

PROJEKT „TRAWELL“

Projektdokumentation

22.06.2017

Projekt Geoinformatik SS2017

Prof. Dr. Thomas Brinkhoff

Lennard Gabriel, Lisa Haltermann,
Maximilian Herbers, Nils Kirsch, David Post

Inhaltsverzeichnis

1	Aufgabenstellung	2
2	Studie und Anforderungsanalyse	2
3	Systemarchitektur	3
4	Implementierung.....	4
4.1	Benutzeroberfläche und Layout.....	5
4.2	Google Maps und Places API	8
4.3	Weather API	11
4.4	Beziehung von Zugverbindungsdaten über GTFS.....	11
4.5	Routenfindung.....	14
4.6	Datenmodell der Touren	15
4.7	Social Media Export.....	17
5	Ausblick und Zusammenfassung	18
6	Literaturverzeichnis.....	20
7	Abbildungsverzeichnis.....	20
8	ANHANG	22
	Anhang I Lastenheft	22
	Anhang IV Entwurf des Layouts.....	23
	Anhang V JSON Google Places API.....	26
	Anhang VI JSON Weather API.....	27
	Anhang VII API's	28
	Anhang VIII Dijkstra-Algorithmus	Fehler! Textmarke nicht definiert.

1 Aufgabenstellung

Im Rahmen der Forschungsinitiative *mFund* des Bundesministeriums für Verkehr und digitale Infrastruktur (BMVI) werden Forschungs- und Entwicklungsprojekte rund um digitale datenbasierte Anwendungen für die Mobilität 4.0 gefördert. Hierdurch sollen Geschäftsideen, die auf Mobilitäts-, Geo- und Wetterdaten basieren unterstützen und gefördert werden (Bundesministerium für Verkehr und digitale Infrastruktur, kein Datum). Das *mFund* dient als Anregung für das folgende dargelegte Projekt im Modul Projekt Geoinformatik des Sommersemester 2017.

Im Zuge dieses Projektes wird eine Android-Applikation im Kontext von Bahnreisen durch Europa entstehen. Ziel ist es, dass die entwickelte Applikation den Anwendern ermöglicht, eine Reise durch europäische Städte auf Grundlage des intereuropäischen Bahn-Tickets, dem sogenannten Interrailpass, zu planen. Zudem soll die Applikation einen hilfreichen digitalen Reisebegleiter zu den ausgewählten Reisezielen bieten und Informationen zu z.B. Unterkünften und Wetterdaten anzeigen.

2 Studie und Anforderungsanalyse

Die zu entwickelnde Anwendung für die Android Plattform soll besonders von Reiseinteressierten und Besitzern eines Interrailpasses genutzt werden. Unter diesem Gesichtspunkt ist vor allem auf eine intuitive benutzerfreundliche Bedienung zu achten, da es sich hier auch um nicht technisch versierte Anwender handeln kann. Der Benutzer sollte in wenigen einfachen Schritten die von ihm gewünschten Aktionen durchführen können. Damit die Applikation einem breiten Anwenderspektrum zugänglich gemacht werden kann, ist die Mindestanforderung hier Android 4.1 (API Level 16) mit einem Verbreitungsgrad im Juli 2017 von 98,6% (developer.android.com, 2017)

Zu den von der App bereitgestellten Funktionen zählen die Auswahl der zu bereisenden Städte in einer Karte und die Auswahl einer Unterkunft in diesen Städten. Des Weiteren soll der Abruf einer „Alles-auf-einen-Blick“ bzw. eine Überblick-Anzeige mit Informationen zum aktuellen Ziel der Reise, wie beispielsweise die aktuelle Wetterlage oder der Zeitpunkt der nächsten Abfahrt, möglich sein.

Die Route und dementsprechend die Zugverbindungen sollen automatisiert und offline verfügbar gespeichert, über einen Algorithmus bestimmt und angezeigt werden. Abschließend soll eine fertig geplante Reise über soziale Medien mittels einem je nach der Route generierten Bild und individuellen Text geteilt werden können.

Die Zielbestimmungen und Funktionen wurden in einer frühen Fassung in einem Lastenheft festgehalten, welches der Dokumentation als Anhang beiliegt (ANHANG I).

3 Systemarchitektur

Das Benutzerinterface der Android Applikation setzt sich aus zwei Android-Activities und mehreren Fragments zusammen. Die MainActivity stellt hierbei die Hauptkomponente dar. In diese werden je nachdem welche Informationen vom Anwender aufgerufen werden die jeweiligen Fragments geladen. Eine zweite separate Activity (NewTourActivity) lädt die entsprechenden Fragmente zum Anlegen einer neuen Tour (Abbildung 1: Übersicht Aufbau des Benutzerinterface).

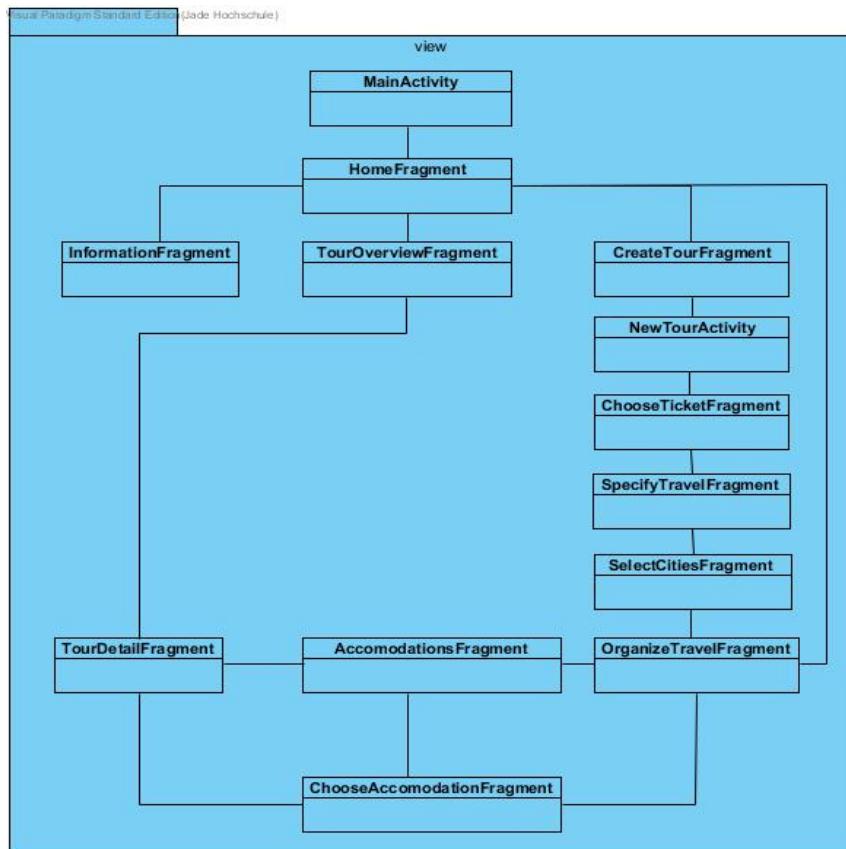


Abbildung 1: Übersicht Aufbau des Benutzerinterface

Die Anwendungslogik setzt sich unter anderem aus einigen Modellklassen zusammen, die wiederum über Adapter im Package Graph zur Berechnung der Routen oder zur Darstellung von Tourinformationen eingesetzt werden können. Zusätzlich werden Schnittstellen für Wetter oder Unterkünfte eingebunden.

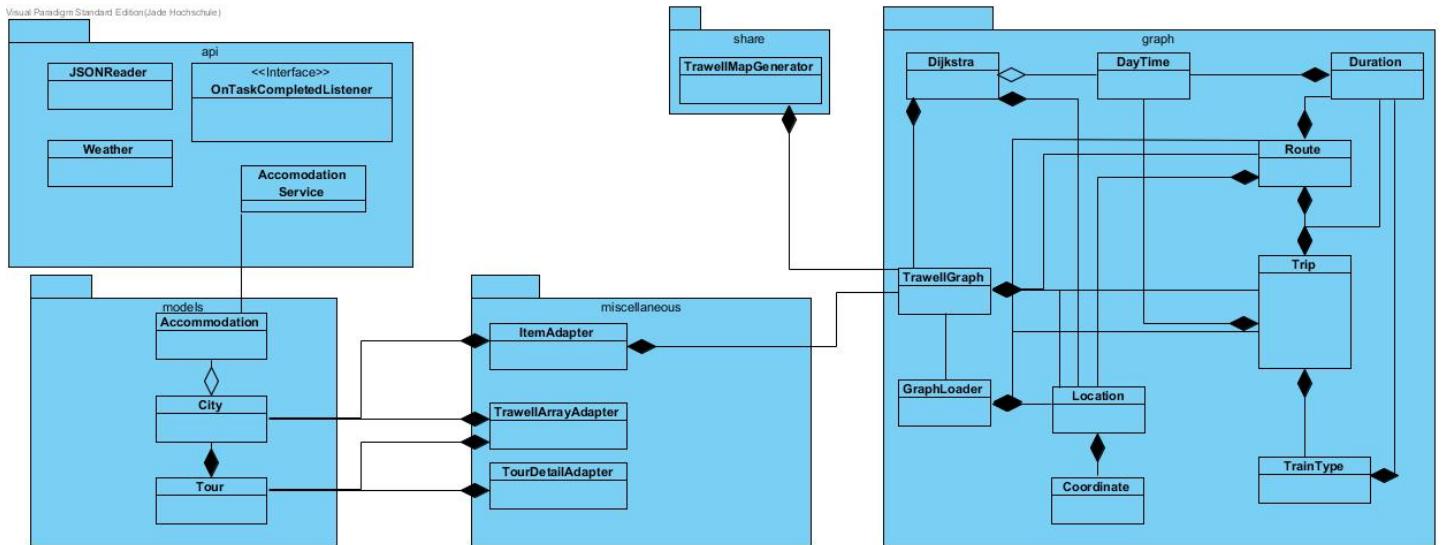


Abbildung 2: Anwendungslogik

4 Implementierung

Zur Implementierung der geplanten Systemarchitektur wurde der Aufbau des Benutzerinterfaces zuerst realisiert. Durch die Implementierung des Layouts konnte ein erster Eindruck zu Farbgestaltung und den Größenverhältnissen gewonnen werden. Sie orientierte sich am Corporate-Design des Interrail Unternehmens.

Im nächsten Arbeitsschritt wurde eine umfassende Datengrundlage aufgebaut. Es wurden passende Datenschnittstellen (APIs) identifiziert und auf ihre Nutzungsbedingungen geprüft. Eine Datendarstellung wurden über die entsprechenden Fragmente in der Android Applikation realisiert. Zur Routenfindung wurde der Dijkstra-Algorithmus implementiert, näheres hierzu unter 4.5 Routenfindung. Des Weiteren wurde eine grundlegende Selektion von Reisezielen in einer Karte ermöglicht. Zusätzlich wurde eine Datenspeicherung implementiert. Abschließend wurde eine Routenfindung mittels Dijkstra-Algorithmus realisiert.

Zur Quellcode-Versionierungsverwaltung wurde das Versionierungswerkzeug Git verwendet. In der Form des Online-Dienst GitHub ermöglicht es zusätzlich ein agiles Projektmanagement mit protokollierter Aufgabenverteilung und einem klar abgesteckten zeitlichen Rahmen. Eine parallele Softwareentwicklung über mehrere private Rechner ist somit einfach möglich. GitHub selber hostet eine Vielzahl von überwiegend Open Source Softwareprojekten (lernmoment.de Was ist GitHub?, kein Datum).

Das Repository ist unter dem folgenden Link öffentlich erreichbar:

<https://github.com/luxn/geoinf>

Eine fertige ZIP-Datei des vollständigen Repositories im Master-Branch kann hier heruntergeladen werden:

<https://github.com/luxn/geoinf/archive/master.zip>

Im Ordner Dokumentation befinden sich alle Dokumentationsdateien zum Projekt. Der Ordner Trawell beinhaltet ein Android-Studio Projekt mit allen relevanten Quellcode und Ressourcen Dateien. Außerdem gibt es einen Ordner GTFS, in welchem alle relevanten Dateien im Markdown-Format sowie die entsprechenden Python-Skripte sowie eine ZIP Datei der Binaries für die später angesprochene Transitfeed-Bibliothek zur Analyse von GTFS-Feeds. Die Feeds selber sind unter /gtfs/Feeds zu finden.

Im Folgenden wird nun auf einzelne Teilbereiche der Implementierung detaillierter eingegangen.

4.1 Benutzeroberfläche und Layout

Ein wesentlicher Bestandteil der Anwendung ist die grafische Benutzeroberfläche, die alle technischen Funktionalitäten für die Benutzer bedienbar macht. Die GUI soll zum einen auf die Zielgruppe zugeschnitten sein und zum anderen auch mit dem Interrail-Design harmonieren. Für ein einheitliches Design wurden die Farben des Interrail-Logos in der Anwendung verwendet und lediglich um neutrale Farbe ergänzt. Die Benutzeroberfläche soll intuitiv und einfach gestaltet sein und dem Benutzer einen hohen Bedienkomfort bieten. Die Anwendung soll weiterhin klar und übersichtlich strukturiert sein.

Die Dialoggestaltung zwischen dem Benutzer und der Anwendung soll ebenfalls selbsterklärend und einfach gestaltet werden, sodass der Benutzer bei der Steuerung der App auch die Rückmeldungen erhält, die er erwarten würde. Zudem sollen dem Benutzer bei der Eingabe von Informationen oder bei der Auswahl nur Möglichkeiten gegeben werden die auch durchführbar sind. Es sollen also keine überflüssigen Informationen angezeigt oder zur Auswahl bereitgestellt werden. Des Weiteren sollen unnötige Interaktionen (Klicks) vermieden werden und dem Benutzer nur die Aktionen zur Verfügung stehen, die auch ausführbar sind. Durch Toasts soll der Benutzer auf Fehler oder nicht ausführbare Aktionen hingewiesen werden. Ein Toast ist ein einfaches schlankes Popup, das für eine Moment eine Nachricht z.B. eine Rückmeldung anzeigt.

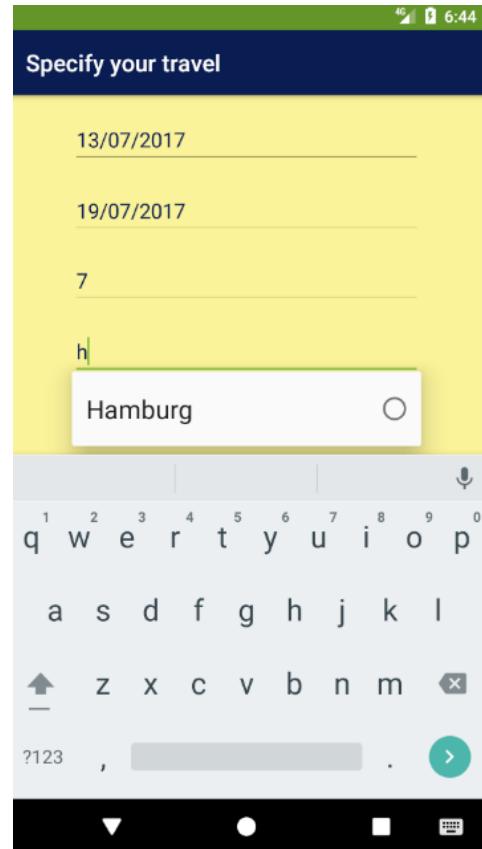


Abbildung 3: Eingabe einer Stadt mit vorgegebenen Möglichkeiten

Für den Aufbau der Benutzeroberfläche werden Activities und Fragments verwendet. Eine Activity ist ein Bestandteil einer Anwendung, die einen Bildschirm/ ein Fenster bereitstellt, in dem die Benutzeroberfläche, mit dem der Benutzer interagieren kann, dargestellt wird. Ein Fragment ist ein modularer Bereich einer Activity und kann in mehreren Activities verwendet werden. Activities und Fragments haben jeweils eigene Lebenszyklen. Jedoch verhalten sich die Fragments, die in einer Activity enthalten, bei Zustandsänderungen der Activity gleich. Für die Anwendung wurden zwei Activities erstellt die dann wiederum einzelne Fragments verwalten. Zum einen gibt es eine MainActivity, die beim Starten der Anwendung ausgeführt wird und zu Beginn das HomeFragment anzeigt. Zusätzlich zeigt die MainActivity unabhängig von dem angezeigten Fragment immer eine Navigationsleiste. Die zweite Activity, die NewTourActivity dient zur Dialogsteuerung für die gesamte Erstellung einer neuen Tour. Die NewTourActivity verwaltet also alle Fragments, die unmittelbar im Zusammenhang mit dem Erstellen einer neuen Tour stehen. Die beiden Activities enthalten ansonsten keine einzelnen UI-Elemente sondern dienen sozusagen nur als Container für Fragments. Die Fragments dagegen setzen sich aus vielen verschiedenen UI-Elementen (Buttons, TextViews, ImageViews, ListViews, Layouts, usw.) zusammen. Die Interaktionen der UI-Elemente werden von dem jeweiligen Fragment verwaltet und verarbeitet.

Die Anordnung der UI-Elemente in den einzelnen Fragments werden durch das hier verwendete ConstraintLayout festgelegt. Das constraintLayout beschreibt Bedingungen (relative Abhängigkeiten) zwischen den UI-Elementen, die zur Positionierung dienen. Jedes Element kann an allen Seiten eine Verbindung zu einem anderen UI-Element haben. Die Verwendung des ConstraintsLayout ermöglicht eine vollständige Entwicklung des Layouts im Design-Modus und es lassen sich somit schnell Layouts erstellen. Im Hintergrund werden die Layouts für die einzelnen Fragments in einer XML-Datei gespeichert und können auch manuell angepasst werden.

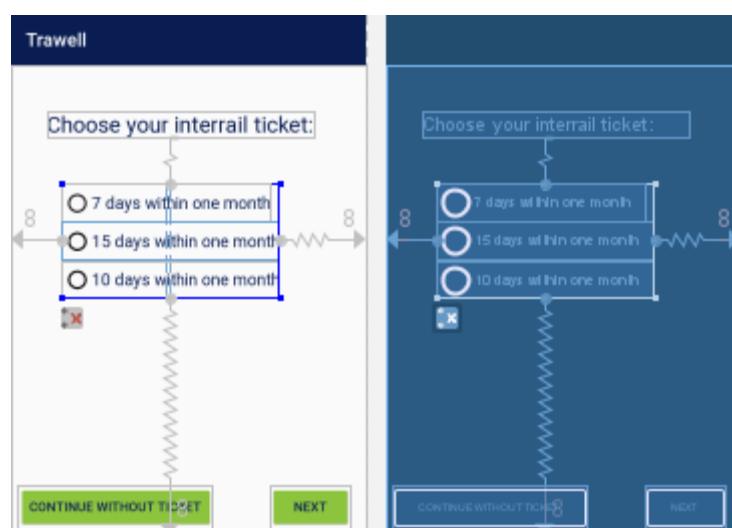


Abbildung 4: Constraints für ein UI-Element

Projekt Geoinformatik SS2017 – Trawell

Alle Activities und Fragments sind unter der Verwendung des ConstraintLayouts aufgebaut worden. Im Folgenden werden einige Layout-Entwürfe mit den endgültigen Ergebnissen verglichen. Viele der Entwürfe wurden auch zum Großteil so umgesetzt und meist nur geringe Veränderungen vorgenommen. Jedoch wurde die Anwendung um einige neue Fragments/Layouts erweitert.

Das HomeFragment hat sich im wesentlich dahin geändert, dass anstatt von einzelnen Buttons und ImageButtons eine Navigationsleiste eingefügt wurde. Zudem werden zu dem aktuell angezeigten Bild und der zugehörigen Stadt Informationen über das Wetter eingeblendet.



Abbildung 5: Entwurf für das HomeFragment



Abbildung 6: Ergebnis für das HomeFragment

Die Entwürfe des Layouts finden Sie im Anhang IV.

4.2 Google Maps und Places API

Eine der Anforderungen war die Auswahl/Selektion der zu bereisenden Städte in einer Karte. Um diese Funktionalität zu realisieren wurde auf die Google Maps Android API zugegriffen. Mit der Google Maps API können Karten basierend auf Google Maps-Daten zu der Anwendung hinzugefügt und über ein von Google bereitgestelltes MapView-Element angezeigt werden. Die API verarbeitet das Herunterladen der Daten, die Kartenanzeige sowie die Benutzerinteraktionen automatisch. Weiterhin bietet die API noch Funktionen zum Setzen von Markern oder geometrischen Figuren. Für die Anwendungen wurden lediglich die Marker, die ein Symbol an einer bestimmten Position auf der Karte darstellen, verwendet. Alle Marker sind automatisch interaktiv und erhalten ein *click*-Ereignis. Zudem können die Marker selbst gestaltet werden oder durch ein eigenes Icon dargestellt werden. In der Anwendung wurden somit alle Städte die zur Auswahl stehen mit einem Marker an deren Entsprechender Position versehen und können über einen Klick ausgewählt und abgewählt werden. Für die Marker wurde ein Standardsymbol verwendet und es wird lediglich die Farbe beim Auswählen angepasst.



Abbildung 7: Kartendarstellung mit Markern in der App

Um die Google Maps Android API zu verwenden benötigt man ein Google-Account über den man dann seinen Eigenen API-Schlüssel für ein bestimmtes Projekt generieren lassen kann. Dieser Schlüssel kann danach für alle weiteren APIs von Google verwendet werden. Zur Sicherheit kann der Schlüssel nur auf Android-Apps oder andere Anwendungen (Websites, IP-Adressen) eingeschränkt werden. Jedoch werden im weiteren Verlauf noch weitere Google APIs verwendet und die Einschränkung des Schlüssels für diese Anwendung ist somit nicht möglich gewesen. Darüber hinaus wird die Verwendung des API-Schlüssels durch ein Kontingent an Datenabfragen der einzelnen APIs begrenzt. Für die Google Maps API ist das Kontingent allerdings unbegrenzt und es können so viele Daten wie nötig abgefragt werden.

Um den API-Schlüssel in der Anwendung verwenden zu können, muss dieser in die Manifest-Datei des Projekts eingebunden werden. Die Manifest-Datei ist eine XML-Datei, die in jedem Android Projekt

```
<meta-data  
    android:name="com.google.android.geo.API_KEY"  
    android:value="AIzaSyAgxZyKMWkDMuwcYZEfIAPvsFghJC04NDY" />
```

Abbildung 8: Ausschnitt aus der Manifest-Datei

enthalten sein muss. Die Datei trägt den Namen *AndroidManifest.xml* und enthält wichtige Informationen über die Applikation. Diese Angaben werden benötigt um die App zu installieren und ausführen zu können. Um den API-Schlüssel in die Manifest-Datei zu integrieren, wird ein neues Element in die XML-Datei eingefügt. Das Element enthält die Attribute *name* und *value* (API-Key), über die der API-Schlüssel festgelegt wird.

Für die ausgewählten Reiseziele sollten über eine weitere Funktion Unterkünfte zu den entsprechenden Städten ausgewählt werden können. Zur Realisierung wurden ebenfalls APIs von Google genutzt. Um die passenden Unterkünfte für eine Stadt zu erhalten wurden im ersten Schritt die Google Places IDs für bestimmte Unterkünfte über die Google Places API Web Service herausgefiltert. Dafür wurde eine sogenannte Nearby Search-Anforderung für jede Stadt durchgeführt. Mithilfe einer Nearby Search-Anforderung können gezielt Orte (Places) innerhalb eines definierten Gebietes gesucht werden. Die Nearby-Search-Anforderung ist eine HTTP URL, bei der die Suchanforderungen über Schlüsselwörter definiert werden können. Folgende Parameter sind bei jeder Anfrage erforderlich. Zum einen der *API-Schlüssel*, die *Location* der Stadt (Längen- und Breitengrad), ein *Radius* in dem gesucht wird sowie *rankby* (Reihenfolge der Ergebnisse). Wobei es für die Angaben Radius und rankby noch Sonderreglungen gibt, sodass rankby nicht angegeben werden muss, wenn ein Radius definiert wird. Die Suche nach den Unterkünften beinhaltete ebenfalls diese Parameter. Zusätzlich wurde ein Parameter *type* hinzugefügt, der die Suche nach Unterkünften (*type: lodgings*) einschränkt. Für die jeweiligen Städte wurde innerhalb eines Radius von 5.000 m um den Mittelpunkt der Stadt nach Unterkünften gesucht. Als Antwort der Nearby Search-Anforderung erhält man ein JSON-Objekt (JavaScript Object Notation) oder eine XML-Objekt (Extensible Markup Language). Das Format, dass

Projekt Geoinformatik SS2017 – Trawell

man erhalten möchte, wird ebenso in der HTTP URL angegeben. Für die Anwendung wurde ein JSON angefordert, dass anschließend nach der Places ID durchsucht wurde.

Mithilfe der Google Places IDs konnten danach im zweiten Schritt die Informationen zu den Unterkünften (Orte mit dem type lodging) über die Google Places Android API ermittelt werden. Durch die zuvor bestimmten Places IDs können die Orte direkt anhand der ID angefordert werden. Dafür stellt

```
html_attributions:  
▶ next_page_token: "CpQCAQEAC006en1qT9pPuQW..4i2MXwacOjR88j3D9Nbm7CA"  
▼ results:  
  ▼ 0:  
    ▼ geometry:  
      ▼ location:  
        lat: 48.5851274  
        lng: 7.746176  
      ▶ viewport: Object  
    ▶ icon: "https://maps.gstatic.com/pi/icons/lodging-71.png"  
    id: "e8f7d9f6471e447437dccdad0c89b31c8635f95f"  
    name: "Hotel Sofitel Strasbourg Grande Ile"  
    ▶ opening_hours: Object  
    ▶ photos: [1]  
    place_id: "ChIJ_ecXE07I1kcR-TndJQuTfJs"  
    rating: 4.2  
    ▶ reference: "CmRSAAAAz418eTQpNB-pjsrE..jojr6U9s3oyK0vdCl_4aTng"  
    scope: "GOOGLE"  
    ▶ types: [5]  
    vicinity: "4 Place Saint-Pierre-le-Jeune, Strasbourg"  
  ▼ 1:  
    ▼ geometry:  
      ▼ location:  
        lat: 48.58486389999999  
        lng: 7.745490599999999  
      ▶ viewport:  
        ▶ northeast: Object  
        ▶ southwest: Object  
    ▶ icon: "https://maps.gstatic.com/mapfiles/place_api/icons/lodging-71.png"  
    id: "d76b5f6ff73f316243e6686422a9b330bacf2b49"  
    name: "Hotel Mercure Strasbourg Centre"  
    ▶ opening_hours:  
      open_now: true  
      weekday_text:  
    ▶ photos:  
      ▶
```

Abbildung 9: Antwort einer Nearby Search im JSON-Format

```
  ▼ html_attributions:  
    ▼ 0: "<a href=\"https://maps.google.com/maps/contrib/102646412341187735725,  
  ▼ photo_reference: \"CmRaAAAA3UHCX6VHicjqTLe0sECRa6R6OrfbKppi1LttHH1B4HmFIR7Ac0qwt-epZf-yp/  
    width: 5312  
  place_id: \"ChIJs517ek7I1kcRc7K1BQz4Yd4"
```

die Google Places Android API das Interface *GeoDataApi* mit der Methode *getPlaceById* bereit. Über die Methode werden dann alle Orte die zu der ID passen in einem *PlaceBuffer* zurückgegeben. Jedoch

kann jede Orts-ID nur auf einen Ort verweisen und somit ist das erste Objekt der gesuchte Ort (Place). Die Objekte in dem *PlaceBuffer* werden als *Place* repräsentiert und enthalten alle Informationen (Ortsdaten) zu ihrem Ort. Zu den Ortsdaten zählen eine Vielzahl von Daten. Für die Unterkünfte wurden folgende Daten abgefragt: Name der Unterkunft, Adresse, Bewertung der Unterkunft, Telefonnummer und die Internetadresse sofern diese verfügbar ist.

Für die Google Places API Android und Web Service konnte der gleiche API-Schlüssel wie für die Google Maps API verwendet werden. Allerdings gab es für diese beiden APIs andere Kontingentbegrenzungen. Die Google Places API für Android und Web Services wurden auf 1.000 Anfragen pro Tag begrenzt.

4.3 Weather API

Die benötigten Wetterdaten für die Städte wurden durch die API „OpenWeatherMap“ bezogen. Dazu wurde eine Anfrage an den Dienst gestellt, bei welcher nur der Name des angeforderten Ortes angegeben werden musste. Die Antwort des Dienstes lag im JSON-Format vor. Für eine kurze und übersichtliche Darstellung des Wetters in einem Ort sind vor allem die Temperatur, die Niederschlagswahrscheinlichkeit sowie eine allgemeine Wetterinformation (Sonne, Regen, Wolken etc.) ausschlaggebend. Da die verwendete API jedoch leider keine Informationen zur Niederschlagswahrscheinlichkeit liefert, wurde alternativ die Luftfeuchtigkeit verwendet. Die allgemeine Wetterinformation wird mit einem Icon veranschaulicht. Die kostenlose API beschränkt den Nutzer auf maximal 60 Aufrufe pro Minute, was jedoch ausreichend ist. Bei einer Vermarktung der App könnte man die Begrenzung durch einen Premiumzugang leicht aufheben. Eine beispielhafte Antwort der API im JSON-Format liegt dieser Dokumentation als Anhang VI bei.

4.4 Beziehung von Zugverbindungsdaten über GTFS

Um eine passende Navigation mit den Zugverbindungsdaten benötigt man diese. Hierbei bietet sich das GTFS Format an, welche mehrere Verkehrsunternehmen bereitstellen.

GTFS (*General Transit Feed Specification*) ist ein von Google entworfenes Austauschformat um Fahrpläne für den ÖPNV und zusätzliche Informationen, wie die Geokoordinaten der Haltestellen, auszutauschen. Es ist ein textbasiertes Format und wird mit einem ZIP-Archiv dargestellt, welche intern aus von sechs bis zu 13 CSV-Dateien besteht. Dabei soll das GTFS Format den öffentlichen sichtbaren Fahrplan eines Verkehrsunternehmens darstellen. Die CSV-Dateien stellen ein Abbild einer relationalen Datenbank mit Fahrplänen des Unternehmens dar. Eine ZIP-Datei mit den CSV-Dateien wird auch als GTFS-Feed bezeichnet (Wikipedia, 2017)

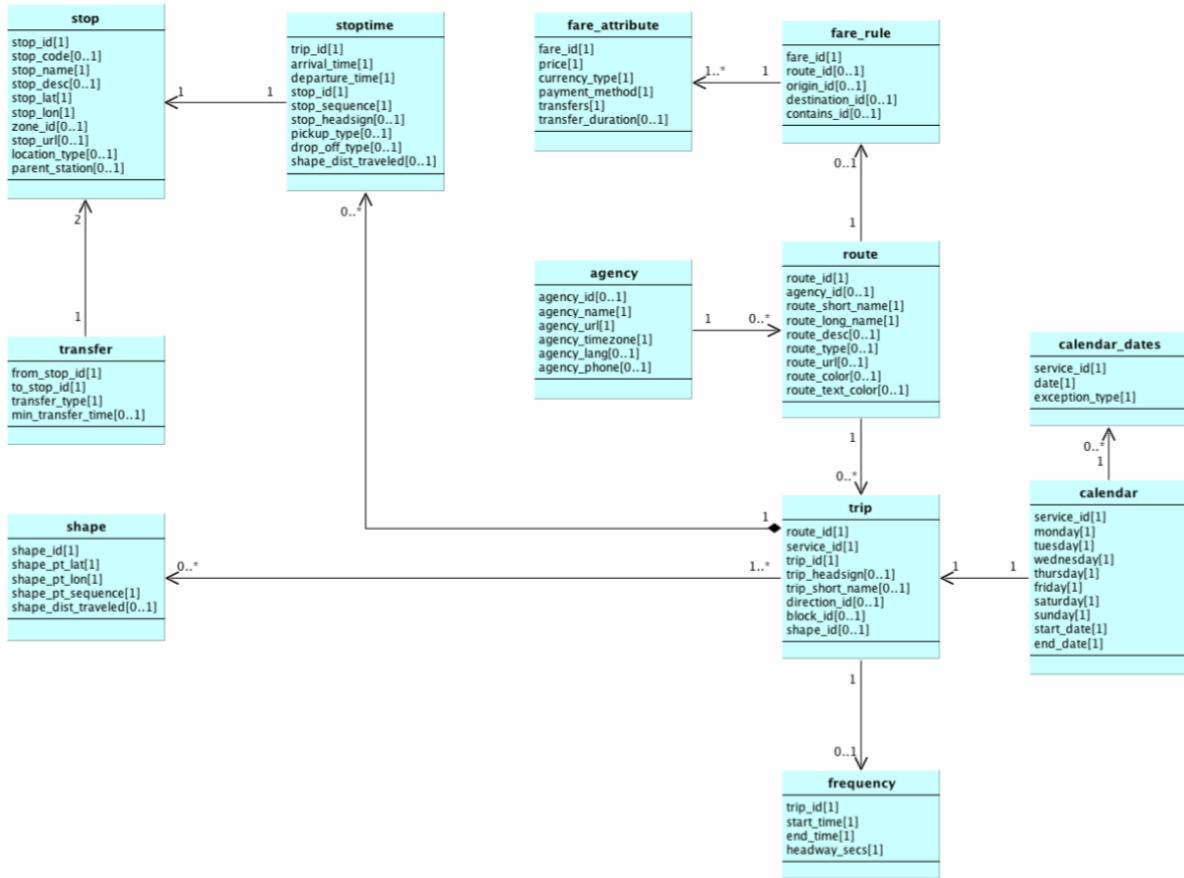


Abbildung 10: Relationen in einem GTFS-Feed (developers.google.com, 2016)

Zur Beziehung und Recherche der passenden GTFS-Feeds wurde die Seite transitfeeds.com verwendet. Diese Seite bietet einen Index, um passende GTFS Feeds für jeweilige Länder und Städte aufzuspüren.

Hier bestanden nun Probleme: Da es in einen übergeordneten, transnationalen einzelnen GTFS Feed für ganz Europa nicht gibt, und die bestehenden GTFS Feeds eher von lokalen Verkehrsunternehmen und Verkehrsbunden verwendet wird, dazu die nationalen GTFS Feeds eher auf freiwilliger Basis von Dritten zusammengestellt worden sind, musste hier eine Auswahl auf nur bestimmte Städte und Länder innerhalb Europas und des Geltungsbereiches der Interrail-Tickets gesetzt werden. Weitere Schwierigkeiten treten dabei auf, dass aufgrund der überwiegenden Regionalität der GTFS-Feeds ein internationaler Zugverkehr mit wenigen Ausnahmen nicht abgebildet werden kann bzw. wurde, da die Züge meist an der nächsten Grenzstadt im jeweiligen Ausland endeten.

Aus diesen Gründen können die Daten der GTFS-Feeds nicht einfach so übernommen werden und bspw. im relationalem Modell einer SQLite Datenbank auf Android abgebildet werden. Deshalb müssen die Feeds weiter analysiert und gefiltert werden. Hierbei wurde die Python Bibliothek `gtfslib-python`¹ verwendet. Sie bietet eine einfache Möglichkeit sich u.a. einen bestehenden GTFS-

¹ <https://github.com/afimb/gtfslib-python>

Feed in eine SQLite Datenbank zu transferieren und sich komfortabler bspw. mit einfachen Schleifen Orte und Zugnummer anzeigen zu lassen wie das untere Listing zeigt.

```
from gtfslib.dao import Dao
dao = Dao("db.sqlite")
dao.load_gtfs("mygtfs.zip")
for stop in dao.stops():
    print(stop.stop_name)
for route in dao.routes(filter=Route.route_type == Route.TYPE_BUS):
    print("%s: %d trips" % (route.route_long_name, len(route.trips)))
Listing 1: gtfslib-python Snippet – Es werden alle Routen ausgegeben, welche von Typ „Bus“ sind und die Anzahl ihrer Trips auf dieser Route
```

Um trotzdem ein funktionales und offline verfügbares Routing zu bekommen, welche auch den transnationalen Zugverkehr beinhaltet, müssen diese vereinzelten Routen und Fahrten aus den jeweiligen Feeds entnommen werden, welche von einer größeren Stadt im Interrail-Ticket zu einer nächsten, auch im Ausland, entnommen werden. Dazu wurde sich ebenfalls einer Python Bibliothek bedient, nämlich der eigens von Google erstellen Bibliothek namens transitfeed². Transitfeed selber kann auch als Kommandozeilen Programm und als vorkompilierte Windows Version verwendet werden. Es gibt zum einen ein Python Package ähnlich der ersten Bibliothek. Zudem gibt es einen Feed Validator, welche per übergebenen GTFS ZIP-Archiv möglich Fehler und Verbesserungen in einer HTML-Seite ausgibt und zudem prüft, ob der Feed den allgemeinen Anforderungen entspricht. Des Weiteren gibt es Tools zum Kombinieren zweier Feeds („Merge“), einen KML-Writer oder auch einen ScheduleViewer. Der ScheduleViewer wurde in der weiteren Analyse verwendet, um die Zugrouten zu identifizieren, welche die intranationalen Verbindungen darstellen, da diese optisch leicht mit den hinterlegten Google Maps Karten zu identifizieren und anklickbar sind. Dabei werden weitere Informationen angezeigt für die jeweiligen Trips und die Abfahrts- und Ankunftszeiten.

Bei der nun getätigten Analyse fiel zudem noch auf, dass die Feeds auch unterschiedlich in der Datenqualität sind, einige bieten Busverbindungen mit, andere nur Zugverbindungen. Zudem ist bei einem Feed ein Bahnhof nur ein Bahnhof, während woanders jedes Gleis des Bahnhofs als eigener Bahnhof geführt wird. Dies führte nun dazu, dass eine weitere Vereinfachung getätigkt werden muss, nämlich, dass die Gleise nicht weiter berücksichtigt werden. Dies folgte im Schluss nun zu folgenden Vereinfachungen und Städten/Ländern die nun von der App abgedeckt werden:

- Es wurden GTFS-Feeds aus Deutschland, Frankreich, Niederlande, Belgien und Luxemburg verwendet.
- Auf Gleise und verschiedene Bahnhöfe in einer Stadt (Paris Nord, Paris Est) wurde nicht weiter unterschieden.
- Es werden nur Zugfernverkehr und Direktverbindungen übernommen.

² <https://github.com/google/transitfeed>

- Es wird angenommen, dass die Züge zu den Zeiten täglich immer fahren, unabhängig von Feiertragen o. Ä. und zudem sind nur zwischen 3 und 5 Verbindungen täglich übertragen worden

Daraus ergab sich nun ein Streckennetz aus den folgenden Städten:

Amsterdam, Rotterdam, Antwerpen, Brüssel, Luxembourg, Hamburg, Berlin, Köln, München, Frankfurt, Paris, Strasbourg, Lyon, Nice, Marseille, Montpellier, Bordeaux

Die relevanten Daten wurden nun ein drei eigene erstellen CSV Dateien übertragen und später zur Laufzeit der App in ein Graphenmodell geladen.

Die Orte/Städte befinden sich in der locations.csv und bestehen aus dem Namen, dem Land, der Position in Longitude und Latitude und einer GoogleID für GoogleMaps/GooglePlaces. Sie stellen in einem Graphenmodell die Vertices, Nodes bzw. Knoten bereit.

Zu einem Ort gehören die zugehörigen Verbindungen zu anderen Orten welche in der routes.csv zu finden sind. Dort sind Ursprungsort und Zielort hinterlegt. Es handelt sich um einen gerichteten Graphen und die Routes stellen hier die Edges bzw. Kanten dar. In der CSV sind allerdings alles Verbindungen nur einmal hinterlegt, da die inverse Verbindung beim Einlesen automatisch gesetzt wird.

Zuletzt fehlen noch die Abfahrtszeiten, welche in der Datei trips.csv gespeichert sind und beim Einlesen der jeweiligen Route als Liste zugeordnet werden. Hier sind auch Zugnummer und Zug-Typ hinterlegt.

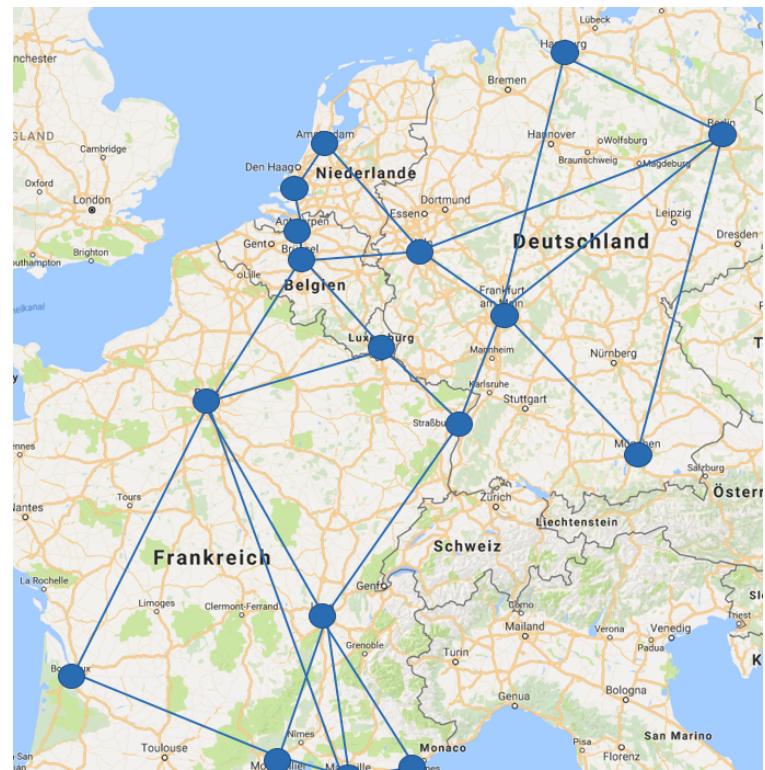


Abbildung 11: Graph-Übersicht

4.5 Routenfindung

Aus dem obigen Graphmodell wird zu Laufzeit ein Graphmodell basierend auf Java-Klassen realisiert. Die drei CSV-Dateien werden über die Klasse GraphLoader im Paket graph eingelesen. Dabei ist er so designed, dass es immer nur eine Instanz des Graphen geben kann. Beim erstmaligen Anfrage an den

Graphen wird er geladen und sonst immer nur Referenzen gebildet (Singleton Pattern). Die Klasse TrawellGraph bietet hier den Einstiegspunkt und stellt die APIs zur Verwendung bereit.

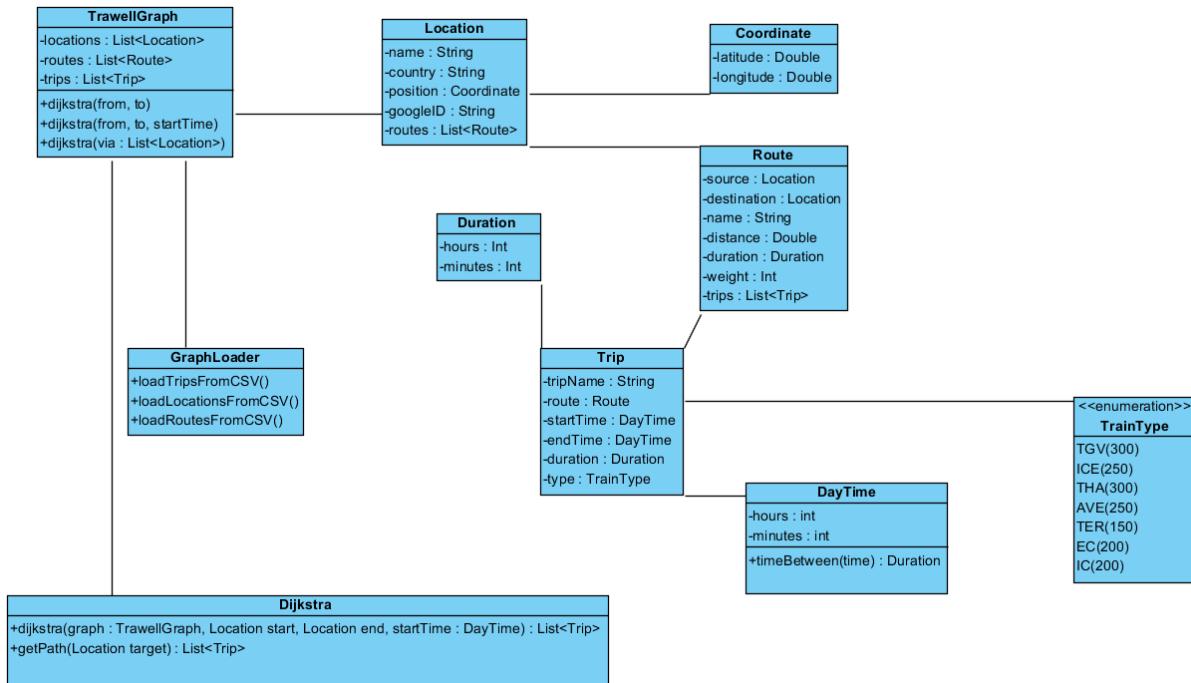


Abbildung 12: Klassendiagramm zum Paket `graph` mitsamt Hilfsklassen zur Zeitberechnung

Die Routenfindung wird durch den Dijkstra-Algorithmus realisiert. Dieser beschreibt den kürzesten Weg in einem Graphen mit nicht negativen Gewichten. Dafür müssen zu den Kanten die Informationen zur Entfernung und zu den Kosten dieser Fahrt, welche durch die Fahrzeit in Minuten berechnet wird, angegeben werden. Der Dijkstra-Algorithmus berechnet und besucht nun von einem Startort aus alle Verbindungen und erhöht jeweils immer das Gewicht der darauffolgenden Verbindung um die dortige Wartezeit an den nächsten Ort. Dabei wird immer ein Vorgängerort samt Trip der genommen wurde gesetzt, um schlussendlich durch eine inverse Routenbildung, ähnlich dem Backtracking, aufgrund von Zeigern/Referenzen berechnen zu können. Das Ergebnis ist eine Liste von passenden Trips zeitlich korrekt zum Zielort. Man führt nun auf dem Graphen den Dijkstra-Algorithmus samt Startort und Startzeit aus, es werden an jedem Ort die jeweiligen Vorgänger gesetzt und man kann anschließen bequem jeden Verbindungsablauf durch das Backtracking von einem Zielort abfragen.

4.6 Datenmodell der Touren

Die Touren, die der Benutzer mit Hilfe der Anwendung erstellt, sollen anschließend auch gespeichert werden und jederzeit Abrufbar sein. Ebenfalls sollen gespeicherte Touren gelöscht und ausgewählte Daten nachträglich angepasst werden können.

Um die zugehörigen Daten einer Tour sinnvoll zu speichern wurden durch Objektrelationalen Abbildungen (object-relational mapping, ORM) die Tour-, Stadt- und Unterkunfts-Objekte aus JAVA

direkt in ein relationales Datenmodell überführt. Für diesen Schritt wurde die Bibliothek Sugar ORM (<http://satyan.github.io/sugar/>) verwendet. Mithilfe der Bibliothek wird automatische eine lokale SQLite Datenbank auf dem Endgerät angelegt und die gekennzeichneten Klassen werden zu relationalen Tabellen überführt. Diese Bibliothek eliminiert das Schreiben von SQL-Abfragen und kümmert sich gleichzeitig um das Erstellen einer Datenbank. Des Weiteren wird die Verwaltung von Objektbeziehungen erleichtert. ORM hat dabei den Vorteil, dass man sich bei einfachen Anwendungen nicht selbst mit der Datenbank befassen muss, dies bedeutet dass man das eigentliche Attribut des Objects direkt ändern kann. Allerdings ist es möglich, dass wenn man auf speziellere Datenbank abfragen angewiesen ist, diese wieder in SQL schreiben muss (Cremer, 2017).

Nachdem eine Tour erstellt wurde, werden zuerst die Daten einer Tour abgelegt und anschließend die ausgewählten Städte (Reiseziele). Im zweiten Schritt können die Unterkünfte ausgewählt und abgespeichert werden. Die Unterkünfte können auch nachträglich ausgewählt oder geändert werden. Hingegen kann eine bestehende Tour mit ihren Städten nicht nachträglich bearbeitet werden. Das Tour-Objekt enthält folgende Attribute:

- startCity: String – Name der Ausgangsstadt
- finalCity: String – Name der Zielstadt
- start: Date – Datum des Reisebeginns
- end: Date – Datum des Reiseendes
- duration: int – Dauer der Reise

Die Stadt-Objekte (intern City) enthalten jeweils ein Attribut Tour, dass auf die zugehörige Tour verweist und werden mit folgenden Attributen abgespeichert:

- name: String – Name der zu bereisenden Stadt
- duration: int – Aufenthaltsdauer in der Stadt
- tour: Tour – Verweis auf das zugehörige Tour-Objekt

Die Unterkunfts-Objekte (intern Accommodations) enthalten jeweils ein Attribut City, dass auf die zugehörige Stadt verweist und werden mit folgenden Attributen abgespeichert:

- name: String – Name der Unterkunft
- adresse: String – Adresse der Unterkunft
- bewertung: String – Bewertung der Unterkunft
- phoneNumber: String – Telefonnummer
- url: String – Internetadresse zu der Unterkunft
- city: City – Verweis auf das zugehörige City-Objekt

4.7 Social Media Export

Abschließend soll eine fertig geplante Reise über soziale Medien mittels einem je nach der Route generierten Bild und individuellen Text geteilt werden können.

Die Teilen-Funktion der Tour ist in der App über das standardmäßige Teilen der Android Plattform realisiert worden. Aus den im Datenmodell der Touren referenzierten Orte, die im Graphenmodell mit Koordinaten hinterlegt worden sind, kann mittels der *android.graphics.Canvas* Klasse auf eine bestehende Bilddatei gezeichnet werden. Die Canvas Klasse ist von der Benutzung ähnlich der in vorherigen Veranstaltungen kennengelernten Klasse *Canvas* aus dem *java.awt* Paket. Zuerst muss ein Hintergrundbild verfügbar sein. Es wurde ein Hintergrundbild aus Open Street Map Kacheln erzeugt. Dieses Bild liegt dem Android Applikation in den applikationsinternen Ressourcen bei. Zudem muss eine Georeferenzierung bekannt sein, welche vorher mit QGIS und einem WMS durchgeführt wurde. Die Parameter der Transformation, sprich die entstandene World-File, wurden in der Klasse *TrawellMapGenerator* in den jeweiligen Methoden zur Umrechnung von Lat/Lon Koordinaten in Pixelkoordinaten hinterlegt. Danach kann ausgehend von einer Liste von Orten die besucht werden auf eine veränderbare Canvas Instanz, welche vorher das Bild als Hintergrund gesetzt bekommen hat, gezeichnet werden. Dabei stehen Methoden zum Zeichnen von Text, Kreisen, Ovalen, Rechtecken u.v.m. zur Verfügung. Hiermit kann nun ein individuelles Bild passend zur jeweiligen Tour, die der Benutzer teilen möchte, erstellt werden.

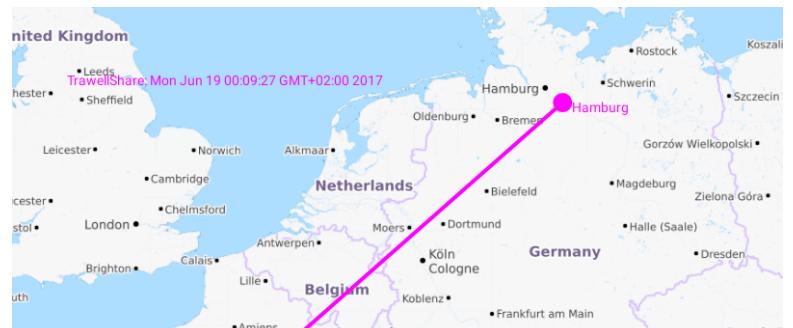
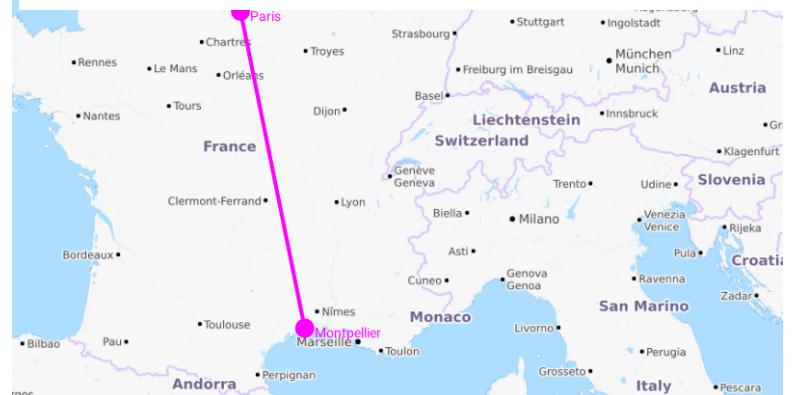


Abbildung 13: Bildexport der Beispieltour Hamburg-Paris-Montpellier



Das hieraus entstandene Bild zum Teilen wurde in den temporären Speicher der App gelegt. Eine Speicher- bzw. Dateireferenz in Form einer URI (Uniform Resource Identifier) mit zugehörigen Beschreibungstext und MIME-Type (*image/**) per Intent an das Android-System übergeben. Der Android-Betriebssystem erstellt nun automatisch ein Teilen-Dialog, welcher alle Apps aufführt, welche bei ihrer Installation dem System mitgeteilt haben, diesen MIME-Type empfangen zu können. Somit ist es möglich das nun erstellte Bild als Email-Anhang zu senden, in die Fotobibliothek zu importieren

oder mit der Twitter-App zu öffnen, welche dort einen Tweet mit angehängtem Bild zum Teilen bereitstellt.

5 Ausblick und Zusammenfassung

Trawell ist eine App, mit der eine Interrail Reise zwischen verschiedenen europäischen Städten geplant werden kann. Aktuell handelt es sich hierbei leider nur um eine Auswahl von Städten. In Zukunft sollte diese Auswahl erweitert werden, sodass zumindest alle Hauptstädte der beteiligten Länder in der App vertreten sind. Zur Umsetzung dieses Vorhabens müssten alle Zugverbindungen verfügbar sein, was zurzeit nicht der Fall ist. Einige Zugstrecken sind momentan zudem nur beispielhaft beziehungsweise simulativ dargestellt. Diese würden im Falle einer besseren Verfügbarkeit der Zugverbindungen genauer und somit könnte dem Anwender eine Auswahl von Zugverbindungen präsentiert werden, aus der Letzterer die gewünschte Abfahrtszeit heraussuchen könnte.

Des Weiteren wäre die App insofern erweiterbar, dass der Benutzer verschiedene Points of Interest (POI) zu den anzureisenden Städten auf einer Karte angezeigt bekommt. Hierfür müsste nur das bisherige Tour- beziehungsweise Accomodationsmodell erweitert werden und ein neues Fragment zur App hinzugefügt werden.

Auch das Buchen eines Interrail Tickets aus der App heraus wäre wünschenswert, jedoch müsste dafür eine Kooperation mit dem Parlament der EU eingegangen werden. Gleicherweise von Vorteil wäre eine Kooperation mit einem Anbieter von Unterkünften - zum Beispiel Airbnb, sodass eine Buchung aus der App heraus vorgenommen werden könnte. Die Kooperation mit Airbnb hätte darüber hinaus den Vorteil, dass dem App-Benutzer mehr günstige Ferienwohnungen und Hostels angeboten werden könnten, die der Hauptzielgruppe der App sicherlich mehr zu sagen würden.

Ein ebenfalls an der jungen Zielgruppe ausgerichteter Verbesserungsvorschlag ist die Erweiterung der Sharing-Funktion. Die Sharing Funktionalität könnte ansprechender gestaltet und hinsichtlich der Individualität erweitert werden. Der Benutzer sollte die Möglichkeit haben, dem geposteten Bild einen Text beizufügen und eventuell eigene Bilder integrieren zu können. Die bildlich dargestellte Tour könnte außerdem die Aufenthaltsdauer in den einzelnen Städten darstellen sowie die Start- und Zielstadt der Interrail Reise hervorheben.

Natürlich müssten bei einer weiteren und tiefer gehenden Bearbeitung der Anwendung ebenfalls die in der App vereinzelt auftretenden Fehler behoben werden.

6 Literaturverzeichnis

Bundesministerium für Verkehr und digitale Infrastruktur. (kein Datum). *bmvi.de*. Abgerufen am 06.

07 2017 von

<http://www.bmvi.de/DE/Themen/Digitales/mFund/Foerderung/foerderung.html>

Cremer, M. (kein Datum). "The Smart Programmer". Abgerufen am 11. 07 2017 von

<https://www.smart-programmer.de/einfuehrung-in-orm-object-relational-mapping/4>

developer.android.com. (07 2017). Abgerufen am 14. 06 2017 von Android Developer:

<https://developer.android.com/about/dashboards/index.html>

developers.google.com. (12. 06 2016). Abgerufen am 14. 06 2017 von

<https://developers.google.com/transit/gtfs/reference/>

github/gtfslib-python. (kein Datum). Abgerufen am 14. 07 2017 von

<https://github.com/afimb/gtfslib-python>

github/transitfeed. (7. 10 2014). Abgerufen am 14. 07 2017 von

<https://github.com/google/transitfeed/wiki>

lernmoment.de Was ist GitHub? (kein Datum). Abgerufen am 14. 07 2017 von

<http://www.lernmoment.de/alle/was-ist-github/>

Wikipedia. (2017). *General Transit Feed Specification*. Abgerufen am 13. 07 2017 von de.wikipedia.org:

https://de.wikipedia.org/wiki/General_Transit_Feed_Specification

7 Abbildungsverzeichnis

Abbildung 1: Übersicht Aufbau des Benutzerinterface	3
Abbildung 2: Anwendungslogik.....	4
Abbildung 3: Eingabe einer Stadt mit vorgegebenen Möglichkeiten.....	5
Abbildung 4: Constraints für ein UI-Element.....	6
Abbildung 5: Entwurf für das HomeFragment	7
Abbildung 6: Ergebnis für das HomeFragment	7
Abbildung 7: Kartendarstellung mit Markern in der App	8
Abbildung 8: Ausschnitt aus der Manifest-Datei	8
Abbildung 9: Antwort einer Nearby Search im JSON-Format.....	10
Abbildung 10: Relationen in einem GTFS-Feed.....	12
Abbildung 11: Graph-Übersicht	14

Projekt Geoinformatik SS2017 – Trawell

Abbildung 12: Klassendiagramm zum Paket graph mitsamt Hilfsklassen zur Zeitberechnung 15

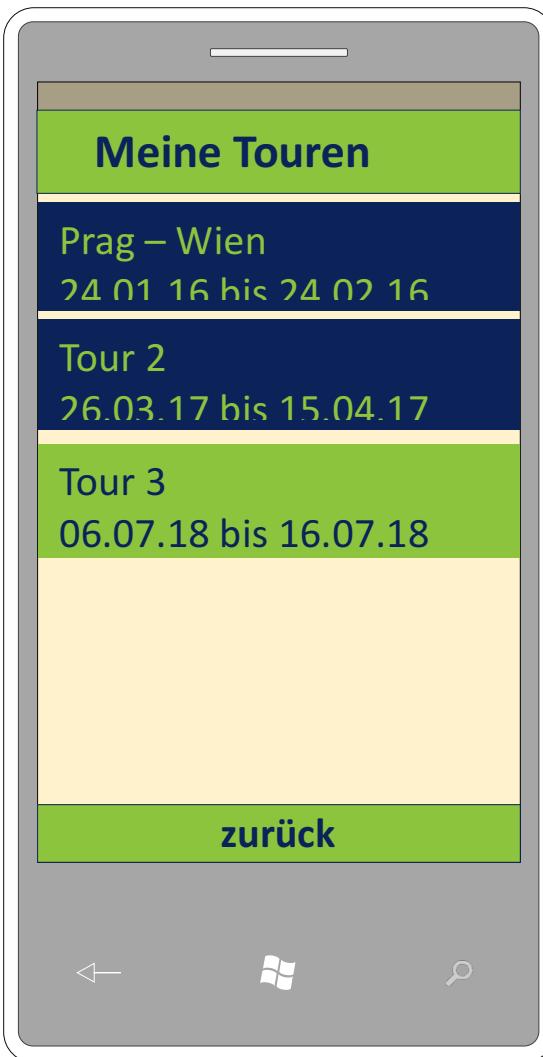
Abbildung 13: Bildexport der Beispieltour Hamburg-Paris-Montpelier 17

8 ANHANG

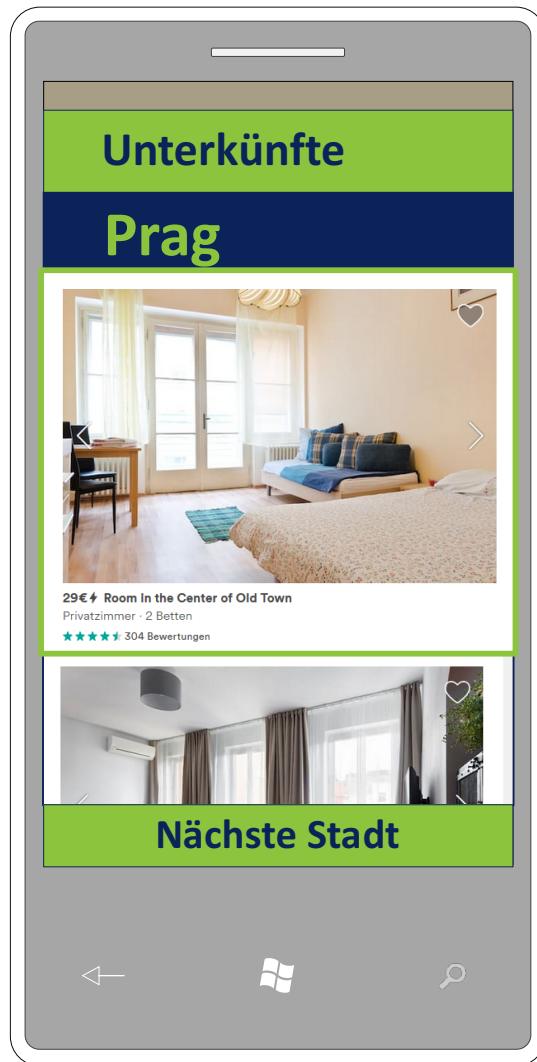
Anhang I Lastenheft

Siehe PDF-Datei Trawell_Lastenheft.pdf

Anhang IV Entwurf des Layouts







Anhang V JSON Google Places API

```
▼ geometry:  
  ▼ location:  
    lat: 51.509963  
    lng: -0.129797  
  ▼ viewport:  
    ▼ northeast:  
      lat: 51.5112819302915  
      lng: -0.12731615  
    ▼ southwest:  
      lat: 51.5085839697085  
      lng: -0.13178755  
  ▼ icon: "https://maps.gstatic.com/mapfiles/place_api/icons/lodging-71.png"  
  id: "7fda34a493402e7b24ace4a0a0bedd012ef6e7c0"  
  name: "Radisson Blu Edwardian, Hampshire"  
  ▼ photos:  
    ▼ 0:  
      height: 370  
      ▼ html_attributions:  
        ▼ 0: "<a href='https://maps.google.com/maps/contrib/103398050898135161305/photos'>Radisson Blu Edwardian, Hampshire</a>"  
      ▼ photo_reference: "CmRaAAAAGsy7u0KSE_VjvpqP3-h9-ffrxQTdqB8A-G1dabinpF0X1ouP-RzRe9kGnWePHFdqM2jdXpq1GmzHKWggOVEs7S8ZAztZPgrF03DYyGvdh6Fgou7Nc0tKOjrawhgtnly6mEhBG8BEgLAPGO9XuvYz02hfkGhTi-rbotkSq0wAzeU1Q8k10jf7Yww"  
      width: 650  
    place_id: "ChiJF-WCA9IEdkgRnhW704RFvCI"  
    rating: 4.1  
  ▼ reference: "CmRRAAAAc0cTZ01W3NTMT1Cz8NYauXByf8rny4f-mA-3f14bwW3sNrKMFa211vYYSkShWxUJQd3z1aLB-ymWlHyQEsXwHAX3tZ7Zcc72xAwQ3V0V38rrLaag16_sIOR9ubQ_pkRGEhAAhaz8nb1fpPJA6iGHn-GhTLgg9dLuHQt1sCbNhzYXrc70iUQ"  
  scope: "GOOGLE"  
  ▼ types:  
    0: "lodging"  
    1: "point_of_interest"  
    2: "establishment"  
  vicinity: "31-36 Leicester Square, London"
```

Aufgerufen mit der URL : <https://maps.googleapis.com/maps/api/place/nearbysearch/json?location=51.51,-0.13&radius=5000&type=lodging&key=AIzaSyAgxZyKMWkDMuwcYZEfIAPvsFghJC04NDY>

Anhang VI JSON Weather API

```
▼ coord:  
  lon: -0.13  
  lat: 51.51  
▼ weather:  
  ▼ 0:  
    id: 500  
    main: "Rain"  
    description: "light rain"  
    icon: "10d"  
  base: "stations"  
▼ main:  
  temp: 287.64  
  pressure: 1010  
  humidity: 72  
  temp_min: 286.15  
  temp_max: 289.15  
  visibility: 10000  
▼ wind:  
  speed: 3.6  
  deg: 240  
▼ rain:  
  3h: 0.815  
▼ clouds:  
  all: 92  
  dt: 1499750400  
▼ sys:  
  type: 1  
  id: 5091  
  message: 0.0035  
  country: "GB"  
  sunrise: 1499745409  
  sunset: 1499804084  
  id: 2643743  
  name: "London"  
  cod: 200
```

Aufgerufen mit der URL :

<http://samples.openweathermap.org/data/2.5/weather?q=London,uk&appid=b1b15e88fa79722541>

2429c1c50c122

Anhang VII API's

