HTWG Konstanz

Fakultät für Informatik

DeepRain

Regenvorhersage mit Neuronalen Netzen und die Visualisierung dieser in einer App

MSI AS - Master Informatik, Autonome Systeme

Autoren: Simon Christofzik
Paul Sutter
Till Reitlinger

Version vom: 19. August 2020

Betreuer: Prof. Dr. Oliver Dürr

Zusammenfassung

2-3 Sätze über die Netzte. (Das reimt sich auch noch! :)) Des weiteren wurde eine App entwickelt in der die Regenvorhersagen visualisiert werden. Außerdem bietet Sie die Möglichkeit, bei bevorstehenden Regen, den Benutzer zu benachrichtigen.

Inhaltsverzeichnis

Abbildungsverzeichnis

1 Gesamtsystem

Die komplette Datenpipeline von dem Server des DWD bis zur Darstellung in der App sieht wie folgend aus.

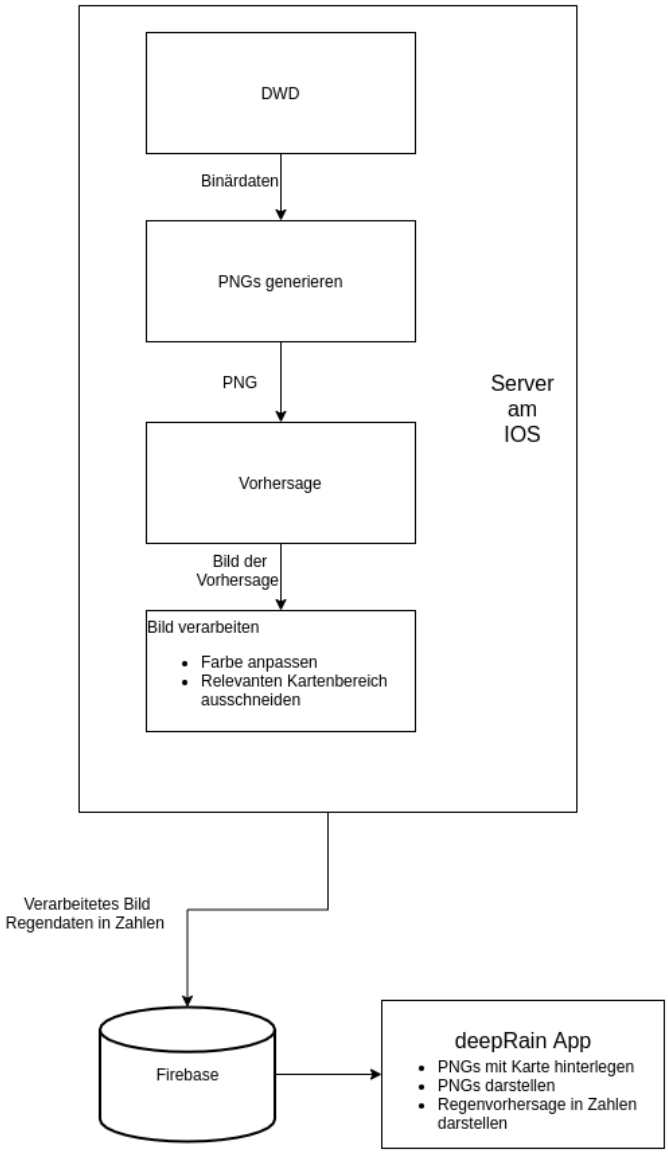


Abbildung 1: Das Gesamtsystem

Der DWD stellt alle fünf Minuten die neusten Regendaten für Deutschland im Binär Format zur Verfügung. Diese werden heruntergeladen und in der Datenaufbereitung verwendet um PNGs zu generieren (Siehe Kapitel “Datenaufbereitung”). Dieses PNGs werden im Anschluss verwendet um eine Regenvorhersage zu machen (Siehe Kapitel “Regenvorhersage / Netze”). Der Output der Regenvorhersage ist wieder ein PNG. Dieses PNG wird in der Firebase gespeichert und auf allen Geräten mit der installierten App angezeigt (Siehe Kapitel “App und Datenbank”). Der Übersichtlichkeit halber haben wir das gesamte Projekt in die drei Komponenten “Datenbeschaffung |& Vorverarbeitung”, “Vorhersage” und “App |& Datenbankhandling”

aufgeteilt. Die Komponente “Datenbeschaffung |& Vorverarbeitung” reicht vom Download der Binär - Daten beim DWD bis zu den daraus generierten PNG’s. In der Komponente “Vorhersage” werden die Regenvorhersage mithilfe von neuronalen Netzen gemacht. In der Komponente “App |& Datenbankhandling” geht es um das Datenmanagement mit der Firebase, und um die App welche die Daten darstellt. (Hier sollten wir gegen Ende des Projektes eventuell noch ein bisschen genauer beschreiben wie das alles genau funktioniert. Das wird unseren Nachfolgern sehr stark helfen um sich schnell einzuarbeiten)

2 Die Daten

Die Datengrundlage für die Netze werden von dem DWD in Form des Radar Online Aneichungs verfahren (RADOLAN) zur Verfügung gestellt. Das RADOLAN verfahren kombiniert die Messungen der 18 Radarstation und den punktuellen Messungen von über 2000 Bodenniederschlagsstationen (https://www.dwd.de/DE/leistungen/radolan/radolan.html) Eine dieser Bodenniederschlagsstationen befindet sich in Konstanz.

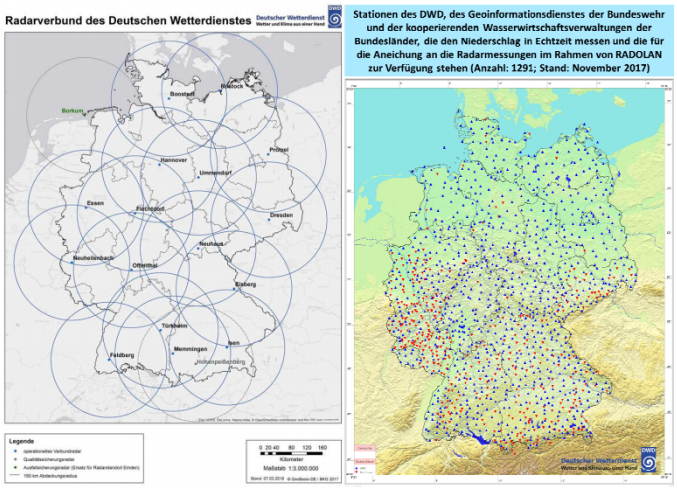


Abbildung 2: Übersicht über alle Boden und Radarstationen

Um die Qualität der aus den RADAR Daten gewonnenen PNGs zu überprüfen, wurden die PNGs mit den Niederschlagsdaten von der Bodenstation in Konstanz verglichen. Die Daten der Station stehen in ein und zehn minütiger Auflösung zur Verfügung. Da sich herausstellte, dass bei der ein minütigen Auflösung Daten Fehlen, wurden die Daten der 10 minütigen Auflösung verwendet um die PNGs zu validieren. Die RADOLAN Daten für die Netze werden über den Opendata Server vom DWD zur Verfügung gestellt. Die Binärdaten werden je nach Jahr in Form eines 1100x900 oder 900x900 Pixel Gitter über Deutschland gelegt. Dabei entspricht jeder Pixel 1km x 1km. Jeder Pixel in dem Pixel Koordinatensystem hat dabei zugehörige Höhen und Breitengrad Koordinaten. In folgender Abbildung ist der Aufbau der Binärdaten zu sehen.

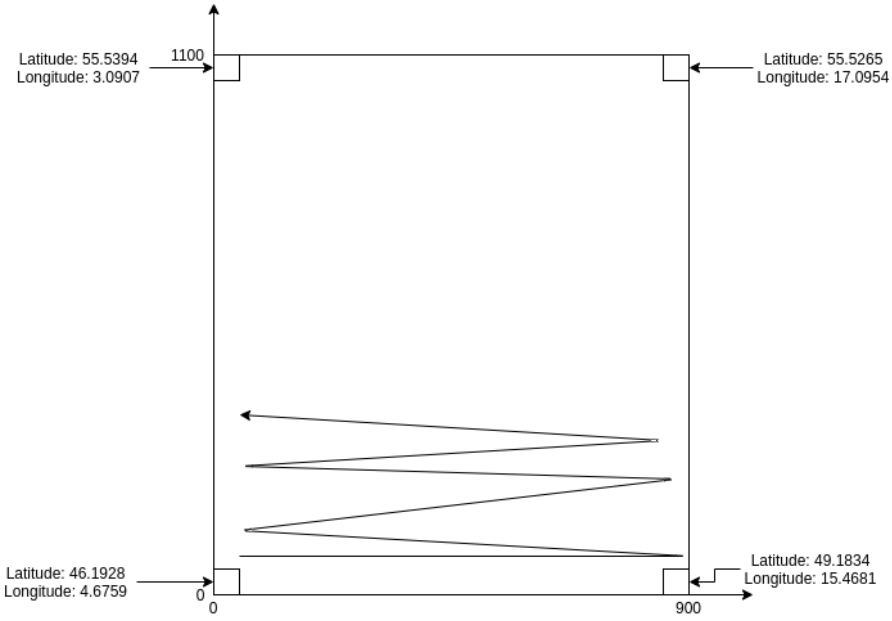


Abbildung 3: Der Aufbau des Koordinatensystems der Radar Daten

Wie zu sehen ist, befinden sich der Pixel (0|0) in der Ecke unten links. Bei der Datenvorverarbeitung wird das Array mit den Pixeln jedoch von oben Links beginnend gefüllt, was bei dem entstehenden PNG zu einer Spiegelung von 90 grad um die vertikale Achse führt (Siehe Kapitel Datenaufbereitung). Des weiteren ist zu beachten, dass die Breitengrade in den Ecken nicht übereinstimmen. So hat die Ecke unten Links einen größeren Breitengrad als die Ecke oben Links. Das gleiche ist auch bei den Höhengraden zu beobachten. Die Ursache hierfür ist die Transformation der Daten von einer 3D Kugel auf eine 2D Karte. In Abbildung 4 ist das daraus resultierende Ergebnis abgebildet.

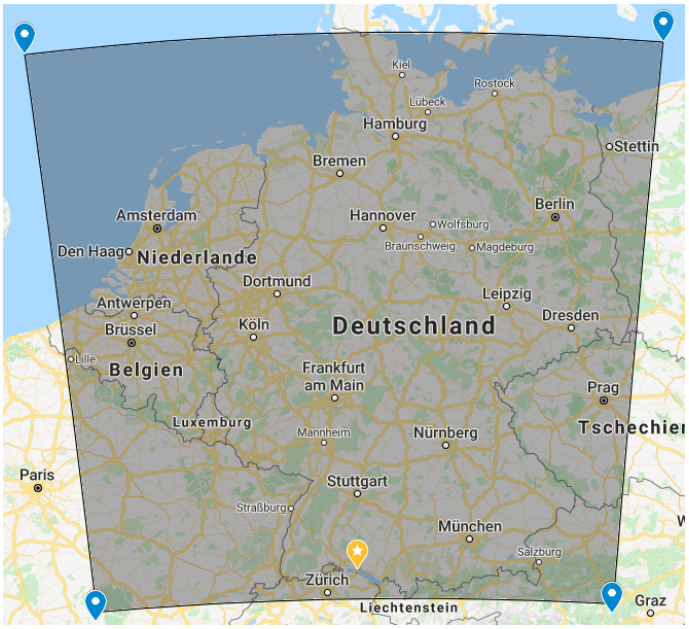


Abbildung 4: Die von den Daten abgedeckte Fläche auf einer 2D Karte

3 Die Datenaufbereitung

Die vom Deutschen Wetterdienst (im Folgenden DWD abgekürzt) in fünf minütiger Auflösung bereitgestellten Radardaten, müssen für das Training der Netze und deren Vorhersage in ein Bildformat umgewandelt werden. Bei den bereits umgewandelten Bildern viel während der Entwicklung der Baseline auf, dass die Bilder weit weniger Regentage abbilden als es tatsächlich regnet. Daraufhin wurde das bisher vorgenommene Preprocessing evaluiert. Um die Radardaten in ein Bild umzuwandeln, muss jeder Radardatenpunkt in ein Pixelfarbwert transformiert werden. Bisher wurde dafür ein Sakalierungsfaktor berechnet mit dem jeder Radardatenpunkt multipliziert wurde. Der Faktor ergab sich aus dem zur Verfügung stehenden Wertebereich (0 bis 255), welcher durch den Maximalwert der Radardaten geteilt wurde. So erhält man transformierte Radarwerte in einem Bereich von 0 bis 255. Anschließend folgte eine Inspizierung der Daten. Hierfür wurde exemplarisch die Radardaten von Juni 2016 herangezogen.

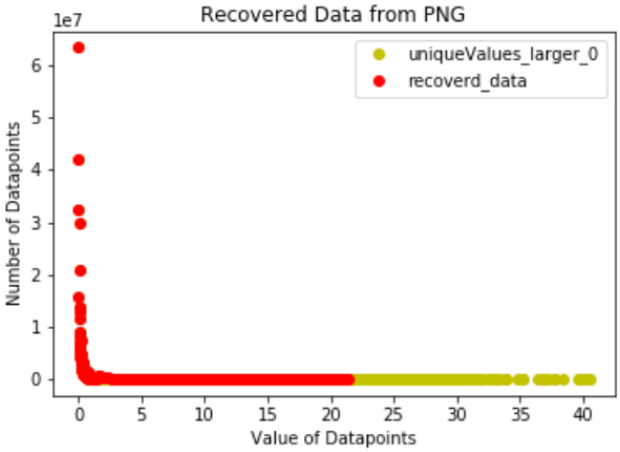


Abbildung 5: Hier muss eine Beschreibung hin.

Geplottet werden alle auftretenden Werte sowie dessen Häufigkeit. Hierbei stechen zwei Ausreißer hervor, welche sehr viel häufiger vorkommen als die restlichen Werte. Der Wert –9999 steht dabei dafür, dass keine Daten verfügbar sind und der zweite Ausreißer ist bei 0, was für “kein Regen” steht. In dem folgenden Plot werden die beiden Ausreißer gefiltert, da die relevanten Informationen in den restlichen Datenpunkten stecken.

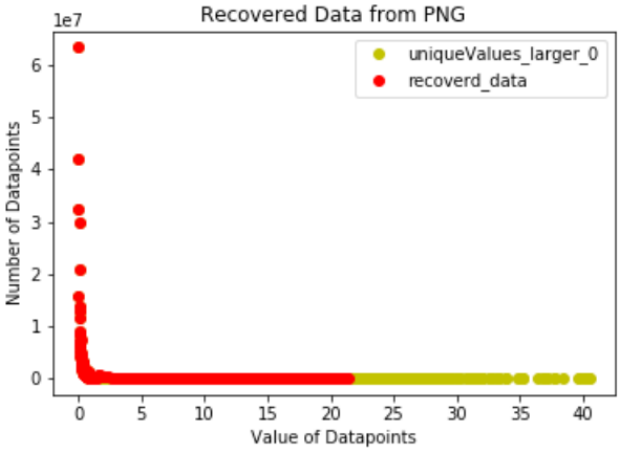


Abbildung 6: Hier muss eine Beschreibung hin.

In diesem Plot dargestellt sind die Radardaten bei denen es regnet. Es wird deutlich, dass ein Großteil der Datenpunkte klein ist. Der Mittelwert liegt bei 0,1744 und zeigt das Problem der bestehenden preprocessing Methode: Radardatenpunkte deren Wert auch nach der Multiplikation mit dem berechneten Skalierungsfaktor kleiner eins sind werden zu null. Aufgrund der vorliegenden Datenverteilung führt das zu einem erheblichen Fehler weshalb eine andere Methode entwickelt werden muss. Die beiden folgenden Plots zeigen verschiedene Perzentile und machen so die Datenverteilung deutlich.

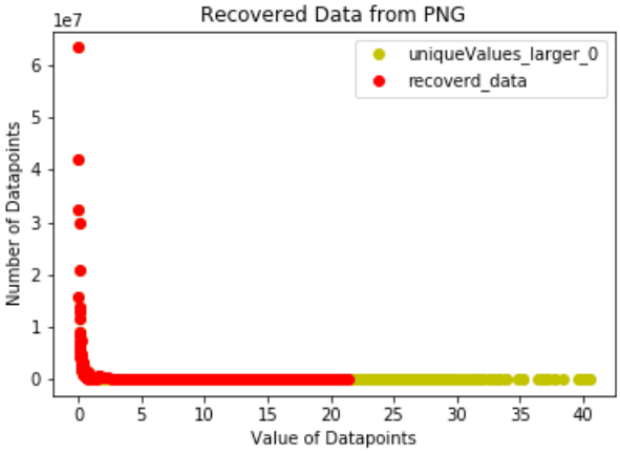


Abbildung 7: Hier muss eine Beschreibung hin.

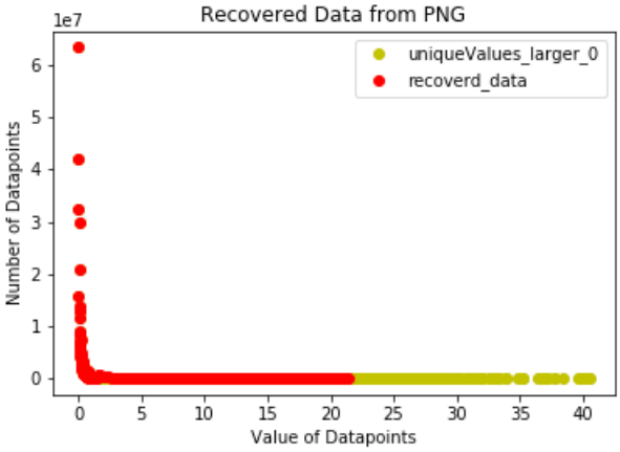


Abbildung 8: Hier muss eine Beschreibung hin.

Das 99 Prozent-Perzentil beinhaltet 138 verschiedene Werte. Wenn jedem dieser Werte ein Farbwert zugeordnet wird, werden 99 Prozent der daten bereits abgebildet und es verbleit ein Wertebereich von 117 mit welchem das letzte Prozent der Daten dargestellt werden kann. Der folgende Plot zeigt die Verteilung der noch verbleibenden Daten.

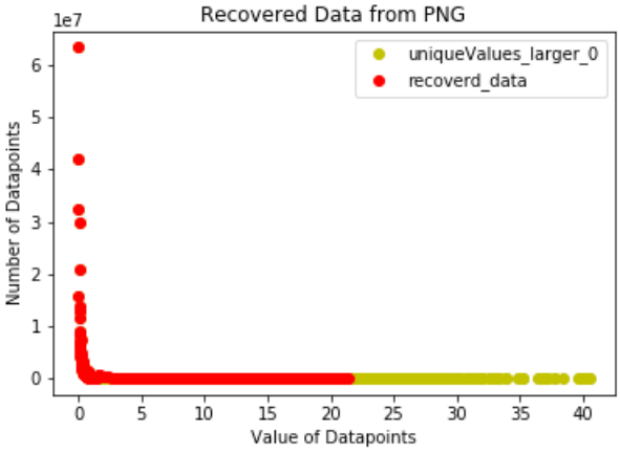


Abbildung 9: Hier muss eine Beschreibung hin.

Noch immer befindet sich der größte Teil der Datenpunkte im kleinem Wertebereich. Da für die verbleibenden Daten eine lineare Transformation ähnlich dem bereits bestehenden preprocessing Vorgang eingesetzt wird, entstehen Rundungsfehler. Da für die Berechnung des Skalierungsfaktors auch nicht der tatsächliche Maximalwert genutzt wird werden Werte die nach der Transformation über 255 sind auf 255 gesetzt. Diese Fehler sind verkraftbar, da es zum einen wichtiger ist alle Datenpunkte abzubilden und zum anderen die Auflösung der verschiedenen Regenstärken immer noch höher ist, als die der menschlichen Wahrnehmung. Radardaten welche transformiert werden müssen:

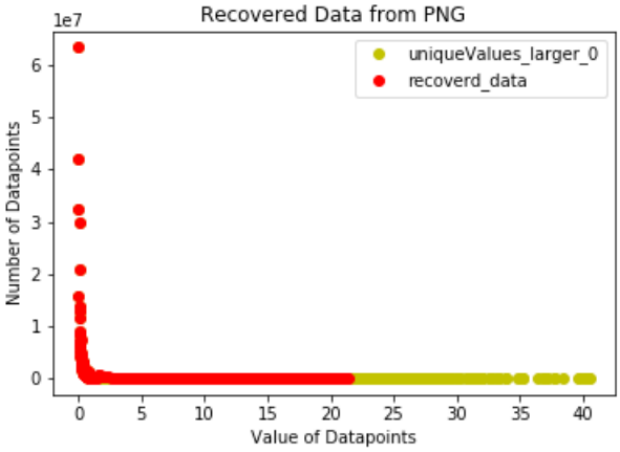


Abbildung 10: Hier muss eine Beschreibung hin.

Die transformierten Daten befinden sich in einem Wertebereich von 0 bis 255.

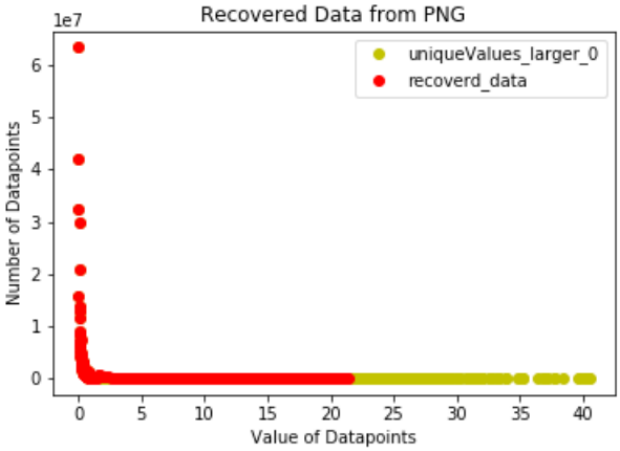


Abbildung 11: Hier muss eine Beschreibung hin.

Rekonstruiert man aus dem PNG Daten wieder die Radardaten ergibt sich der folgende Plot. Die quantitativ wiederhergestellte Anzahl der Datenpunkte beträgt 100

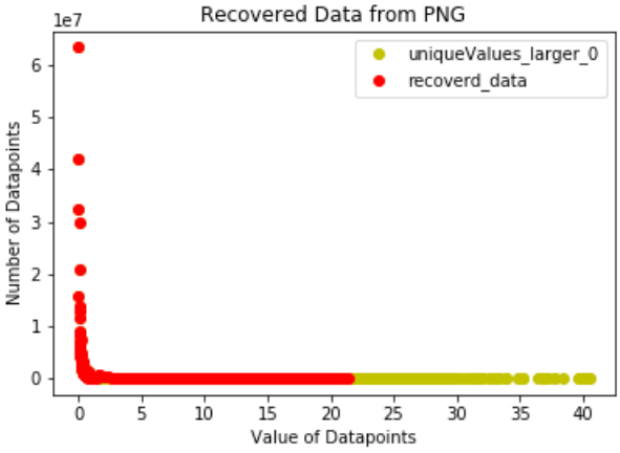


Abbildung 12: Hier muss eine Beschreibung hin.

4 Die neuronalen Netze

Hier kommt alles über das Kernstück unserer Arbeit hin

5 Die DeepRainApp und das Datenbankhandling

Im folgenden werden die Komponenten des Projektes betrachtet, welche benötigt werden um die Vorhersage Daten in einer App zu Visualisieren. Dazu gehört die App selbst, die Datenbank und die Serverkomponenten, welche für das Veröffentlichen der Vorhersagen verantwortlich ist.

5.1 Übersicht

Die Komponente “App und Datenbankhandling” besteht zum wesentlich aus diese drei Unterkomponenten.

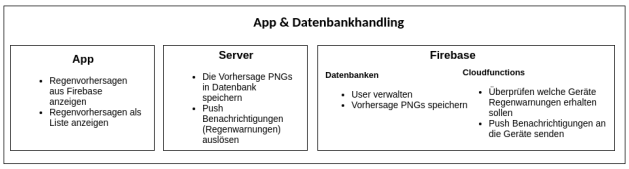


Abbildung 13: Die Komponente App und Datenbankhandling

Die App dient zur Visualisierung der Daten und macht somit die Regenvorhersagen für den Endnutzer brauchbar. Auf dem Server werden die Vorhersagen berechnet und, in dem für dieses Kapitel relevanten Teil, für die App zur Verfügung gestellt. Die Firebase erfüllt im wesentlichen zwei verschiedene Aufgaben, die Datenbankaufgaben und das senden der Push Benachrichtigungen, welche für die Regenwarnungen gebraucht wird.

5.2 Firebase

Firebase ist eine Entwicklungsplattform für mobile Apps. Diese stellt verschiedene Services zur Verfügung welche es ermöglichen effizient Apps für IOS und Android zu entwickeln. Die Firebase ist ein zentraler Baustein der Komponente und wird in jeder Unterkomponente verwendet, weshalb hier einführend ein Überblick gegeben werden soll. Firebase stellt eine Datenbank und einen Cloudspeicher zur Verfügung welcher genutzt wird um die User zu verwalten und die Vorhersage PNGs auf den Geräten anzuzeigen und mit dem Server zu synchronisieren. Des Weiteren werden die In-App Messaging dienste von Firebase verwendet um die Regenwarnungen in Form von Push Benachrichtigungen zu senden. Die genaue Funktionsweise wird in dem Kapitel “Push Nachrichten” beschrieben. Firebase ist bis zu einem gewissen Punkt der Verwendung komplett kostenlos. Wird dieser Punkt überschritten, sind die kosten

von der tatsächlichen Nutzung abhängig. In der kostenlosen Version enthalten sind 1 GB Cloudspeicher, von welchem aktuell nur ein Bruchteil für die 20 PNGs verwendet wird. Des Weiteren können am Tag bis zu 20.000 Dokumente geschrieben und 125.000 Dokumente gelesen werden. Durch die komplett überarbeitete Datenbank und Softwarearchitektur wurden die Datenbanknutzung so weit verringert, dass diese Limitierungen bei weitem nicht erreicht werden sollten.

5.3 Datenbank und Cloudspeicher

Die verwendete Datenbank ist ein Firestore von Firebase. Firestore ist ein Cloud NOSQL Datenbanksystem. Die Daten werden in sogenannte Kollektionen und Dokumente eingeteilt. Dabei gehören zu jeder Kollektion Dokumente, in welchen die eigentlichen Daten gespeichert sind. In Abbildung 14 ist der Datenbankaufbau zu sehen.

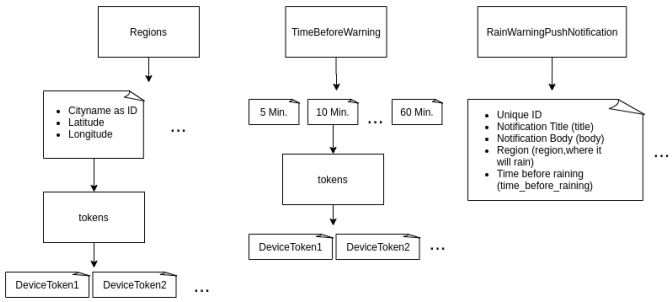


Abbildung 14: Der Aufbau der Kollektionen und Dokumente in der Firebase

Jedes Gerät besitzt einen einmaligen Device Token welcher in zwei Kollektionen gespeichert wird. Die Eine Kollektion steht für den Zeitpunkt in dem die Regenwarnung gesendet werden soll, die andere Kollektion steht für die Region in der die App verwendet wird. Diese beiden Kollektionen werden für das Senden der Push Benachrichtigungen benötigt, auf welches in dem Kapitel 5.6.1 eingegangen wird. In den Einstellungen der App kann eingestellt werden wann die Regenwarnung als Push-Benachrichtigung gesendet werden soll. Je nachdem was der User einstellt, wird sein Devicetoken in eine andere Kollektion gespeichert. Dabei gibt es für jede einstellbare Zeit ein eigenes Dokument in der Kollektion TimeBeforeWarning. Wenn die Netze Regen vorhersagen wird vom Server ein Dokument in RainWarningPushNotification gepusht. Dieses Dokument wird von einer Cloud-Function (Siehe Kapitel 5.6.1) verwendet um die Push Benachrichtigungen an die richtigen Geräte zu senden. Der Cloud Storage von Firebase wird verwendet um die PNGs mit der jeweiligen Vorhersage zu speichern. Dabei lädt der Server alle 5 Minuten die neuen Vorhersagen hoch. Diese PNGs werden in der App angezeigt.

5.4 Server

Mit der Serverkomponente ist der Teil des Servers gemeint, der für die Kommunikation mit der Firebase verantwortlich ist. Dazu gehört das bereitstellen der aktuellsten Regendaten im Bildformat sowie das triggern von Push Benachrichtigungen. Da zu beginn des Projektes noch keine Vorhersagen von den Netzen zur Verfügung stand, wurde ein Programm entwickelt, dass den Server simuliert und zufällig generierte Regendaten in der Firebase speichert. Somit konnte die App unabhängig und parallel zu den Netzen entwickelt werden. Wenn die Netze eine neue Regenvorhersage getroffen haben, werden die daraus resultierenden Vorhersage Bilder in den Firestore hochgeladen. Um die Pushbenachrichtigungen auszulösen, muss für jede Region in der sich ein Nutzer befindet, der Pixel der Region berechnet werden. Dafür werden für alle vorhandenen Regionen, in denen Device Tokens gespeichert sind, es also Nutzer in der Region gibt, über den jeweiligen Breiten und Höhengrad der dazugehörige Pixel für die Region berechnet. Der Wert der jeweiligen Pixel wird ausgelesen. Liegt dieser Farb, oder auch Regenwert, über einem bestimmten Grenzwert, wird ein Dokument in der Kollektion RainWarningPushNotification gespeichert. Hierdurch wird eine Cloudfunktion getriggert, welche sich um das senden der Pushbenachrichtigungen kümmert.

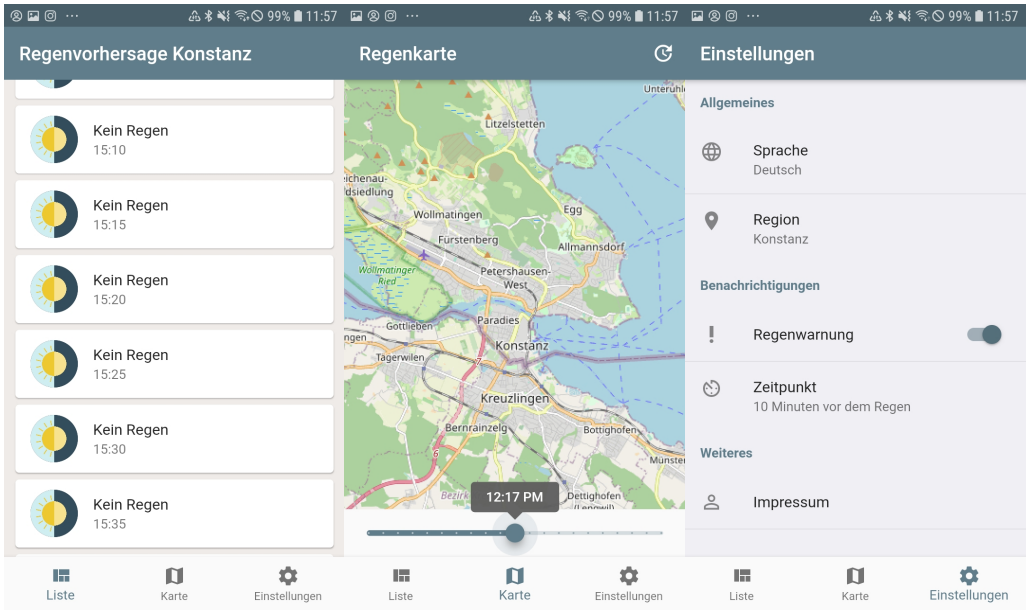
5.5 Die App

5.5.1 Funktionen der App

Die App soll die von den Netzen berechnete Vorhersagen visualisieren und dem Benutzer zur Verfügung stellen. Die Daten werden dabei sowohl in Tabellarischer als auch in Form einer Karte dargestellt. Dabei wird gewährleistet, dass immer die aktuellsten Daten zur Verfügung stehen. Außerdem wird der Benutzer benachrichtigt sobald es eine Regenwarnung gibt. Der Zeitpunkt der Regenwarnung kann eingestellt werden.

5.5.2 Screens der App

Im Wesentlichen besteht die App aus drei Screens. Einem Screen zum Anzeigen der Daten in Listenform, einem zum Anzeigen der Daten auf einer Karte und den Einstellungen. Die jeweiligen Screens können über die Bottomnavigation erreicht werden, somit ist es möglich intuitiv zwischen den einzelnen Screens zu wechseln.



(a) (b) (c)

Abbildung 15: Die drei Hauptscreens der App

Regenvorhersage als Liste

Auf diesem Screen werden die von den Netzen berechneten Regenvorhersagen angezeigt. Dabei wird in die drei Kategorien “Kein Regen”, “Leichter Regen” und “Starker Regen” unterschieden. Je höher die berechnete Regenintensität ist, je dunkler wird der Regenschirm welcher zu Beginn jedes einzelnen Listeneintrages zu sehen ist.

Regenvorhersage als Karte

Auf diesem Screen werden die von den Netzen erzeugten PNGs visualisiert. Dafür werden die PNGs mit einer Karte hinterlegt auf welcher der User frei navigieren kann, um die aktuelle Regensituation an jedem beliebigen Ort zu prüfen. Dabei wird standardmäßig der Kartenausschnitt von der Region angezeigt, die in den Einstellungen eingestellt wurde. Mit dem Slider kann der Zeitpunkt eingestellt werden, in dem die Regenvorhersage angezeigt werden soll. Mit dem Aktualisieren Button in der Actionbar können die neusten Bilder vom Server heruntergeladen werden. Im Normalfall werden die Bilder einmalig bei dem App Start heruntergeladen.

Einstellungen

Auf diesem Screen können alle relevanten Einstellungen gemacht werden. Dazu gehört z.B. die Sprache der Benutzeroberfläche. Außerdem kann die Region eingestellt werden. Die hier ausgewählte Region wird standardmäßig auf der Karte angezeigt und nur für diese Region werden Regenwarnungen gesendet. Unter der Benachrichtigung’s Kategorie können die Regenwarnungen Aktiviert werden sowie der Zeitpunkt der

Regenwarnung eingestellt werden. Jede Aktion in dieser Kategorie löst eine Datenbankaktion aus. Wenn die Regenwarnung aktiviert wird, wird der Device Token von dem Gerät in die Datenbank hochgeladen, beim Deaktivieren wird der Device Token gelöscht. Wenn der Zeitpunkt der Regenwarnung verändert wird, wird der Device Token in der Datenbank von einer Kollektion in eine andere Kollektion verschoben. Alle gemachten Einstellungen werden in sogenannten Shared Preferences gespeichert, damit sie auch nach App Start immer noch vorhanden sind. In Abbildung 16 ist der Datenfluss beim anpassen des Zeitpunktes für die Regenwarnung nachvollzogen werden. In der folgenden Abbildung kann der Datenfluss nachdem der Zeitpunkt der Regenwarnung verändert wurde nachvollzogen werden.

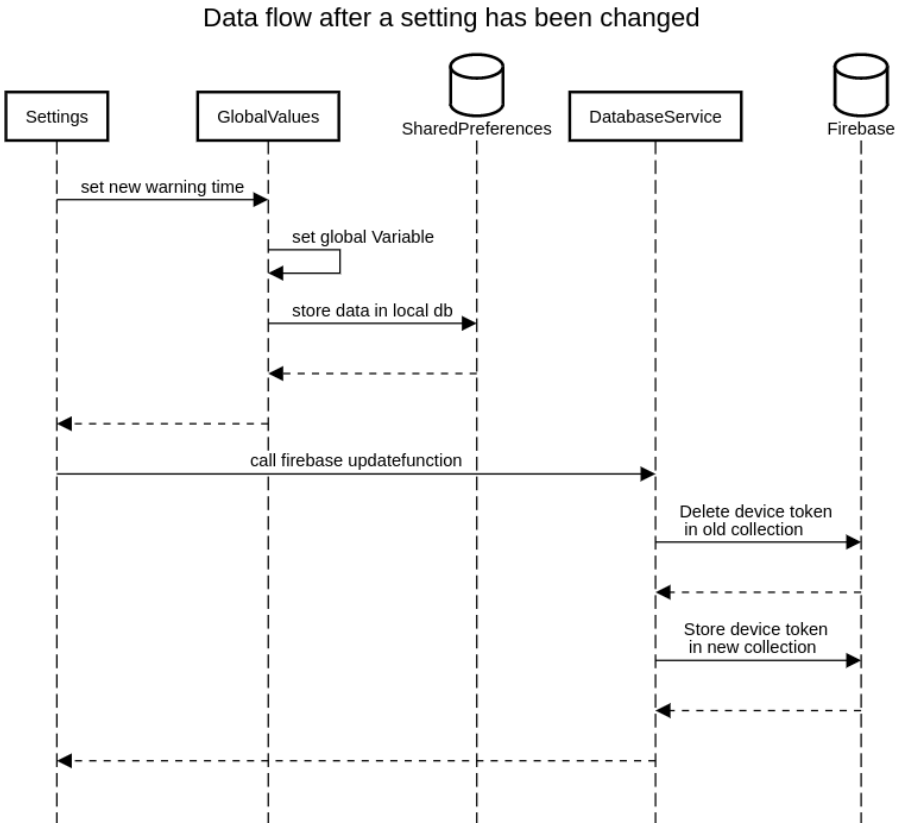


Abbildung 16: Der Datenfluss beim ändern des Zeitpunktes der Regenwarnung

5.5.3 Der Appstart

Während dem Appstart wird die App für die Verwendung vorbereitet, Einstellungen werden wieder hergestellt und die aktuellen Regenvorhersagen werden aus der Datenbank heruntergeladen. Die Einstellungen können dabei aus den Shared Preferences ausgelesen werden. Die Shared Preferences sind eine lokale Key-Value Datenbank, in der alle benutzerspezifischen Daten gespeichert werden. Bei dem App Start werden die Einstellungen aus den Shared Preferences gelesen und auf globale Variablen gespeichert, somit sind sie während der Laufzeit der App dynamisch verfügbar. Außerdem wird bei dem ersten Appstart die Position in dem Vorhersage Bild berechnet. Diese wird

benötigt um aus den Vorhersage Bildern den richtigen Pixel auszulesen und in der Vorhersageliste anzuzeigen. Der Appstart ist in folgender Abbildung schematisch dargestellt.

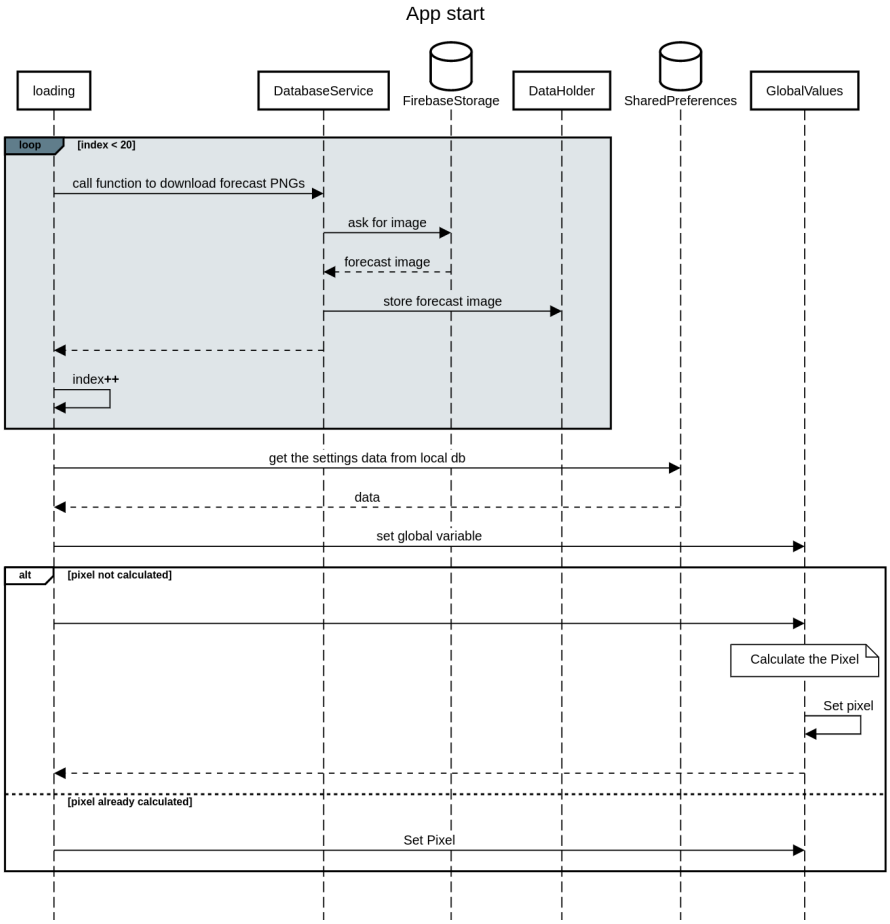


Abbildung 17: Datenfluss beim starten der App inklusive der Datenbankabfragen (Server und Lokale Datenbank)

5.5.4 Die Berechnung der Regenintensität

Zu beginn wurde angenommen, dass die App nur in Konstanz verwendet werden soll. Im Laufe des Projektes stellte sich jedoch heraus, dass die entwickeleten Netze in der Lage sind, Vorhersagen für ganz Deutschland zu machen. Da die Software Architektur nicht für eine solche Anwendung ausgelegt war, mussten einige Änderungen vorgenommen werden. Bis zu diesem Zeitpunkt wurden die Vorhersage Daten für jeden Pixel auf dem Server berechnet und im Anschluss in der Firebase gespeichert. Bei verschiedenen Nutzern in verschiedenen Regionen kommt diese Architektur allerdings schnell an seine Grenzen. Hat die App bspw. 1000 Nutzer in verschiedenen Regionen, müssen für jeden der 1000 Nutzer, alle fünf Minuten, 20 Vorhersage Daten hochgeladen werden. Daher musste der Datenfluss so umstrukturiert werden, dass der neue Regenwert direkt in der App berechnet wird. Dabei muss das Handy den entsprechenden, eigenen, Pixel auf der Karte berechnen. Die Berechnung hierfür

ist verhältnismäßig aufwändig, da mit großen Listen (810.000) Einträgen gearbeitet werden muss. Auf diese Berechnung wird in Abschnitt 5.5.5 eingegangen.

Wenn die Bilder beim Appstart oder bei einem Vorhersageupdate heruntergeladen werden, wird von jedem Bild der Regenwert in dem entsprechnden Pixel berechnet. Es wird eine Liste mit ForecastListItem Objekten erstellt. Diese wird global gespeichert und in der Vorhersage Liste angezeigt. Aktuell können nur beim App Start und durch manuelles auslösen eines Updates die Vorhersage Daten aktualisiert werden.

5.5.5 Berechnung des Pixels

Um die Regensituation an dem jeweiligen Ort des Users auszuwerten, muss der Pixel in dem Vorhersagebild berechnet werden. Hierfür dienen drei verschiedene Listen. Diese Listen wurden in Python mit der Wradlib erstellt, und anschließend im JSON Format in die App übertragen. Zwei der Listen enthalten alle Latitude bzw. Longitude Werte. Die Koordinaten in den einzelnen Indizes Kombiniert geben die Position der einzelnen Pixel im Weltkoordinatensystem an. In der dritten Liste steht zu jedem Index die jeweilig zugehörige Pixel Koordinate im Bild. Somit steht jeder Index für eine Position im Weltkoordinatensystem, ausgedrückt durch Höhen und Breitengrad Informationen, und der Abbildung dieser Position auf den Vorhersagebild. Da das Bild eine Auflösung von 900x900 Pixeln hat, sind diese Listen 810.000 Elemente groß.

listLatitude
46.95351109003129
46.95444001727318
46.955367192755986

listLongitude
3.6010757050026125
3.6132220366153205
3.625368944704319

listCoordinates
[0, 1]
[0, 2]
[0, 3]

Abbildung 18: Der Exemplarische Aufbau der Listen zur Berechnung des eigenen Pixels

Jeder User hat eine eigene Position in Deutschland, welche in Form von Höhen und Breitengrad angaben bekannt ist. Diese wird durch die in den Einstellungen festgelegte Region bestimmt. Es wird nun also ein Algorithmus gesucht, der mithilfe der Höhen und Breitengrad Informationen den Pixel im Bild findet, der am besten zu diesen Koordinaten passt. Nun wäre es natürlich möglich, durch alle Indizes zu Iterieren und somit den richtigen Pixel zu finden. Dieses Verfahren ist aber sehr Zeit und Rechenintensiv und daher nicht geeignet um es auf einem Smartphone auszuführen.

Wir brauchen noch eine Lösung!!

5.5.6 Framework Entscheidung

Bei der Entwicklung einer App steht die Frage der zu bedienenden Plattformen an erster Stelle. Soll die App zum Beispiel nur unternehmensintern verwendet werden oder ist das Gerät auf dem sie verwendet wird eine Neuanschaffung kann es ausreichend sein nativ auf einer Plattform zu entwickeln. Soll jedoch, wie bei den meisten Apps, eine breite Zielgruppe angesprochen werden, ist es unerlässlich die App auf IOS und Android zur Verfügung zu stellen. Je nach dem auf welchen Betriebssystemen die App verwendet werden soll, muss eine komplett andere Framework wahl getroffen werden. So würde man, wenn man entweder nur für IOS oder Android entwickeln möchte, zu einer der nativen Lösungen greifen. Je nach Anforderungen kann auch, wenn beide Betriebssysteme bedient werden sollen, zu der nativen Lösung gegriffen werden. In dem Fall muss der komplette Code natürlich doppelt geschrieben werden. Daher wird normalerweise auf Frameworks zurück gegriffen welche beide Betriebssysteme bedienen. Einige bekannte Frameworks sind zum Beispiel Xamarin, React Native oder Flutter. Jedes dieser Frameworks ist zukunftsträchtig und wurde von großen Unternehmen auf den Markt gebracht. So steht Microsoft hinter Xamarin, Facebook hinter React Native und Google hinter Flutter. Je nach Framework Wahl kann von ca. 80-100 Prozent von dem kompletten Code für beide Betriebssysteme verwendet werden. Dafür sind Cross-Plattform Frameworks oft nicht so performant wie die nativen. Dies fällt besonders bei rechenaufwändigen Apps und Spielen ins Gewicht. Bei so einer leichten App wie DeepRain fällt dieser Nachteil nicht ins Gewicht. Außerdem hätten wir auch nicht genug Kapazitäten um zwei native und voneinander unabhängige Apps zu entwickeln. Ein großer Vorteil von Flutter ist, dass 100 Prozent der Codebasis für Android und IOS übernommen werden können. Außerdem hat die Prominenz von Flutter in den letzten Jahren seit der Veröffentlichung stark zugenommen. In der folgenden Abbildung sind die Google Suchanfragen für den Begriff Flutter abgebildet. Das in Kombination mit eigenem Interesse an dem Framework, ist der Grund dafür, dass die DeepRain App mit Flutter entwickelt wurde.

Abbildung 19: Entwicklung der Suchanfragen für den Begriff 'Flutter'

5.5.7 Technischer Aufbau von Flutter

Flutter Anwendungen werden in der Programmiersprache Dart geschrieben und können anschließend für IOS, Android, Windows, Linux, MacOs und als WebApp

veröffentlicht werden. Dart bringt dabei im Vergleich zu Java Script den Vorteil mit, dass es objektorientiert ist, was vor allem in größeren Softwarearchitekturen zum tragen kommt. Auch alle Bibliotheken um Code asynchron auszuführen werden bereits von Dart mitgeliefert. Die wohl größte Rolle für jeden Dart Entwickler spielen die sogenannten Widgets. Widgets sind einzelene Bausteine welche die UI repräsentieren. Jedes UI element ist dabei ein eigenes Widget. Dabei werden widgets oft ineinander geschachtelt, was es ermöglicht, komplexere UI's zu entwerfen. Dabei werden Widgets in Stateless und Statefull Widgets unterschieden. Während ein Stateless Widget keine Daten und somit keinen Zustand speichern kann, ist das mit einem Stateful Widget möglich.

Die Grundlage von Flutter sind Widgets.
Wie funktioniert flutter?
Was ist der Unterschied zu anderen hybriden Frameworks?
Nur eine Codebasis
Keine weiteren Frameworks nötig
Alles dabei, UI, Widgets, Animationen

5.5.8 Projektstruktur

Der gesamte Code der App ist in die fünf Ordner DataObjects, global, screens, services und Widgets aufgeteilt. Die erste verwendete Datei ist main.dart. In dieser wird festgelegt welcher Screen als erstes nach dem Start aufgerufen wird und die Bottom Navigation wird konfiguriert. Für jeden Screen der App gibt es eine .dart datei in dem Ordner Screens. Welche Files gibt es?
Welche Klassen gibt es?
Wie arbeitet was zusammen?

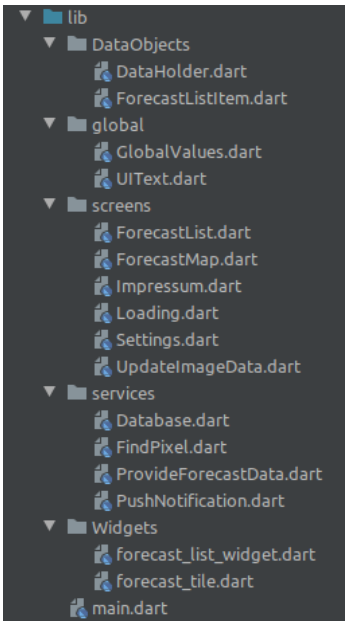


Abbildung 20: Die Projektstruktur der App

5.6 Cloudfunktionen

Mit den Cloud Funktionen von Firebase kann Backend-Code direkt auf den Servern von Google gespeichert und ausgeführt werden. Dabei ist es möglich, auf bestimmte Datenbank Aktionen zu reagieren. Diese Funktionalität wird verwendet, um Push Benachrichtigungen zu senden. Der Code wird in Java Script geschrieben und anschließend als Cloud Funktion hochgeladen.

5.6.1 Push Benachrichtigungen

Zum Warnen der Benutzer, wenn es eine Regenvorhersage gibt, werden Push Nachrichten verwendet. Der Zeitpunkt der Push Benachrichtigungen kann in der App eingestellt werden. So kann man sich zwischen 5 und 60 Minuten vor dem bevorstehenden Regen warnen lassen. Um eine Push Benachrichtigung zu versenden speichert der Server ein Dokument in der Firebase welches alle für die Push Benachrichtigung relevanten Informationen enthält. Dazu gehört zum Beispiel der Push Benachrichtigungs Titel und Text, sowie die Zeit bis der Regen eintritt. Sobald das neue Dokument mit den Informationen für die Push Benachrichtigung hochgeladen wurde, wird eine Callback Funktion in form einer Cloudfunktion aufgerufen. Je nach dem zu welchem Zeitpunkt ein User die Regenwarnung erhalten möchte, wird sein Device Token in eine andere Kollektion gespeichert. Außerdem wird je nach Region in der sich der User befindet, sein Token in einer anderen Kollektion gespeichert. Nur die Tokens, die sowohl mit dem Zeitpunkt, als auch mit der Region, aus dem vom Server hochgeladenen Dokument übereinstimmen, sollen eine Push Benachrichtigung erhalten. Die Funktion findet diese Schnittmenge und weiß somit, welche Geräte eine Pushbenachrichtigung

erhalten sollen.

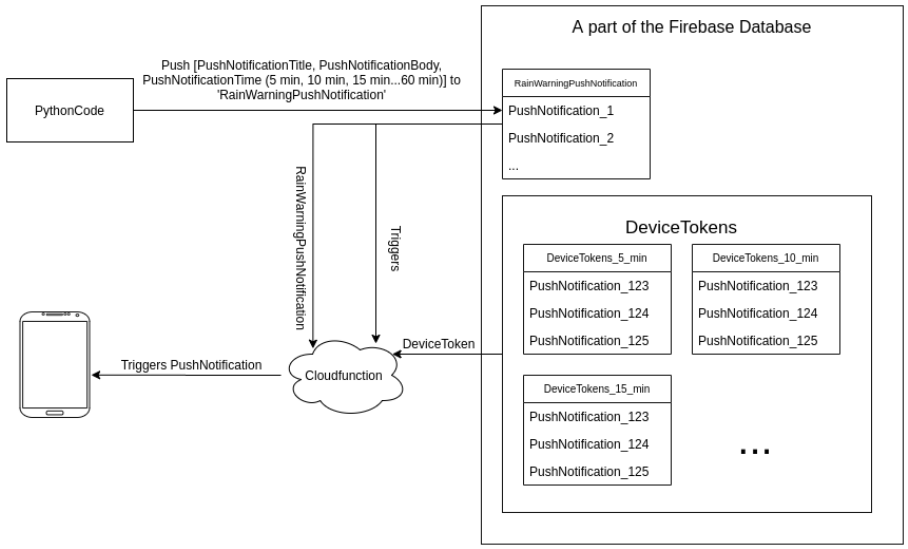


Abbildung 21: Funktionsweise des Prozesses zum senden von Push - Benachrichtigungen.

5.7 Vorgehen bei Entwicklung

In Flutter entwickelte Apps können sowohl auf Android als auch auf IOS ausgeführt werden. Um eine Flutter App auf einem IPhone auszuführen, wird allerdings MacOS als Betriebssystem benötigt. Da während des Entwicklungsprozesses nur ein Linux Rechner zur Verfügung stand, wurde die App lange Zeit nur unter Android getestet. Erst gegen ende des Projektes wurde der Code für IOS kompiliert und für das IPhone angepasst. Da die gesamte Pipeline zu beginn noch nicht funktioniert hat, wurde der für die App relevante Teil der Pipeline simuliert. Dazu wurde ein Python Programm geschrieben, welches reale Daten in die Firebase pushed. Somit war es möglich, gekapselt vom rest des Projektes zu arbeiten, und die App fertig zu stellen.