Dokumentation des Softwareprojekts BarFinder

Ein Projekt von
Thomas Millan, Marc Müller und Erik Saibel

Verfasser:	**Thomas Millan (tm108), Marc Müller (mm286), Erik Saibel (es116)**
Kurs:	Projekt 119400a
Betreuer:	Prof. Walter Kriha
Abgabetermin:	26.07.2021

Inhaltsverzeichnis

1. Motivation

Unsere **Motivation** für **BarFinder** beruht sich auf eigenem **Interesse** und der potenziellen **Marktlücke im App Store**, da es eine große Bandbreite an Apps für Restaurants u.Ä. gibt, jedoch nichts in Richtung Bars und Events. Die ersten Gespräche über die Idee fanden tatsächlich auch in einer Bar statt und von da an hat sich alles entwickelt und es bot sich hervorragend als **Softwareprojekt** an.

2. Projektdefinition

Im Jahr **2021** gibt es immer noch **keine etablierte mobile App** für das Nachtleben **bezüglich Bars und Events**. Man kann nicht dediziert nach Bars **suchen** und gleichzeitig individuell seine **Präferenzen berücksichtigen** und danach **filtern**. Man kann nicht abends spontan auf einen Blick nach **potenziellen Events** Ausschau halten und sich diese, sowie auch Bars, als **Favoriten** abspeichern. Ebenso gibt es aus der Sicht der **Barbesitzer** und **Eventveranstalter** keine mobile Plattform, um für die eigenen Aktivitäten zu **werben**. Dieses Problem wird mit **BarFinder** beseitigt.

Wir entwickeln eine **mobile App**, welche **strukturiert** und **einfach** gestaltet ist, um dir das **Verwalten** deiner **Lieblingsbars** und **Events** so angenehm wie möglich zu machen. Wie bereits oben erwähnt lässt sich das passende Event oder die passende Bar **beliebig** nach deinen **individuellen Präferenzen finden** und **abspeichern**. In der **vollständigen Version** wirst du bei **anstehenden Events**, die zu dir passen **benachrichtigt**, damit du nichts verpasst. Somit ist **BarFinder** deine intuitive "go-to" App mit einer **simplen**, **modernen** und **ansprechenden** Oberfläche, wenn es um **Bars** und **Events** in deiner Stadt geht.

2.1. Projektziele

Allgemein:

· Agiles und dynamisches Arbeiten in der Gruppe soll gegeben sein.

· Die Oberfläche der App wird in Figma gestalten und in Xcode mit Swift umgesetzt.

Design:

· Personas werden passend zu dem Konzept erstellen.

·Eine vollständige Entwicklung der Oberfläche: von Scribbles bis High-Fidelity Prototype

· Das App-Icon zu BarFinder erstellen.

· Light- und Darkmode gestalten.

· Es sollen Usertest zwischen den Designphasen durchgeführt und berücksichtigt werden.

· Das User-Interface und die User Experience stehen im Vordergrund.

Code und Funktionen:
· Einfachgehaltenes Prototpye Backend.
· Mit der Suchleiste soll es möglich sein direkt nach Bars oder Events zu suchen.
· Man soll Bars nach "Geöffnet", "Entfernung" und "Preis" sortieren können.
· Man soll Events nach: "Datum", "Entfernung" und "Preis" sortieren können.
· Bars sollen nach: "Öffnungszeit", "Entfernung", "Preis "und "Tags" (verschiedenen Bararten und Stichworte) gefiltert werden können.
· Events sollen nach: "Entfernung", "Preis" und "Tags" (verschiedene Eventarten und Stichworte) gefiltert werden können.
· Bars und Events können in einer Liste in Form von Kacheln mit den wichtigsten Informationen auf einen Blick angezeigt werden und man soll die Möglichkeit haben sie zu seinen Favoriten hinzuzufügen
· Die Map zeigt jeweils die Bars oder Events auf einer Karte und diese werden als Pinnnadeln dargestellt. Beim Klicken der Pinnnadeln öffnet sich eine vereinfachtet und kleine Übersicht, die einem den Weg zum detaillierten Screen oder eine Route über Maps ermöglicht. Natürlich sieht man auch wo man sich selber auf der Karte befindet und kann sich selbst jederzeit zentrieren.
· Es ist möglich Bars und Events zu favorisieren und diese können dann auch offline abgerufen werden.

Nicht-Ziele:
· Anbindung an Kunden (Barbesitzer oder Eventveranstalter)
· Umsetzung für Android
· Umsetzung für ganz Deutschland
· Vollwertiges Backend
· Integriertes Ratingsystem
· Accountsystem

Alle **Nicht-Ziele** sind **nicht Teil des Projekts** bezüglich der Hochschule. Diese werden zu einem **späteren Zeitraum** verwirklicht, da die App auch **außerhalb der Hochschule weiterentwickelt** wird. Zudem sind alle gelb markierten Ziele sogenannte **Kann-Ziele** und somit **zweitrangig** für unser Projekt.

2.2. Projektumfang

Ein vollwertiges **Interface** der BarFinder App in **Figma**. D.h. jeder Schritt von **Scribbles**, über **User-Tests**, bis **Highfidelity-Prototype** wird berücksichtigt.

Eine **iOS App** mit allen oben genannten Funktionen in **Xcode mit Swift** entwickelt. Dabei wird das gestaltete **Interface aus Figma** als **Vorlage** verwendet.

2.2.1. Schwerpunkt Design

Für das Projekt haben wir uns vor allem das **Design** als Schwerpunkt gesetzt. Das erklärt auch die **Teamaufteilung** von einem **Design-Team aus zwei Personen** und einem **Entwickler-Team aus einer Person**. Beide Teams arbeiteten **zeitgleich** und **dynamisch** miteinander.

Für das Design wurde ein **ausführlicher Prozess** gewählt. Angefangen mit dem **Zweck** und der **Idee**: "Eine Lösung für das **Entdecken** und **Filtern** von **Bars** und **Events** abhängig von den **individuellen Präferenzen**." Hinzukommt noch die aktuelle **Covid-Situation** und somit die Existenzgefährdung von Bars. Mit **BarFinder** soll dem **entgegengewirkt** werden. Man versucht den **Besitzern** mit der **mobilen Plattform** eine **Werbemöglichkeit** darzubieten und den **Nutzern** der App einen einfachen und **intuitiven** Weg zu der neuen Lieblingsbar oder aufregenden Events.

2.2.2. App-Umsetzung

Bei der **App-Umsetzung** hat man zuerst eine **grundlegende Struktur** gelegt und mit den **Funktionen** angefangen. **Code** und **Design** wurden **parallel** entwickelt. Als das **Design vollendet** war, wurde **gemeinsam** an der App in Xcode weiterentwickelt und schließlich auch die **Optik** angepasst. Gegen Ende haben wir uns dann um die optischen **Details** gekümmert, um so nah wie möglich an unsere **Designvorlage** aus Figma heranzukommen.

2.3. Technologien

Im BarFinder Projekt wurde mit **Figma**, **Xcode** und **GitLab** gearbeitet. Figma wurde zur **Gestaltung** der **Oberfläche** und als **Prototyping-Tool** genutzt. In **Xcode** wurde mit **Swift** die App programmiert und auf **GitLab** wurde das Projekt **aufgesetzt** und relevante **Dokumente** dort gelagert.

2.3.1. Software

In **Xcode** haben wir das ganze Projekt in **Swift** geschrieben und uns auf **iOS spezialisiert**.

2.4. Projektablauf

Design

Brief

Personas

Problem

Solution

Competitors

Requierments

Context of use

Content Table

Scribbles

Wireframes

User Flow

Prototype

Design System

Highfidelity

HiFi-Protoype

Dark Mode

App Icon

Code

Create Bar Models

Storyboard Prototype

Implement Filter Logic

Programm Map

Create Seques

Create Delegates

Add Event Models

Add Event Views

Improve Storyboard

Let User Choose Filters

Implement Search

Improve Map

Create Dummy Backend

Implement API Call In App

Add Observer Pattern

Create Detail Views

Add Favourties In Core Data

3.1. Competitors

Mit der Auflistung der Vor- und Nachteile der anderen Apps wollten wir aus ihren Problemen lernen und ein insgesamt überlegenes Produkt entwickeln. Unser Ziel bei der Wettbewerbsanalyse war es, uns ein Bild von der aktuellen Marktsituation zu machen.

	Stärken	Schwächen
Google Maps	• Schlichtes design • Favoriten Liste • Gute Routenplanung möglich	• Filter sehr unübersichtlich • Spezifische Suchen sehr mühevoll • Überladen
Tripadvisor	• Messenger • Onboarding bei Erstbenutzung • Sehr detailliert	• Schlechte bis keine Filterung möglich • Zu überladen • Stark verschachtelt
	• Zeitnahe Empfehlung • Design ansprechend • Ausführlicher Filter	• Filtertags schlecht zu entfernen • Filter nicht direkt zu finden • Überladene Navbar

Emma Weber

Occupation
Mobile Media Student

Location
Stuttgart, Germany

Age
22

Family
Single

Social Media Activity

Instagram

Twitter

Snapchat

Biography

Emma is a Mobile Media Student at Hochschule der Medien in Stuttgart. She lives in Stuttgart since she was born. She ist very athletic and often goes to college sports after college. She knows a couple of bars in Stuttgart and goes there with her friends on the weekend. For a change she goes to different Bars sometimes.

Needs

- She no longer wants to be dependent on social media to discover new spots in her city
- Find balance between college and friends
- She wants to go to bars and meet her friends again

Pain points

Emma need to find a good platform to see what bars are near you and wich events they are currently offering.

3.2. Personas

Um unsere Zielgruppe anzusprechen, mussten wir zunächst diese „definieren“. Um immer vor Augen zu haben, für wen wir designen, haben wir unsere Personas entwickelt. Dies hat uns sehr geholfen, da es uns einen Blick auf die „Pain Points“ und andere wichtige Informationen gab. Durch die Personalisierung unserer Zielgruppe haben wir unser Publikum viel besser verstanden.

Wang Li

Occupation
Exchange Student

Location
Changsha, China

Age
23

Family
Single

Social Media Activity

Instagram

Twitter

Snapchat

Biography

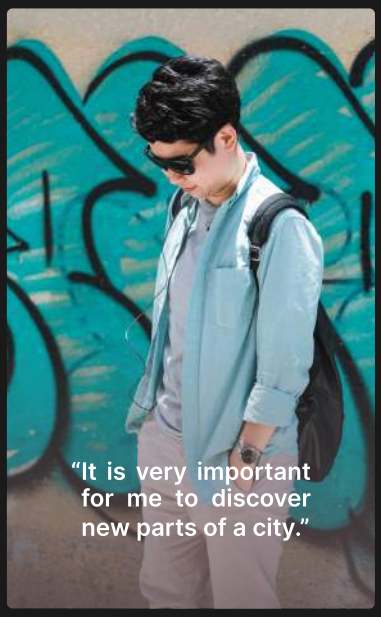
Wang Li is an exchage student from Changsha, China and spends his semester abroad in Stuttgart at the HdM. He is a very hard-working and intense-learning person, who chooses very wisely where to spend his freetime. In his home town he was supposed to go out with his friends almost every friday for balancing his strict student life and have fun together.

Needs

- More freetime because of his working/learning lifestyle
- Find balance between work and fun
- More social connections with other students and people in general

Pain points

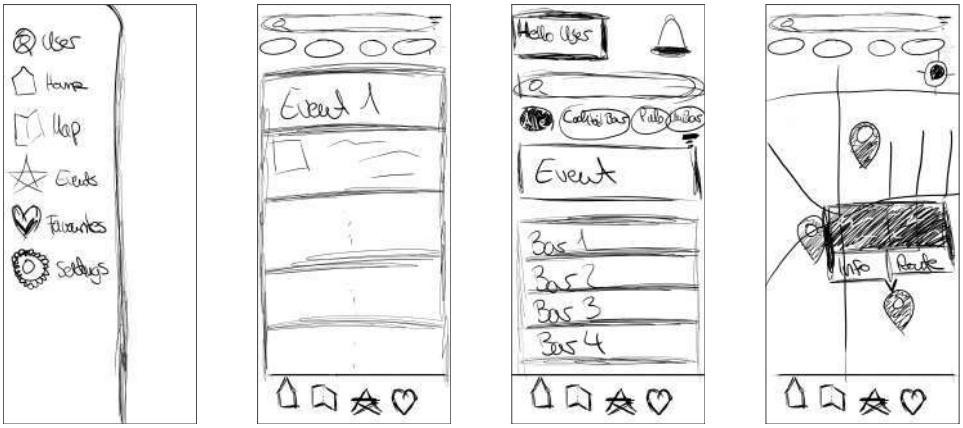
Wang has issues with getting by in the city, because he is new to Stuttgart. He seems kind of lost and doesn't know where to start.



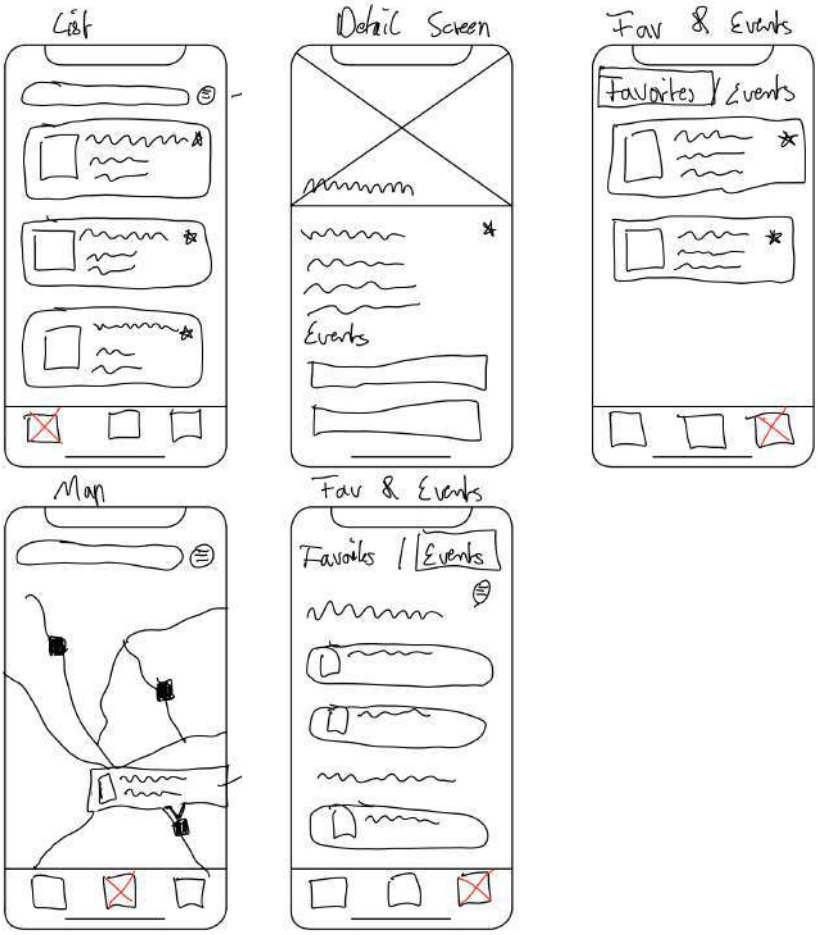
3.3. Scribbles

Nachdem wir eine Inhaltstabelle mit allen notwendigen Informationen erstellt hatten, haben wir uns über die Informationsarchitektur Gedanken gemacht. Durch die Auswertung der Vor- und Nachteile einer Tab-Leiste oder der Navigation über ein Hamburger-Menü haben wir festgestellt, dass sich unsere App leicht in 3 Hauptbereiche unterteilen lässt. Mit diesem Wissen fiel uns die Entscheidung leicht und wir haben uns für die Tab-Navigation entschieden, weil wir eine klare und schnelle Navigation für den Benutzer wollten, damit er immer weiß, wo er sich aktuell beifndet. Vor diesem Hintergrund haben wir die Grundstruktur unserer App sowie das grobe Layout festgelegt.

Version 1

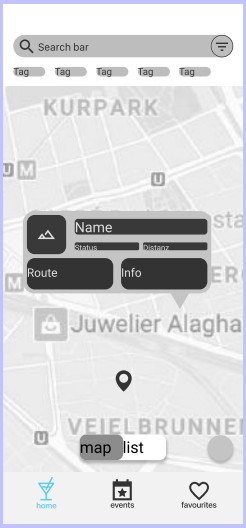


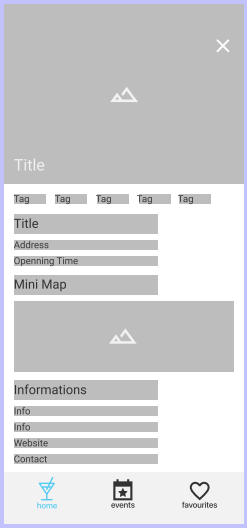
Version 2

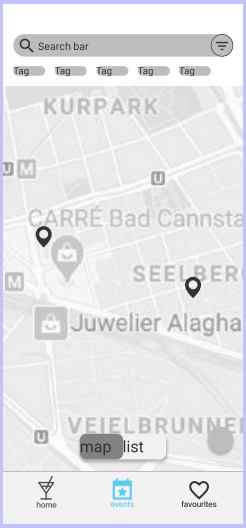


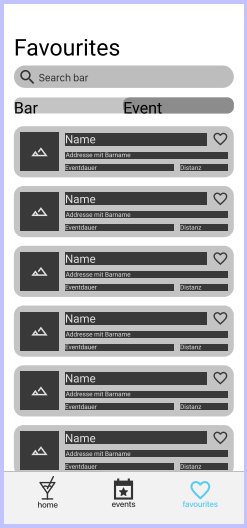
3.4. Wireframes

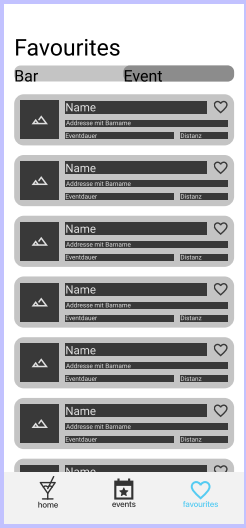
In unserem ersten Ansatz haben wir nur Grundformen verwendet , auch für die späteren Überschriften oder Textfelder, um ein besseres Verständnis von Größe und Ausrichtung zu erhalten. Mit genau diesen Wireframes haben wir unsere ersten Szenarien erstellt und erste User-Tests durchgeführt, um am Benutzerfluss zu arbeiten.

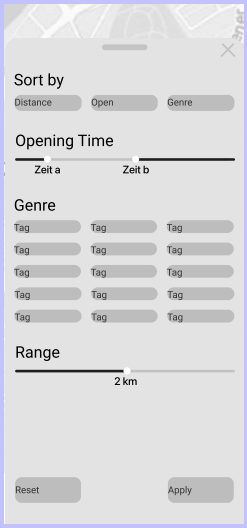












3.5. User-Flow

Um unsere Zielgruppe anzusprechen, mussten wir zunächst „definieren“. Um immer vor Augen zu haben, für wen wir designen, haben wir unsere Personas entwickelt. Dies hat uns sehr geholfen, da es uns einen Blick auf die „Pain Points“ und andere wichtige Informationen gab. Durch die Personalisierung unserer Zielgruppe haben wir unser Publikum viel besser verstanden.

3.6. First User Test

Unser erster User-Test haben wir mit unseren Wireframes durchgeführt. Wir haben grundlegende Funktionen und die Anordnung der Seiten dargestellt und getestet. Mit den Ergebnissen haben wir dann weiter an den Funktionen und der Benutzerfreundlichkeit gearbeitet. Die Tests haben uns erste Einblicke der realen User gegeben, womit wir ein viel besseres Verständnis bekommen konnten.

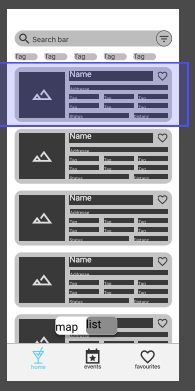
Wir haben den Probanden*innen verschiedene Fragen gestellt die alle Themengebiet abgedeckt haben. Nachdem wir die Tests durchgeführt haben, erstellten wir eine Zusammenfassung und haben die Probleme ausgearbeitet.

Das Ergebnis aus unserem ersten Test fiel sehr gut aus. Die Tester*innen hatten wenige bis keine Probleme sich in der App zurechtzufinden. Natürlich war es für den ein oder anderen etwas schwieriger da wir nur mit den Wireframes getestet haben.

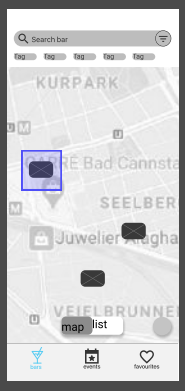
alt

alter Filter

"home" erklärt nicht genau was gemeint ist



Zelle wirkt zu klein

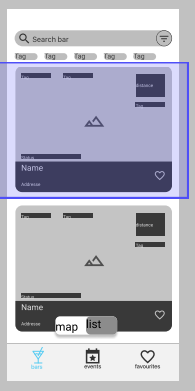


Baricon zu klein für ein Bild

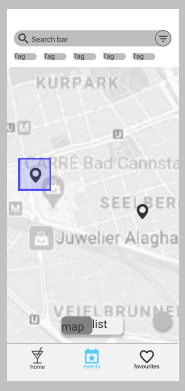
neu

neuer Filter

"home" wurde zu "bars" geändert



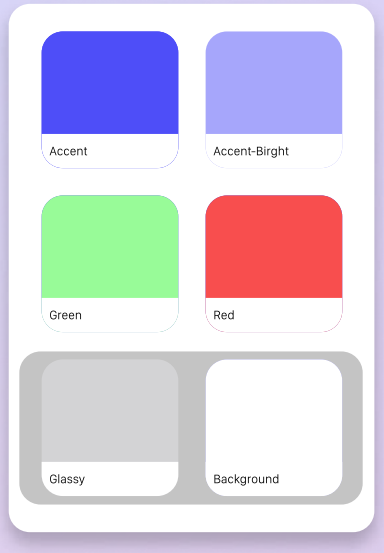
Zellen wurden komplett überarbeitet

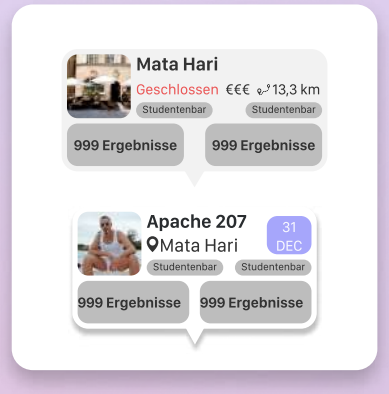


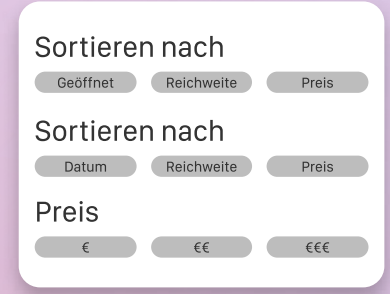
Bar als Nadel dargestellt

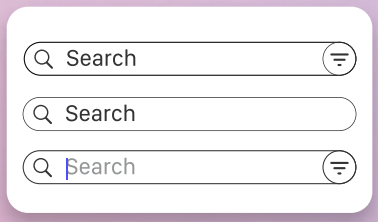
3.7. Design System

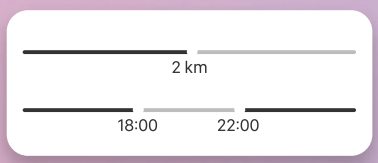
Nachdem wir alle wichtigen Änderungen an unseren frühen Wireframes vorgenommen hatten, war es an der Zeit, ein Designsystem zu entwickeln, bevor wir mit dem High-Fidelity Screens beginnen konnten. Das Designsystem war sehr wichtig, um einen konsistenten und leicht anpassbaren Prototyp zu bauen. Natürlich erfordert die Erstellung eines Designsystems einige Zeit und Mühe, aber im Nachhinein hat sich die Entwicklungszeit mehr als gelohnt. Dadurch konnten wir die High-Fidelity Screens modular aufbauen und jede spätere Anpassung war nur ein paar Klicks entfernt und das gesamte Design wurde global verändert.

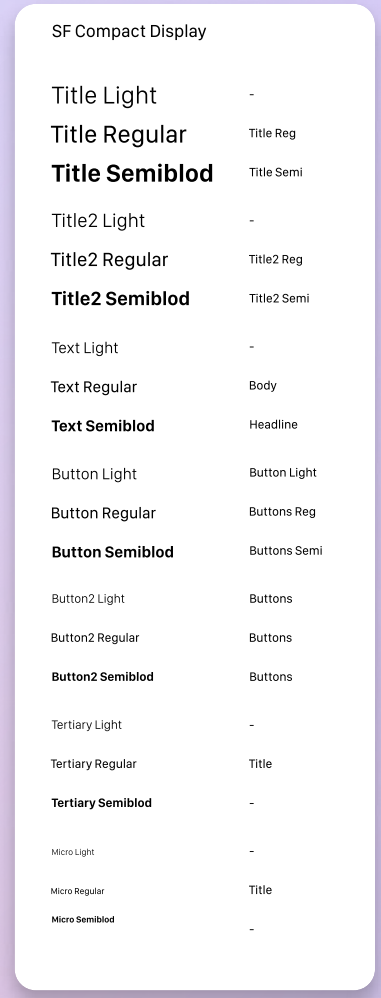


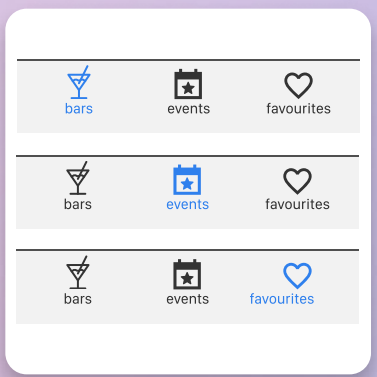


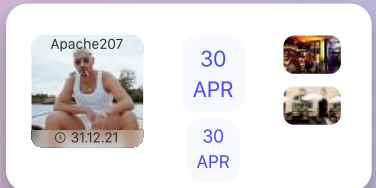


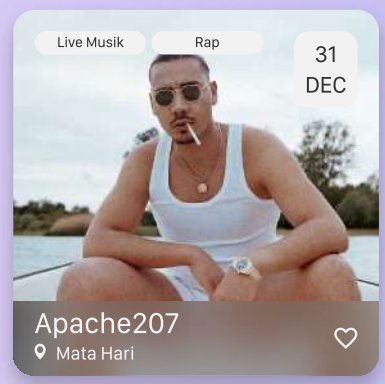


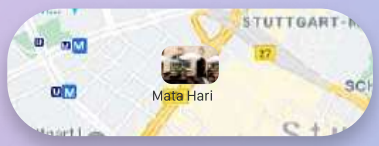


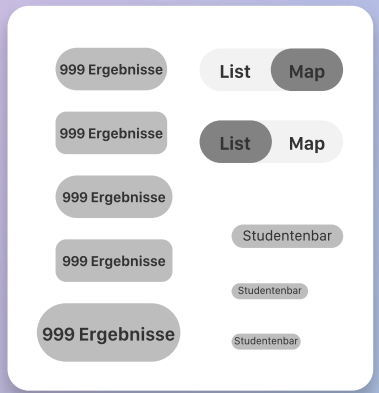


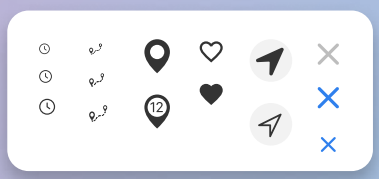






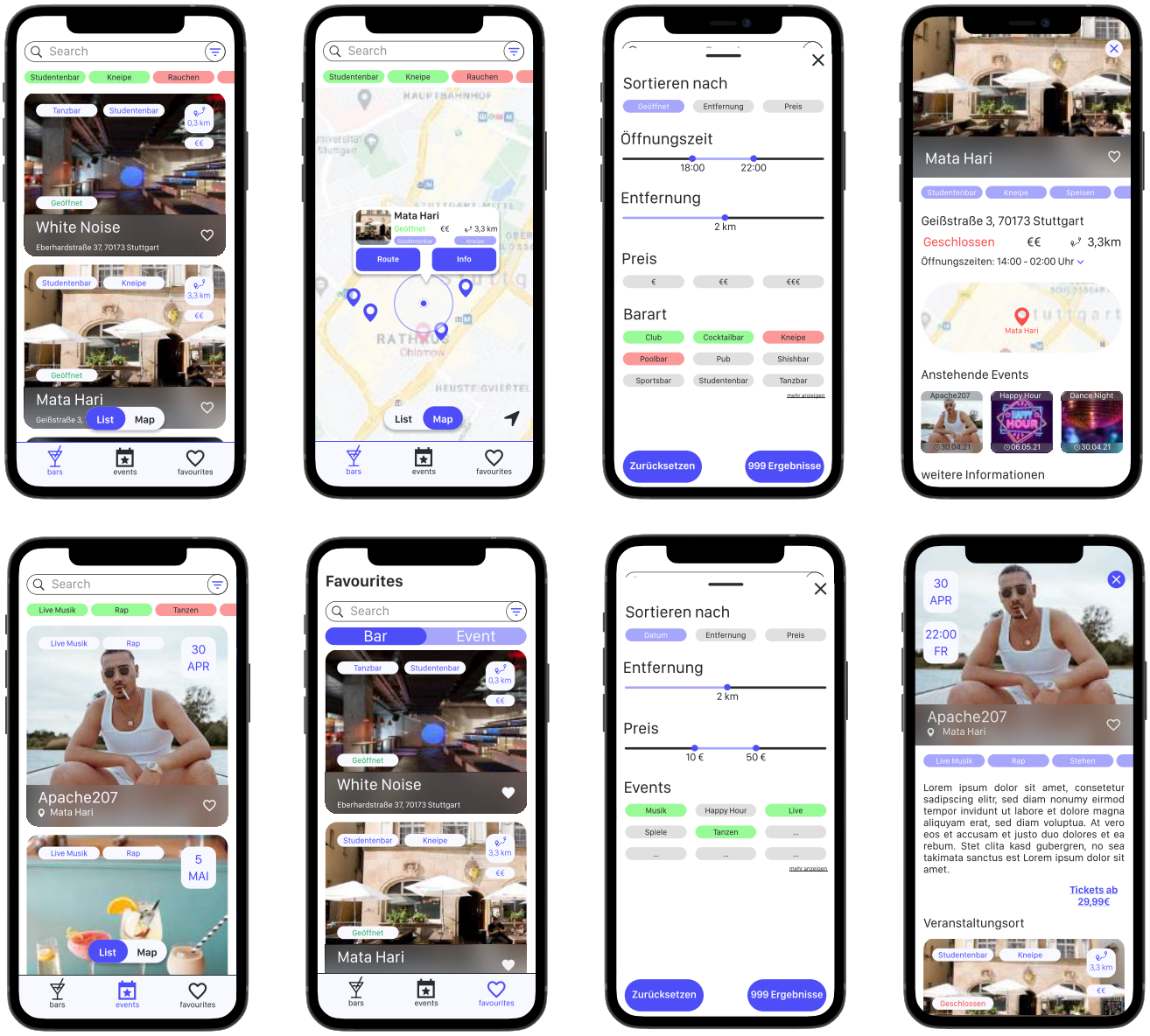






3.8. High Fidelity

Mit der endgültigen Version des Designsystems war es nun an der Zeit, alles zusammenzufügen. In dieser Phase des Projekts waren wir mit einigen Problemen konfrontiert. Theoretisch funktioniert die Idee eines Baukastensystems perfekt, aber in der Realität wird immer eine Überschrift, ein Bild oder einfach eine andere Variante eines Buttons fehlen. Gelegentlich mussten wir das Designsystem sowie einige einzelne Bildschirme anpassen. Unser Designsystem ermöglichte es, sich hauptsächlich auf das Layout zu konzentrieren, da alle Designelemente bereits fertig waren. Mit unserem vordefinierten Layoutraster haben wir deutlich gemacht, dass jeder Bildschirm den gleichen Rand und die gleichen Rinnen zwischen den Elementen hat. Mit der konsequenten Verwendung unserer Primär- und Sekundärfarbe wollten wir einen sauberen und harmonischen Look schaffen.



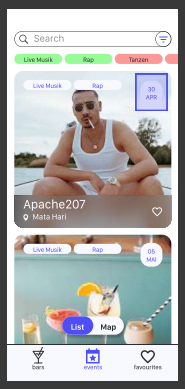
3.9. Second User Test

In unserem zweiten User-Test konnten wir nochmal mit einem weiter entwickelten Prototype die User testen. Wir haben für den Test einen High Fidelity Prototype erstellt und mit verschiedenen Platzhalter gefüllt das der Proband*innen einen besseren Eindruck bekommen konnte.

Auch in unserem zweiten Test haben wir den Probanden*innen verscheiden Fragen gestellt. Zudem haben wir den Probanden*innen auch kleinere Aufgaben gegeben um herauszufinden, ob sich der User gut oder wenig gut zurechtfindet.

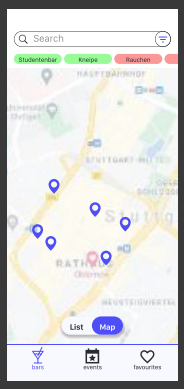
Das Ergebnis aus unseren zweiten Test war sehr positiv ausgefallen. Alle Tester*innen konnten die Aufgaben sehr gut und schnell lösen. Die Kritikpunkte haben wir als Team ausgearbeitet und so gut wie möglich umgesetzt.

alt

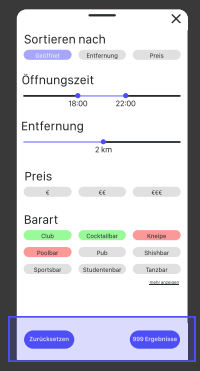


Datum zu klein um zu erkennen

Close Button mit zu wenig Kontrast

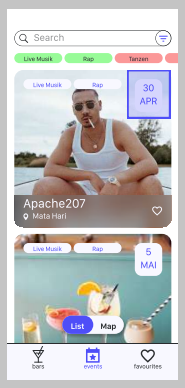


Zu wenig Informationen über den eigenen Standort



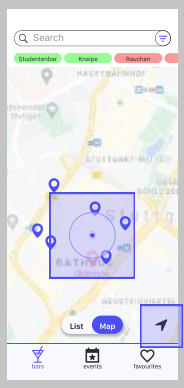
Filter Button zu klein

neu

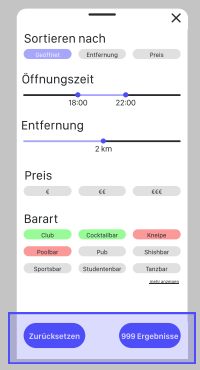


größeres Datum

Neuer Close Button mit besserem Kontrast



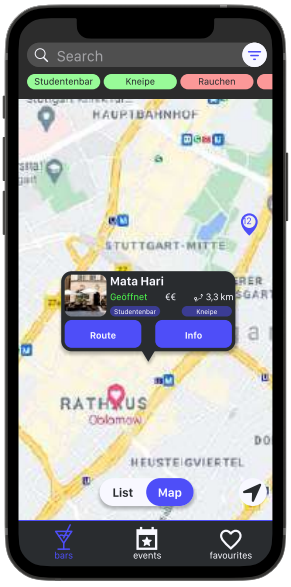
Mehr Informationen über den eigenen Standort

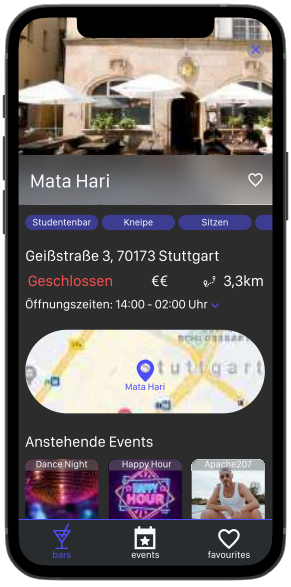


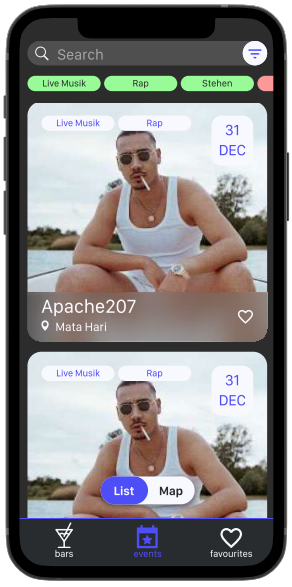
größere Filter Button

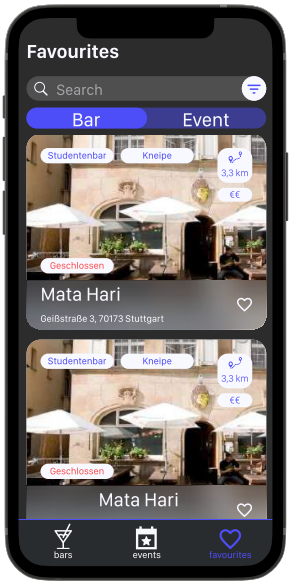
3.10. Dark Mode

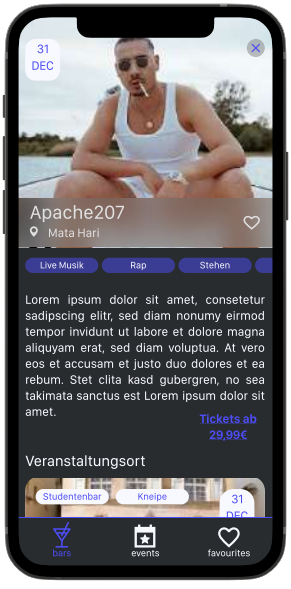
Da der „Darkmode“ eher eine Notwendigkeit als eine Möglichkeit oder Wahl des Designs wurde, war es nicht umstritten, dass wir eine Darkmode-Version unserer App erstellen mussten. Da wir alle Elemente in unserem Designsystem zuvor entworfen haben, war der Dunkelmodus nur wenige Farbanpassungen entfernt. Wir mussten einfach „dunkle“ Varianten unserer bereits vorhandenen Komponenten erstellen. Anstatt nur Negativfarben zu verwenden, ist es uns gelungen, einen hohen Kontrast beizubehalten, während wir weiterhin unsere Primärfarbe verwenden.

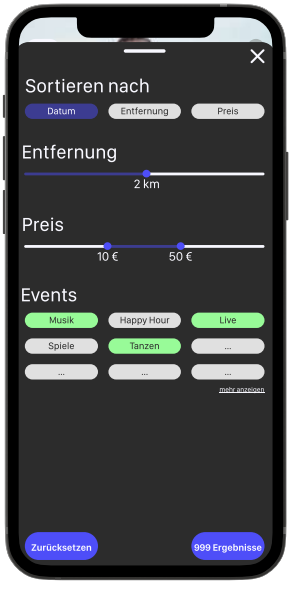












4 Architektur & Code

Als Programmiersprache wurde Swift mit UIKit benutzt, da der Scope des Projektes auf der App und dem Design dieser lag, wurde ein “dummy” Backend in Python mit der Bibliothek “FastApi” erstellt, die Daten im Backend wurden als Json gespeichert.

Swift

In Swift dreht sich alles um das Model View Controller Pattern (MVC) mit den folgenden Komponenten:

-Models, in welchen die Logik, DB Funktionen und Aufrufe etc. gemanaget werden.

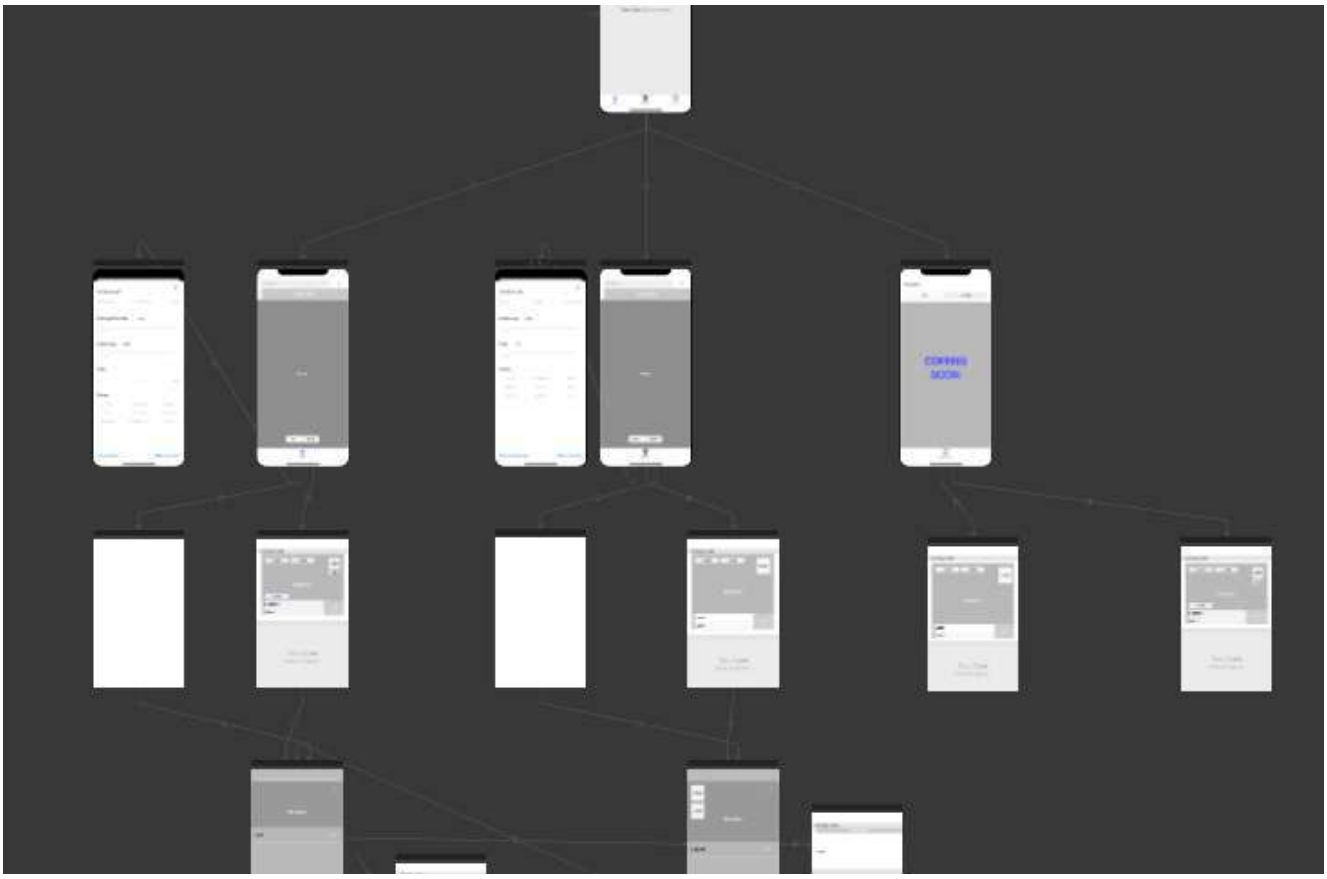
-Views(in Swift sind die Views im Storyboard), in welchen die UI der App definiert wird.

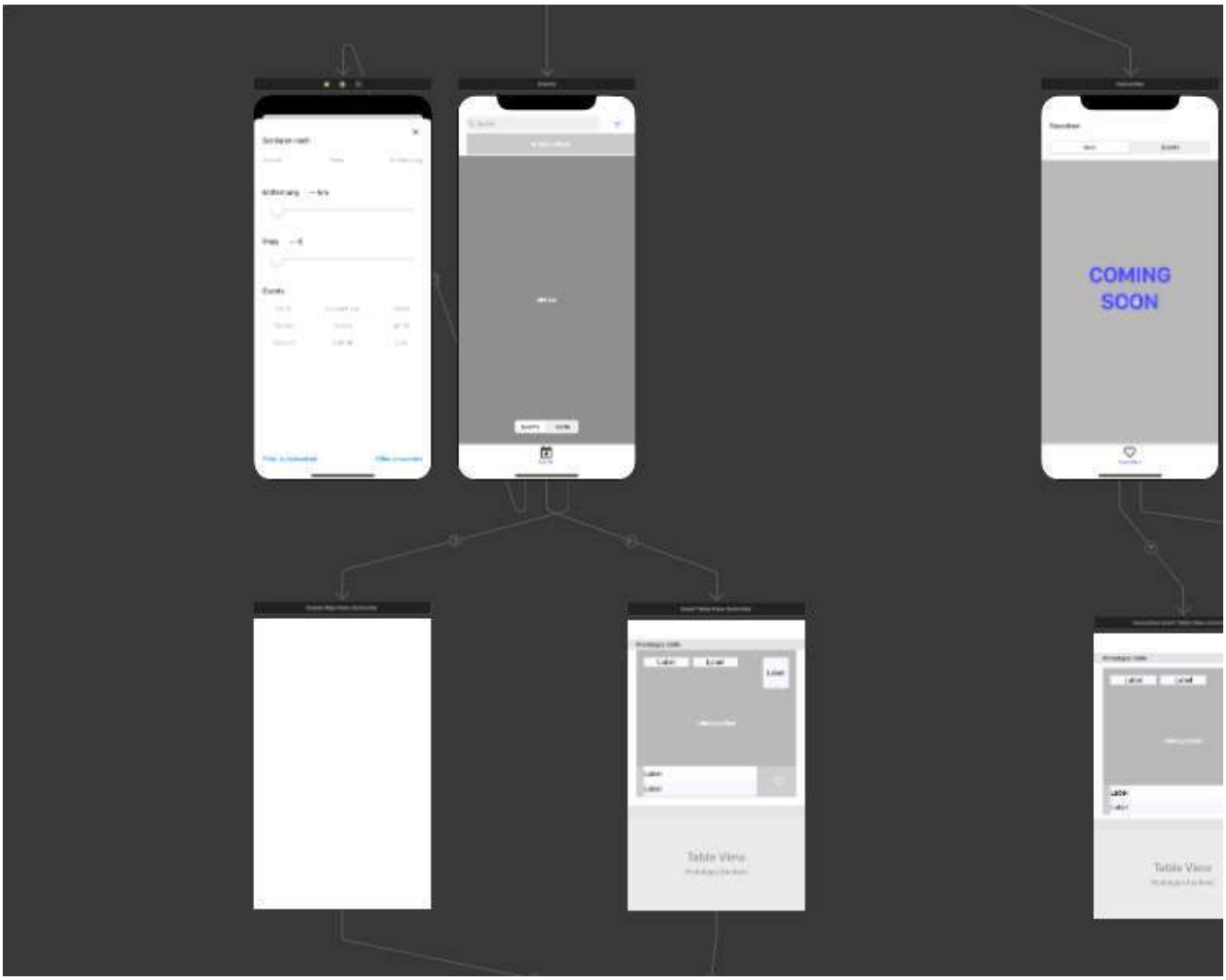
-Controller(in Swift ViewController), welche als Bindeglied zwischen Models und Views fungieren und das Zusammenspiel von Models und Views regeln.

Storyboard

Wir haben uns für unser Projekt für das von Apple bereitgestellte UIKit entschieden, hierbei werden alle Views auf dem Storyboard erstellt und definiert.

Anschließend kann man diese mit ViewControllern verbinden, welche die einzelnen Views untereinander und mit der Logik in den Models verbinden. Unser Storyboard sieht folgendermaßen aus:




COMING
SOON
Table View
Table View

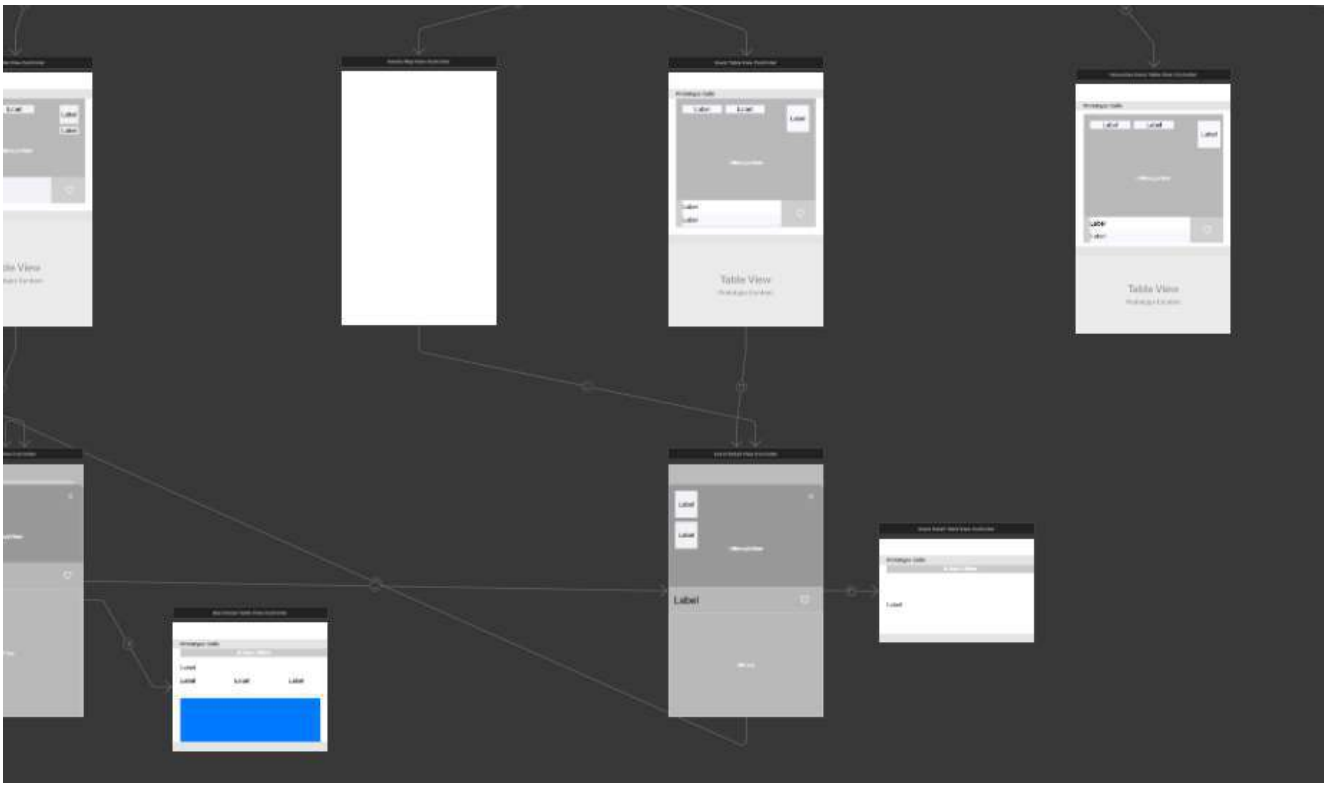


Table View
Table View
Label

Models

In unserer App, dreht sich alles um Objekte der Klasse Bar und der Klasse Event, welche die Eigenschaften definieren, welche eine Bar oder ein Event haben kann.

Aus dem Backend werden bei App Start die Daten geladen und als Instanzen von Bar und Event zu Arrays hinzugefügt. Diese Arrays werden anhand der vom Nutzer angegeben Präferenzen gefiltert und die Objekte im gefilterten Array dann auf der Karte, sowie in der Liste angezeigt. Klickt man auf eines der Elemente auf der Liste oder Karte, werden in einer DetailView alle Informationen über das Element angezeigt. Des Weiteren kann man hier mit einem Klick auf das Favoriten Icon das Element zu seinen Favoriten hinzufügen, wobei die Bar oder Event Instanzen in Core Data (Ausführliche Erklärung unter 4.3) persistent gespeichert werden.

4.1. Ablaufplan

Unsere App hat 3 Hauptthemen: Bars, Events und Favoriten.

Bars

Der Einstieg der App ist im BarsMainViewController, hier werden die Bars und Events über einen API-Aufruf aus dem Backend geladen. Des Weiteren fungiert der Controller als Parrentviewcontroller von BarsTableViewController, sowie EventTableViewController, welche in die View von BarsMainViewController eingebettet und zwischen welchen man mit einem Switch Button wechseln kann.

Klickt man auf den Filterbutton oben rechts, öffnet sich die BarsFilterView als Modal (Fenster, welches sich in das Bild hochschiebt und mit einem einfachen Wisch nach unten wieder geschlossen werden kann), in welcher der Nutzer die Filter seiner Wahl setzen. Die Kommunikation zwischen BarsMainViewController und dem BarsFilterViewController, wurden über Seques und Delegates implementiert.

(Mehr zu Segues und Delegates weiter unten)

Nachdem der Nutzer seine Auswahl an Filtern getroffen hat und auf Filter Anwenden klickt, schließt sich das Modal-Fenster und in BarsMainViewController, wird über das oben beschrieben Delegate Pattern die Funktion “updateFilterList(...)” aufgerufen, und alle ausgewählten Filter als Argumente in Form von Arrays mitgegeben. In BarsMainViewController, wird dann die Filter Funktion aufgerufen, welches anhand der gegebenen Filter alle Bars rausfiltert, welche nicht mit den Filtern übereinstimmen.

Nachdem in dem Array FilterdBars nur noch die ausgewählten Bar Objekte vorhanden sind, “benachrichtigt” BarsMainViewController über das Observer Pattern BarsMapViewController und BarsListViewController.

Beim ObserverPattern, gibt es eine Subject Klasse und mindestens eine, aber in der Regel mehrere Observer-Klassen. Wenn sich eine Klasse als Observer registriert, wird sie “benachrichtigt”, wenn die Subject Klasse die Observer mit neuen Daten versorgt.

Swift bietet mit dem Notfication-Center genau diese Funktionalität, des Observer Patterns mit sich.

Eintrag als Observer:

```
//set observer for bars
NotificationCenter.default.addObserver(self, selector:
    #selector(onNotification(notification:)), name:
    BarsMainViewController.notificationName, object: nil)
```

Notify-Funktion, welche in Subject-Klasse aufgerufen wird, um Observer zu benachrichtigen, wenn sich Daten ändern:

```
//notify Observers
NotificationCenter.default.post(name: BarsMainViewController.notificationName, object:
    nil, userInfo: ["selectedBars": selectedBars, "allEvents": allEvents])
```

Nachdem BarsMapViewController und BarsListViewController die aktualisierte Liste der Bar erhalten haben, aktualisieren sie Ihre Ansicht und zeigen die neue Auswahl an.

Events

EventMainViewController verhält sich sehr ähnlich wie BarsMainViewController, hauptsächlich die Filter Auswahl und die Filter Funktion sind unterschiedlich.

4.2 Kartenfunktionalität

Bei der Karte hatten wir die Auswahl zwischen AppleMaps und dem GoogleMapsSDK, wir haben uns für die Karte von Google entschieden, da die Community darum größer ist und es die Unterstützung für spätere mögliche Funktionen (bsp: Route und Navigation in der App) gibt, welche wir mit AppleMaps nicht hätten.

Wir haben die Karte programmatisch in einen UIViewContainer eingebunden.

```
16 import GoogleMaps
17 import UIKit
18 import GooglePlaces
19 import GoogleMaps
20
21 class MapViewController : UIViewController, GMSMapViewDelegate, MapMarkerDelegate {
22
23     private var infoWindow = MapMarkerWindow()
24     fileprivate var locationMarker : GMSMarker? = GMSMarker()
25
26     var selectedBar: Bar?
27     var selectedBars = [Bar]()
28     var allEvents = [Event]()
29     var mapView = GMSMapView(frame: .zero)
30     var markerImage = UIImage()
31     var markerView = UIImageView()
32
33     override func viewDidLoad() {
34         super.viewDidLoad()
35         markerImage = UIImage(named: "marker")!.withRenderingMode(.alwaysTemplate)
36         markerView = UIImageView(image: markerImage)
37         markerView.tintColor = UIColor(named: "accent_color")
38         //set observer for bars
39         NotificationCenter.default.addObserver(self, selector: #selector(onNotification(notification:)), name:
               BarsMainViewController.notificationName, object: nil)
40         let camera = GMSCameraPosition(latitude: 48.7758459, longitude: 9.1829321, zoom: 14)
41         self.mapView = GMSMapView(frame: .zero, camera: camera)
42         mapView.isMyLocationEnabled = true
43         self.view = mapView
44         setMarkersOnMap()
45         self.infoWindow = loadNiB()
46         mapView.delegate = self
47
48     }
```

Anschließend haben wir die Marker, sowie die Infofenster für die Marker erstellt und personalisiert (Siehe Klassen: BarsMapViewController, EventMapViewController):

```
50      //Info Window:
51      func didTapInfoButton(data: Bar) {
52          selectedBar = data
53          self.performSegue(withIdentifier: "mapinfoseq", sender: self)
54      }
55
56      func didTapRouteButton(data: Bar){
57          if (UIApplication.shared.canOpenURL(URL(string:"comgooglemaps://")!)){
58              //openGooglemaps
59              let urlString = "comgooglemaps://?center="+String(data.lat)+","+String(data.lon)+"&zoom=14&views=traffic"
60              UIApplication.shared.openURL(URL(string: urlString)!)          'openURL' was deprecated in iOS 10.0
61          }else{
62              //openAppleMaps
63              UIApplication.shared.openURL(NSURL(string:
                    "http://maps.apple.com/?ll="+String(data.lat)+","+String(data.lon))! as URL)
64          }
65
66      }
67      func loadNiB() -> MapMarkerWindow {
68          let infoWindow = MapMarkerWindow.instanceFromNib() as! MapMarkerWindow
69          return infoWindow
70      }
71      func mapView(_ mapView: GMSMapView, didTap marker: GMSMarker) -> Bool {
72          var bar: Bar?
73          if let data = marker.userData! as? Bar {
74              bar = data
75          }
76          locationMarker = marker
77          infoWindow.removeFromSuperview()
78          infoWindow = loadNiB()
79          guard let location = locationMarker?.position else {
80              print("locationMarker is nil")
81              return false
82          }
```

```
83          // Pass the spot data to the info window, and set its delegate to self
84          infoWindow.spotData = bar!
85          infoWindow.delegate = self
86          // Configure UI properties of info window
87          infoWindow.nameLabel.text = bar?.name
88          infoWindow.nameLabel.font = infoWindow.nameLabel.font.withSize(20)
89
90          infoWindow.categoryLabel1.text = bar?.categories[0]
91          infoWindow.categoryLabel2.text = bar?.categories[1]
92          infoWindow.distanceLabel.text = "\(String(format: "%.1f", bar!.distance!)) km"
93          //TODO: Switch for open label:
94          //infoWindow.openLabel.text = bar.
95          infoWindow.priceLabel.text = bar?.price
96          infoWindow.img.image = bar?.img
97
98
99          //style cell
100         infoWindow.layer.cornerRadius = 20
101         infoWindow.layer.masksToBounds = true
102
103         infoWindow.infoButton.cornerRadius = 20
104         infoWindow.routeButton.cornerRadius = 20
105
106         infoWindow.categoryLabel1.layer.cornerRadius = 8
107         infoWindow.categoryLabel1.layer.masksToBounds = true
108         infoWindow.categoryLabel2.layer.cornerRadius = 8
109         infoWindow.categoryLabel2.layer.masksToBounds = true
110
111         infoWindow.img.layer.cornerRadius = 16
112         infoWindow.img.layer.masksToBounds = true
113
114
115
116
```

```
126     }
127     func mapView(_ mapView: GMSMapView, didChange position: GMSCameraPosition) {
128         if (locationMarker != nil){
129             guard let location = locationMarker?.position else {
130                 print("locationMarker is nil")
131                 return
132             }
133             infoWindow.center = mapView.projection.point(for: location)
134             infoWindow.center.y = infoWindow.center.y - 75
135         }
136     }
137
138     func mapView(_ mapView: GMSMapView, didTapAt coordinate: CLLocationCoordinate2D) {
139         infoWindow.removeFromSuperview()
140     }
141
142     func setMarkersOnMap(){
143         mapView.clear()
144         for bar in selectedBars{
145             let position = CLLocationCoordinate2D(latitude: bar.lat, longitude: bar.lon)
146             let marker = GMSMarker(position: position)
147             marker.userData = bar
148             marker.iconView = markerView
149             marker.map = mapView
150         }
151     }
152
153     @objc func onNotification(notification:Notification)
154     {
155         selectedBars = notification.userInfo?["selectedBars"] as! [Bar]
156         allEvents = notification.userInfo?["allEvents"] as! [Event]
157         setMarkersOnMap()
158     }
159
```

4.3 Segues & Delegates

Seques und delegates sind meist die beste Wahl, wenn es darum geht Informationen oder Funktionsaufrufe von einem ViewController zu einem anderen zu übertragen.

Segues:

Segues regeln in Swift den Übergang von einem View Controller in den nächsten, Seques können Aufgerufen werden, um eine andere View zu öffnen und indem man die Funktion func prepare(for segue: UIStoryboardSegue, sender: Any?) überschreibt, kann man noch Daten an den sich öffnenden ViewController mitgeben.

Delegates:

Delegation ist ein Pattern der Objekt-Orientierten Programmierung um Objekt-Komposition zu unterstützen. Bei Delegation behandelt ein Objekt A eine Anfrage, indem es an ein weiteres Objekt B delegiert.

Beispiel:

```
protocol Assistant {
      func writeEmail()
}

class WorkerA : Assistant {
     func writeEmail() {
            print("WorkerA: I wrote an Email!")
      }
}

class Boss {
      var delegate : Assistant?
      func work(){
            print("Boss: I have no time to write Emails!“)
            delegate?.writeEmail()
       }
}
```

In diesem Beispiel, delegiert die Klasse Boss die Aufgabe Emails zu schreiben an die Klasse WorkerA, welche das Delegate Protokoll “Assistant” (Protokolle sind ähnlich zu Interfaces in Java) implementiert.

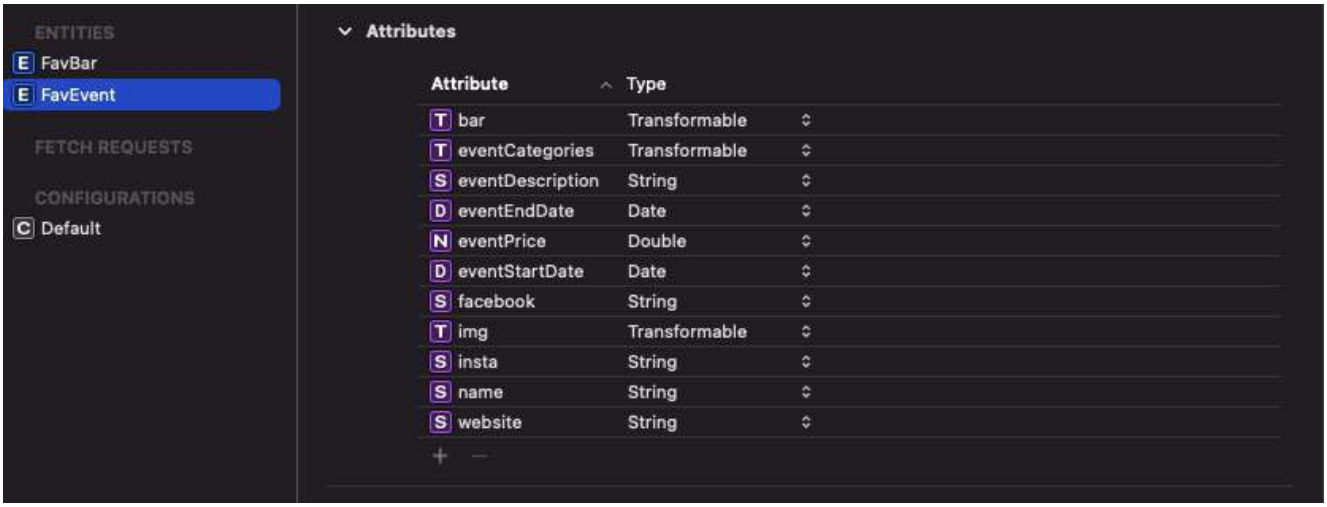
4.4 Datenbank/CoreData

Der Nutzer hat die Möglichkeit, wenn er auf das Favoriten-Herz bei einer Bar klickt, diese in die Liste der Favoriten hinzuzufügen und auch Offline abzurufen.

Diese Funktionalität haben wir mit CoreData implementiert.

Core Data ist ein Framework für die Speicherung von Objekten nach einem Datenmodell. Dabei werden 1:n und n:m Beziehungen zwischen Objekten unterstützt. Das Datenmodell wird dabei in einer .xcdatamodel-Datei abgelegt.

Leider konnten wir die Datenbankfunktionalität bis zur Abgabe, aufgrund des Umfangs des Projektes nicht vollständig Fertigstellen, haben jedoch bereits die Models in der Datenbank erstellt.



5.1. Erfahrungen & Know-how

Marc:

Mitgebrachtes Know-how:

Ich hatte durch die Vorlesung Praktikum Mobile Application Development, bereits erste Erfahrungen in der Programmiersprache Swift. Ebenfalls hatte ich vor Beginn des Projektes in meinem Praxissemester erste Erfahrungen in Python gesammelt, weshalb dies die Programmiersprache der Wahl für das Backend wurde.

Erwartungen:

Meine Erwartung an das Projekt war, meine Kenntnisse in Swift zu verbessern und in einem größeren Projekt anzuwenden. Des Weiteren wollte ich erste Erfahrungen im Erstellen einer API und eines Backends sammeln.

Was ich gelernt habe:

Durch das Projekt konnte ich viel Programmierpraxis in Swift erlernen. Ich habe mich tiefergehend mit in Swift häufig vorkommenden Pattern wie Seques und Delegates beschäftigt und habe einiges über das ModelViewController Pattern gelernt.

Thomas:

Mitgebrachtes Know-how:

Wir hatten in den letzten Semestern viel mit Figma gearbeitet, wodurch wir schon sehr viel Erfahrungen sammeln konnten. Deswegen haben wir uns dazu entschieden das Design in Figma umzusetzen. Durch die Vorlesung MAD 2 habe ich schon erste Einblicke in die Programmiersprache Swift bekommen. Dies war der Grund warum wir uns dazu entschieden haben das Projekt in Swift umzusetzen.

Erwartungen:

Meine größte Erwartung war das wir zusammen eine Fertige App auf die Beine stellen, das wir als Team zusammen arbeiten und natürlich meine eigenen Erfahrungen anwenden sowie ausbauen kann.

Was ich gelernt habe:

Ich konnte noch mehr Erfahrungen mit Figma sammeln und somit auch mein Arbeitsstil weiter ausbauen. Durch das Projekt konnte ich mehrere Erfahrungen sammeln und weiter ausbauen. Zu einem habe ich mehr über die Sprache Swift gelernt, des Weiteren konnte ich auch weitere Erfahrungen im Designprozess erlenernen.

Erik:

Mitgebrachtes Know-how:

Durch vorherige Semester war ich mit Figma und Xcode bereits vertraut und hatte grundlegende Swift-Kenntnisse durch MAD2. Auch der Designprozess war mir bekannt. Vor allem bei Mobile Apps, da wir diese bis jetzt am meisten im Studiengang konzipiert und gestaltet haben. Dazu gehören auch bestimmte Richtlinien und Trends für das User Interface. Letztlich war dies eine solide Grundlage an Fähigkeiten, um das Projekt “BarFinder” in Angriff zu nehmen und umzusetzen.

Erwartungen:

Ich habe mir von dem Projekt den vollständigen Prozess von der Entstehung einer iOS App vorgestellt. Die Entwicklung auf Design- und Programmierseite, als auch das Miteinander arbeiten und Rücksicht aufeinander nehmen gehörten zu meinen Erwartungen dazu. Natürlich fällt darunter auch seinen Fähigkeitenhorizont und sein Wissen in Figma, Swift und alles was Gruppenarbeit angeht zu erweitern.

Was ich gelernt habe:

Definitiv arbeite ich jetzt noch sicherer mit Figma als zuvor, da ich nun auch viel mehr Plugins und Mockups verwendet habe. Das gilt auch für das Arbeiten in einer Gruppe. Mittlerweile schätze ich sehr die Arbeitsatmosphäre und Zuverlässigkeit der Gruppe und diese lief in diesem Projekt angenehm und völlig problemlos ab.

5.2. Reflection

Reflektion Design:

- Da wir viel Wert auf Organisation und Zeitmanagement gesetzt haben, verlief der komplette Designprozess sehr angenehm und ohne Probleme. Natürlich war auch das Arbeiten zu zweit im Design Team praktisch und unkompliziert, weil die unterschiedlichen Aufgaben somit eindeutig verteilt werden konnten.
- Wäre ein größerer Zeitraum gegeben, dann würden wir gerne zwischen den wichtigen Designphasen jeweils einen User-Test durchführen und dementsprechend einen zusätzlichen User-Test verwenden zu können.

Reflektion Programierung:

- Trotz intensiver Beschäftigung mit der Architektur und Planung des Projektes, würden wir nächstes mal zu Beginn noch mehr Zeit in das Planen der Code Architektur und dem Lernen der Swift typischen Programmierpattern investieren, um einige zeitraubende Fehler zu vermeiden.
- Das nächste Mal würde ich die das Programmierteam größer dimensionieren, eventuell noch eine 4. Person, welche sich auch ausschließlich um die Programmierung kümmert.

Reflektion Allgemein:

- Es war sehr gut, dass wir direkt zu Beginn, feste Zeiten vereinbart haben, an denen man sich trifft, um sich auszutauschen, an dem Projekt zu arbeiten und die weiteren Schritte zu planen.
- Das nächste Mal würden wir uns noch genauere Deadlines für die einzelnen Unterziele setzen.
- Es war gut, dass wir direkt zu Beginn angefangen haben, parallel das Design und die Grundklassen des Codes zu erstellen und nicht erst angefangen haben zu programmieren, als das Design final fertig war.

Endergebnis:

Als Endergebnis, haben wir ein komplettes Designsystem inklusive fertigem Prototypen in Figma, welches durch mehrere Usertests und vielen Anpassungen, auf Usability und Benutzerfreundlichkeit perfekt optimiert wurde.
Des Weiteren bildet die in Swift programmierte IOS App fast alle geplanten Funktionen ab, inklusive des Filters der Bars und Events, Implementierung der Karte, lediglich die vollständige Implementierung von Core Data(Favouritenfunktion), sowie der Events API call müssen vor Veröffentlichung der App noch voll funktionsfähig gemacht werden.
Das geplante Prototyp "Dummy" Backend wurde vollständig implementiert und reagiert zuverlässig auf API Anfragen.

Leider konnten wir wie oben erwähnt aufgrund der begrenzten Zeit einzelne Funktionen, wie die Favoriten und einige Feinheiten der App nicht komplett fertigstellen. Wir sind jedoch mit dem Ergebnis aufgrund der Teamgröße und der limitierten Zeit sehr zufrieden.

BARFINDER

by Thomas Millan, Marc Müller
and Erik Saibel