
RS-Codes	Reed-Solomon Codes
UI	User Interface
v.n.l.r	von links nach rechts
XC	Xcode
z.B.	zum Beispiel

Anhang

Anhang 1: Beispieldatensatz

```
    data = """[{
  "name": "COVID-19",
  "targetID": "840539006",
  "type": "human",
  "vaccines": [
    {
      "id": "EU/1/20/1528",
      "name": "COMIRNATY",
      "manufacturer": "Biontech AG"
    },
    {
      "id": "EU/1/20/1507",
      "name": "Spikevax",
      "manufacturer": "Moderna"
    }
  ],
  "age": 3
},
{
  "name": "Rabies",
  "targetID": "000000001",
  "type": "human",
  "vaccines": [
    {
      "id": "EU/0/15/0001",
      "name": "Rabipur",
      "manufacturer": "ACME Inc."
    }
  ],
  "age": 12,
  "countries": ["thailand"]
},
```

```
{
  "name": "Hepatites A",
  "targetID": "000000003",
  "type": "human",
  "vaccines": [
    {
      "id": "EU/0/15/0002",
      "name": "Avaxim",
      "manufacturer": "ACME Inc."
    },
    {
      "id": "EU/0/15/0003",
      "name": "Havrix",
      "manufacturer": "ACME Inc."
    },
    {
      "id": "EU/0/15/0004",
      "name": "VAQTA",
      "manufacturer": "ACME Inc."
    },
    {
      "id": "EU/0/15/0005",
      "name": "Viatim",
      "manufacturer": "ACME Inc."
    },
    {
      "id": "EU/0/15/0006",
      "name": "Twinrix",
      "manufacturer": "ACME Inc."
    }
  ],
  "age": 12,
  "countries": ["greece", "thailand"]
},
{
  "name": "Cat Flu",
  "targetID": "100000001",
  "type": "cat",
  "vaccines": [
    {
      "id": "EMEA/V/C/000090",
      "name": "Purevax RCP",
      "manufacturer": "Boehringer Ingelheim Vetmedica GmbH"
    }
  ]
```

```
},
{
  "name": "Rabies",
  "targetID": "200000001",
  "type": "dog",
  "vaccines": [
    {
      "id": "EMEA/V/C/005090",
      "name": "Purevax RCP",
      "manufacturer": "Intervet International B.V."
    }
  ]
```

Der Quellcode, der im Verlauf dieser Arbeit enstanden ist, ist auf einem USB-Stick beigelegt oder kann unter folgender URL angesehen werden:

https://github.com/julianveerkamp/vaccinapp

Diese Arbeit wurde mithilfe von Pandoc und dem Eisvogel[1] Template verfasst.

[1]https://github.com/Wandmalfarbe/pandoc-latex-template

Quellen

ADC, Denso, 2012. QR Code Essentials. [online]. Oktober 2012. Verfügbar unter: https://www.denso-adc.com/download/qr-code-white-paper/

AMAZON WEB SERVICES, Inc., 2021b. Whitepaper: Security Overview of AWS Lambda. [online]. 2021. [Zugriff am: 25 Mai 2022]. Verfügbar unter: https://aws.amazon.com/de/lambda/security-overview-of-aws-lambda/

AMAZON WEB SERVICES, Inc., 2021a. Whitepaper: Security Overview of Amazon API Gateway. [online]. 2021. [Zugriff am: 2 Juni 2022]. Verfügbar unter: https://docs.aws.amazon.com/whitepapers/latest/security-overview-amazon-api-gateway/security-overview-amazon-api-gateway.pdf

APPLE, Inc., 2018. Cocoa Core Competencies - Object graph. [online]. 2018. [Zugriff am: 29 Juni 2022]. Verfügbar unter: https://developer.apple.com/library/archive/documentation/General/Conceptual/DevPedia-CocoaCore/ObjectGraph.html

APPLE, Inc., [kein Datum] d. Combine Documentation. [online]. [Zugriff am: 29 Juni 2022]. Verfügbar unter: https://developer.apple.com/documentation/combine/receiving-and-handling-events-with-combine

APPLE, Inc., [kein Datum] c. Swift Language Guide. [online]. [Zugriff am: 3 Juni 2022]. Verfügbar unter: https://docs.swift.org/swift-book/LanguageGuide/TheBasics.html

APPLE, Inc., [kein Datum] a. Displaying Data in a List. [online]. [Zugriff am: 26 Mai 2022]. Verfügbar unter: https://developer.apple.com/tutorials/app-dev-training/displaying-data-in-a-list

APPLE, Inc., [kein Datum] b. UIKit Essentials - Creating a List View. [online]. [Zugriff am: 26 Mai 2022]. Verfügbar unter: https://developer.apple.com/tutorials/app-dev-

training/creating-a-list-view

APPLE, Inc., [kein Datum] e. Core Data Documentation. [online]. [Zugriff am: 29 Juni 2022]. Verfügbar unter: https://developer.apple.com/documentation/coredata

BROWN, Alex, 2020. MVVM & Networking with SwiftUI and Combine. [online]. 19 Juni 2020. [Zugriff am: 3 Juni 2022]. Verfügbar unter: https://www.swiftcompiled.com/mvvm-swiftui-and-combine/

DSGVO, 2018. Daten Schutz Grund Verordnung. [online]. 25 Mai 2018. [Zugriff am: 1 Juni 2022]. Verfügbar unter: https://dsgvo-gesetz.de/

EHEALTH NETWORK, 2022b. Guidelines on Technical Specifications for EU Digital COVID Certificates - JSON Schema Specification. [online]. Juni 2022. Verfügbar unter: https://health.ec.europa.eu/system/files/2021-06/covid-certificate_json_specification_en_0.pdf

EHEALTH NETWORK, 2022a. Guidelines on Technical Specifications for EU Digital COVID Certificates Volume 2 - EU Digital COVID Certificate Gateway. [online]. Juni 2022. Bd. 2. Verfügbar unter: https://health.ec.europa.eu/system/files/2022-07/digital-covid-certificates_v2_en.pdf

EHEALTH NETWORK, 2022c. Guidelines on Technical Specifications for EU Digital COVID Certificates Volume 3 - Interoperable 2D Code. [online]. Februar 2022. Bd. 3. Verfügbar unter: https://health.ec.europa.eu/system/files/2022-07/digital-covid-certificates_v3_en.pdf

EHEALTH NETWORK, 2022d. Guidelines on Technical Specifications for EU Digital COVID Certificates Volume 4 - EU Digital COVID Certificate Applications. [online]. Februar 2022. Bd. 4. Verfügbar unter: https://ec.europa.eu/health/system/files/2022-02/digital-covid-certificates_v4_en_0.pdf

EUROPEAN COMMISSION, 2021. Durchführungsbeschluss (EU) 2021/1073. [online]. 28 Juni 2021. [Zugriff am: 23 Mai 2022]. Verfügbar unter: https://eur-lex.europa.eu/legal-content/DE/TXT/?uri=CELEX:32021D1073

FARBTONKARTE, 2022. Wirkung, Assoziationen und Bedeutung der Farbe Blau. [online]. 2022. [Zugriff am: 3 Juni 2022]. Verfügbar unter: https://farbtonkarte.de/farbe-blau/

FOWLER, Martin, 2015. *Patterns of Enterprise Application Architecture*. Pearson Education Limited. ISBN 978-0-321-12742-6

GAMMA, Erich, Richard HELM, Ralph JOHNSON und John VLISSIDES, 1994. *Design patterns*. Addison Wesley. ISBN 978-0-201-63361-0

GOOGLE, Inc., 2022. Material Design - Onboarding. [online]. 2022. [Zugriff am: 30 Juni 2022]. Verfügbar unter: https://material.io/design/communication/onboarding.html

LAPLANTE, Phillip A., 2009. *Requirements Engineering for Software and Systems*. 2nd. CRC Press,Auerbach Publications. ISBN 978-1-4200-6468-1

SHAI MISHALI, Florent Pillet und Marin TODOROV, 2021. *Combine: Asynchronous Programming with Swift* [online]. 3rd. Razorware LLC. Verfügbar unter: https://www.raywenderlich.com/books/combine-asynchronous-programming-with-swift

WALS, Donny, 2022. *Practical Core Data: A modern Guide to the Core Data framework* [online]. 2022. donnywals.com. Verfügbar unter: https://donnywals.gumroad.com/l/practical-core-data

YAPSTUDIOS, 2022. SQLite version (bundled with OS). [online]. 4 Mai 2022. [Zugriff am: 26 Mai 2022]. Verfügbar unter: https://github.com/yapstudios/YapDatabase/wiki/SQLite-version-(bundled-with-OS)