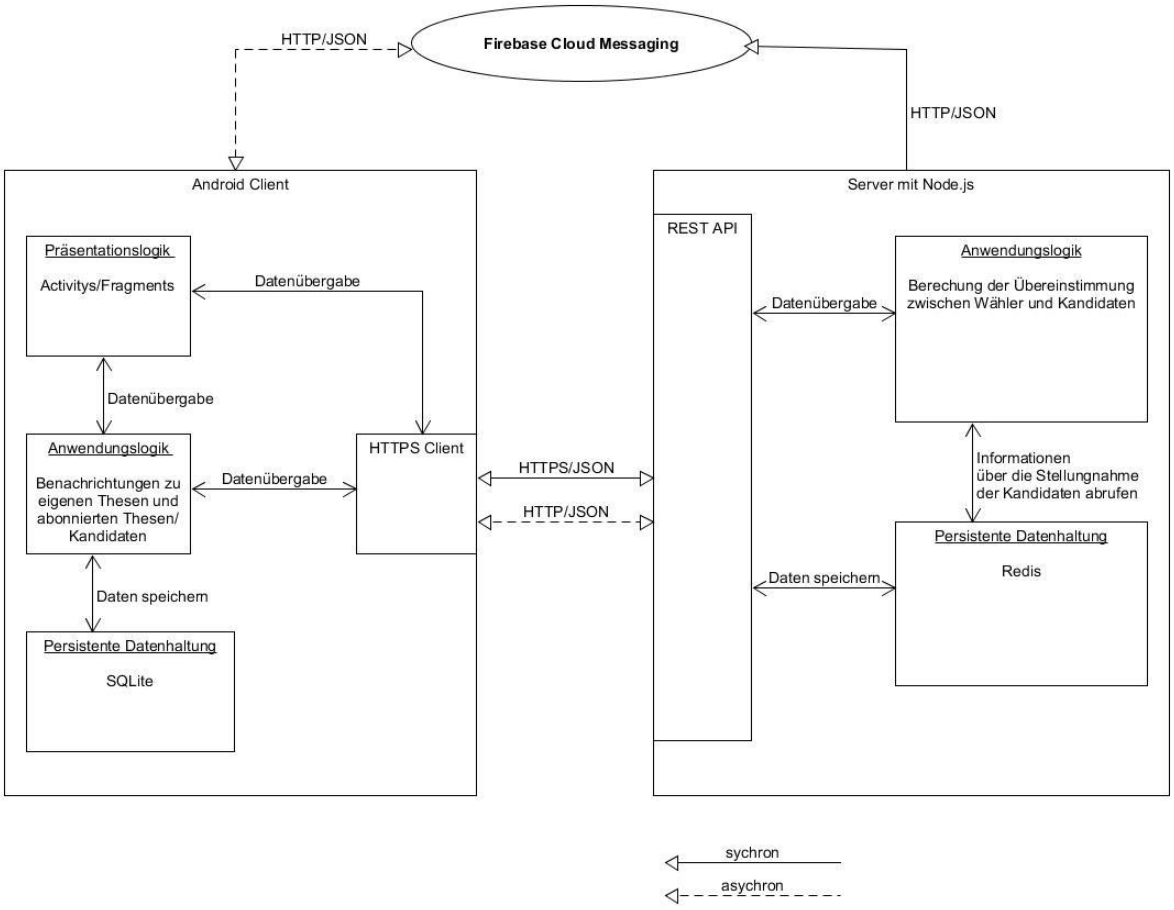
Architektur und Architekturdiagramm



Der Server soll als eine REST API mit Node.js und Javascript entwickelt werden. Die synchrone und asynchrone Client-Server Kommunikation wird über das HTTPS Protokoll mit dem Datenformat JSON stattfinden. Der Client des Rapid Prototyping wird als Android App entwickelt.

Die Vorteile von einer Entwicklung mit Node.js sind die einfache Implementation von den benötigten Ressourcen (siehe Tabelle ?? im Anhang) mit den generischen HTTP-Verben, sowie die Skalierbarkeit des Servers. Da eine Ressource mehrere Repräsentationen haben könnte, jedoch immer nach einer festgelegten Semantik arbeiten sollte, ergibt sich die Entwicklung des Servers als REST API.

Aufgrund einer zeitlichen Beschränkung des Projektes fokussiert sich die Entwicklung des Projektes nur auf den mobilen Nutzungskontext. Ein Vorteil einer mobilen Android Anwendung ist die Möglichkeit jederzeit unterwegs Thesen zu stellen und Stellung zu beziehen. Bei einer einfachen Webanwendung hingegen könnten die privaten Daten der Wähler nicht ausreichend geschützt werden.

Asynchrone Kommunikation

Die Asynchrone Kommunikation zwischen Server und Client soll über *Googles Firebase Cloud Messaging* (Google Inc., 2016, folgend auch *FCM*) erfolgen. Alternative Möglichkeiten wären *PARSE* oder *Amazon SNS*, jedoch lässt sich *FCM* mit geringsten Aufwand in der Android App implementieren. Um dem strategischen Ziel „Die privaten Daten des Wählers müssen geschützt werden“ zu folgen, sollen die Benutzer vor der Verwendung von *FCM* selbst entscheiden, ob Sie dieser Verwendung zustimmen. Wenn der Benutzer dem zugestimmt hat, kann der Server den Android Client des Benutzers über neue Informationen benachrichtigen, sodass automatisch die aktuellsten Informationen von Thesen oder Kandidaten im Client dargestellt werden. Andernfalls muss der Benutzer manuell neue Informationen über Thesen oder Kandidaten vom Server abfragen.
Der Payload der Benachrichtigung vom Server soll nur die Information enthalten, zu welcher These oder zu welchem Kandidat neue Informationen vorliegen, sodass keine sensiblen Daten übertragen werden.

Anwendungslogik

Die Anwendungslogik des Servers ist die Berechnung der Übereinstimmung zwischen Wähler und Kandidat. Für jede Abweichung zwischen dem Kandidaten und der Position des Wählers erhält der jeweilige Kandidat einen Punkt. Der Kandidat mit den wenigsten Punkten ist der Position des Wählers in Bezug auf die abgefragten Thesen am nächsten, der mit den meisten am fernsten. Für die Differenz zwischen "pro" und "contra" erhält der Kandidat zwei Punkte. Kommt die Bewertung "neutral" auf Seiten des Wählers oder des Kandidaten vor, wird nur ein Punkt vergeben - es sei denn beide Seiten stimmen "neutral", dann erhält der Kandidat keinen Punkt.

Tabelle 5: Punktesystem für die Berechnung

	Wähler: „pro“	Wähler: „neutral“	Wähler: „contra“
Kandidat: „pro“	0	1	2
Kandidat: „neutral“	1	0	1
Kandidat: „contra“	2	1	0

Die öffentliche Position der Kandidaten zu den einzelnen Thesen muss dabei persistent auf dem Server gespeichert werden, während die persönliche Ansicht des Wählers nur kurzfristig für diese Berechnung zum Server übermittelt wird. Um den Datenschutz der privaten Daten des Wählers zu gewährleisten, werden die Daten verschlüsselt auf dem Client persistent abgespeichert und über eine sichere HTTPS Verbindung zum Server übertragen. Dabei müssen die persönliche Position des Wählers zu einer einzelnen These, die Bewertung einer These, die Bewertung einer Begründung einer Position zu einer These und die Ergebnisse der Berechnung der Übereinstimmung beim Client gespeichert werden.

Die Anwendungslogik des Clients sind die Benachrichtigungen zu eigenen Thesen oder abonnierten Thesen/Kandidaten. Sobald sich Kandidaten zu den eigenen oder abonnierten Thesen positioniert haben oder Begründungen ihrer Position hinzugefügt haben, bekommt der Wähler eine Benachrichtigung in der Ansicht seines Profils.

Der Algorithmus der Berechnung der Übereinstimmung

Wenn der Wähler eine Berechnung durchführen lässt, wird ein JSON Array erzeugt, welches für jede These ein JSON Objekt enthält. Dieses Objekt enthält die Thesen ID, die Position des Wählers zur These, ein Feld zur Bestimmung ob diese These schon ausgewertet wurde und gegeben falls die Position des Wählers bei der letzten Auswertung. Beim Server wird dieses JSON Array nun auf vier Arrays (Thesen_IDS[], USERPOS[], GESENDET[], LASTUSERPOS[]) aufgeteilt. Anschließend werden die Thesen aus der Datenbanken geladen und es wird ein Array für die Kandidaten ID’s und mehrere

Array für die Zähler der Punktzahlen erzeugt. Bei jeder These wird mittels einer for-Schleife über das „K_Positionen“ Array iteriert und jede neue Kandidaten ID wird dem KandidatenID Array hinzugefügt. Nun wird jede einzelne Kandidaten Position mit der Position des Wählers zu der jeweiligen These verglichen und die Punktzahl des Kandidaten an die Stelle im Punktzahlen Array geschrieben, an der sich die jeweilige Kandidaten ID im KandidatenID Array befindet. Nach dem alle Positionen des Wählers zu den Thesen verarbeitet wurden, wird ein JSON Array mit dem Ergebnissen erzeugt. Für jeden Kandidaten wird ein JSON Objekt mit seiner KID und den Punktzahlen auf dieses Ergebnis Array gepusht und anschließend wird dieses Ergebnis an den Client zurückgeschickt.

Datenstrukturen

Im Folgenden werden die Datenstrukturen und die persistente Speicherung der Daten bei dem Server und bei den Android Clients beschrieben.

Server

Für die persistente Datenhaltung wurde *Redis* beim Server gewählt, da schon auf herkömmlicher Hardware mehrere Zehntausend Schreibvorgänge pro Sekunde möglich sind und dies die Skalierbarkeit des Servers garantiert. Die verschiedenen Repräsentationen von Wähler, Kandidaten und Thesen werden mit einer einmaligen ID (z.B. „WID_2“, „KID_3“, „TID_4“) als „Key“ in *Redis* gespeichert.

```
var Waehler = {
    username: "MaxMustermann",
    password: "12345",
    wahlkreis: "Oberbergischer Kreis",
    email: "max@mustermann.de",
    WID: "WID_23"
};
```

Zu jedem registrierten Wähler wird sein Username, sein Passwort, sein Wahlkreis und seine Email-Adresse, sowie seine eindeutige WID als String gespeichert.

```
var Kandidat = {
    username: "Kandidat_der_Herzen",
    vorname: "Peter",
    nachname: "Müller",
    password: "12345",
    wahlkreis: "Oberbergischer Kreis",
    email: "peter.müller@email.de",
    webseite: "peter.müller-kandidat.de"
    KID: "KID_34",
    Partei: "Unabhängig",
    Thesen_positioniert: [
        {
            TID: "TID_2",
            POS: "PRO",
            KATEGORIE: "Umwelt"
        },
        {
            TID: "TID_543",
            POS: "CONTRA"
            KATEGORIE: "Lokal"
        }
    ],
    Begruendungen: [
        {
            TID: "TID_3",
            POS: "PRO",
            TEXT: "Diese These ist toll."
        }
    ],
    Biographie:
        {
         Geburtsdatum: "23.1.1975",
         Bildungsweg: "Abitur, Studium",
         Berufe: "Lehrer",
         Mitgliedschaften: "",
        }
    ,
    Wahlprogramm: {
        Text: "Deswegen sollte ich gewählt werden.",
        Link: "link.zum.wahlprogramm.de"
    }
};
```

Zu jedem registrierten Kandidaten wird ebenfalls sein Username, sein Passwort, sein Wahlkreis und seine Email-Adresse und dazu noch seinen Vor- und Nachnamen und seine Parteizugehörigkeit als String gespeichert. Desweiterten wird ein JSON Array „Thesen_positioniert“ für jeden Kandidaten angelegt, in dem seine Position in einem JSON Objekt zu den jeweiligen Thesen festgehalten wird. Dieses JSON Objekt umfasst die TID der These, die konkrete Position des Kandidaten und die Kategorie der These. Außerdem werden die Begründungen des Kandidaten zu Thesen in einem Array namens Begruendungen[] gespeichert. Hinzu kommen noch ein JSON Objekt mit der Biographie des Kandidaten und ein JSON Objekt mit dem Wahlprogramm des Kandidaten.

```
var These = {
    thesentext: "Es sollte für Bildung mehr Geld ausgegeben werden als für Krieg!",
    kategorie: "Wirtschaft",
    wahlkreis: "Oberbergischer Kreis",
    Anzahl_Zustimmung: 23454,
    Anzahl_Ablehnung: 34,
    Anzahl_Neutral: 347,
    Likes: 3785,
    TID: "TID_2",
    K_PRO: [
        {
            UID: "KID_2",
            Text: "Geld ist genug da!",
            likes: 5,
            Kommentare: [
                {
                    UID: "WID_2",
                    USERNAME: "Wähler123"
                    Kommentar: "Endlich sagt es jemand"
                }
            ]
        },
        {
            UID: "KID_5",
            Text: "Bildung ist wichtig!",
            likes: 6,
            Kommentare: []
        }
    ],
    K_NEUTRAL: [],
    K_CONTRA: [],
    W_PRO: [],
    W_NEUTRAL: [],
    W_CONTRA: [],
    K_POSITION: [
        {
            UID: "KID_2",
            POS: "PRO"
        },
        {
            UID: "KID_5",
            POS: "PRO"
        },
    ]
};
```

Zu jeder veröffentlichen These wird der Text der These, die Kategorie der These, der Wahlkreis des Absenders und die Thesen ID (TID) gespeichert. Wenn die Wähler eine Berechnung der Übereinstimmung von Server durchführen lassen, wird ihre Position zu den Zählern Anzahl_Zustimmung, Anzahl_Ablehnung oder Anzahl_Neutral hinzugezählt. Die Begründungen der Positionen von Kandidaten und Wähler zur Thesen werden in verschiedenen JSON Arrays (von K_PRO bis W_CONTRA) gespeichert. Dabei repräsentiert ein JSON Objekt die Begründung und umfasst die UID des Absenders, den Text der Begründung, die Anzahl an „likes“ der Begründung, und ein JSON Array mit Kommentaren zu der Begründung. Da die Begründungen optional von Kandidaten zur These hinzugefügt werden können, werden die Positionen der Kandidaten zu der These in einem zusätzlichen JSON Array namens „K_POSITION“ gespeichert.

Android Client

Um die Android App auch ohne ständige Internetverbindung nutzen zu können, müssen die Daten auch beim Client persistent gespeichert werden. Dafür wurde beim Client das Open-Source *„SQLite“* Datenbanksystem verwendet, welches dem ACID-Prinzip folgt und sich mit relativ geringem Aufwand in die App implementieren lässt.

```
public static final String SQL_CREATE_THESENTABLE =
        "CREATE TABLE " + ThesenTable.TABLE_NAME + " (" +
                ThesenTable.COLUMN_NAME_TID + " STRING PRIMARY KEY," +
                ThesenTable.COLUMN_NAME_THESENTEXT + " TEXT," +
                ThesenTable.COLUMN_NAME_KATEGORIE + " TEXT," +
                ThesenTable.COLUMN_NAME_WAHLKREIS + " TEXT," +
                ThesenTable.COLUMN_NAME_LIKES + " TEXT," +
                ThesenTable.COLUMN_NAME_ANZAHL_PRO + " TEXT," +
                ThesenTable.COLUMN_NAME_ANZAHL_NEUTRAL + " TEXT," +
                ThesenTable.COLUMN_NAME_ANZAHL_CONTRA + " TEXT," +
                ThesenTable.COLUMN_NAME_K_PRO + " TEXT," +
                ThesenTable.COLUMN_NAME_K_NEUTRAL + " TEXT," +
                ThesenTable.COLUMN_NAME_K_CONTRA + " TEXT," +
                ThesenTable.COLUMN_NAME_W_PRO + " TEXT," +
                ThesenTable.COLUMN_NAME_W_NEUTRAL + " TEXT," +
                ThesenTable.COLUMN_NAME_W_CONTRA + " TEXT," +
                ThesenTable.COLUMN_NAME_K_POSITION + " TEXT" +
                " )";
```

Der Aufbau der Thesen Tabelle beim Client ist deckungsgleich zur Repräsentation beim Server.

```
public static final String SQL_CREATE_KANDIDATENTABLE =
        "CREATE TABLE " + KandidatenTable.TABLE_NAME + " (" +
                KandidatenTable.COLUMN_NAME_KID + " STRING PRIMARY KEY," +
                KandidatenTable.COLUMN_NAME_VORNAME + " TEXT," +
                KandidatenTable.COLUMN_NAME_NACHNAME+ " TEXT," +
                KandidatenTable.COLUMN_NAME_PARTEI+ " TEXT," +
                KandidatenTable.COLUMN_NAME_EMAIL + " TEXT," +
                KandidatenTable.COLUMN_NAME_WAHLKREIS + " TEXT," +
                KandidatenTable.COLUMN_NAME_BEANTWORTETETHESEN + " TEXT," +
                KandidatenTable.COLUMN_NAME_PUNKTE_INSGESAMT + " INTEGER," +
                KandidatenTable.COLUMN_NAME_PUNKTE_LOKAL + " INTEGER," +
                KandidatenTable.COLUMN_NAME_PUNKTE_UMWELT + " INTEGER," +
                KandidatenTable.COLUMN_NAME_PUNKTE_AP + " INTEGER," +
                KandidatenTable.COLUMN_NAME_PUNKTE_SATIRE + " INTEGER" +
                " )";
```

Die Kandidaten Tabelle beinhaltet wie bei dem Server verschiedene Informationen zum Kandidaten und wurde um die Spalten „Punkte_Insgesamt“, „Punkte_Lokal“, „Punkte_Umwelt“, „Punkte_AP“ und „Punkte_Satire“ erweitert, um nach dem Matching die Punktzahlen der Auswertung zu speichern und die Kandidaten nach diesen Punktzahlen sortiert darstellen zu können.

```
public static final String SQL_CREATE_USERPOSITIONDATATABLE =
        "CREATE TABLE " + UserpositiondataTable.TABLE_NAME + " (" +
                UserpositiondataTable.COLUMN_NAME_TID + " STRING PRIMARY KEY," +
                UserpositiondataTable.COLUMN_NAME_POSITION + " TEXT," +
                UserpositiondataTable.COLUMN_NAME_SERVER + " TEXT," +
                UserpositiondataTable.COLUMN_NAME_LAST_POSITION + " TEXT" +

        " )";
```

In der Userposition Tabelle wird die jeweilige Position des Benutzers zu jeweiligen These gespeichert. In der Spalte „Server“ wird erfasst, ob die Position des Benutzers schon von dem Server ausgewertet wurde und in der Spalte „Last_Position“ wird die letzte Position des User nach der Auswertung gespeichert, umgegeben falls die Zähler (Zustimmung, Ablehnung, Neutral) der jeweiligen Thesen bei der nächsten Auswertung zu korrigieren.