TravelMatch
Architectural Design Document
Version 1.0

D.J. van den Brand (0772180)
S. He (0810831)
J.M.A.P. van Hoof (0778486)
G.C. Linders (0815449)
M.J.M. Muijsers (0805654)
G.H. Santegoeds (0890429)
L.D. Stooker (0819041)
J.W.J.H. Visser (0828234)

22nd June, 2015
Abstract

This document contains the Architectural Design Document, consisting of descriptions of the architecture, for the TravelMatch application, which is used to help people find their holiday destination. This application is developed in the Software Engineering Project at Eindhoven University of Technology. This document complies with the Software Engineering Standard, as specified by the European Space Agency. [1]
# Contents

**Document Status Sheet**  
- General: 3  
- Document history: 3  

**Document Change Records**  
- General: 4  
- Changes: 4  

1 Introduction  
- 1.1 Purpose: 5  
- 1.2 Scope: 5  
- 1.3 Definitions and abbreviations: 5  
  - 1.3.1 Definitions: 5  
  - 1.3.2 Abbreviations: 6  
- 1.4 References: 7  
- 1.5 Overview: 8  

2 System overview  
- 2.1 Background: 9  
- 2.2 Basic Design and Context: 9  
- 2.3 Design Decisions: 9  
  - 2.3.1 Architecture: 9  
  - 2.3.2 Back end: 10  
  - 2.3.3 Persistent storage: 11  
  - 2.3.4 MySQL database: 12  
  - 2.3.5 Front-end: 12  
  - 2.3.6 JWT: 13  

3 System context  
- 3.1 Facebook: 14  
  - 3.1.1 Implementation of the interface: 14  
- 3.2 Mailgun: 14  
  - 3.2.1 Implementation of the interface: 14  
- 3.3 Google Analytics: 15  
  - 3.3.1 Implementation of the interface: 15  
- 3.4 TradeTracker: 15  
  - 3.4.1 Implementation of the interface: 15  

4 System design  
- 4.1 Design methods: 17  
- 4.2 Decomposition description: 17  
  - 4.2.1 TravelMatch server components: 17  
  - 4.2.2 TravelMatch client components: 18  
- 4.3 File structure: 20  
  - 4.3.1 Server file structure: 20  
  - 4.3.2 Client file structure: 21  
- 4.4 Database Schemas: 22
Document Status Sheet

General

Document title: Architectural Design Document
Document identifier: TravelMatch.Doc.ADD/1.0
Authors:
- D.J. van den Brand (0772180)
- S. He (0810831)
- J.M.A.P. van Hoof (0778486)
- G.C. Linders (0815449)
- M.J.M. Muijsers (0805654)
- G.H. Santegoeds (0890429)
- L.D. Stooker (0819041)
- J.W.J.H. Visser (0828234)

Document status: Final document

Document history

<table>
<thead>
<tr>
<th>Version</th>
<th>Date</th>
<th>Author(s)</th>
<th>Reason</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.1</td>
<td>26-05-2015</td>
<td>G.C. Linders</td>
<td>Initial version.</td>
</tr>
<tr>
<td>0.2</td>
<td>01-06-2015</td>
<td>J.W.J.H. Visser</td>
<td>Chapter 2 draft.</td>
</tr>
<tr>
<td>1.0</td>
<td>22-06-2015</td>
<td>M.J.M. Muijsers</td>
<td>Final version.</td>
</tr>
</tbody>
</table>
Document Change Records

General

Date: 22nd June, 2015
Document title: Architectural Design Document
Document identifier: TravelMatch.Doc.ADD/1.0

Changes

<table>
<thead>
<tr>
<th>Section</th>
<th>Reason</th>
</tr>
</thead>
</table>


Chapter 1

Introduction

1.1 Purpose

The Architectural Design Document (ADD) describes the basic design of the software that will be made by the TravelMatch development team. This document describes the decomposition of the software into components. Then, for each component it describes the relation to external interfaces and the dependencies on other components, and which software requirements they fulfill. Additionally, an overview and the context of the system is given, and the document ends with an estimation of the feasibility and resource requirements.

1.2 Scope

TravelMatch is an application designed for smartphones and tablets, conceived by iLysian B.V. and developed by the TravelMatch development team. The purpose of the application is to assist users in planning a vacation by showing them images from various destinations and hotels or other places to stay. The application employs machine learning to build a profile of the user in order to suggest the ideal trip.

1.3 Definitions and abbreviations

1.3.1 Definitions

<table>
<thead>
<tr>
<th>Definition</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Affiliate Network</td>
<td>A network that enables you to receive money from customer redirection [18]</td>
</tr>
<tr>
<td>Analytics Data</td>
<td>The log of analytics events that is recorded and stored on the database.</td>
</tr>
<tr>
<td>Android</td>
<td>A popular open-source operating system for embedded devices, including smartphones and tablets, created by Google.</td>
</tr>
<tr>
<td>Angular JS</td>
<td>An open-source web application framework maintained by Google.</td>
</tr>
<tr>
<td>Cosine similarity</td>
<td>A measure of similarity between two vectors of an inner product space that measures the cosine of the angle between them.</td>
</tr>
<tr>
<td>Destination advice</td>
<td>The city, and selection of hotels, that is advised to a user after performing one or more interest analyses.</td>
</tr>
<tr>
<td>Destination attributes or tags</td>
<td>Each destination will have one or more destination attributes with an associated numerical relative value, those attributes cover the same preferences as the DNA attribute.</td>
</tr>
<tr>
<td>DNA attribute or tags</td>
<td>These are the attributes that the client wants to use to compose the DNA of. In the beginning 10 attributes are chosen and each image shall have a relative numerical value on one or more of the attributes. Attributes can be added or removed later for new and existing images and DNA.</td>
</tr>
<tr>
<td>Google Play Store</td>
<td>A public repository of free and paid apps for Android, managed by Google.</td>
</tr>
<tr>
<td>Guest user</td>
<td>An user that does not provide login details but still uses the TravelMatch app.</td>
</tr>
<tr>
<td>Term</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------------</td>
<td>----------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td><strong>Hotelstars rating</strong></td>
<td>A hotel classification with common criteria and procedures in participating countries to rate a hotel’s quality. See [21].</td>
</tr>
<tr>
<td><strong>iLysian</strong></td>
<td>Short for iLysian B.V., a software engineering company situated in Eindhoven, Netherlands. The client for the TravelMatch project.</td>
</tr>
<tr>
<td><strong>Interest analysis</strong></td>
<td>The action the user will do of judging the images.</td>
</tr>
<tr>
<td><strong>iOS</strong></td>
<td>A popular closed-source operating system for smartphones and tablets created by Apple.</td>
</tr>
<tr>
<td><strong>iOS App Store</strong></td>
<td>A public repository of free and paid apps for iOS, managed by Apple.</td>
</tr>
<tr>
<td><strong>JWT</strong></td>
<td>JSON Web Token: a compact URL-safe means of representing claims to be transferred between two parties, and used in TravelMatch as authentication token, since it is self-validating.</td>
</tr>
<tr>
<td><strong>Relational database management system (RDBMS)</strong></td>
<td>A database management system (a piece of computer software that interacts with users, other applications and a database to capture and analyze data) based on the relational model (commonly based on the relational database model)</td>
</tr>
<tr>
<td><strong>TCP/IP</strong></td>
<td>A computer networking model and set of communication protocols used on the internet and similar computer networks, including the Transmission Control Protocol (TCP) and the Internet Protocol (IP)</td>
</tr>
<tr>
<td><strong>Tinder</strong></td>
<td>A popular dating application for smartphones and tablets featuring a swipe based interface, where a swipe to the left indicates a dislike and a swipe to the right indicates a like.</td>
</tr>
<tr>
<td><strong>Travel DNA</strong></td>
<td>A collection of information about vacation preferences of a specific user or, more specifically, one vacation of that user. This information is stored on the server in a table with values representing the respective gain per attribute for each image the user has swiped.</td>
</tr>
<tr>
<td><strong>TravelMatch</strong></td>
<td>An application for smartphones and tablets that assists users in planning a vacation. The subject of this project.</td>
</tr>
<tr>
<td><strong>TravelMatch team</strong></td>
<td>A team of Computer Science students at Eindhoven University of Technology who will design and implement the TravelMatch application.</td>
</tr>
<tr>
<td><strong>User</strong></td>
<td>The user of the app.</td>
</tr>
<tr>
<td><strong>Waverunner</strong></td>
<td>Waverunner Search Service by Pyton Communication Services; a search service that provides vacation offers and prices of participating travel agencies.</td>
</tr>
</tbody>
</table>

### 1.3.2 Abbreviations

<table>
<thead>
<tr>
<th>Term</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AI</td>
<td>Artificial Intelligence</td>
</tr>
<tr>
<td>APK</td>
<td>Android Application Package</td>
</tr>
<tr>
<td>App</td>
<td>Application for smartphones and tablets</td>
</tr>
<tr>
<td>CMS</td>
<td>Content Management System</td>
</tr>
<tr>
<td>ESA</td>
<td>European Space Agency</td>
</tr>
<tr>
<td>IPA</td>
<td>iOS App Store Package</td>
</tr>
<tr>
<td>MVC</td>
<td>Model View Controller</td>
</tr>
<tr>
<td>OS</td>
<td>Operating System</td>
</tr>
<tr>
<td>RDBMS</td>
<td>Relational Database Management System</td>
</tr>
<tr>
<td>SRD</td>
<td>Software Requirements Document</td>
</tr>
<tr>
<td>TU/e</td>
<td>Eindhoven University of Technology</td>
</tr>
<tr>
<td>URD</td>
<td>User Requirements Document</td>
</tr>
</tbody>
</table>
1.4 References

[1] ESA PSS-05-0 Issue 2, Software requirements and architecture engineering process, February 1991
1.5 Overview

The remainder of this document is structured as follows. In chapter 2 of this document we give a
general overview of the system, including the design decisions and their motivation. Then, in chapter
3 of this document we describe the system context, showing the relation to each external system.
Chapter 4 contains the system design, listing the design methods as well as the system’s components.
The descriptions of these components will be treated in detail in chapter 5. Next, chapter 6 covers the
feasibility and resource estimates. Lastly, we provide a requirements traceability matrix in chapter 7.
Chapter 2
System overview

2.1 Background

The TravelMatch app is related to the project Tinder, which is a dating app that pairs users together if they indicate that they “like” each other’s pictures. Tinder uses a swiping interface where a swipe to the right means a “like” and a swipe to the left means a “dislike”. The interface of the TravelMatch app was directly inspired by this swiping interface featured in the Tinder app. Besides the Tinder app, there are no other projects related to TravelMatch.

2.2 Basic Design and Context

The TravelMatch app runs locally as an application on the phone of the user. The application runs on a browser based web framework and has animations to make it feel responsive to the user. For user accounts, the application connects to the TravelMatch server and potentially also to the Facebook server or Mailgun if the user decides to login using Facebook or e-mail. Furthermore, analytics information will be sent to the servers of Google during the runtime of the app.

The content information inside the app is retrieved from the TravelMatch server, and the app sends user-specific information back to the TravelMatch server, which stores it. The TravelMatch server will retrieve affiliate feeds from the respective affiliate servers. Currently, only TradeTracker is used, in the future the client will make his own affiliate server to connect with travel agencies. An overview of the context of TravelMatch can be found in figure 2.1. Clouds are an indication of components which are connected to each other over the internet.

Figure 2.1: Overview of the TravelMatch context.

2.3 Design Decisions

2.3.1 Architecture

The components of the TravelMatch project follow the Model-View-Controller architecture. This architecture is used in the TravelMatch project because it has a modular design with low coupling
and high cohesion. The MVC architecture increases the re-usability and improves the readability of the code. There are three major components in the MVC architecture that all have their own responsibility:

- **Model**: The model manages the data and behaviour of the application. The state of the application is stored inside the model as well.

- **View**: The view renders the user interface of the application. When the user interacts with the view, it fires events that are sent to the controller. The view gets updated by both the model (state changes) and the controller (events).

- **Controller**: The controller is the decision maker of the MVC. It reacts to events generated by the view and updates the system by notifying the model.

The system of TravelMatch consists of not one, but two different MVCs. The former one resides in the front end of the system while the latter one resides in the back end of the system. This can be seen in figure 2.2, where an overview is shown of the whole architecture of the system. Information about the front end and back end architecture of the system will be discussed in the next two chapters, including the reasoning behind these architectural decisions.

![Figure 2.2: The TravelMatch architecture consisting of two Model-View-Controllers.](image)

### 2.3.2 Back end

The server of TravelMatch is driven by Django [24], which is a free and open-sourced Python Web framework focused on scalability, speed and security. The main reason for selecting Django comes from the usage of Python, which is a relatively easy to learn language and can be quite powerful. This is useful considering the fact that the project group did not have much experience with different programming languages. Furthermore, there are a range of scientific libraries written in Python, which is convenient for the interest analysis and machine learning part of the project. Another reason for selecting Django was because it has a very useful Content Management System built-in. There was no need to pick a third-party CMS saving more time for the rest of the project.

Likewise to AngularJS, Django also uses the Model-View-Controller architecture, albeit in a different form. To start off, the model consists of the database which is used for the storage and retrieval of data,
such as user data. Meanwhile, the controller module consists of all Django controller modules used in the project, including the API controller, affiliate controller and the CMS controller. Finally, the view consists of the view of the built-in CMS. In there, the administrators can manage TravelMatch’s content via a web interface. All these elements make the design modular, which improves the maintainability of the system.

**Version of Django**

Django is an open-source project that is under continuous development. Proper functioning of TravelMatch is only guaranteed when installed on a server running an appropriate version of Django. TravelMatch is developed using version 1.8.0 of the Django framework.

**Models**

Models describe the logical grouping of data and functionality, like classes in the object-oriented paradigm. Django uses the word model where object-oriented paradigm uses the word class. Objects are called model instances in Django. Django models are compatible with the object-oriented paradigm. Therefore, the TravelMatch models include all attributes as described in the class diagram of the SRD. In addition, implementation-specific attributes are added, as described in the DDD [5].

**Templates**

Templates define the looks of the user interface of a Django application. Templates consist of HTML code with special tags, which are replaced with information from specific model instances. When the template engine, an essential component of the Django MTV system, renders a template, it processes the template file and replaces all tags with the appropriate information.

**Views**

Views define the structure of a Django application. They group units of user functionality together and they describe the possible user interactions with the models.

**Mapping of URIs to views**

Each HTTP request contains at least a Uniform Resource Identifier (URI). Django can map URIs to specific views (i.e. user functionality). It uses regular expression pattern matching to map a URI to the appropriate view.

**2.3.3 Persistent storage**

To enable persistent storage of data, Django ships with an Object-Relational Mapper (ORM). The ORM transparently keeps a persistent storage of all model instances created and modified in the Django application. The actual storage is done through a MySQL database. For the programming perspective, the ORM translates between the object-oriented way of accessing data and the queries required to retrieve the correct data from the database. Figure 2.3 illustrates the position of the ORM in the architecture of an object-oriented application that uses a database for persistent storage.
2.3.4 MySQL database

MySQL is a widely used RDBMS. It was chosen because it runs as a separate service and is connected to via TCP/IP: which means that it can be run physically separate from the normal TravelMatch server, making it easier to migrate parts of the back end to different physical machines. Additionally, it allows multiple concurrent connections with access, meaning that future services analyzing the data in the database can run simultaneously with the existing TravelMatch server Django instance. Lastly, there are various user interfaces available for directly monitoring or modifying the database content, such as the MySQL Workbench developed by Oracle, or PHPMyAdmin.

2.3.5 Front-end

The front end is driven by the Ionic framework [20]. The Ionic framework is an open-sourced SDK which utilizes AngularJS [27] in order to create rich and robust applications. This framework has been chosen after careful considerations because it compiles both to iOS and Android, because it is efficient and because it is free. Everything in the front end part of figure 2.2 is devised from AngularJS only.

AngularJS uses scopes as the model to store and retrieve the state of the user. The view is the interface which is shown to the user, and the controller updates the state and can update the view directly in case of e.g. scrolling. A special part of the controller is the communication to the services of AngularJS. In fact, these are used for calls to the server of TravelMatch; the API calls are made using these services. The architecture of the client side of the application follows the model-view-controller (MVC) architectural design pattern. The MVC pattern divides the application into three different parts: the model, the view and the controller. AngularJS is a structural framework for dynamic web apps. It lets you use HTML as your template language and lets you extend HTML’s syntax to express your application’s components clearly and succinctly. AngularJS brings all of the services that are often on the server side of the MVC pattern to the client side, thus the whole MVC pattern is implemented on the client side. This reduces the burden on the backend. The implementation of the MVC pattern in AngularJS has several key features that are discussed in the following sections.
Model
In AngularJS, a model is simply a JavaScript object that provides an internal representation of the outside world. All application logic is contained in the model. An overview of the models contained in the application can be found in Section 4.2.2.

View
The view is a representation of the model that the user can see and interact with. The view does not contain any functional behavior. An overview of the views contained in the application can be found in Section 4.2.4.

Controller
The controller is responsible for the construction of the model and connects the model to one or more views. The scope sits between the controller and the view. In AngularJS, the scope, which is provided by Angular, is an object that refers to the application model. Its most important job is to detect changes to model objects and propagate such changes throughout the system. Thus, it facilitates the communication between the model and the views and forwards events to the controller. Controllers should be straightforward and simply provide an interface for a view to manipulate the model and retrieve the models content for the view. An overview of the controllers contained in the application can be found in Section 4.2.3.

Ionic
Ionic is a mobile development framework and SDK, that allows you to build hybrid apps with web technologies such as HTML and JavaScript. Ionic provides support for various native components, and uses Cordova to embed the created web app into a platform-specific app, including the Android and iOS platforms.

2.3.6 JWT
For authentication from the client to the server, TravelMatch uses JWT: JSON Web Tokens, which are self-validating tokens; the claims in a JWT are encoded as a JSON object that is digitally signed using JSON Web Signature (JWS). This means that the client only has to store the JWT, and the server can validate it even without storing it, which makes it much easier to create and keep sessions.
Chapter 3
System context

This chapter describes the connections that TravelMatch has with external systems. TravelMatch must be able to operate properly without these connections. TravelMatch interfaces with Facebook, Mailgun, Google Analytics and TradeTracker. Each of these interfaces is defined in their respective following sections.

3.1 Facebook

The purpose of the Facebook interface is explained in section 2.5.2 of the SRD [3]. Facebook is a social network, that allows users to create a profile. Users can use this profile to authenticate themselves in applications other than Facebook.

3.1.1 Implementation of the interface

The procedure for registering or logging into an account via Facebook is described in section 2.7.2 of the SRD [3]. Facebook will provide the TravelMatch client with an ID and authentication token, that are then sent to the TravelMatch server, so that the server can validate these with the Facebook servers.

3.2 Mailgun

The purpose of the Mailgun interface is to send e-mails to users, specifically to verify their e-mail address. Mailgun is addressed through Django.

Sending the verification e-mail is a part of User Authentication, as described in section 3.1.2 of the SRD [3].

3.2.1 Implementation of the interface

Mailgun is called when a user registers in the TravelMatch app with their e-mail. For this, Mailgun requires the use of a specific URL and API key. For TravelMatch:

- the URL is https://api.mailgun.net/v3/gotravelmatch.com/messages
- the API key is key-07bd4dbc95e6f5d62192e1d4d6a7ace5

These are to be replaced when connecting a different Mailgun account (using the information provided by Mailgun).

The mailgun API requires the following data to send an e-mail:

- from A sender e-mail address (currently for TravelMatch: noreply@gotravelmatch.com)
- to A receiver e-mail address
- subject The subject line of the e-mail
- html The content of the e-mail in HTML format
- text The content of the e-mail in plaintext format
3.3 Google Analytics

The purpose of Google Analytics is to provide insight in how users use the app. Google Analytics comes as a plugin to the Angulartics module for AngularJS.

3.3.1 Implementation of the interface

For Google Analytics, a specific tracking ID can be created via the web interface offered at http://analytics.google.com. This tracking ID is sent to the Google Analytics servers whenever the TravelMatch application is started, through a standard code snippet provided by Google. Google Analytics will then track the user’s actions by hooking into Angulartics (no further action is taken by the TravelMatch system). The current temporary tracking ID for the TravelMatch app is UA-63205703-2, registered to a private Gmail. The ID must be replaced by one that is registered in Google Analytics as a Website, not a Mobile app. Google Analytics tracks various values, including, but not limited to:

- Live number of active users on the app
- User’s geographical location, language, country and city
- Whether sessions belong to new or returning users
- Average session duration
- Users flow (flow of use of the application)
- Statistics for screens in the application
- User’s browser, operating system, service provider

3.4 TradeTracker

The purpose of using the TradeTracker interface is to get trip data from advertisers, and make sure that TravelMatch is known as the referrer of those trips when they are booked. TravelMatch can use any data feed when a parser is written that converts the affiliate data to the TravelMatch data model. TradeTracker can provide a great number of API URLs, each for their own data feed. Currently, TravelMatch implements a parser for the Arke and De Jong Intra feeds that TradeTracker provides. The API URLs for these feeds are saved in the database and can be added via the CMS. An example API URL for the Arke feed of Spain is is http://pf.tradetracker.net/?aid=218916&encoding=utf-8&type=xml-v2&fid=472017&categoryType=2 &additionalType=2.

3.4.1 Implementation of the interface

The TradeTracker API will respond by providing an XML file (which differs per feed). For the implemented Arke feed this XML is parsed into the TravelMatch data model by using the following properties available in the XML file (original property ⇒ TravelMatch property):

- name ⇒ name
- description ⇒ description
- city ⇒ city
- region ⇒ region
- country ⇒ country
- stars ⇒ hotel_stars
For the implemented De Jong Intra feed this XML is parsed into the TravelMatch data model by using the following properties available in the XML file (original property ⇒ TravelMatch property):

• name ⇒ name
• description ⇒ description
• city ⇒ city
• (This feed provides no information suitable for region)
• country ⇒ country
• stars ⇒ hotel_stars
• price ⇒ price
• URL ⇒ link
• images/image ⇒ image
• persons_minimum ⇒ min_nr_people
• departureDate ⇒ departure_date
• duration ⇒ duration
• true ⇒ with_flight (always true for De Jong Intra feed)
• (This feed provides no information suitable for user_rating)
Chapter 4

System design

This chapter describes the technical aspects of the design of TravelMatch.

4.1 Design methods

The TravelMatch server is implemented as a web server based on the Django framework. The framework supplies many useful components for the development of web servers. The structure and terminology of the framework determine the structure and terminology used in the description of the TravelMatch server. The TravelMatch client is implemented as a web application based on the Ionic framework, which allows the application to use AngularJS, and which generates binaries for the application for iOS and Android.

Section 4.2 defines the components of TravelMatch and their dependencies.

4.2 Decomposition description

The decomposition of TravelMatch into components is based on the requirements of the URD [2] and the SRD [3]. Section 4.2.1 lists the components in the TravelMatch server, whereafter section 4.2.2 lists the components in the TravelMatch client, which are all further described in chapter 5.

4.2.1 TravelMatch server components

On the server-side of TravelMatch, we can identify the following components:

- **Affiliate** The Affiliate component pulls data from an affiliate feed and translates it into the TravelMatch data model. It contains functionality to add, modify and remove feeds and parsers for the feeds.

- **AI** The AI component contains the implementation of the decision making algorithms in TravelMatch. The AI component can calculate which images should be presented next in interest analysis, and give a holiday recommendation based on Travel DNA.

- **Authentication token verification** The Authentication token verification component provides functionality to verify the authenticity of JSON Web Tokens.

- **CMS** The CMS component contains the hooks of the Content Management System, so that it can trigger the affiliate parser, and allow administrators to change data via a user interface.

- **Database** The Database component contains the TravelMatch data model, and stores all data according to it.

- **Facebook authentication** The Facebook authentication component checks with the Facebook servers whether the Facebook token provided by the client is valid.

- **Interest analysis API** The Interest analysis API component holds the API functions for the interest analysis swiping. It allows functionality to start, query and update vacations, get new images for interest analysis and record likes and dislikes.
- **Recommendation API** The Recommendation API component holds the API functions for the holiday recommendations: getting a recommendation, getting a location’s trips, and saving, loading and deleting a location overview.

- **Registration / login API** The Registration / login API component holds the API functions for user registration and login. This includes registering and logging in via e-mail or Facebook, activating an e-mail account, delete accounts, querying and updating user details and creating and deleting guest accounts.

![Figure 4.1: Overview of the dependencies between server components.](image)

### 4.2.2 TravelMatch client components

On the client-side of TravelMatch, we can identify the following components:

- **about** The about module consists of the model, view, controller and service for the about screen, handling and showing everything available in the screen. It displays information about the app’s licenses and the company that made it.

- **details** The details module consists of the model, view, controller and service for the vacation details screen. This component allows users to make and modify vacation details.

- **front** The front module consists of the model, view, controller and service for the front screen. This component allows a guest user to choose to use the TravelMatch app with or without an authentication method.

- **hotelOverview** The hotelOverview module consists of the model, view, controller and service for the hotel overview and hotel detail screen. This component allows users to view their recommendations and book a trip.

- **registration** The registration module consists of the model, view, controller and service for the registration of users. This component allows for the registering of an e-mail user and authentication of a Facebook user.

- **main** The main module consists of the authentication service, http injector and the constant values. It handles the authentication of a user.
• **language** The language module consists of the language files for each language one. It supplies language specific descriptions for language dependent items.

• **analytics** The analytics module consists of a service and provider for recording the analytics.

• **login** The login module consists of the model, view, controller and service of the login screen for all authentication providers. This component allows the user to log in with all authentication providers.

• **navigation** The navigation module consists of the models, views and controller of the header, sidebar and tabs of the login and registration screen. This component handles the navigation throughout the TravelMatch app, the views consists of the header, back button and sidebar visible throughout the app.

• **swipe** The swipe module consists of the models, views, controller and service of the interest analysis named after the associated swiping of images. This component handles the interest analysis. It will retrieve images for the user to judge.

• **user** The user module consists of the model, view, controller and service of the user detail screen. This component allows for user details to be made and changed for an authenticated user.

Figure 4.2: Overview of the dependencies between modules, where main, language and Analytics are three different modules.

**Dependencies between the TravelMatch client-side components**

All of the TravelMatch app components consist of at least one of model, view, controller and service, usually all four of them. In figure 4.3 the internal relation between those parts is shown, leaving out the http injector and constant as they are only indirectly connected to all Services, who use http calls. Also the language module is left out as it is also deployed everywhere there is a view, to change the view according to the language.
4.3 File structure

In this section, the file structures of the TravelMatch server and the TravelMatch client will be outlined. All folders names have / as suffix.

4.3.1 Server file structure

django-db/ ...........................................Django folder for data model conversion
__all.json .................................Django file for data model conversion
__all_unfiltered.json ....................Django file for data model conversion
django/ ...........................................Django configuration files
__mysql_personal_settings.cnf .........................MySQL settings
__requirements.v1.txt .................. List of dependencies and their versions
travelmatch/ .....................................TravelMatch source code
affiliate/ ......................................Affiliate networks code
  __tradetracker/ ...................................TradeTracker parsers
    __init__.py ...........................................
    __akte_parser.py ..............................Parser for Arke feeds
    __dji_parser.py ..............................Parker for De Jong Intra feeds
    __init__.py ...........................................
  __admin.py .......................................CMS feed parsing
  __models.py ...................................Object-relational model definitions
  __serializer.py ..............................REST framework serialization
  __tests.py ....................................Unit tests
  __views.py ......................................Django affiliate views (none)
ai/ ..............................................AI (recommendation engine)
  __init__.py ...........................................
  __admin.py .......................................CMS recommendations and interest analysis
  __entropy.py .....................................Entropy calculation for interest analysis
  __models.py ...................................Object-relational model definitions
  __recommender_system.py ......................Recommender system
  __serializers.py ..............................REST framework serialization (locations and trips)
  __tests.py ....................................Unit tests
  __views.py ......................................Django AI views (none)
appusers/ ........................................Users
  __email/ ..............................................E-mail users
    __email_already_activated.html .........Page for already activated e-mails
### 4.3.2 Client file structure

For the client file structure, we use * to denote that for every client-side module, the file structure includes the paths with * replaced by the module name.

```plaintext
app/ .........................................................Main app directory
  _fonts/ ..................................................Fonts
  _img/ ..................................................Images
  _index.html ............................................Main Ionic app entry point
  _lib/ ...................................................Libraries managed by Bower
  _scripts/ ..............................................App components
    _app.js ..............................................App configuration
    _app.module.js .....................................App module
    _app.router.js .....................................URL router
    */ ...................................................App components
      */.module.js ..................................Module specification
      */.directive.js .................................Module directives
      */.controller.js ...............................Module controllers
      */.html ...........................................Module templates
      */.service.js ....................................Module services
  _styles/ ................................................LESS stylesheets
  _bower.json ........................................List of app dependencies
  _config.xml ........................................Cordova configuration and plugin dependencies
  _coverage/ ..........................................Istanbul reports
  _docs/ ...............................................Documentation generated by ngdocs
  _gulpfile.js ........................................Gulp task configuration
  _hooks/ ...............................................Cordova lifecycle hooks
```
4.4 Database Schemas

As we are using MySQL for the RDBMS, we can identify the tables that are created in the database, and their respective schemas. Below we list these tables, in SQL notation, where adding `CREATE` in front of the tables would make a valid SQL statement to create a table with identical columns. The last few tables are used by Django to store data that describes the dataset and the operation of the Django server.

- **TABLE** `affiliate_abstractparsermodel` ( `parser_id` integer NOT NULL PRIMARY KEY AUTOINCREMENT);

- **TABLE** `affiliate_affiliatefeed` ( `id` integer NOT NULL PRIMARY KEY AUTOINCREMENT, `url` varchar(2048) NOT NULL, `parser_id` integer NOT NULL );

- **TABLE** `affiliate_arkeparsermodel` ( `abstractparsermodel_ptr_id` integer NOT NULL, PRIMARY KEY(`abstractparsermodel_ptr_id`) );

- **TABLE** `affiliate_djiparsermodel` ( `abstractparsermodel_ptr_id` integer NOT NULL, PRIMARY KEY(`abstractparsermodel_ptr_id`) );

- **TABLE** `affiliate_prijsvrijparsermodel` ( `abstractparsermodel_ptr_id` integer NOT NULL, PRIMARY KEY(`abstractparsermodel_ptr_id`) );

- **TABLE** `affiliate_trip` ( `id` integer NOT NULL PRIMARY KEY AUTOINCREMENT, `description` text NOT NULL, `city` text NOT NULL, `region` text, `country` text, `hotel_stars` integer, `price` real NOT NULL, `link` varchar(2048) NOT NULL, `image` varchar(2048) NOT NULL, `min_nr_people` integer, `departure_date` date NOT NULL, `duration` integer NOT NULL, `with_flight` text NOT NULL, `user_rating` real, `created_on` datetime NOT NULL, `name` varchar(64) NOT NULL );

- **TABLE** `affiliate_tripoffer` ( `offer_id` integer NOT NULL PRIMARY KEY AUTOINCREMENT, `name` varchar(64) NOT NULL, `description` text NOT NULL, `hotel_stars` integer, `price` real NOT NULL, `link` varchar(2048) NOT NULL, `image` varchar(2048) NOT NULL, `min_people` integer, `dept_date` date NOT NULL, `duration_days` integer NOT NULL, `with_flight` bool NOT NULL, `user_rating` real, `loc_id` integer NOT NULL, `priority` integer NOT NULL );
- **TABLE** `ai_imageblacklistitem` ( `id` integer NOT NULL PRIMARY KEY AUTOINCREMENT, `img_id` integer NOT NULL, `vac_id` integer NOT NULL, `created_on` datetime NOT NULL );

- **TABLE** `ai_imagedimension` ( `id` integer NOT NULL PRIMARY KEY AUTOINCREMENT, `width` integer NOT NULL, `height` integer NOT NULL );

- **TABLE** `ai_imagetag` ( `id` integer NOT NULL PRIMARY KEY AUTOINCREMENT, `img_id` integer NOT NULL, `tag_id` integer NOT NULL, `value` integer NOT NULL );

- **TABLE** `ai_location` ( `loc_id` integer NOT NULL PRIMARY KEY AUTOINCREMENT, `city_name` varchar(64) NOT NULL, `country_name` varchar(64) NOT NULL, `active` bool NOT NULL, `region_name` varchar(64) NOT NULL );

- **TABLE** `ai_locationblacklistitem` ( `id` integer NOT NULL PRIMARY KEY AUTOINCREMENT, `loc_id` integer NOT NULL, `vac_id` integer NOT NULL );

- **TABLE** `ai_locationtag` ( `id` integer NOT NULL PRIMARY KEY AUTOINCREMENT, `value` integer NOT NULL, `loc_id` integer NOT NULL, `tag_id` integer NOT NULL, `last_modified_by_id` integer, `initial_value` integer NOT NULL );

- **TABLE** `ai_swipeimage` ( `img_id` integer NOT NULL PRIMARY KEY AUTOINCREMENT, `created` datetime NOT NULL, `original_filename` varchar(100) NOT NULL, `active` bool NOT NULL, `uploaded_by_id` integer NOT NULL );

- **TABLE** `ai_tag` ( `tag_id` integer NOT NULL PRIMARY KEY AUTOINCREMENT, `name` varchar(256) NOT NULL, `created_on` datetime NOT NULL, `created_by_id` integer NOT NULL, `priority` integer NOT NULL, `active` bool NOT NULL );

- **TABLE** `ai_traveldNA` ( `id` integer NOT NULL PRIMARY KEY AUTOINCREMENT, `like` bool NOT NULL, `img_id` integer NOT NULL, `vacation_id` integer NOT NULL );

- **TABLE** `ai_vacationtag` ( `id` integer NOT NULL PRIMARY KEY AUTOINCREMENT, `sum_value` integer NOT NULL, `tag_id` integer NOT NULL, `vac_id` integer NOT NULL );

- **TABLE** `appusers_appuser` ( `user_id` integer NOT NULL PRIMARY KEY AUTOINCREMENT, `name` varchar(500) NOT NULL, `gender` varchar(20) NOT NULL, `birthday` date NOT NULL, `activation` bool NOT NULL );

- **TABLE** `appusers_fbappuser` ( `appuser_ptr_id` integer NOT NULL PRIMARY KEY(auto_increment), `fbid` varchar(256) NOT NULL, `password` varchar(256) NOT NULL, `email` varchar(256) NOT NULL, PRIMARY KEY(appuser_ptr_id) );

- **TABLE** `appusers_guestappuser` ( `appuser_ptr_id` integer NOT NULL UNIQUE, `timestamp` datetime NOT NULL, `device_id` varchar(250) NOT NULL, PRIMARY KEY(device_id) );

- **TABLE** `appusers_mailappuser` ( `appuser_ptr_id` integer NOT NULL, `password` varchar(256) NOT NULL, `email` varchar(256) NOT NULL, PRIMARY KEY(appuser_ptr_id) );

- **TABLE** `appusers_pendingactivation` ( `id` integer NOT NULL PRIMARY KEY AUTOINCREMENT, `timestamp` datetime NOT NULL, `key` varchar(255) NOT NULL, `user_id` integer NOT NULL );

- **TABLE** `appusers_savedlocation` ( `id` integer NOT NULL PRIMARY KEY AUTOINCREMENT, `loc_list_id` integer NOT NULL, `loc_id` integer NOT NULL, `user_id` integer NOT NULL );

- **TABLE** `appusers_triplist` ( `trip_list_id` integer NOT NULL PRIMARY KEY AUTOINCREMENT );
- **TABLE** `appusers_triplistentry` ('id' integer NOT NULL PRIMARY KEY AUTOINCREMENT, 'cached_name' varchar(64) NOT NULL, 'trip_list_id' integer NOT NULL, 'trip_offer_id' integer);

- **TABLE** `appusers_vacationdetail` ('internal_id' integer NOT NULL PRIMARY KEY AUTOINCREMENT, 'vac_id' mediumint unsigned NOT NULL, 'vac_name' varchar(256) NOT NULL, 'start_date' date NOT NULL, 'start_dateExtend' mediumint NOT NULL, 'end_date' date NOT NULL, 'end_dateExtend' mediumint NOT NULL, 'persons' mediumint unsigned NOT NULL, 'personsChildren' mediumint unsigned NOT NULL, 'budget' integer unsigned NOT NULL, 'user_id' integer NOT NULL, 'lastModified' datetime NOT NULL, 'start_date' date NOT NULL);

- **TABLE** `auth_group` ('id' integer NOT NULL PRIMARY KEY AUTOINCREMENT, 'name' varchar(80) NOT NULL UNIQUE);

- **TABLE** `auth_group_permissions` ('id' integer NOT NULL PRIMARY KEY AUTOINCREMENT, 'group_id' integer NOT NULL, 'permission_id' integer NOT NULL);

- **TABLE** `auth_permission` ('id' integer NOT NULL PRIMARY KEY AUTOINCREMENT, 'content_type_id' integer NOT NULL, 'codename' varchar(100) NOT NULL, 'name' varchar(255) NOT NULL);

- **TABLE** `auth_user` ('id' integer NOT NULL PRIMARY KEY AUTOINCREMENT, 'password' varchar(128) NOT NULL, 'is_superuser' bool NOT NULL, 'username' varchar(30) NOT NULL UNIQUE, 'first_name' varchar(30) NOT NULL, 'last_name' varchar(30) NOT NULL, 'email' varchar(254) NOT NULL, 'is_staff' bool NOT NULL, 'is_active' bool NOT NULL, 'dateJoined' datetime NOT NULL, 'lastLogin' datetime);

- **TABLE** `auth_user_groups` ('id' integer NOT NULL PRIMARY KEY AUTOINCREMENT, 'user_id' integer NOT NULL, 'group_id' integer NOT NULL);

- **TABLE** `auth_user_user_permissions` ('id' integer NOT NULL PRIMARY KEY AUTOINCREMENT, 'user_id' integer NOT NULL, 'permission_id' integer NOT NULL);

- **TABLE** `authtoken_token` ('key' varchar(40) NOT NULL, 'created' datetime NOT NULL, 'user_id' integer NOT NULL UNIQUE, PRIMARY KEY(('key')));

- **TABLE** `corsheaders_cormodel` ('id' integer NOT NULL PRIMARY KEY AUTOINCREMENT, 'cors' varchar(255) NOT NULL);

- **TABLE** `django_admin_log` ('id' integer NOT NULL PRIMARY KEY AUTOINCREMENT, 'action_time' datetime NOT NULL, 'object_id' text, 'object_repr' varchar(200) NOT NULL, 'action_flag' smallint unsigned NOT NULL, 'change_message' text NOT NULL, 'content_type_id' integer, 'user_id' integer NOT NULL);

- **TABLE** `django_content_type` ('id' integer NOT NULL PRIMARY KEY AUTOINCREMENT, 'app_label' varchar(100) NOT NULL, 'model' varchar(100) NOT NULL);

- **TABLE** `django_migrations` ('id' integer NOT NULL PRIMARY KEY AUTOINCREMENT, 'app' varchar(255) NOT NULL, 'name' varchar(255) NOT NULL, 'applied' datetime NOT NULL);

- **TABLE** `django_session` ('session_key' varchar(40) NOT NULL, 'session_data' text NOT NULL, 'expire_date' datetime NOT NULL, PRIMARY KEY(('session_key')));

This table is based by Django on the entity-relational model in figure 4.4 that can also be found in chapter 2.7.3 of the SRD [3].

For readability purposes, we also present an overview of the tables in relational notation, in which the primary key is underlined:
affiliate_abstractparsermodel(parser_id)
affiliate_affiliatefeed(id, url, parser_id)
affiliate_arkeparsermodel(abstractparsermodel_ptr_id)
affiliate_djiparsermodel(abstractparsermodel_ptr_id)
affiliate_prijsvrijparsermodel(abstractparsermodel_ptr_id)
affiliate_trip(id, description, city, region, hotel_stars, price, link, image, min_nr_people, departure_date, duration, with_flight, user_rating, create_on, name)
affiliate_tripoffer(offer_id, name, description, hotel_stars, price, link, image, min_people, dept_date, duration_days, with_flight, user_rating, loc_id, priority)
ai_imageblacklistitem(id, img_id, vac_id, created_on)
ai_imagedimension(id, width, height)
ai_imagetag(id, img_id, tag_id, value)
ai_location(loc_id, city_name, country_name, active, region_name)
ai_locationblacklistitem(id, loc_id, vac_id)
ai_locationtag(id, value, loc_id, tag_id, last_modified_by_id, initial_value)
ai_swipeimage(img_id, created, original_filename, active, uploaded_by_id)
ai_tag(tag_id, name, created_on, created_by_id, priority, active)
ai_traveldna(id, like, img_id, vacation_id)
ai_vacationtag(id, sum_value, tag_id, vac_id)
appusers_appuser(user_id, name, gender, birthday, activation)
appusers_fbappuser(appuser_ptr_id, fbid, appuser_ptr_id)
appusers_guestappuser(appuser_ptr_id, timestamp, datetime, device_id)
appusers_mailappuser(appuser_ptr_id, password, email)
appusers_pendingactivation(id, timestamp, key, user_id)
appusers_savedlocation(id, loc_list_id, loc_id, user_id)
appusers_triplist(trip_list_id)
appusers_triplistentry(id, cached_name, trip_list_id, trip_offer_id)
appusers_vacationdetail(internal_id, smallintunsigned, vac_name, start_date, end_date, persons_adults, persons_children, budget, user_id, last_modified, start_date)
auth_group(id, name)
auth_group_permissions(id, group_id, permission_id)
auth_permission(id, content_type_id, codename, name)
auth_user(id, password, is_superuser, username, first_name, last_name, email, is_staff, is_active, date_joined, last_login)
auth_user_groups(id, user_id, group_id)
auth_user_user_permissions(id, user_id, permission_id)
authtoken_token(key, created, user_id)
corsheaders_corsmodel(id, cors)
django_admin_log(id, action_time, object_id, object_repr, action_flag, change_message, content_type_id, user_id)
django_content_type(id, app_label, model)
django_migrations(id, app, name, applied)
django_session(session_key, session_data, expire_date)
Chapter 5
Components descriptions

In this chapter, we describe every component of the TravelMatch client and server in detail.

5.1 Server-side components

5.1.1 Affiliate

Type
Server component

Purpose
SR210, SR211, SR212, SR213, SR214, SR215, SR216, SR217

Function
The Affiliate component pulls data from an affiliate feed and translates it into the TravelMatch data model.

Subordinates
none

Dependencies
This component depends on the following TravelMatch components:

- **Database**: For retrieving and storing all needed and resulting data.

Interfaces
Note: the Affiliate component provides abstract interfaces for affiliate parsers and trip attributes.

Add an affiliate parser that parses data from an affiliate feed

- Input: Affiliate parser
- Output: none
- Condition: The affiliate parser is an implementation of the relevant interface

Designate an affiliate feed to an affiliate parser

- Input: Affiliate parser, affiliate feed
- Output: none
- Condition: The affiliate parser is an implementation of the relevant interface

Query current affiliate parsers
• Input: none
• Output: List of affiliate parsers
• Condition: none

Query current affiliate parsers’ feeds
• Input: Affiliate parser
• Output: List of affiliate feeds
• Condition: The affiliate parser is an implementation of the relevant interface

Remove an affiliate parser
• Input: Affiliate parser
• Output: none
• Condition: The affiliate parser is present in the current list of parsers

Remove an affiliate feed from an affiliate parser
• Input: Affiliate parser, affiliate feed
• Output: none
• Condition: The affiliate parser is an implementation of the relevant interface

Retrieve data from all feeds of a parser, parse it and add it to the database
• Input: Affiliate parser
• Output: none
• Condition: The affiliate parser is an implementation of the relevant interface

Resources
An affiliate feed URL must be accessible when its data is being retrieved.

References
The precise relation to affiliate networks is described in section 2.5.1 of the SRD. A class diagram of the affiliate networks component in described in section 2.7.4 of the SRD. The interaction between the affiliate networks and other components is described in section 2.7.5 of the SRD.

Processing
An affiliate parser interface exists, that is implemented through a number of concrete parsers, and that has at least a number of affiliate feeds associated with it. A trip attribute interface exists, that is implemented through a number of concrete trip attributes. A central subcomponent handles the adding, querying and removing of parsers.

Data
A central subcomponent stores all parsers, and every parser keeps track of its associated affiliate feeds.
5.1.2 AI

Type
Server component

Purpose

Function
The AI component contains the implementation of the decision making algorithms in TravelMatch.

Subordinates
none

Dependencies
This component depends on the following TravelMatch components:

- **Database**: For retrieving and storing all needed and resulting data.

Interfaces

**Rank locations in similarity to Travel DNA**
- Input: Travel DNA (as array of choices made by the user, with tag values), array of locations (each with tag values)
- Output: True or false (whether the authentication token is valid)
- Condition: Travel DNA and locations are vectors in a positive space based on their tag values

**Determine next interest analysis images**
- Input: Vacation ID, number of new images requested (as integer)
- Output: List of image references (as URLs)
- Condition: none

Resources
none

References
The general capabilities of the AI module are described in section 2.2.2 of the URD.

Processing
The AI component does not keep track of data between function calls, but instead executes a specific algorithm every time a function is called.

Data
none
5.1.3 Authentication token verification

Type
Server component

Purpose
SR48, SR49, SR200

Function
The Authentication token verification component provides functionality to verify the authenticity of JSON Web Tokens.

Subordinates
none

Dependencies
This component does not depend on any TravelMatch components.

Interfaces
Check authentication token validity
- Input: Authentication token (as a JSON Web Token that is self-authenticating)
- Output: True or false (whether the authentication token is valid)
- Condition: none

Resources
none

References
none

Processing
This component serves as a wrapper for using JSON Web Tokens and as such its processing consists purely of verifying JSON Web Tokens through the JWT library and returning the result.

Data
none
5.1.4 CMS

Type
Server component

Purpose

Function
The CMS component contains the hooks of the Content Management System, so that it can trigger the affiliate parser.

Subordinates
none

Dependencies
This component depends on the following TravelMatch components:

- Database: For retrieving and storing all needed and resulting data.
- Affiliate: For triggering the retrieval and parsing of data from affiliate feeds.

Interfaces
The CMS provides a user interface to modify any data in the database, following the TravelMatch data model. The user interface allows additions, modifications and deletions of properties such that the data stays consistent. The input of these interface functions is always the new or replacement data, and the output is either success or a detailed failure (in case some of the input was not valid). The data that can be manipulated through the CMS is, arranged by their listing in the CMS user interface:

- Affiliate
  - Affiliate feeds
  - Trip offers
  - Trips

- AI
  - Image dimensions
  - Location tags
  - Locations
  - Swipe images
  - Tags
  - Travel DNAs

App users
- Facebook app users
- Guest app users
- E-mail app users
- Pending activations
- Vacation details

**Authentication and Authorization**

- Groups
- Users

**Authentication token**

- Tokens

Additionally, the CMS provides an interface to retrieve data from an affiliate feed, parse the result, and add it to the database. Also, the CMS can be logged into, so the CMS provides the following functions as well (which is implemented by Django):

**Log in an administrator**

- Input: Username, password
- Output: The CMS landing page or a detailed error message in the login page (if the condition does not hold)
- Condition: Username and password are valid and match

**Register an administrator**

- Input: Username, password
- Output: Success or failure (in case condition does not hold)
- Condition: Username and password are valid

**Log out an administrator**

- Input: none
- Output: CMS login page or failure (in case condition does not hold)
- Condition: Some administrator is logged in

**Remove an administrator**

- Input: Username
- Output: Success or failure (in case condition does not hold)
- Condition: The actor has sufficient permissions, and an administrator with the given username exists

**Resources**

none
References
The users of the CMS are described in section 2.4.2 of the URD. The interface of the CMS is described in section A.2 of the SRD.

Processing
The processing for this component is done automatically by Django.

Data
none
5.1.5 Database

Type
Server component: database

Purpose
SR222, SR264, SR272, SR273, SR274, SR275, SR276, SR277, SR279, SR280

Function
The Database component contains the TravelMatch data model.

Subordinates
none

Dependencies
This component does not depend on any TravelMatch components.

Interfaces
The Database component allows other components to look up and manipulate all data in the database, following the TravelMatch data model used in Django. The input of the functionality of this interface is always either a query or update request, and the output is a result in terms of data, or a success or failure signal. The condition for using the functions is that the updates and requests adhere to the TravelMatch data model, but this is always guaranteed by the way that Django updates class instances, based on the data model, by querying the RDBMS correctly.

Resources
The RDBMS that Django is configured to use must be available.

References
The data model of the database is described in section 2.7.3 of the SRD.

Processing
The Database component is mostly implemented by Django, and uses the TravelMatch data model to correctly interface with the RDBMS.

Data
The Database component implicitly contains the Travelmatch data model.
5.1.6 Facebook authentication

Type
Server component

Purpose
SR20, SR21, SR22

Function
The Facebook authentication component checks with the Facebook servers whether the authentication token provided by the client is valid.

Subordinates
none

Dependencies
This component does not depend on any TravelMatch components.

Interfaces
Check Facebook token validity
- Input: Facebook token
- Output: True or false (whether the Facebook token is valid)
- Condition: none

Resources
A connection to the Facebook servers

References
The use case regarding Facebook authentication is described in section A.1.3 of the URD. The relation of TravelMatch to Facebook is described in section 2.5.2 of the SRD. The interaction between Facebook and other components is described in section 2.7.5 of the SRD.

Processing
This component serves as a wrapper for Facebook’s API and as such it only redirects input to the Facebook servers and output to the caller of the functionality.

Data
none
5.1.7 Interest analysis API

Type
Server component: API

Purpose

Function
The Interest analysis API component holds the API functions for the interest analysis swiping.

Subordinates
none

Dependencies
This component depends on the following TravelMatch components:
- Database: For retrieving and storing all needed and resulting data.
- AI: The AI component contains the implementation for the interest analysis method and is used for that purpose.
- Authentication token verification: For authenticating the user making API requests.

Interfaces
Note: the vacation data consists of a start date and end date and the offsets of these days in integers, a budget per person and the number of people (with distinction between adults and children).

Create a vacation
- Input: Authentication token, vacation data
- Output: New vacation ID, or detailed failure (in case condition does not hold)
- Condition: The authentication token and vacation data are valid

Get vacation details
- Input: Authentication token, vacation ID
- Output: Vacation ID, or detailed failure (in case condition does not hold)
- Condition: The authentication token and vacation ID are valid

Modify vacation details
- Input: Authentication token, vacation ID, vacation data
- Output: Success, or detailed failure (in case condition does not hold)
• Condition: The authentication token, vacation ID and vacation data are valid

Get next interest analysis images
• Input: Authentication token, vacation ID, number of new images requested (as integer)
• Output: List of image references (as URLs), or detailed failure (in case condition does not hold)
• Condition: The authentication token, vacation ID and number of new images requested are valid

Record a like or dislike
• Input: Authentication token, vacation ID, image ID, like/dislike (as boolean)
• Output: Image reference, or detailed failure (in case condition does not hold)
• Condition: The authentication token, vacation ID and image ID are valid

Resources
none

References
The general capabilities of the interest analysis are described in section 2.2.1 of the URD. Use cases regarding the interest analysis are described in section A.2 of the URD. The interaction between the interest analysis API and other components is described in section 2.7.5 of the SRD.

Processing
This API component does not do processing apart from calling the other components that can retrieve the requested data: namely the AI component and the Database component, and then checking the data’s validity. In addition, the Authentication token verification component is used to validate authentication tokens.

Data
none
### 5.1.8 Recommendation API

#### Type
Server component: API

#### Purpose


#### Function

The Recommendation API component holds the API functions for the holiday recommendations.

#### Subordinates

none

#### Dependencies

This component depends on the following TravelMatch components:

- **Database**: For retrieving and storing all needed and resulting data.
- **AI**: The AI component contains the implementation for the holiday recommendation method and is used for that purpose.
- **Authentication token verification**: For authenticating the user making API requests.

#### Interfaces

- **Get a recommendation in the form of locations and trips**
  - **Input**: Authentication token, vacation ID
  - **Output**: Locations and trips, or detailed failure (in case condition does not hold)
  - **Condition**: Authentication token and vacation ID are valid

- **Get trips for a location**
  - **Input**: Authentication token, location ID
  - **Output**: Trips, or detailed failure (in case condition does not hold)
  - **Condition**: Authentication token and location ID are valid

- **Save a location overview**
  - **Input**: Authentication token, location ID, array of trip IDs
  - **Output**: Success, or detailed failure (in case condition does not hold)
  - **Condition**: Authentication token, location ID and trip IDs are valid

- **Load a location overview**
  - **Input**: Authentication token, location ID


TravelMatch Architectural Design Document

- Output: Trips, or detailed failure (in case condition does not hold)
- Condition: Authentication token and location ID are valid

Delete a recommended location

- Input: Authentication token, location ID
- Output: Success, or detailed failure (in case condition does not hold)
- Condition: Authentication token and location ID are valid

Resources

none

References

The general capabilities required of the recommendations API are described in section 2.2.3 of the URD. Use cases regarding recommendations are described in section A.2 of the URD. The interaction between the recommendation API and other components is described in section 2.7.5 of the SRD.

Processing

This API component does not do processing apart from calling the other components that can retrieve the requested data: namely the AI component and the Database component, and then checking the data’s validity. In addition, the Authentication token verification component is used to validate authentication tokens.

Data

none
5.1.9 Registration / login API

Type
Server component: API

Purpose

Function
The Registration / login API component holds the API functions for user registration and login.

Subordinates
none

Dependencies
This component depends on the following TravelMatch components:
- **Database**: For retrieving and storing all needed and resulting data.
- **Facebook authentication**: The Facebook authentication component is called to query whether Facebook tokens are valid.
- **Authentication token verification**: For authenticating the user making API requests.

Interfaces
Register a user via e-mail
- Input: E-mail, password
- Output: Success, or detailed failure (in case condition does not hold)
- Condition: The e-mail is a valid e-mail address and there is no user with the e-mail yet

Activate a user’s account if using e-mail
- Input: Activation key
- Output: Success, or detailed failure (in case condition does not hold)
- Condition: The activation key exists

Log in a user via e-mail
- Input: E-mail, password
- Output: Authentication token, or detailed failure (in case condition does not hold)
- Condition: There exists an user with given e-mail and password

Register or log in a user via Facebook
- Input: Facebook ID, Facebook token
• Output: Success, or detailed failure (in case condition does not hold)
• Condition: Facebook token is valid

Delete a user’s account
• Input: Authentication token, identification
• Output: Success, or detailed failure (in case condition does not hold)
• Condition: The authentication and identification match an existing user

Query a user’s details if available (name, gender, birthday)
• Input: Authentication token
• Output: Name, gender and birthday (as available) or detailed failure (in case condition does not hold)
• Condition: Authentication token is valid

Update a user’s details
• Input: Authentication token, name, gender and birthday
• Output: Success, or detailed failure (in case condition does not hold)
• Condition: The authentication token is valid, the name is a valid name, the gender is a valid gender and the birthday is a valid birthday

Log in a guest user
• Input: Device ID
• Output: Authentication token, or detailed failure (in case condition does not hold)
• Condition: Device ID is valid

Delete a guest user’s account
• Input: Device ID
• Output: Success, or detailed failure (in case condition does not hold)
• Condition: Device ID is valid

Resources
A connection to the Mailgun servers

References
Use cases regarding registration and login are described in section A.1 of the URD. The relation of registration and login to the rest of the system is described in section 2.5.2 of the SRD. The interaction between registration and login and other components is described in section 2.7.5 of the SRD.

Processing
This API component does not do processing apart from calling the other components that can retrieve the requested data: namely the Database component, and then checking the data’s validity. In addition, the Authentication token verification component is used to validate authentication tokens and the Facebook authentication component is used to validate Facebook tokens.

Data
none
5.2 Client-side components

5.2.1 About

Type
Module

Purpose
SR248, SR281, SR282

Function
It displays information about the app's licenses and the company that made it.

Subordinates
none

Dependencies
This component does not depend on any other components.

Interfaces
none

Resources
It uses several external software libraries: Angular, jQuery, Angulartics, Ionic and Cordova

References
none

Processing
none

Data
none
5.2.2 Analytics

Type
Module

Purpose
SR208

Function
It records all analytic events.

Subordinates
none

Dependencies
This component depends on the following TravelMatch components:
- **Main**: For obtaining user identity info that is sent with every analytics event.

Interfaces
The user does an action
- **Input**: Action
- **Output**: none
- **Condition**: The action can be recorded

Resources
It uses several external software libraries: Angular, jQuery, Angulartics, Ionic and Cordova

References
The relation of TravelMatch to the analytics is described in section 2.5.3 of the SRD.

Processing
It records actions in the TravelMatch app via the Service.

Data
none
5.2.3 Details

Type
Module

Purpose

Function
This component allows users to make and modify vacation details.

Subordinates
none

Dependencies
This component depends on the following TravelMatch components:

- **Analytics**: For sending an analytics event when the user submits the vacation details form.
- **Language**: For providing strings in the correct language used in the user interface of the vacation detail screen.
- **Main**: For providing user authentication info required for communication with the back end API, and for intercepting common HTTP error codes.

Furthermore, this component depends on the following external library components:

- **Ionic**: For providing pop-up functionality.

Interfaces
Create a vacation

- **Input**: Vacation details
- **Output**: Vacation ID
- **Condition**: Valid vacation details.

Modify a vacation

- **Input**: Vacation details and vacation ID
- **Output**: none
- **Condition**: Valid vacation ID.

Get a vacation

- **Input**: vacation ID
- **Output**: vacation details
- **Condition**: valid vacation ID.
Resources
It uses several external software libraries: Angular, jQuery, Angulartics, Ionic and Cordova

References
The data model of the vacation details is described in section 2.7.3 of the SRD. The vacation details in relation to other components are described in the class diagram of the TravelMatch app in section 2.7.4 of the SRD. The interaction of the vacation details with other components is described in section 2.7.5 of the SRD.

Processing
The view subcomponent provides the interface described above, the controller hears the request. The request is send to the Service to handle a call to the Server with the vacation and or vacation details. Depending on the response of the Server, the user can start the interest analysis.

Data
none
5.2.4 Front

Type
Module

Purpose

Function
This component allows a guest user to choose to use the TravelMatch app with or without an authentication method.

Subordinates
Main

Dependencies
This component depends on the following TravelMatch components:
- Analytics: For sending an analytics event if the guest login or Facebook login options are used.
- Main: For checking if the user is already authenticated, as the front screen is skipped if that is the case.
- Language: For providing strings in the correct language used in the user interface of the front screen.

Furthermore, this component depends on the following external library components:
- Ionic: For providing pop-up functionality.

Interfaces
Guest login
- Input: Device ID
- Output: Access to the app
- Condition: User has a valid device ID.

Resources
It uses several external software libraries: Angular, jQuery, Angulartics, Ionic and Cordova

References
The user interface of the front screen is described in section A.1.2 of the SRD.

Processing
The view subcomponent provides the interface described above, the controller hears the request. The request is send to the Service to handle a call to the Server with the device ID. Depending on the response of the Server, the user knows if the app can be used.

Data
none
5.2.5 HotelOverview

This component is dependent on the Main and Language component, because it uses a screen with descriptions and connects with the API for registration and authentication.

**Type**

Module

**Purpose**


**Function**

This component allows users to view their recommendations and book a trip.

**Subordinates**

none

**Dependencies**

This component depends on the following TravelMatch components:

- **Analytics**: For sending an analytics event when the first or second recommendation is viewed, or when the user select a hotel.
- **Main**: For providing user authentication info required for communication with the back end API, and for intercepting common HTTP error codes.
- **Language**: For providing strings in the correct language used in the user interface of the hotel overview screen.

Furthermore, this component depends on the following external library components:

- **Ionic**: For providing pop-up functionality, and for providing touch-based scrolling controls.

**Interfaces**

**Save location**

- Input: Location ID and trip IDs
- Output: none
- Condition: Valid location ID.

**Load location**

- Input: Location ID
- Output: Trips information
- Condition: User has valid location ID.

**Retrieve trips**
• Input: Location ID
• Output: Trips information
• Condition: User has valid location ID.

Resources
It uses several external software libraries: Angular, jQuery, Angulartics, Ionic and Cordova

References
The general capabilities of the hotel overview are described in section 2.2.3 of the URD. Use cases regarding the hotel overview are described in section A.2 of the URD. The data model of the hotel overview is described in section 2.7.3 of the SRD. The user interface of the hotel overview is described in section A.1.11 and A.1.12 of the SRD.

Processing
The view subcomponent provides the interface above, the controller hears the request. The request is send to the Service to handle a call to the Server with the location ID and maybe trip IDs. Depending on the response of the Server, the user sees trips or errors.

Data
Location IDs
5.2.6 Language

Type
Package

Purpose
SR263

Function
It supplies language specific descriptions for language dependent items.

Subordinates
none

Dependencies
This component does not depend on any TravelMatch components. However, this component depends on the following external library components:

- **angular-translate**: For providing an interface to defining translation strings.

Interfaces

description mentioned in the app

- Input: Description
- Output: Reaction dependent on response
- Condition: Login details are valid with Facebook.

Resources
It uses several external software libraries: Angular, jQuery, Angulartics, Ionic and Cordova

References
none

Processing
The language subcomponent gives the set language description to the requester.

Data
Translation for each description in each required language.
5.2.7 Login

Type
Module

Purpose
SR15, SR16, SR17, SR18, SR19, SR245

Function
This component allows the user to log in with all authentication providers.

Subordinates
Main, Registration

Dependencies
This component depends on the following TravelMatch components:

- **Analytics**: For sending an analytics event when the user logs in, either with an e-mail and password or with Facebook.
- **Language**: For providing strings in the correct language used in the user interface of the login screen.
- **Main**: For storing user authentication info after a successful login, and for intercepting common HTTP error codes.
- **Registration**: For initializing the e-mail input field to the last e-mail address that was used to register an account in the current session, and for creating a new TravelMatch account in the back-end server upon the first time a user logs in with Facebook.

Furthermore, this component depends on the following external library components:

- **Ionic**: For providing pop-up functionality.
- **Phonegap Facebook plugin**: For providing a native interface that allows TravelMatch to launch the mobile Facebook app if it is installed on the user’s device.

Interfaces
Login User

- Input: E-mail and password
- Output: Reaction dependent on response
- Condition: E-mail and password are in the database

Resources
It uses several external software libraries: Angular, jQuery, Angulartics, Ionic and Cordova

References
Use cases regarding login are described in section A.1 of the URD. The relation of login to the rest of the system is described in section 2.5.2 of the SRD. The interaction between login and other components is described in section 2.7.5 of the SRD. The user interface of the login screen is described in section A.1.7 of the SRD.
Processing
The view part of the component provides the interface described above, the controller reads the user input when requested. The user input is send to the Server by the Service. Depending on the Servers response the user can be told the result of logging in. The view part of the component also provides access to the Registration module for Facebook authentication.

Data
none
5.2.8 Main

Type
Module

Purpose
SR223, SR224, SR251, SR252

Function
It handles the authentication of a user.

Subordinates
none

Dependencies
This component does not depend on any other components.

Interfaces
Request to back end

- Input: Configuration
- Output: Data JWT encoded + authentication token
- Condition: User has an authentication token.

Resources
It uses several external software libraries: Angular, jQuery, Angulararts, Ionic, jsjws and Cordova

References
none

Processing
If there is a request to the back end the request is injected with the authentication token if available by the subcomponent.

Data
Identity and token.
5.2.9 Navigation

Type
Module

Purpose
SR241, SR254, SR256

Function
This component handles the navigation throughout the TravelMatch app, the views consists of the header, back button and sidebar visible throughout the app.

Subordinates
none

Dependencies
This component depends on the following TravelMatch components:

- **Analytics**: For sending an analytics event when the user returns to the vacation details screen from the hotel overview screen.
- **Language**: For providing strings in the correct language used in the user interface of the sidebar menu.
- **Main**: For checking if a user is currently authenticated, so that the sidebar menu can be adapted to this.

Furthermore, this component depends on the following external library components:

- **Ionic**: For providing pop-up functionality.

This component is dependent on the Language component. Each entry in the sidebar needs language specific translation.

Interfaces
Sidebar redirecting
- Input: Redirect
- Output: Change of view
- Condition: The view is available in the sidebar

Back button
- Input: Redirect
- Output: Change of view
- Condition: The back button is available.

Sidebar
- Input: Redirect
- Output: Change of view
- Condition: The sidebar is available and the user has the rights for the new view.

Logout
- Input: none
- Output: Redirect
- Condition: The user is not logged out yet.

Resources
It uses several external software libraries: Angular, jQuery, Angulartsic, Ionic and Cordova

References
The user interface of the navigation is described in section A.1 of the SRD.

Processing
The view component provides the interface described above. If the user wants to redirect it interacts with the sidebar, which the controller will handle. A description of the redirect is given to indicate what will happen if the user interacts. On the event of logging out, the controller is called which calls a subcomponent of Main to complete logging out. In the event of the user interacting with the back button, the controller handles the redirect to the specified back location.

Data
none
5.2.10 Registration

Type
Module

Purpose
SR1, SR2, SR3, SR4, SR5, SR20, SR21, SR22, SR23, SR24, SR246

Function
This component allows for the registering of an e-mail user and authentication of a Facebook user.

Subordinates
Main

Dependencies
This component depends on the following TravelMatch components:

- **Language**: For providing strings in the correct language used in the user interface of the registration screen.
- **Main**: For storing user authentication info after a successful Facebook registration, and for intercepting common HTTP error codes.

Furthermore, this component depends on the following external library components:

- **Ionic**: For providing pop-up functionality.

Interfaces
Register user

- Input: E-mail, password and password2
- Output: Reaction dependent on response
- Condition: Password matches password2

Facebook login

- Input: Login details Facebook
- Output: Reaction dependent on response
- Condition: Login details are valid with Facebook.

Resources
It uses several external software libraries: Angular, jQuery, Angulartics, Ionic and Cordova

References
Use cases regarding registration are described in section A.1 of the URD. The relation of registration to the rest of the system is described in section 2.5.2 of the SRD. The interaction between registration and other components is described in section 2.7.5 of the SRD. The user interface of the registration screen is described in section A.1 of the SRD.
Processing

The view subcomponent provides the interface described above, the controller reads the user input when requested. The user input is validated by the Service and send to the Server. Depending on the Servers response the user can be told the result of registering. If the user wants to authenticate with Facebook, a service will handle calling the Facebook browser or app to authenticate. Depending on the result of this authenticating, the obtained data will be send to the Server for validating. Depending on the response of the Server the Service will report the response to user. With a successful response from the Server the received data will be communicated with the Main module.

Data

none
5.2.11  Swipe

Type
Module

Purpose

Function
This component handles the interest analysis. It will retrieve images for the user to judge.

Subordinates
Main and Details

Dependencies
This component depends on the following TravelMatch components:

- **Analytics**: For sending an analytics event when the user (dis)likes an image.
- **Details**: For providing vacation details info required for communication with the back end API.
- **HotelOverview**: For receiving a recommendation when the interest analysis has finished.
- **Language**: For providing strings in the correct language used in pop-ups in the interest analysis screen.
- **Main**: For providing user authentication info required for communication with the back end API, and for intercepting common HTTP error codes.

Furthermore, this component depends on the following external library components:

- **Ionic**: For providing pop-up functionality.

Interfaces
Judging an image

- Input: Judgement
- Output: New image or recommendation
- Condition: Images are available or interest analysis is done.

Retrieve images

- Input: none
- Output: Images
- Condition: Images are available, user has an authentication token and vacation id.

Resources
It uses several external software libraries: Angular, jQuery, Angulartics, Ionic and Cordova.
References
The general capabilities of the interest analysis are described in section 2.2.1 of the URD. Use cases regarding the interest analysis are described in section A.2 of the URD. The interaction between the interest analysis API and other components is described in section 2.7.5 of the SRD. The user interface of the interest analysis is described in section A.1.9 of the SRD.

Processing
Each time the user is in the screen, the TravelMatch app asks for images until a set amount images is judged. If the user has made a judgment about a photo, the server is called to record the judgment.

Data
none
5.2.12 User

Type
Module

Purpose

Function
This component allows for user details to be made and changed for an authenticated user.

Subordinates
Main

Dependencies
This component depends on the following TravelMatch components:

- **Language**: For providing strings in the correct language used in the user interface of the user info screen.
- **Main**: For providing user authentication info required for communication with the back end API, and for intercepting common HTTP error codes.

Furthermore, this component depends on the following external library components:

- **Ionic**: For providing pop-up functionality.

Interfaces

Change user details
- **Input**: Name, gender and birthday
- **Output**: Reaction dependent on response
- **Condition**: Input data has the correct format.

Retrieve user details
- **Input**: none
- **Output**: Name, gender and birthday
- **Condition**: User is authenticated

Resources
It uses several external software libraries: Angular, jQuery, Angulartics, Ionic and Cordova

References
The data model of the user info is described in section 2.7.3 of the SRD. The user info in relation to other components is described in the class diagram of the TravelMatch app in section 2.7.4 of the SRD. The interaction of the user info with other components is described in section 2.7.5 of the SRD. The user interface of the user info screen is described in section A.1.8 of the SRD.
Processing

The view subcomponent provides the interface described above, the controller reads the user input when requested. The user input is verified by the Service and send to the Server. Depending on the Servers response the user can be told the result of the registering.

Data

none
Chapter 6
Feasibility and resource estimates

This chapter gives an estimation of the computer resources that are needed to develop and operate TravelMatch. These are the specifications under which TravelMatch has been developed. This does not mean that Travelmatch will not run on machines with lower specifications but does mean that it will most likely run better on computers with at least these specifications.

The requirements for the development of TravelMatch are:

- **CPU**: $\geq 1.0 \text{ GHz } x86$ or equivalent
- **Memory**: $\geq 2 \text{ GB RAM}$
- **Hard disk**: $\geq 2 \text{ GB free on disk (or more if you wish to put more data in the database)}$
- **Operating System**: Linux (Mint 17) / Windows (7, 8) / MAC OS X 10.9 / Chrome OS 44
- **Software**: MySQL 14.14 Distrib 5.5.43 or SQLite 3.8.10.2, Node Package Manager 2.7.4, Gulp 3.9.0, Cordova 5.1.1, Ionic 1.5.0, Google Chrome 43, Django 1.8.0, Git 2.4.4, Android 4.4.2 SDK (API 19), Xcode 7

The requirements for operating TravelMatch are:

- **Client-side**:
  - **CPU**: $\geq 1.0 \text{ GHz } x86$ or equivalent
  - **Memory**: $\geq 512 \text{ MB RAM}$
  - **Hard disk**: $\geq 30 \text{ MB free on disk}$
  - **Operating System**: Linux (Mint 17) / Windows (7, 8) / Android 4.4.2 / iOS 8.3
  - **Software**: Google Chrome 43

- **Server-side**:
  - **CPU**: $\geq 1.0 \text{ GHz } x86$ or equivalent
  - **Memory**: $\geq 2 \text{ GB RAM}$
  - **Hard disk**: $\geq 30 + 2 \text{ GB free on disk (based on 1000 images in the CMS and 100,000 users in the database)}$
  - **Operating System**: Linux (Ubuntu 14.04.2)
  - **Software**: MySQL 14.14 Distrib 5.5.43 or SQLite 3.8.10.2, Django 1.8.0
Chapter 7
Requirements traceability matrix

This chapter describes how the SRD [3] is connected to the ADD, by means of mapping the Software Requirements to components. This way, every developer can look up what a component’s purpose is.

Most requirements have at least two components that fulfill them as they pose a requirement to both the client and the server. NI denotes that a software requirement, by design, is not fulfilled by any component. NCS denotes that a software requirement is not component-specific but puts a requirement on the client or server as a whole.

7.1 SR to components

<table>
<thead>
<tr>
<th>Requirement</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SR1</td>
<td>Registration / login API, Registration</td>
</tr>
<tr>
<td>SR2</td>
<td>Registration / login API, Registration</td>
</tr>
<tr>
<td>SR3</td>
<td>Registration / login API, Registration</td>
</tr>
<tr>
<td>SR4</td>
<td>Registration</td>
</tr>
<tr>
<td>SR5</td>
<td>Registration / login API, Registration</td>
</tr>
<tr>
<td>SR6</td>
<td>Registration / login API</td>
</tr>
<tr>
<td>SR7</td>
<td>Registration / login API</td>
</tr>
<tr>
<td>SR8</td>
<td>Registration / login API</td>
</tr>
<tr>
<td>SR9</td>
<td>Registration / login API</td>
</tr>
<tr>
<td>SR10</td>
<td>Registration / login API</td>
</tr>
<tr>
<td>SR11</td>
<td>Registration / login API</td>
</tr>
<tr>
<td>SR12</td>
<td>Registration / login API</td>
</tr>
<tr>
<td>SR13</td>
<td>Registration / login API</td>
</tr>
<tr>
<td>SR14</td>
<td>Registration / login API</td>
</tr>
<tr>
<td>SR15</td>
<td>Registration / login API, Login</td>
</tr>
<tr>
<td>SR16</td>
<td>Registration / login API, Login</td>
</tr>
<tr>
<td>SR17</td>
<td>Registration / login API, Login</td>
</tr>
<tr>
<td>SR18</td>
<td>Registration / login API, Login</td>
</tr>
<tr>
<td>SR19</td>
<td>Registration / login API, Login</td>
</tr>
<tr>
<td>SR20</td>
<td>Facebook authentication, Registration / login API, Registration</td>
</tr>
<tr>
<td>SR21</td>
<td>Facebook authentication, Registration / login API, Registration</td>
</tr>
<tr>
<td>SR22</td>
<td>Facebook authentication, Registration / login API, Registration</td>
</tr>
<tr>
<td>SR23</td>
<td>Registration / login API, Registration</td>
</tr>
<tr>
<td>SR24</td>
<td>Registration / login API, Registration</td>
</tr>
<tr>
<td>SR25</td>
<td>Registration / login API, User</td>
</tr>
<tr>
<td>SR26</td>
<td>Registration / login API, User</td>
</tr>
<tr>
<td>SR27</td>
<td>Registration / login API, User</td>
</tr>
<tr>
<td>SR28</td>
<td>Registration / login API, User</td>
</tr>
<tr>
<td>SR29</td>
<td>Registration / login API, User</td>
</tr>
<tr>
<td>SR30</td>
<td>Registration / login API, User</td>
</tr>
<tr>
<td>SR31</td>
<td>Registration / login API, User</td>
</tr>
<tr>
<td>SR32</td>
<td>Registration / login API, User</td>
</tr>
<tr>
<td>SR33</td>
<td>Registration / login API, User</td>
</tr>
<tr>
<td>SR34</td>
<td>Registration / login API, User</td>
</tr>
<tr>
<td>SR35</td>
<td>Registration / login API, User</td>
</tr>
</tbody>
</table>
SR36  Registration / login API, User
SR37  Registration / login API, User
SR38  Registration / login API, User
SR39  Registration / login API, User
SR40  Registration / login API, User
SR41  Registration / login API, Front
SR42  Registration / login API, Front
SR43  Registration / login API, Front
SR44  Registration / login API, Front
SR45  Registration / login API, Front
SR46  Registration / login API, Front
SR47  Registration / login API, Front
SR48  Authentication token verification, Details
SR49  Authentication token verification, Details
SR50  Interest analysis API, Details
SR51  Interest analysis API, Details
SR52  Interest analysis API, Details
SR53  Interest analysis API, Details
SR54  Interest analysis API, Details
SR55  Interest analysis API, Details
SR56  Interest analysis API, Details
SR57  Interest analysis API, Details
SR58  Interest analysis API, Details
SR59  Interest analysis API, Details
SR60  Interest analysis API, Details
SR61  Interest analysis API, Details
SR62  Interest analysis API, Details
SR63  Interest analysis API, Details
SR64  Interest analysis API, Details
SR65  Interest analysis API, Details
SR66  Interest analysis API, Details
SR67  Interest analysis API, Details
SR68  Interest analysis API, Details
SR69  Interest analysis API, Details
SR70  Interest analysis API, Details
SR71  Interest analysis API, Details
SR72  Interest analysis API, Details
SR73  Interest analysis API, Details
SR74  Interest analysis API, Details
SR75  Interest analysis API, Details
SR76  Interest analysis API, Details
SR77  Interest analysis API, Details
SR78  Interest analysis API, Details
SR79  Interest analysis API, Details
SR80  Interest analysis API, Details
SR81  Interest analysis API, Details
SR82  Interest analysis API, Details
SR83  Interest analysis API, Details
SR84  Interest analysis API, Details
SR85  Interest analysis API, Details
SR86  Interest analysis API, Details
SR87  Interest analysis API, Details
SR88  Interest analysis API, Details
SR89  Interest analysis API, Details
<table>
<thead>
<tr>
<th>ID</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SR90</td>
<td>Interest analysis API, Details</td>
</tr>
<tr>
<td>SR91</td>
<td>Interest analysis API, Details</td>
</tr>
<tr>
<td>SR92</td>
<td>Interest analysis API, Details</td>
</tr>
<tr>
<td>SR93</td>
<td>AI, Interest analysis API, Swipe</td>
</tr>
<tr>
<td>SR94</td>
<td>Interest analysis API, Swipe</td>
</tr>
<tr>
<td>SR95</td>
<td>AI, Interest analysis API, Swipe</td>
</tr>
<tr>
<td>SR96</td>
<td>AI, Interest analysis API, Swipe</td>
</tr>
<tr>
<td>SR97</td>
<td>Interest analysis API, Swipe</td>
</tr>
<tr>
<td>SR98</td>
<td>AI, Interest analysis API, Swipe</td>
</tr>
<tr>
<td>SR99</td>
<td>AI, Interest analysis API, Swipe</td>
</tr>
<tr>
<td>SR100</td>
<td>AI, Interest analysis API, Swipe</td>
</tr>
<tr>
<td>SR101</td>
<td>AI, Interest analysis API, Swipe</td>
</tr>
<tr>
<td>SR102</td>
<td>AI, Interest analysis API, Swipe</td>
</tr>
<tr>
<td>SR103</td>
<td>AI, Interest analysis API, Swipe</td>
</tr>
<tr>
<td>SR104</td>
<td>Interest analysis API, Swipe</td>
</tr>
<tr>
<td>SR105</td>
<td>Interest analysis API, Swipe</td>
</tr>
<tr>
<td>SR106</td>
<td>Interest analysis API, Swipe</td>
</tr>
<tr>
<td>SR107</td>
<td>Interest analysis API, Swipe</td>
</tr>
<tr>
<td>SR108</td>
<td>Interest analysis API, Swipe</td>
</tr>
<tr>
<td>SR109</td>
<td>Interest analysis API, Swipe</td>
</tr>
<tr>
<td>SR110</td>
<td>Interest analysis API, Swipe</td>
</tr>
<tr>
<td>SR111</td>
<td>AI, Interest analysis API, Swipe</td>
</tr>
<tr>
<td>SR112</td>
<td>Interest analysis API, Swipe</td>
</tr>
<tr>
<td>SR113</td>
<td>AI, Interest analysis API, Swipe</td>
</tr>
<tr>
<td>SR114</td>
<td>AI, Interest analysis API, Swipe</td>
</tr>
<tr>
<td>SR115</td>
<td>Interest analysis API, Swipe</td>
</tr>
<tr>
<td>SR116</td>
<td>AI, Interest analysis API, Swipe</td>
</tr>
<tr>
<td>SR117</td>
<td>Interest analysis API, Swipe</td>
</tr>
<tr>
<td>SR118</td>
<td>AI, Interest analysis API, Swipe</td>
</tr>
<tr>
<td>SR119</td>
<td>AI, Interest analysis API, Swipe</td>
</tr>
<tr>
<td>SR120</td>
<td>AI, Interest analysis API, Swipe</td>
</tr>
<tr>
<td>SR121</td>
<td>AI, Interest analysis API, Swipe</td>
</tr>
<tr>
<td>SR122</td>
<td>AI, Interest analysis API, Swipe</td>
</tr>
<tr>
<td>SR123</td>
<td>AI, Interest analysis API, Swipe</td>
</tr>
<tr>
<td>SR124</td>
<td>AI, Recommendation API, HotelOverview</td>
</tr>
<tr>
<td>SR125</td>
<td>Recommendation API, HotelOverview</td>
</tr>
<tr>
<td>SR126</td>
<td>AI, Recommendation API, HotelOverview</td>
</tr>
<tr>
<td>SR127</td>
<td>AI, Recommendation API, HotelOverview</td>
</tr>
<tr>
<td>SR128</td>
<td>Recommendation API, HotelOverview</td>
</tr>
<tr>
<td>SR129</td>
<td>AI, Recommendation API, HotelOverview</td>
</tr>
<tr>
<td>SR130</td>
<td>AI, Recommendation API, HotelOverview</td>
</tr>
<tr>
<td>SR131</td>
<td>AI, Recommendation API, HotelOverview</td>
</tr>
<tr>
<td>SR132</td>
<td>AI, Recommendation API, HotelOverview</td>
</tr>
<tr>
<td>SR133</td>
<td>AI, Recommendation API, HotelOverview</td>
</tr>
<tr>
<td>SR134</td>
<td>AI, Recommendation API, HotelOverview</td>
</tr>
<tr>
<td>SR135</td>
<td>AI, Recommendation API, HotelOverview</td>
</tr>
<tr>
<td>SR136</td>
<td>AI, Recommendation API, HotelOverview</td>
</tr>
<tr>
<td>SR137</td>
<td>AI, Recommendation API, HotelOverview</td>
</tr>
<tr>
<td>SR138</td>
<td>AI, Recommendation API, HotelOverview</td>
</tr>
<tr>
<td>SR139</td>
<td>AI, Recommendation API, HotelOverview</td>
</tr>
<tr>
<td>SR140</td>
<td>AI, Recommendation API, HotelOverview</td>
</tr>
<tr>
<td>SR141</td>
<td>AI, Recommendation API, HotelOverview</td>
</tr>
<tr>
<td>SR142</td>
<td>AI, Recommendation API, HotelOverview</td>
</tr>
<tr>
<td>SR143</td>
<td>AI, Recommendation API, HotelOverview</td>
</tr>
</tbody>
</table>
SR144 Recommendation API, HotelOverview
SR145 Recommendation API, HotelOverview
SR146 Recommendation API, HotelOverview
SR147 Recommendation API, HotelOverview
SR148 Recommendation API, HotelOverview
SR149 Recommendation API, HotelOverview
SR150 Recommendation API, HotelOverview
SR151 Recommendation API, HotelOverview
SR152 Recommendation API, HotelOverview
SR153 Recommendation API, HotelOverview
SR154 Recommendation API, HotelOverview
SR155 Recommendation API, HotelOverview
SR156 Recommendation API, HotelOverview
SR157 Recommendation API, HotelOverview
SR158 Recommendation API, HotelOverview
SR159 Recommendation API, HotelOverview
SR160 Recommendation API, HotelOverview
SR161 Recommendation API, HotelOverview
SR162 Recommendation API, HotelOverview
SR163 Recommendation API, HotelOverview
SR164 Recommendation API, HotelOverview
SR165 Recommendation API, HotelOverview
SR166 Recommendation API, HotelOverview
SR167 Recommendation API, HotelOverview
SR168 Recommendation API, HotelOverview
SR169 Recommendation API, HotelOverview
SR170 Recommendation API, HotelOverview
SR171 Recommendation API, HotelOverview
SR172 Recommendation API, HotelOverview
SR173 Recommendation API, HotelOverview
SR174 Recommendation API, HotelOverview
SR175 Recommendation API, HotelOverview
SR176 Recommendation API, HotelOverview
SR177 Recommendation API, HotelOverview
SR178 Recommendation API, HotelOverview
SR179 CMS
SR180 CMS
SR181 CMS
SR182 CMS
SR183 CMS
SR184 CMS
SR185 CMS
SR186 CMS
SR187 CMS
SR188 CMS
SR189 CMS
SR190 CMS
SR191 CMS
SR192 CMS
SR193 CMS
SR194 CMS
SR195 CMS
SR196 CMS
SR197 CMS
SR198  
SR199  
SR200 Authentication token verification  
SR201  
SR202 AI, Interest analysis API, Swipe  
SR203 HotelOverview  
SR204 HotelOverview  
SR205 AI  
SR206 AI  
SR207 AI  
SR208 Analytics  
SR209  
SR210 Affiliate  
SR211 Affiliate  
SR212 Affiliate  
SR213 Affiliate  
SR214 Affiliate  
SR215 Affiliate  
SR216 Affiliate  
SR217 Affiliate  
SR218 CMS  
SR219 CMS  
SR220 CMS  
SR221 CMS  
SR222 Database  
SR223 Interest analysis API, Recommendation API, Registration / login API, Main  
SR224 Interest analysis API, Recommendation API, Registration / login API, Main  
SR225 Interest analysis API, Recommendation API, Registration / login API  
SR226 NCS  
SR227 NCS  
SR228 NCS  
SR229 NCS  
SR230 NI  
SR231 NI  
SR232 NI  
SR233 NI  
SR234 NI  
SR235 NI  
SR236 NI  
SR237 NI  
SR238 NCS  
SR239 Interest analysis API, Recommendation API, Registration / login API  
SR240 NCS  
SR241 Navigation  
SR242 Details  
SR243 Swipe  
SR244 User  
SR245 Login  
SR246 Registration  
SR247 HotelOverview  
SR248 About  
SR249 HotelOverview  
SR250 Front  
SR251 Main
SR252 Main
SR253 Swipe
SR254 Navigation
SR255 Front
SR256 Navigation
SR257 
SR258 
SR259 
SR260 
SR261 Registration / login API
SR262 Front
SR263 Language
SR264 Database
SR265 NCS
SR266 NCS
SR267 NCS
SR268 CMS
SR269 CMS
SR270 
SR271 AI, Recommendation API, HotelOverview
SR272 Database
SR273 Database
SR274 Database
SR275 Database
SR276 Database
SR277 Database
SR278 
SR279 Database
SR280 Database
SR281 About
SR282 About

7.2 Components to SR

**Server side**

**Affiliate**
SR210, SR211, SR212, SR213, SR214, SR215, SR216, SR217

**AI**

**Authentication token verification**
SR48, SR49, SR200

**CMS**

**Database**
SR222, SR264, SR272, SR273, SR274, SR275, SR276, SR277, SR279, SR280

**Facebook authentication**
SR20, SR21, SR22
TravelMatch Architectural Design Document

**Interest analysis API**

**Recommendation API**

**Registration / login API**

**Client side**

About
SR248, SR281, SR282

Analytics
SR208

Details

Front

HotelOverview

Main
SR223, SR224, SR251, SR252

Language
SR263

Login
SR15, SR16, SR17, SR18, SR19, SR245

Navigation
SR241, SR254, SR256

Registration
SR1, SR2, SR3, SR4, SR5, SR20, SR21, SR22, SR23, SR24, SR246
<table>
<thead>
<tr>
<th>Category</th>
<th>SR Numbers</th>
</tr>
</thead>
</table>