Technology
Arts Sciences
TH Köln

Entwicklungsprojekt interaktive Systeme

WS 18/19

Installationsdokumentation, Fazit und Prozessassessment

PlaceToBe

Dozenten

Prof. Dr. Gerhard Hartmann
Prof. Dr. Kristian Fischer

Betreuer

Robert Gabriel

Projekt

von

Elena Correll
Mike Klement

Inhaltsverzeichnis

Installationsdokumentation **3**

1. Use Case 3

2. Übersicht der relevantesten Klassen 4

2.1 Client 4

2.2 Server 5

3. Client 5

3.1 Benötigte Anforderungen zur Nutzung des Clients 5

3.2 Präsentationslogik 6

3.3 Anwendungslogik 6

3.3.1 Main1Activity, Main2Activity, Main3Activity, 6

Main4Activity, Main5Activity und Main6Activity 6

3.1.2 Main6Activity 7

Einbindung der API’s 9

3.1.3 MainMapActivity 9

3.1.4 DorfA, DorfB, DorfC 14

4. Server 15

4.1 Datenhaltung 15

4.2 Zuweisungsalgorithmus 16

5. Abweichungen zu MS2 18

Implementationsideen, die nicht vollendet wurden 18

II. Fazit **19**

III. Prozessassessment **21**

I. Installationsdokumentation

In dieser Installationsdokumentation werden wir den UseCase “Wohnort-Finder anwenden” nochmals erläutern und anschließend die einzelnen Installationsschritte auf Client und Serverseite aufzeigen.

1. Use Case

Im Use Case geht es darum, dass der User Fragen beantwortet auf Grund derer der Server ihm geeignete Wohnorte ausgibt.

Name	Wohnort-Finder anwenden	
Auslösender Aktor	Wohnort-Suchender	
Weitere Aktoren		
Auslöser	Benutzer ist auf der Suche nach einem neuen Wohnort	
Vorbedingung	Der Benutzer sucht einen Wohnort in einem Gebiet, dass im System ist	
Nachbedingung	Das System hat dem Benutzer Vorschläge zu Wohnorten ausgegeben	
Haupt-Szenario	*Benutzer*	*System*
	Benutzer gibt an, dass er zum Wohnort Finder möchte	
	Benutzer füllt erforderliche Fragen aus	
		System sucht nach passenden Orten

		System gibt Übersicht aller passenden Orte aus
	Benutzer klickt einen Ort an	
		System gibt Übersicht des Ortes aus

2. Übersicht der relevantesten Klassen

2.1 Client

Java Klasse	XML File	Beschreibung
Main1Activity.java	activity_main.xml	Frage 1 des Wohnort-Finders
Main2Activity.java	activity_main.xml	Frage 2 des Wohnort-Finders
Main3Activity.java	activity_main.xml	Frage 3 des Wohnort-Finders
Main4Activity.java	activity_main.xml	Frage 4 des Wohnort-Finders
Main5Activity.java	activity_main.xml	Frage 5 des Wohnort-Finders
Main6Activity.java	activity_main.xml	Frage 6 des Wohnort-Finders, Erstellung eines User Objekts und Umwandlung in JSON, POST Request an Server, Umwandlung und Zuweisung der Response
MainMapActivity.java	activity_map.xml	Anzeigen der Karte im gesuchten Gebiet und der drei ermittelten Orte, Zuweisung der Buttonnamen
DorfA.java	dorf_a.xml	Übersicht des ersten Ortes

DorfB.java	dorf_b.xml	Übersicht des zweiten Ortes
DorfC.java	dorf_c.xml	Übersicht des dritten Ortes

2.2 Server

Datei	Beschreibung
index.js	Kreieren eines Servers, Behandlung der Requests, Zuweisungsalgorithums der geeigneten Wohnorte, inkrementieren des Feldes Anzahl (zur Statistikerhebung)
dorf.json	Beinhaltet relevante Daten über die verfügbaren Dörfer (Alternative zur Datenbank)

3. Client

3.1 Benötigte Anforderungen zur Nutzung des Clients

Die App wurde mit dem Emulator Nexus 5X API 24 und der Version 7.0 getestet, daher empfehlen wir auch damit zu testen.

Diese folgende Berechtigungen müssen für die Verwendung der App erlaubt werden:

- Anlegen oder Einloggen in einem Google Account
- GPS : für das tracken von Standorten und des eigenen Standort
- Internet: für den Verbindungsaufbau mit GoogleMaps und dem Server

3.2 Präsentationslogik

Für den ausgewählten Use Case haben wir zehn Seiten erstellt. Die ersten sechs zeigen die Fragen an, die gestellt werden, um einen geeigneten Wohnort ermitteln zu können.

Die Elemente hierfür sind:

- Eine Überschrift, die anzeigt, dass man sich beim Wohnort-Finder befindet
- Die Frage an sich
- Ein interaktives Feld zur Beantwortung der Frage in Form eines Textfeldes oder eines verschiebbaren Reglers
- Ein Fortschrittsbalken, der anzeigt, bei welcher Frage man sich befindet
- Ein Button mit der Aufschrift “Weiter”, der zur nächsten Frage führt

Da wir eine SeekBar haben, der vom Nutzer wahrscheinlich mehrmals verschoben wird, ist der automatische Übergang nach der Benutzereingabe zur nächsten Frage nicht sinnvoll. Daher wurde die Entscheidung getroffen einen Button hinzuzufügen. Diese Designentscheidung weicht von der Modellierung ab. Aus Gründen der Einheitlichkeit wird der Button auf jeder der sechs Frage-Seite angezeigt (activity_main - activity_main6.xml).

Die Seite activity_map zeigt dem Benutzer eine Karte, die auf die gewählte Region ausgerichtet ist und darauf drei Buttons, die zu der Übersichtsseite des jeweiligen vom Server ermittelten Dorfes führen.

Diese Seiten DorfA, DorfB und DorfC zeigen jeweils den Namen des Ortes an, ein Video, das den Ort präsentieren wird und Informationen in drei Text-Views (allgemeine Informationen, Natur, Aktivitäten).

3.3 Anwendungslogik

Bevor wir die Anwendungslogik erläutern möchten wir darauf hinweisen, dass die Implementierung unseres Projektes lediglich einen vertikalen Prototypen des gewünschten Gesamtsystems darstellt. Uns ist bewusst, dass die Fragen, der Zuweisungsalgorithmus und die Informationen zu den Wohnorten auf jeden Fall ausbaubar sind, um für den Benutzer gezieltere Ergebnisse ermitteln zu können.

3.3.1 Main1Activity, Main2Activity, Main3Activity,

Main4Activity, Main5Activity und Main6Activity

Da die Anwendungslogik dieser sechs Klassen fast identisch ist, werden wir diese zusammenfassend beschrieben.

Als erstes wird die jeweilige View gesetzt, die die Frage anzeigt. Der Button wird definiert und ein OnClickListener erstellt. Sobald dieser angesprochen wird, wird der Wert der Benutzereingabe in einer statischen Variable gespeichert und die nächste Seite aufgerufen.

seekbar()

Bei den Seiten, die die Antwort des Benutzers über einen Regler entgegennehmen wurde die Funktion seekbar() implementiert. Diese liest den integer Wert ab, sobald der Regler verschoben wird und speichert den Wert in der statischen Variable.

Die statischen Variablen der sechs Frageseiten heißen wie folgt:
ortName, radius, miete, natur, akt und aktivitaet; für jede dieser Variablen wurde eine getter-Methode implementiert

3.1.2 Main6Activity

Neben den oben genannten Funktionen besitzt Main6Activity noch weitere

checkNetwork()

Hier wird mit `ConnectivityManager` überprüft, ob der Client mit dem Netzwerk verbunden ist.

fillUser()

Es wird festgelegt, was für ein Typ der Benutzer ist. Wenn der Wert natur größer als aktivitaet ist, so ist der Typ Natur, ist der Wert Miete kleiner als 600 so ist der Typ Miete und trifft keine der beiden Aussagen zu so ist der Typ Aktivitaet.
Nun wird ein neues Objekt der Klasse User erstellt. Dazu werden mit get-Methode die Variablen geholt, die die Antworten beinhalten.

```
User u1 = new User(Main1Activity.getName(),
Main2Activity.getRadius(), Main3Activity.getMiete(),
Main4Activity.getNatur(), Main5Activity.getAkt(),
Main6Activity.getAktivitaet(), typ);
```

Das erzeugte Objekt wird in JSON umgewandelt und zurückgegeben.

doInBackground()

Diese Methode steht in AsynkTask. Es wird ein OkHttpClient erstellt, denn hier möchten wir ein POST Request mit dem erstellten JSON Objekt von fillUser() an den Server senden.

```
MediaType JSON = MediaType.parse("application/json;charset=utf-8");
RequestBody body = RequestBody.create(JSON, jsons.toString());
```

```
//ein POST Request wird kreiert
Request post = new Request.Builder()
        .url("http://10.0.2.2:3000/userwish")
        .post(body)
        .build();
```

Hierbei ist `jsons` das erstellte Objekt mit den Antworten und dem Typ des Benutzers. Dies wird auf die Ressource **`/userwish`** gepostet.

Der Server berechnet die passenden Wohnorte (siehe Implementierung des Servers). Dem User werden diese automatisch ausgegeben. Intern schickt der Server die passenden Ort in der POST Response. Wie wir diese Daten verwerten sieht man im Folgenden.

```
res = client.newCall(post).execute();
String data = res.body().string();
```

Die Response wird in einen String umgewandelt und in data gespeichert.

```
JSONArray JA = new JSONArray(data);
```

Daraufhin wird data in ein JSONArray verwandelt. Nun wird das JSONArray abgefragt und den Variablen für die drei Dörfer, die ausgegeben werden soll, zugewiesen. Im Anschluss sieht man beispielhaft die Zuweisung von Dorf A.

```
//Dorf A
//Das erste Element des JSON Arrays wird geholt
JSONObject jsonA = (JSONObject) JA.get(0);

//Das JSON Objekt allgemeinA beihnaltet alle allgemeinen
Informationen über den Ort
JSONObject allgemeinA = (JSONObject) jsonA.get("allgemeineInfos");
//ortNameA benötigen wir für die Aufschrift auf dem Button in
MainMapActivity und die Überschrift in DorfA
ortNameA = (String) allgemeinA.get("name");
//allgemeine Informationen unter dem Video
allgA = "Bundesland: " + allgemeinA.get("bundesland") + "\n" +
        "Region: " + allgemeinA.get("region") + "\n" +
        "Einwohnerzahl: " + allgemeinA.get("einwohnerzahl");

//JSON Objekt mit Natur Informationen
JSONObject naturaA = (JSONObject) jsonA.get("natur");
//Array mit allen Naturvorkommen im Ort
JSONArray natur = (JSONArray) naturaA.get("natur");

//StringBuilder um alle Elemente des Arrays natur als String
anzeigen zu können
```

```
StringBuilder sb = new StringBuilder();

for (int n = 0; n < natur.length(); n++){
   sb.append(natur.get(n));
   sb.append(" \n");
   naturaa = String.valueOf(sb);
   System.out.println(naturaa);
}

naturA = "Gruenflaeche pro Einwohner: " +
naturaA.get("gruenflaeche") + "\n" +
       " Natur in der Umgebung: " + naturaa;

JSONArray aktivitaeta = (JSONArray) jsonA.get("aktivitaeten");
StringBuilder sb1 = new StringBuilder();

for (int m = 0; m < aktivitaeta.length(); m++){
   sb1.append(aktivitaeta.get(m));
   sb1.append(" \n");
   aktaa = String.valueOf(sb1);
}

aktA = "Freizeitaktivitaten in " + ortNameA + ": \n" + aktaa;
```

Die Variablen ortNameA, allgA, naturA und aktA sind statische Strings, die später in der Klasse DorfA aufgerufen werden.

Einbindung der API's

3.1.3 MainMapActivity

Um überhaupt die GoogleMapsAPI sowie die Youtube API nutzen zu können muss vorher ein Google Account sowie eine Bankverbindung angeben werden. Danach kann man Keys generieren und ganze 16 API's von Google nutzen. In unseren Programm wurden 3 API's eingebunden:

```
implementation'com.google.android.gms:play-services-maps:16.0.0'
implementation'com.google.android.gms:play-services-location:16.0.0'
implementation files('libs/YouTubeAndroidPlayerApi.jar')
```

Die Verfügbarkeit, Statistiken und die De- und Aktivierung einzelner API's konnte man bei der Google Console einsehen und managen. Jedoch gab es noch einen Schritt um sicherstellen zu können dass die API's aktuell und einsatzbereit sind. Hierbei musste der Google PlayStore immer auf der neusten Version laufen, weil darüber die einzelnen API's

verwaltet werden. Dies wurd in der Methode public boolean isServiceOK() mit mehreren If-Anweisungen für bestimmte Fälle sichergestellt und eingebunden.

```
int availalbe =
GoogleApiAvailability.getInstance().isGooglePlayServicesAvailable(M
ainActivity.this);
     if (availalbe == ConnectionResult.SUCCESS) {
  } else if
(GoogleApiAvailability.getInstance().isUserResolvableError(availalb
e)) {

GoogleApiAvailability.getInstance().getErrorDialog(MainActivity.thi
s,availalbe, ERROR_DIALOG_REQUEST);
   dialog.show();
} else
     return false;
```

Sollte der Service überprüft sein und alles funktionieren wir die GoogleMaps Karte ausgeben. Das wurde mit der Methode protected void onCreate gemacht.

```
super.onCreate(savedInstanceState);
setContentView(R.layout.activity_main);

init();
```

Wenn die Karte erreichbar ist wird dann über public void init() durch einen Button mit einem Listener die Anzeige ausgeführt.

```
if (isServicesOK()) {
   Button btnMap = (Button) findViewById(R.id.btnMap);
   btnMap.setOnClickListener(new View.OnClickListener() {
       @Override
       public void onClick(View v) {
           Intent intent = new Intent(MainActivity.this,
MapActivity.class);
           startActivity(intent);
       }
   });
```

In der nächsten Klasse MapMainActivity wurde die Ausgabe der Informationen, Local permission, das Keyevent für die Karte und das GPS Tracking eingebunden.

Über die Instanz einer Schnittstelle für ein MapFragment- oder MapView-Objekt das festgelegt wurde, wird die onMapReady-Methode (GoogleMap) ausgelöst, wenn die Karte

zur Verwendung bereit ist. Sollte das nicht der Fall sein wird eine nicht-leere Instanz von GoogleMap bereitgestellt. Jedoch bevor das passiert werden die Permission gecheckt.

```
public void onMapReady(GoogleMap googleMap) {
     mMap = googleMap;

if (mLocationPermissionGRANTED) {
       getDeviceLocation();
if (ActivityCompat.checkSelfPermission(this,
Manifest.permission.ACCESS_FINE_LOCATION)
!= PackageManager.PERMISSION_GRANTED &&
ActivityCompat.checkSelfPermission(this,
Manifest.permission.ACCESS_COARSE_LOCATION) !=
PackageManager.PERMISSION_GRANTED) {

 return;
       }
       mMap.setMyLocationEnabled(true);
```

Drei Buttons wurden zur Weiterleitung zu den Übersichtsseiten des jeweiligen Dorfes genutzt, weil die Marker von Google Maps aus nicht funktionierten. Die Schrift des Buttons wird auf den Namen des Dorfes gesetzt.

```
Button buttonA = (Button) findViewById(R.id.buttonA);
buttonA.setText(Main6Activity.getOrtNameA());
```

Über die Methoden Fine/Coarse sowie Device Location werden die Nutzungsrichtlinien der Koordinaten und des Gerätes des Users festgelegt. Diese sind von Google vorgegeben und sollten nicht verändert werden.

```
private void getLocationPermission(){
String[] permission = {Manifest.permission.ACCESS_FINE_LOCATION,
Manifest.permission.ACCESS_COARSE_LOCATION};

private static final String Fine_Location =
Manifest.permission.ACCESS_FINE_LOCATION;

private static final String COARSE_Location =
Manifest.permission.ACCESS_COARSE_LOCATION;
```

Im public boolean onEditorAtion() wird einfach festgelegt, dass die Suche bei einem Enter auf der Rechnertastatur die Suchanfrage startet.

```
                if (actionId == EditorInfo.IME_ACTION_SEARCH
                || actionId == EditorInfo.IME_ACTION_DONE
                || keyEventevent.getAction() == KeyEvent.ACTION_DOWN
```

```
                || keyEventevent.getAction() ==
KeyEvent.KEYCODE_ENTER)
```

geoLocate() wird für die Identifizierung der Standorte genutzt und in einer ArrayList gespeichert. Damit mehrere Standorte gleichzeitig ausgeben werden. Da dies aber nicht mit den Markern von GoogleMaps funktioniert, kann es nur in der Console angezeigt werden.

```
 String searchString = sucheOrt;
private void geoLocate() {
        String searchString  = sucheOrt;
        Geocoder geocoder = new Geocoder(MapActivity.this);
        List<Address> list = new ArrayList<>();       try {
            list = geocoder.getFromLocationName(searchString, 1);
        } catch (IOException e) {
            Log.e(TAG, "geoLocate: IOException: " + e.getMessage());
        }
        if (list.size() > 0) {
            Address address = list.get(0);
```

Der Fused Location Provider ist eine der Standort-APIs in Google Play-Diensten über den man den letzten Standort, in dem der Nutzer sich eingeloggt hat, bekommt. Beim Start der App wird diese Position anzgezeigt.

```
mFusedLocationProviderClient =
LocationServices.getFusedLocationProviderClient(this);

        try {
            if (mLocationPermissionGRANTED) {

final Task location =
mFusedLocationProviderClient.getLastLocation();
                location.addOnCompleteListener(new
OnCompleteListener() {
```

onComplete(): wenn der Listener aufgerufen wird, dann ist ein Task abgeschlossen.

```
public void onComplete(@NonNull Task task) {
                    if (task.isSuccessful()) {
Location currentLocation = (Location) task.getResult();

moveCamera(new LatLng(currentLocation.getLatitude(),
currentLocation.getLongitude()),Default_Zoom "My Location");
```

In moveCamera() werden die einzelnen Koordinaten der übergebenen Suchanfrage genommen und damit die Kamera zur den bestimmten Ort mit einem definierten Zoom angezeigt. Dabei wird auch ein Marker zum bestimmten Suchort angelegt.

```
   private void moveCamera(LatLng latLng, float zoom, String title)
{

Log.d(TAG, "moveCamera: moving the camera to: lat: " +
latLng.latitude + ",lng:" + latLng.longitude);
 mMap.moveCamera(CameraUpdateFactory.newLatLngZoom(latLng, zoom));

        MarkerOptions options = new MarkerOptions()
                .position(latLng)
                .title(title);
        mMap.addMarker(options);
    }
```

In der Methode nehmen wir uns von der GoogleMapAPI die benötigten Fragments um uns die Karte ausgeben zu lassen.

```
        private void initMap(){
            Log.d(TAG, "initMap: Karte anwendungsbereit");
            SupportMapFragment mapFragment = (SupportMapFragment)
getSupportFragmentManager(). findFragmentById(R.id.map);
                    mapFragment.getMapAsync(MapActivity.this);
```

onRequestPermissionResult() wird vor der Ausgabe aufgerufen und überprüft, ob der Nutzer den Tracking-Zugriff zugestimmt hat und erst dann wird die Karte ausgeben. Für die weitere Implementation wurde ein Loop angelegt damit gleich mehrere Zugriffe auf die Karte genehmigt werden können.

```
         switch (requestCode) {
         case Location_Permission_REQUEST_CODE: {
          if (grantResults.length > 0)
           for (int i = 0; i < grantResults.length; i++) {
            if (grantResults[i] !=
PackageManager.PERMISSION_GRANTED) {
             mLocationPermissionGRANTED = false;
                   return;
             }
          }
                  mLocationPermissionGRANTED = true;
```

In hideSoftKeyboard() wurdeder Bug behoben dass die Tastatur auf dem Geräte bleibt um dem Nutzer die Kartenübersicht verdeckt.

```
this.getWindow().setSoftInputMode(WindowManager.LayoutParams.SOFT_INPUT_STATE_ALWAYS_HIDDEN);
    }
```

3.1.4 DorfA, DorfB, DorfC

Die Anzeige der Dörfer besteht aus der Überschrift, dem Video, dem Abspiel-Button und drei TextView Feldern, in denen die Information stehen soll.

Über die get-Methode werden die vom Server empfangenen Werte in Main6Activity geholt und in das jeweilige Textfeld gesetzt. Hier wird der Typ des Benutzers beachtet.
Ist der Typ Natur werden als erstes, also in das oberste Textfeld, die Informationen zu Natur gesetzt. Bei Typ Miete die allgemeinen Informationen und bei Typ Aktivitaet die Aktivitaeten.

Durch die Youtube API können wir über die Methode onCreate und einen Listener und der YouTubeAndroidPlayerApi.jar die API einbinden. Nun müssen wir nur noch den Player einen Youtube Link geben und das Videofeld und den Button in der XML.Datei einbinden und damit kann das Video abgespielt werden.

```
        btnPlay = findViewById(R.id.btnPlay);

                public void onInitializationSuccess(YouTubePlayer.Provider provider,
YouTubePlayer youTubePlayer, boolean b) {
                                                (Link)
                youTubePlayer.loadVideo("ZNrBe7f6JEg");
            }

       btnPlay.setOnClickListener(new View.OnClickListener() {
       @Override
       public void onClick(View view) {
```

Zu Beachten ist das der API Key von der Klasse YoutubeConfig mit eingebunden wird.

```
mYoutubePlayerview.initialize(YoutubeConfig.getApiKey(),
mOnInitializedListener);
            }
        });
```

4. Server

Über NodeJS wird ein Server erstellt.

```
var server = app.listen(3000, listening);
```

4.1 Datenhaltung

Aus Zeitgründen haben wir die Daten zu den einzelnen Dörfern und Städten manuell in einer JSON Datei gespeichert. Die Datenerfassung von PlaceToBe sollte in Zukunft automatisch über die GoogleMapsAPI und Wiki-Data-API erfolgen und in der Datenbank MongoDB gespeichert werden.

Die Struktur der JSON Datei ist hier abgebildet.

```
{ "dorf": [
    {
       "id":,
       "mapDaten":[{
           "naechsteStadt": "",
           "entfernung": }],
       "allgemeineInfos":{
           "name": "",
           "bundesland": "",
           "region": "",
           "einwohnerzahl": ,
           "bevölkerungsdichte": ,
           "mietspiegel": },
       "aktivitaeten": ["", ""],
       "natur":{
           "gruenflaeche": ,
           "natur": ["", ""]}
       },
      "anzahl":
}
]
```

4.2 Zuweisungsalgorithmus

Erhält der Server ein POST auf userwish so berechnet er direkt, welche Orte geeignet sein könnten. Dazu braucht er folgende Variablen (vgl. POST Request auf Client Seite):

```
var name = req.body.ortName;
var miete = req.body.miete;
var radius = req.body.radius;
var aktivitaeten = req.body.aktivitaeten;
var typ = req.body.typ;
```

Daraufhin wird eine erste Auswahl getroffen, welche Dörfer für den Benutzer geeignet sein könnten. In dorf.dorf sind alle Dörfer gespeichert.

```
for (i=0; i<dorf.dorf.length; i++){
   for(j=0; j<dorf.dorf[i].mapDaten.length; j++){
     if (dorf.dorf[i].mapDaten[j].naechsteStadt == name &&
dorf.dorf[i].mapDaten[j].entfernung <= radius &&
       dorf.dorf[i].allgemeineInfos.mietspiegel <= miete) {

         for(l=0; l<dorf.dorf[i].aktivitaeten.length; l++){
           if (dorf.dorf[i].aktivitaeten[l] == aktivitaeten){
             result.push(dorf.dorf[i]);
     (...)
```

Hier wäre zu bemängeln, dass der Server alle Dörfer durchläuft, was bei unserer Anzahl nicht problematisch ist, jedoch bei einer größeren Anzahl von Daten schnell zu schlechter Performance führen kann, sowie einer ineffizienten Nutzung der Ressourcen.

In der Endfassung von PlaceToBe sollen die Felder “naechste Stadt” und “Entferung” in der Datenbank nicht mehr existieren. Die GoogleMapsAPI, die eingebunden werden soll, wird den gewünschten Ort und den Radius vom Client Request entgegennehmen und dem Server mitteilen, welche Orte anhand der geografischen Position überhaupt in Frage kommen. So wird das oben genannte Problem behoben.

In dem Array result [ ] haben wir nun alle Orte gespeichert, die passend wären. Dem Benutzer sollen nur drei Dörfer zurückgegeben werden. Dazu sortieren wir das Array mit einem Selection Sort nach unterschiedlichen Kriterien, die am Typ des Benutzers festgemacht werden.

- Hat die Variable typ den Wert Natur, werden die drei Orte mit der meisten Grünfläche zurückgegeben.
  ```
  if (typ == "Natur" && result.length >1){
  res.send(selectionSortNatur(result)); }
  ```

- Ist der Benutzer vom Typ Aktivitaet, so sortiert der Algorithmus die Dörfer nach Anzahl der Aktivitäten.

```
if (typ == "Aktivitaet" && result.length>1){
res.send(selectionSortAktivitaet(result)); }
```

- Beim Typ Miete werden die drei Orte mit der geringsten Miete zurückgegeben.

```
if (typ == "Miete" && result.length >1){
res.send(selectionSortMiete(result));
```

Hier beispielhaft `selectionSortNatur()`:

```
function selectionSortNatur (result){

   var i, j,  minIx, minVal;
   var ergebnis = [];

   for (i=0; i<result.length; ++i){
     minVal = result[minIx=i].natur.gruenflaeche;
     for (j= i+1; j<result.length; j++){
       result[j].natur.gruenflaeche < minVal && (
            minVal = result[minIx = j].natur.gruenflaeche);
       temp = result[i];
       result[i] = result[minIx];
       result[minIx] = temp;
     }
   }
   ergebnis[0] = result[result.length-1];
   ergebnis[1] = result[result.length-2];
   ergebnis[2] = result[result.length-3];

   return ergebnis;
}
```

Nun wurden die drei für den Benutzer interessantesten Orte ermittelt und der Server sendet die JSON Datei als Response des POST Requests an den Client.

5. Abweichungen zu MS2

Folgende Dienste wurden nicht mit eingebunden die: Wiki-Data-API,OpenWeatherMap sowie die Datenbank von MongoDB und die Stream.oi Cloud.

Die Wiki-API und die Datenbank in MongoDB wurden nicht mit eingebunden, weil die Zeit dafür nicht mehr gereicht hat. Alternativ zur MondoDB wurde für den Use-Case eine JSON-Datei mit allen für den Zuweisungsalgorithmus erforderlichen Daten gespeichert.
Die OpenWeatherMap API würde als nicht benötigt erachtet und sollte nur als Zusatzdienstleistung für eine Fragen dienen.
Die Streamcould.io wurde mit der Youtube API ausgewechselt Grund dafür war das diese besser mit dem System funktioniert und leichter einzubinden ist. Außerdem hat diese den gleichen Installationspfad wie die GoogleMapsAPI.
Änderung in der Systemarchitektur war, dass die GoogleMapAPI im Client integriert wurde und nicht im Server. Für das Endergebnis von PlaceToBe ist die Einbindung der GoogleMapAPI auf Serverseite dringend erforderlich.

Implementationsideen, die nicht vollendet wurden

Es wurde versucht über GPS-Tracking dem User eine Möglichkeit zu bieten ohne Sucheingabe eine direkte Auskunft zu bieten, ob in seiner jetzigen Umgebung möglicherweise passende Orte in Frage kommen um das Intresse der Suche zu steigern.

Es war geplant mit den Markern von GoogleMaps zu arbeiten um eine direkte Weiterverknüpfung zu den einzelnen Seiten der Dörfer zu erlangen. Da dies nicht funktionierte wurde Übergangsweise mit Buttons von Android Studio gearbeitet um die Weiterleitung zu gewährleisten.

Das Suchfeld sollte eigentlich auf eine externe Seite liegen, jedoch gab es Verknüpfungsprobleme, weswegen das Suchfeld jetzt noch auf der GoogleMaps Ansicht liegt.

II. Fazit

Um den Grad der Zielerreichung feststellen zu können haben wir die Ziele aus MS2 genauer betrachtet. Danach können wir entscheiden, ob das Projekt erfolgreich war die Ziele bis zum Showcase erreicht wurden. Bis zur fertigen App sind es zwar noch viele Schritte, jedoch muss man auch bedenken, dass dieses System in solch weniger Zeit nicht umzusetzen ist.

Die taktischen und strategischen Ziele des Projektes wurden wie folgt definiert.

Taktische Ziele die erreicht wurden:

- Das Vorgehen im Projekt muss nach dem Vorgehensmodell nach ISO 9241- 210 erfolgt sein.
- Um den Nutzungskontext zu verstehen muss eine umfangreiche Stakeholderanalyse durchgeführt worden sein.
- Es sollen User Profiles und Personae vorliegen.
- Es sollen Use Cases und Szenarien vorliegen.
- Aus dem Nutzungskontext heraus müssen funktionale, organisatorische sowie qualitative Anforderungen analysiert worden sein.
- Eine Gestaltungslösung muss erarbeitet, evaluiert und iterativ verbessert worden sein.
- Die Systemarchitektur soll genauer spezifiziert und Techniken festgelegt worden sein. (Auswahl der Datenbank und der APIs)
- Die Werteberechnung auf Client und Serverseite muss erfolgreich implementiert sein, sodass die Berechnungen ein korrektes Ergebnis liefern.
- Die Benutzeroberfläche soll unter Beachtung der Grundsätze der Dialoggestaltung nach ISO 9241- 110 implementiert worden sein.

Taktische Ziele die nicht erreicht wurden, mit Begründung:

- Die erforderlichen Städte-spezifischen Daten (Name, Einwohnerzahl, durchschnittlicher Mietpreis) müssen erfasst und in einer Datenbank gespeichert sein, nachdem die Datenstruktur festgelegt wurde.

Aus zeitlichen Gründen wurde die Datenhaltung in einer JSON-Datei realisiert und daher fällt dieser Punkt weg.

- Es soll eine REST Spezifikation vorliegen.

Es gibt keine REST Spezifikation, da in der Modellierungsphase noch nicht alle Request festgelegt wurden und der Zuweisungsalogrithmus noch nicht derart ausgereift um REST-Methoden festlegen zu können.

-Es kann eine Topic Modellierung durchgeführt worden sein.

Die Topic Modellierung wurde rausgelassen, weil wir kein PUB/SUB benötigen.

Strategische Ziele die erreicht wurden:

- Das System muss eine verteilte Anwendungslogik haben.
- Den Benutzern soll die Entscheidung zu einem Wohnort doppelt so schnell gelingen.

Wir können sagen, dass dieses Ziel erreicht wurde, da das System dem benutzer drei passende Wohnorte ausgibt und sogar Informationen zu den Orten zur Verfügung stellt. Der Benutzer spart sich so folgende Schritte der Recherche:

- Recherche der Umgebung: welche Orte befinden sich in der gewünschten Region
- Recherche zu den Orten: wo liegt der Ort, wie viele Einwohner hat der Ort, welche Freizeitaktivitäten gibt es dort, welche Naturattraktionen, ..

Die weitere Implementation des Systems soll zudem noch weitere Informationen beachten, wie zum Beispiel das Wetter, ein Video, dass die Kultur repräsentiert, Einkaufsmöglichkeiten und Infrastruktur Informationen, wie Verkehrsanbindung.

Strategische Ziele die nicht erreicht wurden mit Begründung:

- Die Landflucht soll innerhalb von 10 Jahren um 5 Prozent zurückgehen.

Das Ziel konnte nicht erreicht werden weil, zu wenig Zeit vergangen ist um darüber eine Entscheidung zu treffen.

- Die Oberfläche des Clients soll benutzerfreundliche sein, was anhand der Evaluation und Feedback der Benutzer festgestellt werden kann.

Auch hier können wir keine klare Aussage treffen, da aus zeitlichen Gründen keine Evaluation und Feedback zum vertikalen Prototypen durchgeführt wurden.

III. Prozessassessment

Nach Abschluss des Projektes möchten wir noch einmal kurz die Zeitplanung und Aufgabenbewältigung reflektieren.

Im Rahmen der Konzeptentwicklung wurde ein Projektplan erstellt, zur Erleichterung des Zeitmanagements. Wir möchten nun kurz zusammenfassen in wie fern dieser im Laufe des Projektes PlaceToBe eingehalten wurde und welche Probleme aufgetreten sind.

Im Sinne der zu erledigenden Schritte wurde der Projektplan gut eingehalten. Allerdings hat sich die zeitliche Einteilung teilweise sehr verschoben. Je näher ein Abgabetermin gekommen ist, desto produktiver sind wir geworden wohingegen die Motivation am Anfang einer Phase eher gering war. Allgemein war die Erfolgskurve unseres Projektes am Anfang und am Ende des Projektes am höchsten, zum Ender der Modellierungsphase hatten wir ein Tief.

In der Phase des Konzepts haben wir alle Ziele, die wir uns gesetzt hatten, erreicht. Nur das Ergebnis des Rapid Prototyping war mangelhaft. Dort sind wir mit der Einstellung rangegangen keine guten Programmierer zu sein und hatten auch zu wenig Zeit eingeplant. Diese Einstellung haben wir zum Glück im weiteren Verlauf des Projektes abgelegt.

In der Modellierungsphase hingegen, hätten wir die Zeit die wir in den ersten beiden Wochen vertrödelt haben dringend gebraucht. Die Lücken, die in der der zweiten Phase entstanden sind haben wir in der Implementierungsphase schließen müssen. Dennoch wurden die gesteckten Ziele in der Implementierungsphase erreicht.

Bezüglich der Formulierung der Aufgaben und Meilensteine im Projektplan ist die Genauigkeit mit jeder Phase gesunken. Die Aufgaben der Konzipierung waren daher sehr klar, wohingegen wir uns in dieser Phase noch kein deutliches Bild der Aufgaben in der Implementierungsphase machen konnten. Der Projektplan war dementsprechend ungenau und manche Aufgaben waren nicht einmal mehr erforderlich. In Zukunft würden wir daher empfehlen diesen nach der Modellierungsphase noch einmal zu überarbeiten.

Der Grund warum wir trotz teilweise fehlender Motivation und aufkommenden unerwarteten Problemen das Projekt zufriedenstellend abschließen konnten, war der Einbau und die Nutzung der Puffer im Projektplan.

Auch das Vorgehensmodell der MCI wurde größtenteils eingehalten. Allerdings hat die Zeit für eine Iteration, besonders in der Implementierungsphase gefehlt. Wir empfehlen daher dringend Zeit für Evaluation, Iteration und Verbesserung im Projektplan mit einzubeziehen.

Trotz der Komplikationen und einigen Lücken haben wir den wichtigsten UseCase “Wohnort-Finder anwenden” implementiert. Der momentane vertikale Prototyp gibt jedoch nur eine Vorstellung, wie das System in Zukunft funktionieren soll.

Wir sehen Potenzial in der Idee und würden eine Weiterentwicklung in Betracht ziehen. Drei folgenden Punkte bei einer Weiterentwicklung wären uns wichtig.

- Die Datenhaltung sollte nicht in einer JSON Datei liegen, sondern auf einer Datenbank. Zudem soll die Datenerhebung automatisch über eingebundene APIs erfolgen

- Um individuelle Ergebnisse liefern zu können müssen die Informationen eines Dorfes erweitert werden und die Fragen, die dem User gestellt werden müssen mehr Platz für Facetten bieten.
- Um den Benutzern eine bessere Vorstellung zu einem ort zu geben und somit seine Entscheidung zu erleichtern, soll über das System ein Video erstellt und angezeigt werden, dass die Stimmung und Kultur des Ortes repräsentiert.

Im Großen und Ganzen sind wir mit dem Ergebnis des Projektes zufrieden und hoffen auf eine gute weitere Ausarbeitung und Erfolg von PlaceToBe.