Entwicklerhandbuch

Businect-App

LCD SS 2019

Gruppe 1

BUSINECT

Inhaltsverzeichnis

1 Einleitung

Dies ist das Entwicklerhandbuch zu der Businect-App.

Die App wurde im Zuge des Capstone Projektes an der Universität zu Köln im Sommersemester 2019 entwickelt. Die Entwicklung wurde von Gruppe 01 mit Projektbegleitung von Accenture durchgeführt, die aus den folgenden Mitgliedern besteht:

● Nina Erlacher: SPOC, iOS-Entwicklung

(nerlach1@smail.uni-koeln.de)

● Muqarab Afzal: AR-Development, Business Analyst

(mafzal1@smail.uni-koeln.de)

● Edriss Mosafer: Entwickler, Design

(mmosafe1@smail.uni-koeln.de)

● Maximilian Donner: Scrum-Master, Datenbankmanager

(m.donner@smail.uni-koeln.de)

Das vorliegende Entwicklerhandbuch umfasst 8 Kapitel, die weitestgehend unabhängig zu einander gelesen werden können.

1.1 Testaccounts

E-Mail-Adresse	**Passwort**
nina@test.com	123456789
m.donner@web.de	uni12345
muqarabafzal@icloud.com	123456789

1.2 Formatierung

Objekt	**Beispiel**	**Beschreibung**
Klasse	RegisterViewController.swift	Klassen beginnen mit einem Großbuchstaben. Der Anfang jedes weiteren Wortes wird durch einen Großbuchstaben gekennzeichnet. Im Fließtext werden Klassennamen unterstrichen.
Objekte/ Attribute/ Variablen	*errorLabel*	Der erste Buchstabe von Objekten, Attributen und Variablen wird klein geschrieben. Der

		Anfang jedes weiteren Wortes wird durch einen Großbuchstaben gekennzeichnet. Sie werden kursiv geschrieben.
Methoden	clickContinue()	Der erste Buchstabe von Methoden wird klein geschrieben. Der Anfang jedes weiteren Wortes wird durch einen Großbuchstaben gekennzeichnet. Sie werden kursiv geschrieben und enden mit (). Parameter werden in der Regel nicht mit angegeben.
Dateinamen	*file.png*	Dateinamen werden mit Sternchen gekennzeichnet und enthalten die Typ-Endung.

1.3 Abkürzungen

Abkürzung	**Erläuterung**
RTDB	Firebase Realtime Database
AR	Augmented Reality
SDK	Software Development Kit
DB	Datenbank
MVP	Minimum Viable Product
QR-Code	Quick-Response-Code

2 Projektbeschreibung

Das Ziel unserer Idee ist, den Geschäftsleuten das Netzwerken zu vereinfachen und zugänglicher zu machen. Uns ist aufgefallen, dass wenn Geschäftsleute aus den relevanten Berufen (Selbstständige, Unternehmensberater, Coaches, Vertriebler, Politiker etc.) ihr berufliches Netzwerk erweitern möchten, sie zum großen Teil neben der regulären Arbeitszeit ihre Freizeit investieren müssen, um zum Beispiel Vorträge, Workshops oder Seminar-Events zu besuchen. Unsere Zielgruppe besteht aus geschäftlich bedingten Vielreisenden, die teilweise viel Zeit in Zügen wie ICE nutzlos aussitzen müssen. Genau diese Zeit kann unsere Zielgruppe genauso gut zum Netzwerken nutzen, da in einem ICE der bspw. Montag Morgens von Düsseldorf nach Frankfurt viele Geschäftsleute sitzen.

Das Projekt bestand darin einen MVP zu entwickeln, der genau diese Geschäftsidee in einer Software umsetzen kann. Dazu haben wir eine iOS-App entwickelt, mit der sich Geschäftsleute ein Profil im Businect-Netzwerk erstellen können und anschließend mit Hilfe von QR-Codes auf Reisen in der Bahn netzwerken können.

Dazu kann der eine Benutzer einen QR-Code eines anderen Mitglieds scannen und bekommt folglich neben seinem Gesicht seine generellen Business Informationen *augmented* angezeigt.

3 Voraussetzungen zur Entwicklung

Für die Entwicklung einer iOS-App wird ein Apple-Rechner vorausgesetzt, da das Software Development Kit nur auf Mac-OS läuft. Für die Entwicklung und Weiterentwicklung sind Kenntnisse in Swift und dem Arbeiten in der Entwicklungsumgebung Xcode notwendig. Im Bezug auf unsere App haben wir mit Xcode Version 10.2.1 gearbeitet.
Die kostenlose Entwicklungsumgebung Xcode kann im App Store heruntergeladen werden. Für den privaten Gebrauch und zur Weiterentwicklung wird eine Apple-ID benötigt.
Für den Zugriff auf die Datenbank, also Firebase (Authentication, Realtime Database, Storage), benötigt man ein Google-Konto und das Zugriffsrecht auf das Projekt ScrumMadeDB unter dem die ganze Businect App läuft.

Folgende Systemvoraussetzungen für Xcode werden bei einem Mac empfohlen:

- Intel i5 oder i7 Prozessor mit 1,5 Ghz Taktfrequenz
- Mindestens 4 GB Arbeitsspeicher 8 GB werden empfohlen
- Mindestens 10 GB Festplattenspeicher

Xcode ermöglicht es die App auf nahezu allen mobilen Apple-Geräten zu simulieren, für den QR-Code-Scanner, der die Kamera des Gerätes verwendet muss und die AR-Umgebung anzeigt, benötigen Sie jedoch ein physisches Endgerät, um diese Funktion testen und ausführen zu können.

4 Entwicklerumgebung

Xcode ist ein Tool, welches zum Entwickeln von iOS, macOS, tvOS und watchOS verwendet wird. Es handelt sich dabei um eine von Apple generierte integrierte Entwicklerumgebung. Die Businect-App wurde mit Hilfe von Xcode (Version 10.2.1) realisiert. Hauptsächlich werden die Programmiersprachen Swift und Objective-C mit dem Cocoa-Framework verwendet.
Bei der Arbeit mit Xcode übernimmt die Programmsammlung alles, was für die SoftwareEntwicklung notwendig ist.

Um grafische Benutzeroberflächen wie (GUI) für Mac OS und iOS zu erstellen, existiert der Interface Builder seit Version 4.0 in Xcode. Die Anbindung bestimmter Objekte mit Aktionen ist auch darüber möglich.

Ein weiterer großer Vorteil ist der iPhone Simulator verschiedene iPhone Versionen darstellen kann. Das ermöglicht Entwicklern damit Programme zu schreiben und diese auf einem virtuellen Gerät zu testen, ohne selbst ein iPhone physikalisch in der Nähe zu haben oder zu besitzen. Es existieren jedoch kleine Einschränkungen.

Damit die Performance von Programmen in Xcode überprüft werden kann existiert das integrierte Programm Instruments. Es enthält Analysefunktionen, um CPU-Auslastung, Speicherverbrauch und weitere Eigenschaften von geschriebenen Programmen zu überwachen. Instruments kann daher auch beim Debugging hilfreich sein.

5 Projektstruktur

5.1 Systemarchitektur

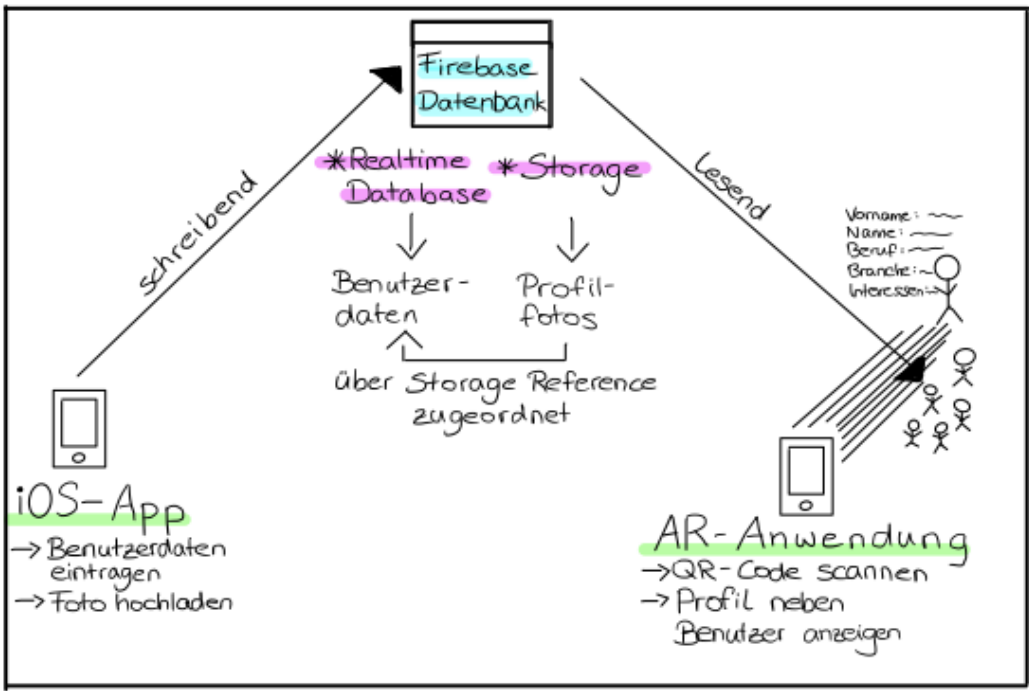

Abbildung 1: Systemarchitektur

5.2 Ordnerstruktur

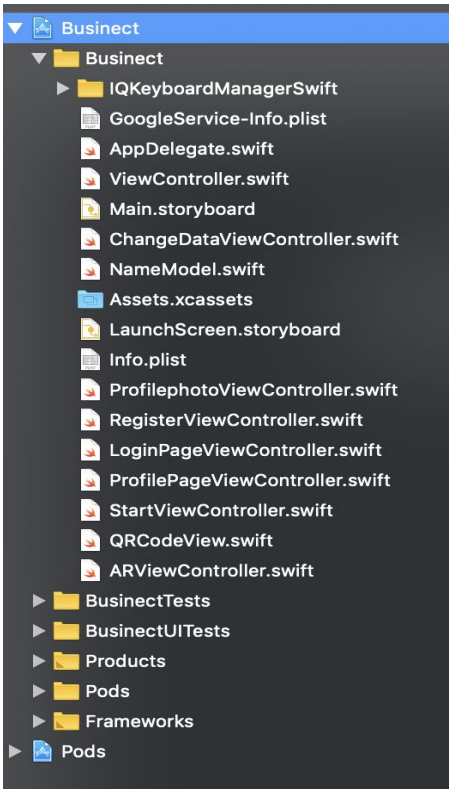

Abbildung 2: Ordnerstruktur

Die Ordnerstruktur des Projektes basiert auf der eines standard iOS Projektes. Auf der obersten Ebene befinden sich die Ordner *Businect*, *BusinectTests*, *BusinectUITests*, *Products*, *Pods* und *Frameworks*.
In dem Ordner *Businect* sind die Klassen enthalten, welche den einzelnen ViewControllern eine Funktion zuweisen.
BusinectTests und *BusinectUITests* sind Ordner, um Testläufe für die App zu generieren.
Anzumerken ist, dass sich hierüber keine AR Tests ausführen lassen, da die AR

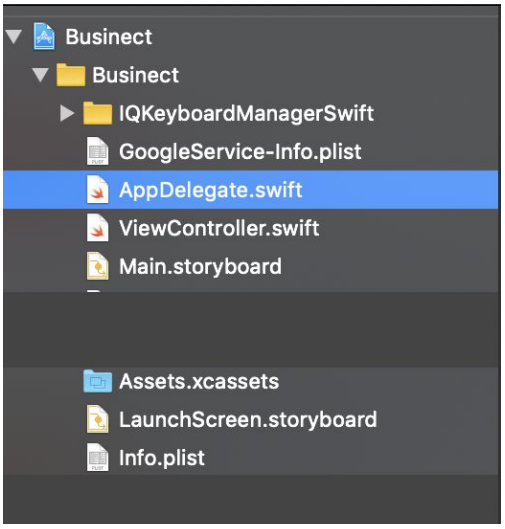

Abbildung 3: App Komponenten

Funktionen der App grundlegend auf der Kamera und auf den Bewegungen eines physischen Gerätes basieren. Diese sind über die Testklassen nur schwer oder gar nicht nachzustellen. Der Ordner *Products* kann fertig kompilierte Versionen der App enthalten. Der Ordner *Pods* enthält die externen Bibliotheken die zu dem Projekt hinzugefügt wurden. In dem Ordner *Frameworks* können Frameworks hinzugefügt werden.

Das *AppDelegate* wird über applikationsweise Ereignisse benachrichtigt - zum Beispiel über Lebenszyklus-Ereignisse wie den Start der App oder die Verlagerung der App in den Hintergrund.

- *ViewController*-Klassen implementieren das Verhalten der Views der App, sie befüllen sie zum Beispiel mit Daten und behandeln Ereignisse.
- *Assets.xcassets* enthält Icons und Grafiken in verschiedenen Auflösungen, unter anderem das App-Icon
- *LaunchSceen.storyboard* enthält den Startbildschirm der angezeigt wird, bis die App vollständig geladen und initialisiert ist.
- *Info.plist* enthält Konfigurationsoptionen, die zur Laufzeit geladen werden, zum Beispiel die Versionsnummer der App oder unterstützte Geräte-Drehungen.

Im Main.storyboard können wir jeden einzelnen ViewController verändern und gestalten. Es besteht auch die Möglichkeit neue ViewController zu erstellen. Jedem ViewController muss eine Klasse mit Funktionen für den ViewController zugeordnet werden. Wie man eine Klasse erstellt und diese einem View Controller zugeordnet wird in Kapitel 7.1 und 7.2 erklärt.

Abbildung 4: Main.Storyboard

5.3 Klassendiagramme

In diesem Kapitel gehen wir auf die Projektstruktur ein. Sie lernen, wie Sie das Projekt öffnen können und sich erfolgreich zu orientieren. Nachdem Sie Xcode heruntergeladen haben, öffnen Sie bitte Xcode.

Abbildung 5: Xcode Startansicht

Danach wird Ihnen rechts das Project “Businect” angezeigt. Sie können dieses direkt öffnen. Falls das Projekt nicht angezeigt wird wählen Sie “Open another project" aus und suchen das Projekt im Finder.

6 Verwendete Technologie

6.1 AR

Augmented Reality (AR) wird übersetzt als erweiterte Realität bezeichnet. Es handelt sich bei AR um eine Computergestützte Wahrnehmung, welche die Reale Welt durch computergenerierte Inhalte erweitert. Außerdem werden virtuelle Objekte

die aus Texten, Grafiken oder Tönen bestehen können in Echtzeit direkt oder indirekt in die Realität integriert. Das ARToolKit von Apple bietet dabei den Entwicklern ein sogenanntes ”out-of-the-box” Konzept zur Realisierung von Augmented Reality Applikationen an, womit sich Anwendungen schnell und einfach realisieren lassen sollen.

7 Entwicklungs-Details iOS-App

7.1 ViewController Klassen erstellen

Um eine ViewController Klasse zu erstellen wählen Sie File aus gehen auf New und dann auf File.

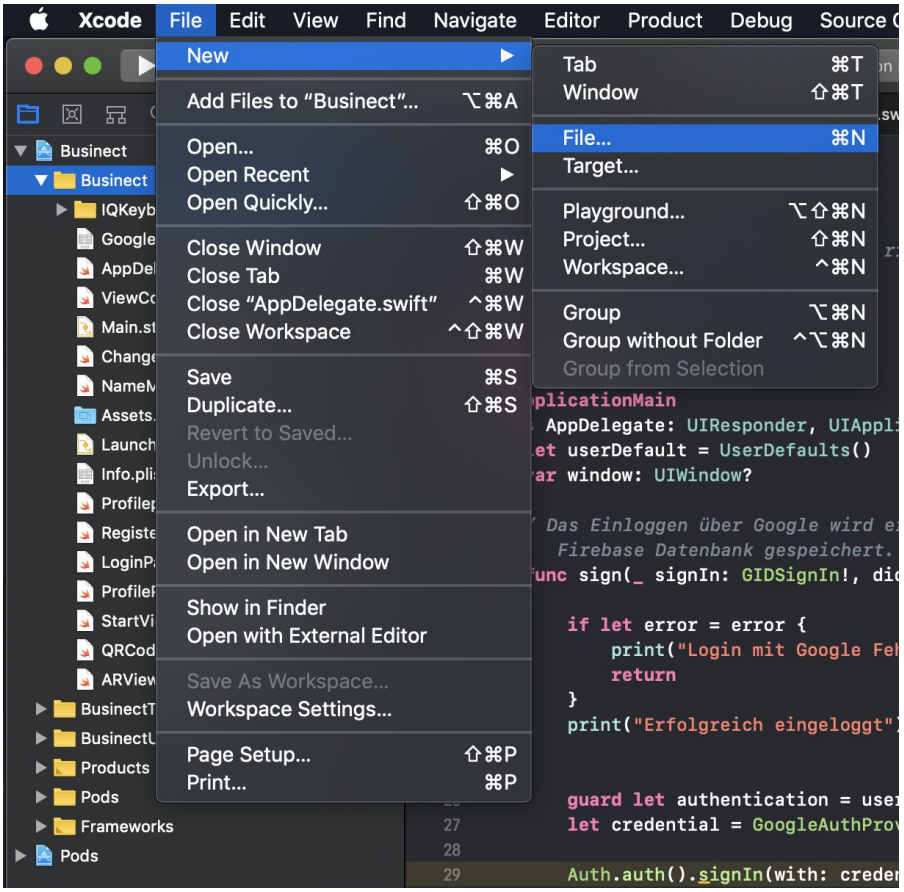

Abbildung 6: Klasse erstellen

Danach gehen Sie auf Cocoa Touch Class und klicken “Next”.

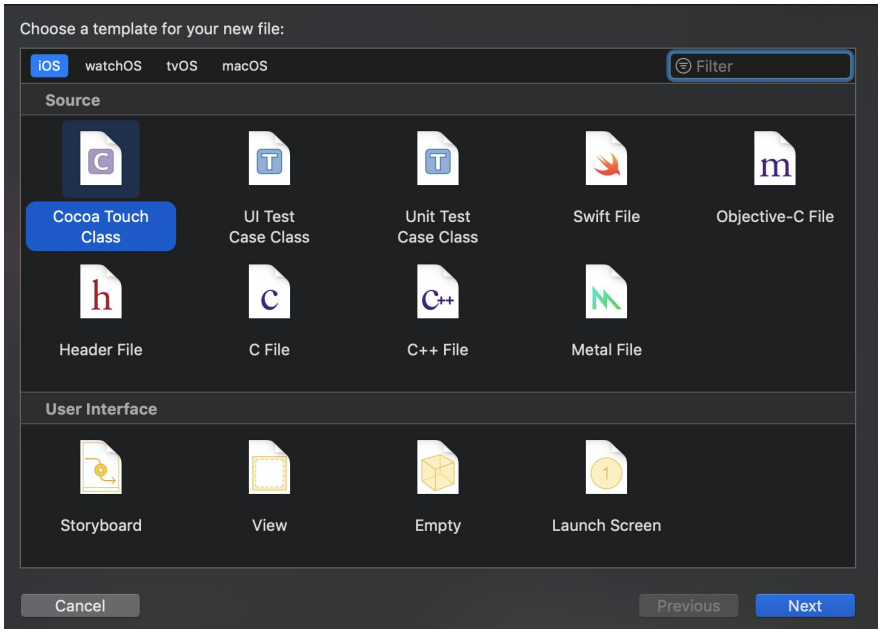

Abbildung 7: Klassentyp auswählen

Im Feld Class kann die Klasse View Controller benannt werden, dabei muss die Subclass “UIViewController” sein. Sobald man auf “Next” klickt, klicken Sie im Finder-Fenster auf “Create” klicken. Dann wird die Klasse in dem Projekt erstellt.

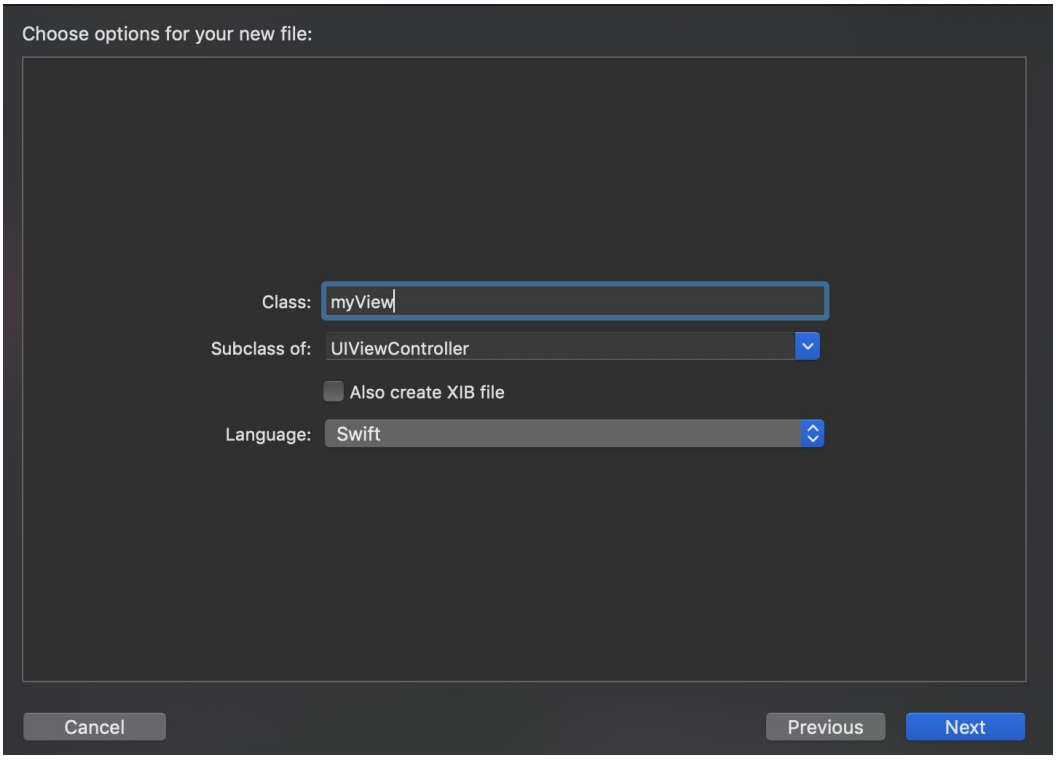

Abbildung 8: ViewController benennen

7.2 ViewController mit der Klasse verknüpfen

Damit Sie in einem ViewController die Klasse zuordnen können, müssen Sie in das Main.Storyboard gehen.

Im Main.Storyboard wählen Sie den ViewController aus, indem auf der Leiste über dem ViewController auf dem gelben Kreis klicken. Mit diesem Klick wählen Sie diesen ViewController aus.

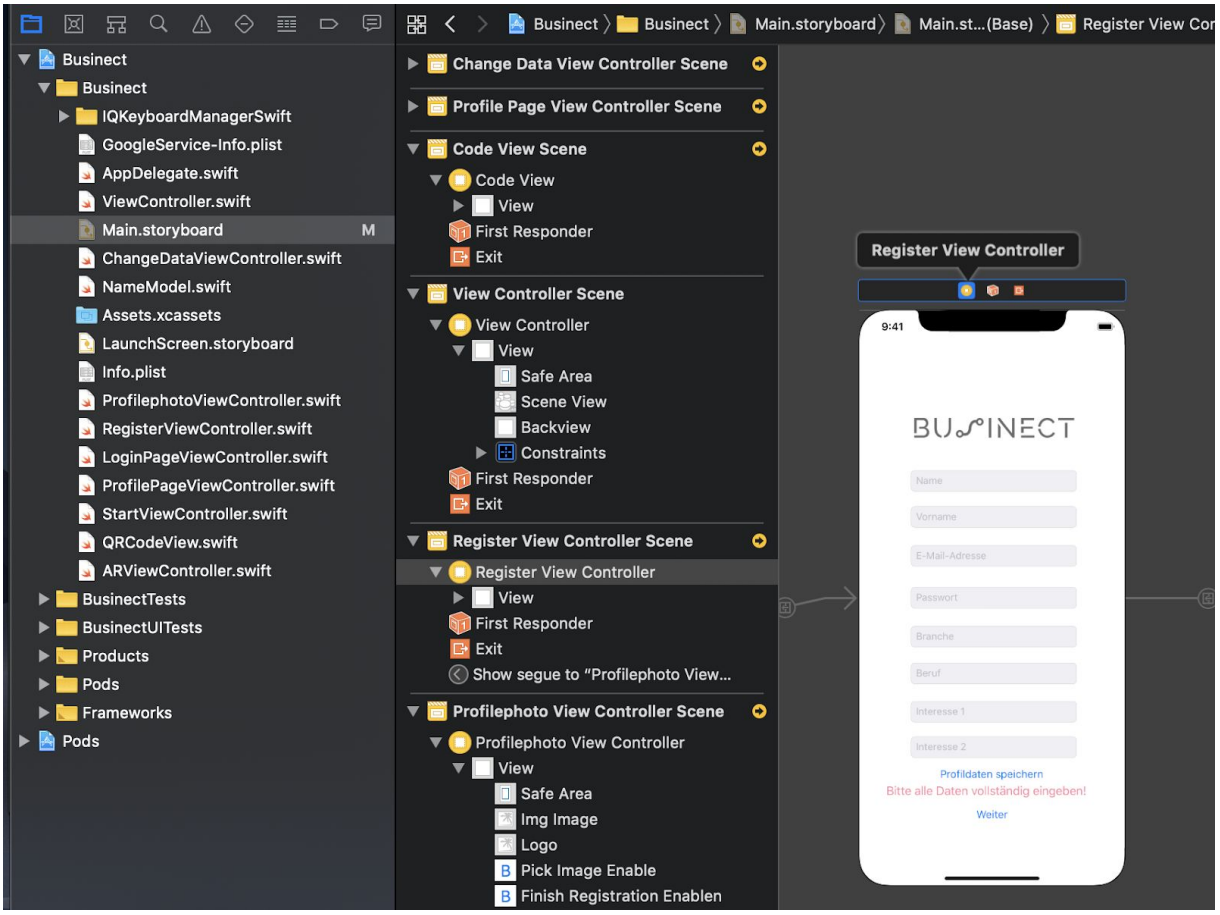

Abbildung 9: ViewController auswählen

Als nächstes wird Ihnen in xCode auf der rechten Seite folgende Leiste angezeigt:

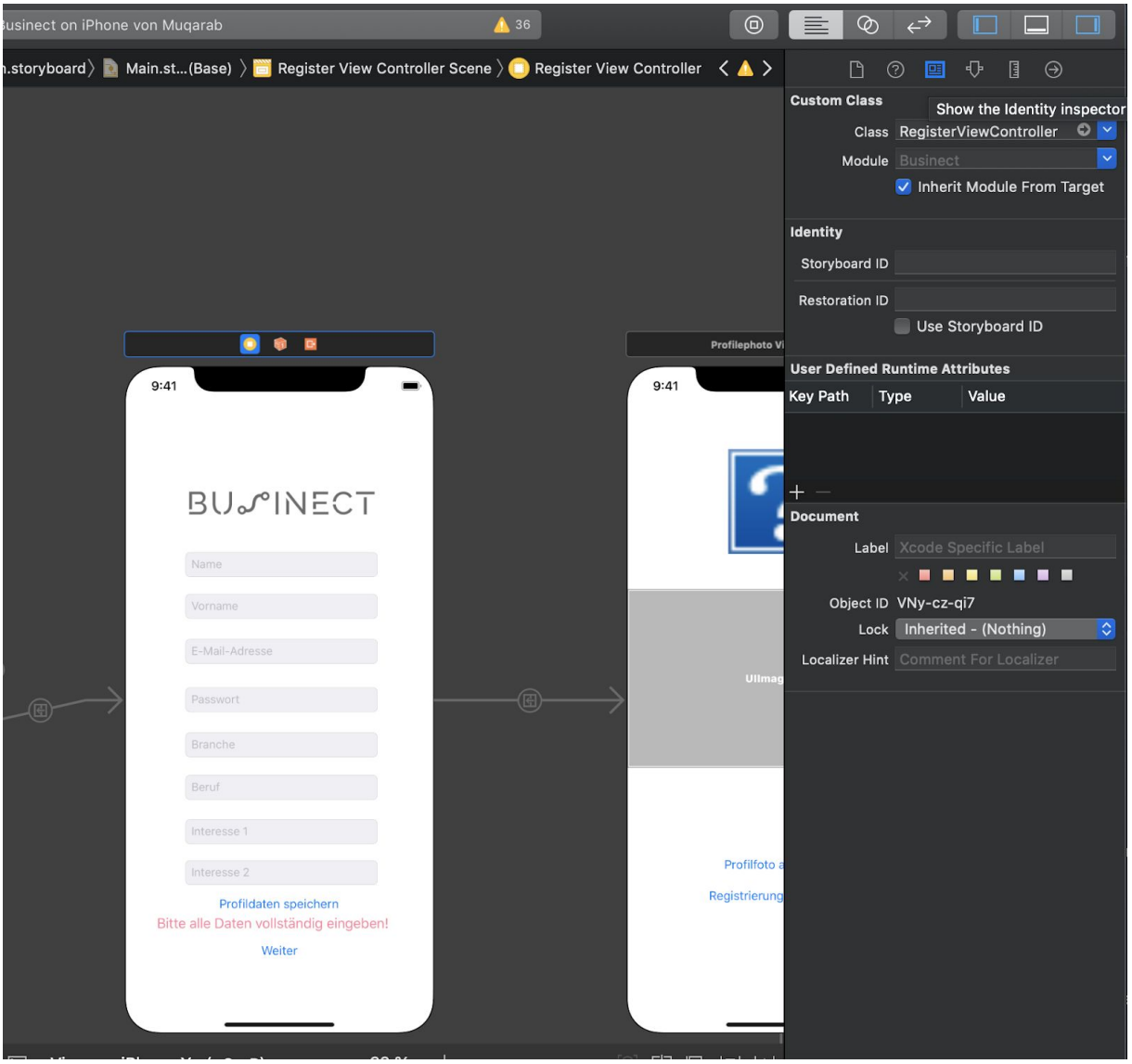

Abbildung 10: ViewController einer Klasse zuordnen

Dort muss der “identify inspector” ausgewählt werden und in “Class” muss die Klasse mit den Funktionen für den ViewController ausgewählt werden.

7.3 Erläuterung der Klassen

7.3.1 StartViewController

Die Klasse “StartViewController.swift” stellt die Startseite der Businect App dar, die beim Starten der App aufgerufen wird. Sie ist der View Controller, der vom Navigation Controller als Startpunkt gesetzt wurde. Somit können Sie mit Hilfe des “Back”-Buttons immer wieder hierhin zurück gelangen.

Auf der Startseite haben Sie die Möglichkeit sich zur Registrierung zu bewegen oder zum Anmeldefenster zu gelangen, falls Sie schon ein Konto haben.
Sind Sie bereits angemeldet haben Sie Zugriff auf Ihre Profilseite und auf den QR-Code-Scanner, und Sie können sich entsprechend hier auch wieder abmelden.

Die Klasse besitzt fünf globale Variablen:

- *loginButton* beinhaltet den Button, der den LoginPageViewController öffnet, also das Anmeldefenster
- *logoutButton* beinhaltet den Button, der den aktuell angemeldeten Benutzer abmeldet
- *profileLink* beinhaltet den Button, der die Profilseite den ProfilePageViewController öffnet
- *registerButton* beinhaltet den Button, der zur ersten Registrierseite RegisterViewController führt
- *qrScanner* beinhaltet den Button, der den ARViewController startet

Des Weiteren besitzt die Klasse X Methoden:

- *viewDidLoad*() lädt die Startseite mit dem Businect Logo und den Buttons loginButton, logoutButton, profileLink, registerButton und qrScanner, welche entsprechend bei angemeldeten Nutzern bzw. nicht angemeldeten Nutzern anklickbar sind
- *viewWillAppear*(_ animated: Bool) ist eine Funktion die vor dem Laden der Startseite im Hintergrund abläuft, und prüft, ob man angemeldet ist und entsprechend die richtigen Button freischaltet oder nicht
- *logOutButtonClicked*(_ sender: Any) wird aufgerufen, wenn der Button logoutButton angeklickt wird, und versucht den akltuellen Nutzer in der Firebase Authentication abzumelden, falls einer gefunden wird und dann nur noch die Buttons loginButton und registerButton freigibt

7.3.2 LoginPageViewController

Die Klasse “LoginPageViewController.swift” ist zuständig für das Anmeldefenster, wobei man entweder die Eingabefelder Email und Passwort zur Verfügung hat oder eine Leiste mit der man sich mit seinem schon bestehenden Google-Konto anmelden kann ohne sich bei der Businect manuell registrieren zu müssen.

Die Klasse besitzt drei globale Variablen:

- *userNameTextField* ist das Eingabefeld, wo man seine Email-Adresse seines Businect-Kontos eingeben soll
- *passwordTextField* ist das Eingabefeld, wo man sein Passwort eingibt
- *errorLabel* ist ein festes Textfeld mit dem Text “There was an error with authentication”, welches bei fehlgeschlagenem Anmeldeversuch erscheint

Des Weiteren besitzt die Klasse fünf Methoden:

- *viewDidLoad*() lädt das Anmeldefenster mit den beiden Eingabefeldern, Anmeldebutton und Googleanmeldebutton
- *setupGoogleButtons*() erstellt die Leiste auf die man klicken kann um sich mit seinem GoggleKonto anzumelden
- *loginClickButton(_ sender: Any)* wird aufgerufen, wenn der Anmeldebutton geklickt wird. Dabei wird geprüft ob es ein Konto in der Firebase Authentication mit der eingegebenen Email und zugehörigem Passwort gibt. Falls ja wird man angemeldet und gelangt zur Startseite zurück. Falls nicht wird der errorLabel erscheinen und man kann das Anmelden erneut versuchen
- *touchesBegan(_ touches: Set<UI Touch>, with event: UIEvent?)* ermöglicht es die Tastatur mit einem Klick auf die Anmeldeseite auszublenden
- *textFieldShouldReturn(_ textField: UITextField)* ist zuständig dafür, dass man mit der Return-Taste auf der iPhone-Tastatur auf das nächste Eingabefeld sofort hingelangt

7.3.3 RegisterViewController

Die Klasse “RegisterViewController.swift” beschreibt die Registrierseite bei dem man sowohl seine Kontodaten E-Mail und Passwort eingeben als auch seine Benutzerinformationen Name, Beruf, Branche und Interessen eingeben muss um sich ein Businect-Konto erstellen lassen zu können. Das Passwort darf nicht kürzer als 8 Zeichen lang sein und die Email muss auch gültig sein. Alle anderen Informationen kann man im Nachhinein in seinem Profil zu einem späteren Zeitpunkt ändern.

Die Klasse besitzt 12 globale Variablen:

- *refName* ist eine Referenz zu der Firebase Database mit dem man Zugang auf diese bekommt
- *textFieldName* beinhaltet das Eingabefeld für den Namen des Benutzers
- *textFieldVorname* beinhaltet das Eingabefeld für den Vornamen des Benutzers
- *textFieldEmail* beinhaltet das Eingabefeld für die E-Mail des Benutzers
- *textFieldPasswort* beinhaltet das Eingabefeld für das Passwort des Benutzers
- *textFieldBeruf* beinhaltet das Eingabefeld für den Beruf des Benutzers
- *textFieldBranche* beinhaltet das Eingabefeld für die Branche des Benutzers
- *textFieldInteresse1* beinhaltet das Eingabefeld für die erste Interesse des Benutzers
- *textFieldInteresse2* beinhaltet das Eingabefeld für die zweite Interesse des Benutzers
- *buttonEnablen* ist der Button, der die eingegeben Daten prüft und akzeptiert und dann den “Weiter”-Button freigibt
- *errorLabel* ist das Textfeld, das ausgegeben wird wenn es ein Fehler mit den eingegebene Daten gibt
- *continueButton* ist der Button, der zum nächsten Registrierungsschritt führt und den ProfilephotoViewController aufruft

Des Weiteren besitzt die Klasse sieben Methoden:

- *viewDidLoad*() ruft die Seite auf mit den Eingabefeldern und dem Button “Profildaten” speichern
- isValidEmail(testStr: String) prüft ob der mitgegeben String eine gültige EMail sein kann
- *addName*() erstellt einen neuen Nutzer in der Firebase Authentication mit Email und Passwort, und einen neuen Datensatz in der Firebase Realtime Database mit allen eingeben Informationen unter der ID des Vor- und Nachnamen
- *buttonAddUser*(_ sender: UIButton) wird aufgerufen, wenn man auf den Button “Profildaten speichern” klickt. Hier wird zunächst geprüft ob alle Felder belegt wurden und es wird die Methode isValidEmail() mit der eingegebenen Email geprüft als auch die Länge des Passworts. Wenn alle Daten korrekt waren wird der “Weiter”-Button freigegeben, ansonsten erscheint das errorLabel mit einem Fehler
- *clickContinue(_ sender: Any)* wird beim Klick auf “Weiter” aufgerufen und gibt dem Benutzerkonto in der Firebase Authentication die ID zum Datensatz des Nutzers als Info mit damit man später darauf zugreifen kann und leitet dann den nächsten Schritt zum ProfilePhotoViewController hin
- *touchesBegan(_ touches: Set<UI Touch>, with event: UIEvent?)* ermöglicht es die Tastatur mit einem Klick auf die Anmeldeseite auszublenden
- *textFieldShouldReturn(_ textField: UITextField)* ist zuständig dafür, dass man mit der Return-Taste auf der iPhone-Tastatur auf das nächste Eingabefeld sofort hingelangt

7.3.4 ProfilephotoViewController

Diese Klasse “ProfilephotoViewController.swift” ist dazu da, dass Nutzer der App ein Foto aus ihrer Galerie auf ihrem Smartphone in der Datenbank von Businect hochladen und speichern können. Dieses können sie anschließend in ihrem Profil

einsehen und kann bei weiteren Updates der Businect-App noch zu weiteren Features genutzt werden. Wir haben von Anfang an die Möglichkeit des Foto hochladens implementiert, damit sichergestellt werden kann, dass jeder Nutzer auch ein Profilfoto hinterlegt hat.

Die Klasse besitzt fünf globale Variablen:

- *imgImage* beinhaltet den Anzeigebereich des Fotos in der App.
- *imagePicker* beinhaltet das ausgewählte Foto aus der Galerie
- *pickImageEnablen* beinhaltet die Erlaubnis den Button zu klicken, der die Funktion hat, dass die Galerie des Benutzer geöffnet wird
- *finishRegistrationEnablen* beinhaltet die Erlaubnis des Button zu klicken, welcher das ausgewählte Foto in der Firebase Datenbank speichert und daraufhin die Profilseite der Businect-App öffnet.
- *imageReference* beinhalet das ausgewählte Foto

Des weiteren besitzt die Klasse vier Methoden:

- *viewDidLoad()* wird aufgerufen, sobald sich die Seite zum Profilfoto hochladen öffnet. Diese gibt dann den Button zum Profilfoto auswählen frei, verhindert jedoch das Anklicken des Buttons zum Profilfoto hochladen.
- *galleryTapped()* öffnet die Galerie des Nutzers. Außerdem änders sie die Zustände von *finishRegistrationEnablen* und *pickImageEnablen* werden auf das Gegenteil gesetzt, sodass der Nutzer nun kein Foto mehr auswählen kann, aber das ausgewählte Foto hochladen und gelangt dann auf seine Profilseite.
- *uploadImgToFirebase()* ist mit dem Button “Registrierung abschließen” verknüpft. Sie legt fest, dass sobald dieser Button vom Nutzer geklickt wird, das Foto, welches sich gerade im ImageView befinde, in den Firebase Storage gespeichert wird. Das Foto wird dort im Dateiformat .png gespeichert. Außerdem wird das Foto mit dem Namen der E-Mail Adresse des Nutzers und dem Suffix .png gespeichert.

- *imagePickerController()* bestimmt was passiert, sobald man auf ein Foto in der zuvor geöffneten Galerie klickt. Und zwar wird dieses in das ImageView auf der Seite der App eingefügt.

7.3.5 ProfilePageViewController

Die Klasse "ProfilePageViewController" beinhaltet alle Funktionen der Profilseite in der Busincet-App. Diese zeigt alle benutzerspezifischen Daten des eingeloggten Nutzers an, gibt ihm die Möglichkeit seine Verfügbarkeit auf "An" oder "Aus" zu stellen und die Möglichkeit sein Profilfoto anzeigen zu lassen.

Diese Klasse besitzt 14 globale Variablen:

- *refName* beinhaltet die Referenz der benutzerspezifischen Daten des Nutzers, die in der der Firebase Datenbank gespeichert sind
- *lblVorname* beinhaltet den Vornamen den Nutzers
- *lblBeruf* beinhaltet den Beruf des Nutzers
- *lblBranche* beinhaltet die Branche des Nutzers
- *lblEMail* beinhaltet die EMail des Nutzers
- *lblInteresse1* beinhaltet eine Interesse des Nutzers
- *lblInteresse2* beinhaltet eine Interesse des Nutzers
- *blName* beinhaltet den Namen des Nutzers
- *downloadImage* ist des ImageView, welches das Foto des Nutzers anzeigen kann
- *availability* beinhaltet den Boolean-Wert der Verfügbarkeit des nutzers
- *switchOutlet* beinhaltet den Zustand des Switches
- *stateSwitch* beinhaltet die Veränderung des Switches
- *user* beinhaltet alle Eigenschaften des angemeldeten Nutzers
- *imageReference* beinhaltet das Profilfoto des angemeldeten Nutzers, welches im Firebase Storage gespeichert ist

Des weiteren besitzt diese Klasse vier Methoden:

- *switchAction()* legt fest, was passiert, wenn der Zustand des Switches geändert wird
    - lokale Variablen:
- *stateChanged()* übernimmt den aktuellen Zustand des Switches in die Realtime Firebase Datenbank. Demnach wird dort die Verfügbarkeit auf “true”gesetzt, wenn der Switch an ist, vice versa.
    - lokale Variablen:
- *viewDidLoad()* wird ausgeführt, sobald die Profilseite der Businect-App aufgerufen wird. Sie überträgt Vorname, Name, E-Mail Adresse, Beruf, Branche, Interessen und die Verfügbarkeit aus der Realtime Firebase Datenbank in das Profil.
- *onDownloadTapped()* lädt das Profilfoto des Nutzers in den das ImageView auf der PRofilseite, sobald man auf den Button “Profilfoto anzeigen” klickt.

7.3.6 ChangeDataViewController

Die Klasse “ChangeDataViewController.Swift” führt alle Funktionen aus, um die benutzerspezifischen Daten des angemeldeten Nutzers von ihm geändert werden können. Dafür gibt es eine separate Seite der Businect-App.

Diese Klasse besitzt vier globale Variablen:

- *textFieldBeruf* beinhaltet das Eingabefeld, in das man den aktuellen Beruf eingeben kann
- *textFieldBranche* beinhaltet das Eingabefeld, in das man die aktuelle Branche eingeben kann
- *textFieldInteresse1* beinhaltet das Eingabefeld, in das man eine aktuelle Interesse eingeben kann
- *textFieldInteresse2* beinhaltet Eingabefeld, in das man eine aktuelle Interesse eingeben kann

Des weiteren besitzt diese Klasse vier Methoden:

- *changeBeruf()* überschreibt den vorherigen gespeicherten Beruf, mit dem was in das Textfeld “Beruf ändern” eingegeben wurde, sobald der Benutzer auf den Button “Ändern” hinter dem Textfeld klickt.
- *changeBranche()* überschreibt die vorherige gespeicherte Branche, mit dem was in das Textfeld “Branche ändern” eingegeben wurde, sobald der Benutzer auf den Button “Ändern” hinter dem Textfeld klickt.
- *changeInteresse1()* überschreibt die vorherige gespeicherte Interesse, mit dem was in das Textfeld “Interesse1 ändern” eingegeben wurde, sobald der Benutzer auf den Button “Ändern” hinter dem Textfeld klickt.
- *changeInteresse2()* überschreibt die vorherige gespeicherte Interesse, mit dem was in das Textfeld “Interesse2 ändern” eingegeben wurde, sobald der Benutzer auf den Button “Ändern” hinter dem Textfeld klickt.

7.3.7 QRCodeView

Die Klasse “QRCodeView.Swift” führt alle Funktionen aus, um einen QR-Code zu erstellen, diesen direkt im Firebase Storage zu speichern und diesen zu teilen.

Diese Klasse besitzt drei globale Variablen:

- *myImageView* beinhaltet den Anzeigebereich für den QR-Code
- *button* beinhaltet den Button dem die Funktion für das teilen zugeordnet wird
- *imageReference* gibt eine Referenz für den Screenshot an damit auf die Storage zugegriffen werden kann.

Des weiteren besitze diese Klasse zwei Methoden:

- *viewdidLoad()* ist eine Methode, die in fast jeder Klasse vorhanden ist. Die Methode wird direkt beim Laden der Klasse ausgeführt. Wir benutzen diese Methode um den QR-Code direkt zu generieren, wenn man vorher im Profil den Button “Lade QR-Code” drückt. Außerdem lädt die Methode einen Screenshot des QR-Codes auf unser Firebase Storage hoch.

- *shareMethod()* nutzt die teilen Funktion des iPhones. Diese Methode ermöglicht den Screenshot des QR-Codes zu drucken, in den Fotos zu sichern und vieles mehr.

7.3.8 ARViewController

Die Klasse "ARViewController.swift" organisiert die ganze AR-Umgebung und sorgt dafür, dass die angezeigten Elemente direkt beim gescannten QR-Code bzw. Nutzern angezeigt werden. Für diese Funktionen wird die Kamera des Endgeräts verwendet. Man hat also vor sich ganz normal die Kameraansicht und sobald ein QR-Code in Sichtfeld ist, wird sofort die AR-Umgebung gestartet.
Die Elemente des gescanntes QR-Codes werden in 3D angezeigt und passen sich beim Bewegen entsprechend an. Es gibt hierbei zwei mögliche Ausgaben.
Entweder der gescannte Nutzer ist verfügbar, in dem Fall werde alle seine Informationen in grün angezeigt, oder er hat sich auf nicht verfügbar gesetzt, in dem Fall wird in rot "Nicht Verfügbar" angezeigt.

Die Klasse besitzt zwölf globale Variablen:

- *sceneView* beinhaltet die AR-Umgebung in dem alle Elemente augmented erscheinen
- *backview* beinhaltet die Kameraansicht zu Beginn, die für das Scannen der QR-Codes zuständig ist
- *user* beinhaltet einen Datensatz in der Firebase Database zur Durchsuchung aller Daten
- *refName* ist eine Referenz zu der Firebase Database mit dem man Zugang auf diese bekommt
- *tempName* beinhaltet die Zwischenspeicherung für den Namen des Benutzers
- *tempVorname* beinhaltet die Zwischenspeicherung für den Vornamen des Benutzers
- *tempEmail* beinhaltet die Zwischenspeicherung für die Email des Benutzers

- *tempPasswort* beinhaltet die Zwischenspeicherung für das Passwort des Benutzers
- *tempBeruf* beinhaltet die Zwischenspeicherung für den Beruf des Benutzers
- *tempBranche* beinhaltet die Zwischenspeicherung für die Branche des Benutzers
- *tempInteresse1* beinhaltet die Zwischenspeicherung für die erste Interesse des Benutzers
- *tempInteresse2* beinhaltet die Zwischenspeicherung für die zweite Interesse des Benutzers
- *video* ist die Live-Videoansicht die auf dem backview läuft damit man auf seinem Gerät sieht was man gerade scannt

Des Weiteren besitzt die Klasse zwei Methoden:

- *viewDidLoad()* startet die Session und die Kameraansicht wird gestartet. Falls man den QR-Code- Scanner das erste Mal startet wird man zuerst um Zugriff auf die Kamera gefragt.
- *metaDataOutput()* läuft im Hintergrund und wird aufgerufen sobald ein QR-Code entdeckt wird. Hier wird dann erstmal der Nutzers des QR-Codes ermittelt und dann überprüft, ob er verfügbar ist. Falls ja werden seine Daten angezeigt, falls nicht wird “nicht verfügbar” ausgegeben

7.3.9 AppDelegate

Die Klasse “AppDelegate” wird beim erstellen einer App mit den Xcode Templates automatisch erstellt. Diese Klasse soll für die Verarbeitung von Ereignissen im Anwendungslebenszyklus vorgesehen sein, d.h. als Reaktion auf das Starten der Anwendung, das Hinterlegen, das Empfangen von Daten usw.. Sie besteht am Anfang aus mehreren leeren Methoden, deshalb werden wir nur auf die Methoden eingehen die wir verwendet oder verändert haben.

Diese Klasse besitzt eine globale Variable:

- *window* beinhaltet das Fenster, in dem der visuelle Inhalt der App auf dem Hauptbildschirm des Geräts angezeigt wird.

Des weiteren besitze diese Klasse acht Methode. Auf drei davon werden wir genauer eingehen:

- *sign()* übergibt dem Google Sign in Button die Funktion.
  Außerdem wird ermöglicht, dass der Google Benutzer Name und die E- Mail in der Firebase Datenbank gespeichert werden
- *application(_* application: UIApplication, didFinishLaunchingWithOptions launchOptions: [UIApplication.LaunchOptionsKey: Any]?*)* wird benutzt um einzelne Schnittstellen aufzurufen und in der ganzen App zu aktivieren
- *application(_* application: UIApplication, open url: URL, options: [UIApplication.OpenURLOptionsKey : **Any**]*)* benutzt eine bestimmte Google URL, um uns für das Google Sign in auf Google weiterzuleiten.

7.3.10 NameModel

Die Klasse “NameModel.swift” enthält alle Daten, die den Benutzer beschreiben.

Diese Klasse besitzt 10 globale Variablen:

- *beruf* beinhaltet den Beruf des Nutzers
- *vorname* beinhaltet den Vornamen des Nutzers
- *id* beinhaltet die Id des Nutzers (Kombination aus Vorname und Nachname)
- *branche* beinhaltet die Branche des Nutzers
- *eMail* beinhaltet die E-Mail-Adresse des Nutzers
- *interesse1* beinhaltet eine Interesse des Nutzers
- *interesse2* beinhaltet eine Interesse des Nutzers
- *name* beinhaltet den Namen des Nutzers
- *passwort* beinhaltet das Passwort des Nutzers
- *verfuegbarkeit* beinhaltet die Verfügbarkeit des Nutzers

Des weiteren wird in dieser Klasse der Benutzer initiiert. Und zwar so, dass er alle globalen Variablen dieses Klasse enthält.

8 Datenbank

In diesem Kapitel werden wir näher erläutern, mit welchen Technologien wir die Benutzer der Businect App bei einer Registrierung oder eines Logins authentifizieren lassen und ihre Profildaten und -Fotos speichern.

Bei einer Recherche wie wir die gewünschten Funktionen am besten umsetzen können, haben wir uns zunächst verschiedene Anbieter von Datenbanken angeschaut. AWS, Azure und andere kleinere serverbasierte Anbieter entsprachen nicht in vollen Zügen unseren Vorstellungen, da wir von Anfang an einen Anbieter gesucht haben mit dem wir Benutzer authentifizieren und deren Daten speichern können. Da wir mit Google's Dienst "Firebase" in der Realtime Database Benutzerdaten speichern und verwalten, mit der Authentication die Benutzer authentifizieren und im Storage Bilddateien speichern können, fiel die Entscheidung auf die Firebase. Auch nach dem Kriterium der verfügbaren Hilfestellungen haben wir entschieden, wobei hier die Firebase z. B. gegenüber AWS Vorteile besitzt. Die Integration von Firebase in iOS-Apps wird von Google selbst hilfreich beschrieben, was wir noch durch die Recherche in Entwickler Foren ergänzen konnten. Da wir zu Beginn noch mit der Entwicklung an der HoloLens geplant haben, mussten wir auch im Vorfeld planen ob und wie wir diese mit der Datenbank verbinden können. Da uns aufgefallen war, dass die auch die Dokumentation für die Unity Integration ausführlich von Google beschrieben ist, war dies wieder ein Pluspunkt für die Firebase. Dies ist später aber überflüssig geworden, hat im Entscheidungsprozess aber trotzdem eine Rolle gespielt.

8.1 Firebase Realtime Database

Mit der Firebase Realtime Database (RTDB) kann man Daten in einer NoSQL-Cloud-Datenbank speichern und über alle Clients hinweg in Echtzeit synchronisieren lassen. Sie ist in einer Cloud gehostet und Daten werden im JSON Format gespeichert. Wenn man plattformübergreifende Anwendungen mit z. B. iOS-SDKs erstellt, teilen sich alle Clients eine Echtzeit Datenbankinstanz und erhalten automatisch Updates mit den neuesten Daten.

Die Realtime Database von Firebase nutzen wir um die Benutzer und deren Daten zu speichern.

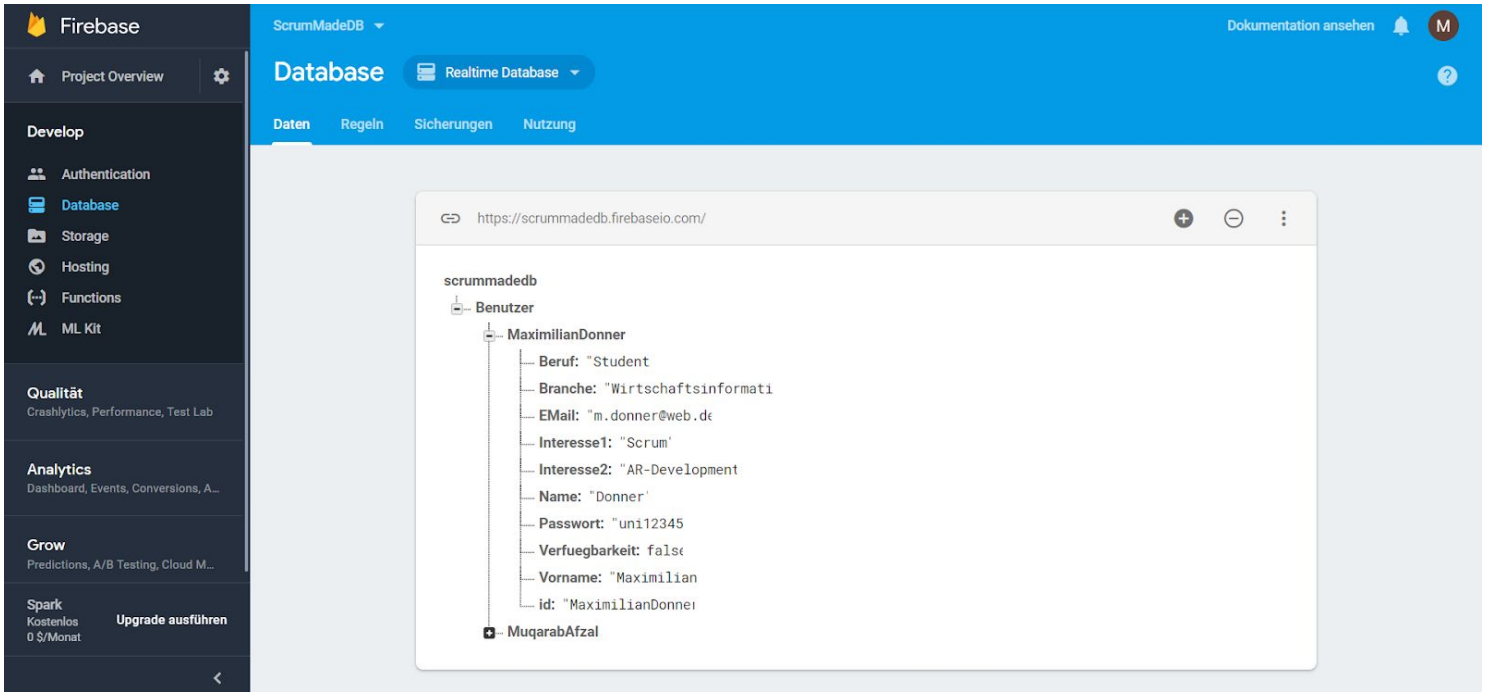

Abbildung 11: Realtime Database Benutzer

In der Abbildung 11 kann man die simple Struktur der RTDB erkennen: unsere gesamte Datenbank wird von dem Wurzelknoten “scrummadedb” (der Name, der für das aktive Firebase Projekt gewählt wurde) angeführt, wonach nur noch der Knoten “Benutzer” hinterlegt ist. Registriert sich nun ein neuer Nutzer über die Businect App,

wird ein neuer Child-Knoten von Benutzer erstellt, der nach dem Schema “VornameNachname” seinen Namen erhält, was sich in der Abbildung 11 erkennen lässt (MaximilianDonner & MuqarabAfzal).

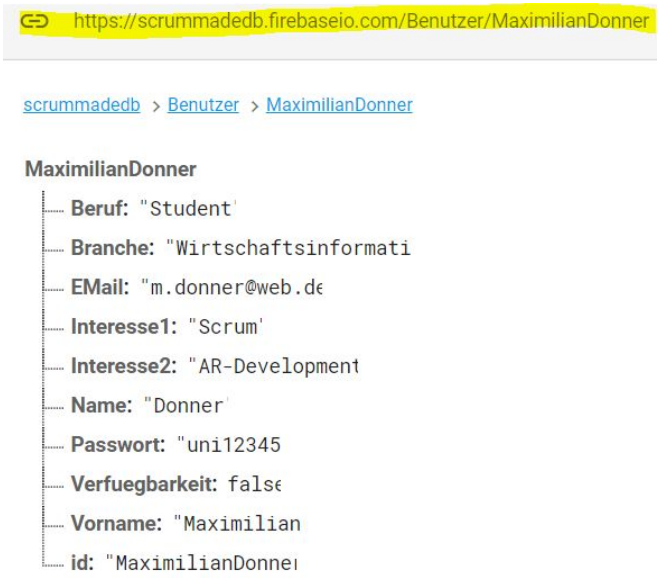

Abbildung 12: Speicherung eines Benutzers

In diesem Screenshot lässt sich nun erkennen, dass jeder gespeicherte Benutzer über eine URL, die dem Knoten Pfad folgt, geöffnet werden kann (gelb markiert), falls man Zugriff besitzt. Unter “MaximilianDonner” kann man sehen, dass die bei der Registrierung eingegebenen Informationen gespeichert worden sind und sich alle zukünftigen Änderungen unter diesem Knoten in den entsprechenden Feldern synchronisieren.

8.2 Firebase Authentication

Firebase Authentication bietet Backend-Services, SDKs und UI-Libaries zur Authentifizierung von Benutzern in unserer App. Es authentifiziert die Businect-Benutzer mit Passwörtern und E-Mail Adressen, sowie über den Identitätsanbieter Google. Zudem lässt sie sich gut mit anderen Firebase-Services

integrieren und nutzt Industriestandards wie OAuth 2.0 und OpenID Connect, so dass wir es in unser Backend implementieren konnten.

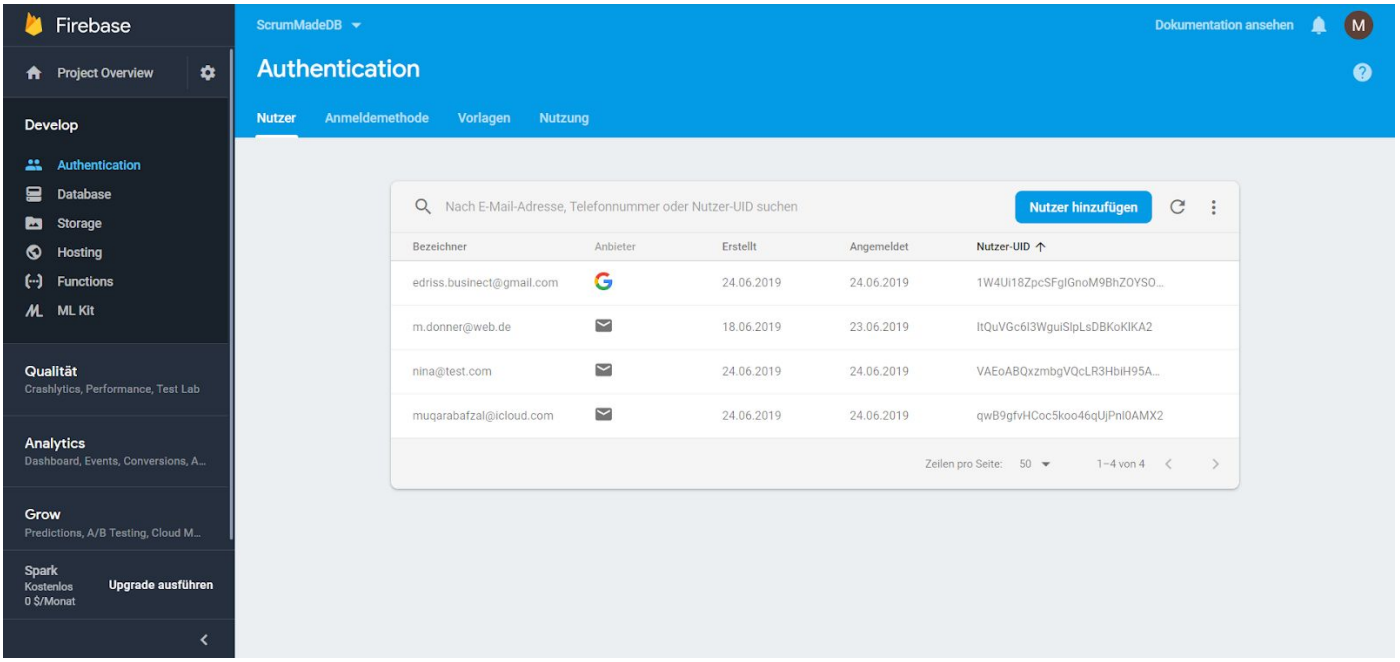

Abbildung 13: Firebase Authentication

Um einen Benutzer in der App anzumelden, erhalten wir zunächst Authentifizierungsdaten vom Benutzer. Diese Anmeldeinformationen bestehen aus der E-Mail-Adresse und dem gewählten Passwort des Benutzers oder einem OAuth-Token von Google als Identitätsanbieter (Google-Sign-In). Anschließend werden diese Anmeldeinformationen an die Firebase Authentication SDK übergeben. Die Firebase Backend-Services überprüfen dann diese Anmeldeinformationen und geben eine Antwort an den Client zurück.

Nach der erfolgreichen Anmeldung kann die App auf die zu dem aktiven Benutzer referenzierten Profilinformationen zugreifen. Zudem kann die App über die authentifizierten Benutzer Daten aus der RTDB in Echtzeit lesen und schreiben. Dies mussten wir über den Zugriff in der RTDB einstellen (siehe Abbildung 13)

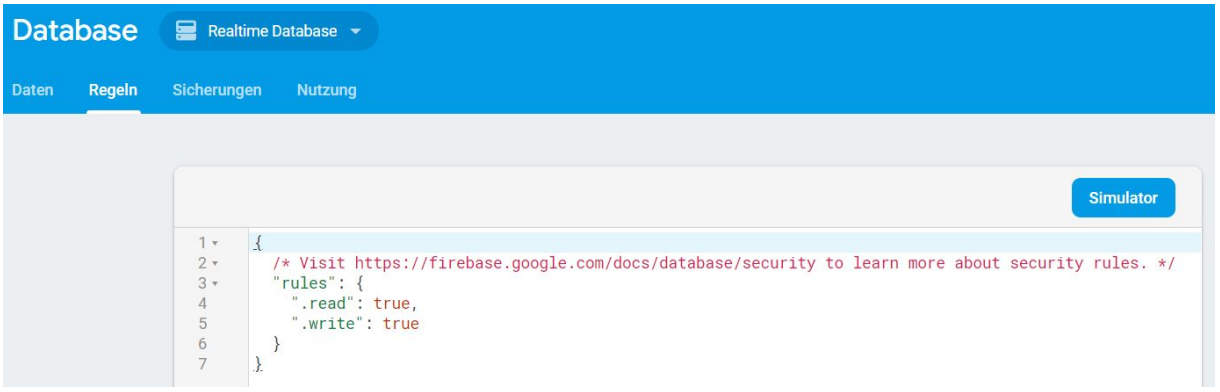

Abbildung 14: RTDB Regeln

8.3 Firebase Storage

Firebase Storage kann benutzergenerierte Inhalte wie Fotos oder Videos speichern und bereitstellen. Der Objektspeicherdienst, der in das iOS-Backend eingebaut werden kann, bietet Google Security Standards, eine hohe Skalierbarkeit und robuste Operationen, die unabhängig von der Netzqualität laufen.

Wir nutzen den Firebase Storage, um die Profilfotos und die von der App generierten QR-Codes unserer Benutzer zu speichern.

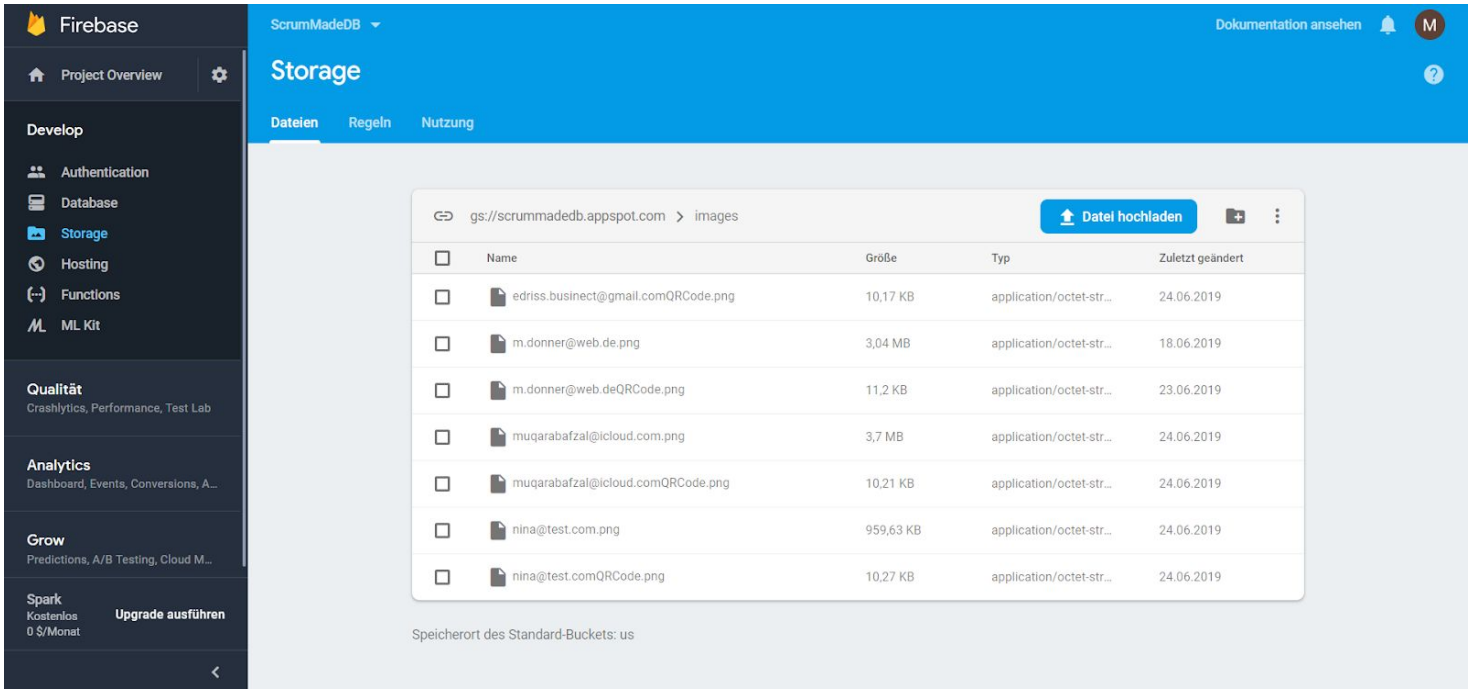

Abbildung 15: Storage mit Profilfotos und QR-Codes

In Abbildung 15 lässt sich erkennen, wie die PNG Dateien im Storage gespeichert werden: die Profilfotos werden nach der E-Mail Adresse des Benutzers und die QR-Codes mit “QR-Code” am Ende nach der E-Mail gespeichert. Mit der Bezeichnung garantieren wir eine eindeutige und redundanzfreie Speicherung der Bilddateien.