TravelMatch

Detailed 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 Detailed Design Document 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 ESA software engineering standard [1].

Contents

Document Status Sheet

General

Document title:	Detailed Design Document	
Document identifier:	TravelMatch.Doc.DDD/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

Version	Date	Author(s)	Reason
0.1	17-06-2015	D.J. van den Brand, S. He, J.M.A.P. van Hoof, G.C. Linders, M.J.M. Muijsers	Initial version.
1.0	19-06-2015	D.J. van den Brand, S. He, G.C. Linders, M.J.M. Muijsers	Final version.

Document Change Records

General

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

Changes

Section	Reason

Chapter 1 Introduction

1.1 Purpose

This Detailed Design Document (DDD) describes the implementation of the TravelMatch system on the most detailed level. The implementation of all components of the TravelMatch system, as defined in the Software Requirements Document[3] and Architecture Design Document[4], is described in this document.

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

Affiliate Network	A network that enables you to receive money from customer redirection [18]
Analytics Data	The log of analytics events that is recorded and stored on the database.
Android	A popular open-source operating system for embedded devices, including smartphones and tablets, created by Google.
Angular JS	An open-source web application framework maintained by Google.
Cosine similarity	A measure of similarity between two vectors of an inner product space that measures the cosine of the angle between them.
Destination advice	The city, and selection of hotels, that is advised to a user after performing one or more interest analyses.
Destination attributes or tags	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*.
DNA attribute or tags	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.
Google Play Store	A public repository of free and paid apps for Android, managed by Google.
Guest user	An user that does not provide login details but still uses the TravelMatch app.
Hotelstars rating	A hotel classification with common criteria and procedures in participating countries to rate a hotel's quality. See [21].

iLysian	Short for iLysian B.V., a software engineering company situated in Eindhoven, Netherlands. The client for the TravelMatch project.
Interest analysis	The action the user will do of judging the images.
iOS	A popular closed-source operating system for smartphones and tablets created by Apple.
iOS App Store	A public repository of free and paid apps for iOS, managed by Apple.
JWT	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.
Relational database management system (RDBMS)	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)
TCP/IP	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)
Tinder	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.
Travel DNA	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.
TravelMatch	An application for smartphones and tablets that assists users in planning a vacation. The subject of this project.
TravelMatch team	A team of Computer Science students at Eindhoven University of Technology who will design and implement the TravelMatch application.
User	The user of the app.
Waverunner	Waverunner Search Service by Pyton Communication Services; a search service that provides vacation offers and prices of participating travel agencies.

1.3.2 Abbreviations

ADD	Architecture Design Document
ADT	Abstract Data Type
AI	Artificial Intelligence
APK	Android Application Package
App	Application for smartphones and tablets
CMS	Content Management System
DDD	Detailed Design Document
DOM	Document Object Model
ESA	European Space Agency
HTTP	Hypertext Transfer Protocol
IDE	Integrated Development Environment
IPA	iOS App Store Package
NPM	Node Package Manager
OS	Operating System
SRD	Software Requirements Document
TU/e	Eindhoven University of Technology

UML	Unified Modeling Language
URD	User Requirements Document

1.4 References

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

[2] TravelMatch team. User Requirement Document. Version 1.2.1. 22 June 2015.

[3] TravelMatch team. Software Requirements Document. Version 1.0. 22 June 2015.

[4] TravelMatch team. Architectural Design Document. Version 1.0. 22 June 2015.

[5] TravelMatch team. Detailed Design Document. Version 1.0. 22 June 2015.

[6] TravelMatch team. Software User Manual. Version 1.0. 22 June 2015.

[7] TravelMatch team. Software Transfer Document. Version 1.0. 22 June 2015.

[8] TravelMatch team. Unit Test Plan. Version 1.0. 22 June 2015.

[9] TravelMatch team. Integration Test Plan. Version 1.0. 22 June 2015.

[10] TravelMatch team. Acceptance Test Plan. Version 1.0.2. 22 June 2015.

[11] TravelMatch team. Software Configuration Management Plan. Version 1.0. 22 June 2015.

[12] TravelMatch team. Software Project Management Plan. Version 1.0. 22 June 2015.

[13] TravelMatch team. Software Quality Assurance Plan. Version 1.0. 22 June 2015.

[14] TravelMatch team. Software Verification and Validation Plan. Version 1.0. 22 June 2015.

[15] Tom Preston-Werner. Semantic Versioning 2.0.0. Retrieved 6 May 2015. http://www.semver.org/

[16] Coley Consulting. MoSCoW Prioritisation. Retrieved 29 April 2015. http://www.coleyconsulting.co.uk/moscow.htm

[17] Tinder, Inc. Tinder. Retrieved 29 April 2015. http://www.gotinder.com/

[18] Organized Shopping, LLC. Affiliate Network. Marketing Terms. Retrieved 29 April 2015. http://www.marketingterms.com/dictionary/affiliate_network/

[19] Daiycon. About Daisycon. Retrieved 29 April 2015. http://www.daisycon.com/en/about_daisycon/

[20] Drifty Co. Ionic: Advanced HTML5 Hybrid Mobile App Framework. Retrieved 30 April 2015. http://ionicframework.com/

[21] Hotelstars Union. Classification criteria 2015-2020. Retrieved 1 May 2015. http://www.hotelstars.eu/index.php?id=criteria

[22] Django. http://www.django-cms.org/en/

[23] Django administration module. The Django Django admin site. Retrieved 1 June 2015. https://docs.djangoproject.com/en/1.8/ref/contrib/admin/

[24] Django Software Foundation. The Web framework for perfectionists with deadlines — Django. Retrieved 1 June 2015. https://www.djangoproject.com/

[25] Facebook User ID. User IDs and Friends. Retrieved 2 June 2015. https://developers.facebook.com/docs/apps/upgrading#upgrading_v2_0_user_ids

[26] ImageMagick. ImageMagick: Convert, Edit, Or Compose Bitmap Images. Retrieved 2 June 2015. http://www.imagemagick.org/

[27] Google. AngularJS - Superheroic JavaScript MVW Framework. Retrieved 1 June 2015. https://angularjs.org

[28] Adobe Systems Inc. Phonegap: Home. Retrieved 1 June 2015. http://phonegap.com/

[29] Xamarin Inc. Mobile App Development & App Creation Software - Xamarin. Retrieved 1 June 2015. http://xamarin.com/

[30] Eric Raymond. The Jargon File. Version 4.4.7. Retrieved 17 June 2015. http://www.catb.org/jargon/html/

[31] Python Software Foundation. Classes. The Python Tutorial. Retrieved 18 June 2015. https://docs.python.org/2/tutorial/classes.html

[32] Python Software Foundation. PEP 0008 – Style Guide for Python Code. 1 August 2013. https://www.python.org/dev/peps/pep-0008/

[33] Django Software Foundation. Coding style. Retrieved 18 June 2015. https://docs.djangoproject.com/en/1.8/internals/contributing/writing-code/coding-style/

[34] Django Software Foundation. Writing your first Django app, part 1. Database setup. Retrieved 18 June 2015. https://docs.djangoproject.com/en/1.8/intro/tutorial01/#database-setup

[35] Massachusetts Institute of Technology. MIT License. Retrieved 21 June 2015. http://opensource.org/licenses/MIT

[36] Apache Software Foundation. Apache License, Version 2.0. January 2004. http://www.apache.org/licenses/LICENSE-2.0

1.5 Overview

The remainder of this document consists of three chapters plus a number of appendices. In chapter 2 of this document we give a description of the standards and conventions used in the implementation of the TravelMatch system. Section 2.1 describes the standards used in constructing the high-level system design and model. Sections 2.2, 2.3 and 2.4 discuss the standards used in the implementation of the TravelMatch system on a source code level. Section 2.5 describes the tools used in the development of the TravelMatch system.

Then, in chapter 3 of this document we give a general description of the various components of the TravelMatch system. In chapter 4 of this document, we discuss the procedures to build and deploy the TravelMatch application and back end server. Finally, in the appendices of this document, we provide the documentation of the source code in the form of source code listings generated from the source code files.

Chapter 2
Standards and conventions

2.1 Design standards

For the high-level system design, Unified Modeling Language (UML) is used. Specifically, we use UML for the class diagrams and sequence diagrams. For the Entity-Relation diagram, we define the meaning of each diagram component as depicted in figure 2.1.

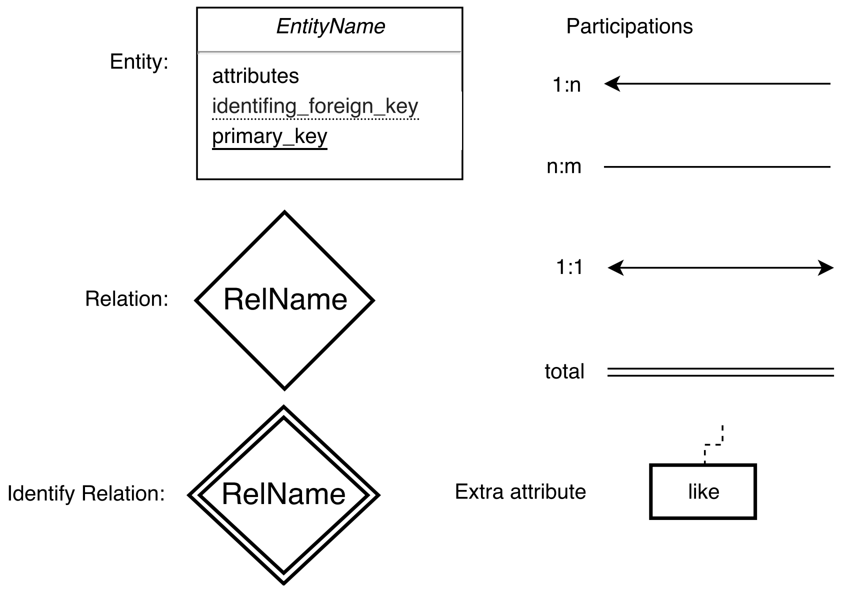


Figure 2.1: Database ER diagram legend

2.2 Documentation standards

2.2.1 General

Every object, class, method and property in the source code must be properly documented inline. We use `ngdoc` notation to document the front end code and `Doxygen` notation for the back end code. For this document, we use the documentation template in section A.1 for the front end and the documentation template in section B.1 for the back end.

For every method a contract must be documented which contains at least the following:

- The name of the method.
- A description of the method's functionality.
- If the method takes parameters, for each parameter the following must be documented:
  - The name of the parameter.

  - The type of the parameter (e.g. a number, string).
  - A description of what the parameter represents.
- If the method returns a value, the following must be documented:
  - The return type (e.g. a number, string).
  - A description of what the returned value represents.

For every object and class, the following must be documented:

- The name of the object or class.
- A description of the object or class.

For every property, the following must be documented:

- The name of the property.
- A description of the property.
- The type of the property (e.g. a number, string).

2.2.2 AngularJS-specific

For AngularJS code in the front end, additionally the following standards apply:

- For every AngularJS component, it must be documented if the object is a module, controller, directive or service.
- For every AngularJS directive, an example of the HTML usage must be defined.
- For every AngularJS directive, if it create a new scope, that must be documented.
- For every AngularJS directive, if the directive takes parameters, for each parameter the following must be documented:
  - The name of the parameter.
  - The type of the parameter (e.g. a number, string).
  - A description of what the parameter represents.

2.3 Naming conventions

2.3.1 Front end

For the front end, the following naming conventions apply.

- File names of AngularJS components must equal the name of that component in lowercase letters, tokenized by periods.
  - For example: `HotelService` is contained in `hotel.service.js`.
- With the exception of the `app.config` module, each file belonging to a certain module must be contained in a folder with the name of that module, with the `app.` and any non-alphabetical characters removed and in lowercase letters.
  - For example: `app.hotel.overview` files are contained in the `hoteloverview` folder.
- Names of AngularJS modules must begin with `app.` and can contain only lowercase letters and periods.

- Names of AngularJS directives follow camelCase practice where the first letter is not capitalized.
- Names of any other AngularJS components follow CamelCase practice where the first letter is capitalized.
- Names of AngularJS controllers must end with `Ctrl`.
- Names of AngularJS services must end with `Service`.
- All names of methods, functions, properties, variables, fields and parameters follow camelCase practice where the first letter is not capitalized, with the exception of parameters for AngularJS directives, which must be entirely lowercase.

2.3.2 Back end

For the back end, the following naming conventions apply.

- All models of an application are written in a file named `tmodels.py` in that application's subdirectory.
  - For example, the models for `ai` are stored in the `ai/models.py`.
- All unit tests of an application are written in a file named `test.py` in that application's subdirectory.
  - For example, the tests for `ai` are stored in the `ai/test.py`.
- Administration modules of an application are written in a file named `admin.py` in that application's subdirectory.
  - For example, the administration modules for `ai` are stored in the `ai/admin.py`.
- The interface that examines and validates the input and output of an application is written in a file named `serializer.py` in that application's subdirectory.
- The view controller of an application is written in a file named `view.py` in that application's subdirectory.
- The url routing related codes of an application are written in a file named `travelmatch/url.py` in that application's subdirectory.
- Naming of the classes, functions, methods and variables follows the standard Python naming conventions.[31]

2.4 Coding standards

Alongside to the above naming standards, the following additional coding standards apply.

2.4.1 Front end

- Use an indenting width of 2 spaces.
- Use spaces, not tabs, for indentation.
- Use the "One True Brace Style" for indentation.[30]
- DOM manipulation is only allowed in controller and directive methods.
- No HTML may be present in JavaScript source files.
- HTTP requests are only allowed in services.
- Public mutable properties in services are not allowed.

2.4.2 Back end

Following coding standards apply for the back end.

- The standard Python coding standards.[32]
- The standard Django coding standards.[33]

2.5 Software development tools

The following software development tool standards apply.

2.5.1 General

- **Git** is used for version control.
- **GitHub** is used for hosting the Git repository.
- **LaTeX** is used for writing documents.

2.5.2 Front end

- **Atom** is used for writing code.
- **NPM** is used for managing development dependencies.
- **Bower** is used for managing app dependencies.
- **Cordova** is used to build the app for Android and iOS.
- **Ionic** is used to test the app on a PC browser.
- **gulp** is used to automate development tasks.
- **ngdoc** is used to generate source code documentation.
- **pandoc** is used to convert ngdoc output to LaTeX.
- **karma** is used to run JavaScript unit tests.
- **Jasmine** is used to write JavaScript unit tests.

2.5.3 Back end

- **PyCharm** is used for writing code in Python.
- **WinSCP** is used for synchronizing the local directory with the remote sever directory.
- **PuTTY** is used for SSH commands.
- **Doxygen** is used to export the Python documentation to Latex.
- **Django** is used to set up the application and to run Python unit tests.

Chapter 3
Component descriptions

3.1 Front end

- **about**
  The about module consists of the model, view, controller and service for the about screen, handling and showing everything available in the screen. This module contains a single, simple controller that shows the iLysian logo and contact information for iLysian.

- **details**
  The details module consists of the model, view, controller and service for the vacation details screen. This module contains a controller that shows the input fields for the vacation details. This includes:

  – The date of departure;
  – The flexibility of the date of departure;
  – The date of return;
  – The flexibility of the date of return;
  – The budget per person;
  – A "surprise me" button for the budget;
  – The number of adults;
  – The number of children.
  – The submission button.

  Furthermore, this module contains a service that manages sending and retrieving vacation details from the back end API.

- **front**
  The front module consists of the model, view, controller and service for the front screen. This module contains a controller that shows the buttons on the front screen. This includes:

  – Continuing without an account (not implemented);
  – Connecting with Facebook;
  – Logging into a TravelMatch account;
  – Registering a TravelMatch account.

- **hotelOverview**
  The hotelOverview module consists of the model, view, controller and service for the hotel overview and hotel detail screen. This module contains a controller that displays the controls on the hotel overview screen. This includes:

  – Buttons to switch between the first advice and second advice;
  – A button to continue the interest analysis;
  – Hotel details for every hotel in the advice.
  – A booking button for every hotel in the advice.

Furthermore, this module contains a service that manages sending and retrieving recommendations from the back end API.

- **registration**
  The registration module consists of the model, view, controller and service for the registration of users. This module contains a controller that shows the input fields of the registration screen. This includes:
  - The e-mail field;
  - The password field;
  - The repeat password field;
  - The submission button;
  - The connect with Facebook button.

  Furthermore, this module contains a service that manages registering in the back end API.
- **main**
  The main module consists of the authentication service, HTTP interceptor and the constant values. This module contains a service that manages the user's session. Furthermore, it contains a service that intercepts all HTTP requests. Any strings in query parameters are encoded to prevent data loss. Also, generic HTTP errors are caught and resolved before they reach the caller. Finally, this module contains a number of global constants for the app.
- **language**
  The language module consists of the language files for each language one. This module contains a language file for Dutch and English language.
- **login**
  The login module consists of the model, view, controller and service of the login screen for all authentication providers. This module contains a controller that shows the input fields of the login screen. This includes:
  - The e-mail field;
  - The password field;
  - The submission button;
  - The connect with Facebook button.

  Furthermore, this module contains a service that manages logging in in the back end API. Additionally, this module contains the directive for the Facebook button, which displays a connect with Facebook button and manages Facebook authentication in the external API.
- **navigation**
  The navigation module consists of the models, views and controller of the header, sidebar and tabs of the login and registration screen. This module contains a main controller which contains common functionality that is used in all other controllers of this module. Furthermore, it contains several directives:
  - A directive for the header that displays the state name, back button and menu button;
  - A directive for the sidebar menu that displays buttons for every state in the sidebar menu;
  - A directive for the menu button in the header;
  - A directive for tabs in the login and registration screens, that allow the user to switch between the two.

- **swipe**
  The swipe module consists of the models, views, controller and service of the interest analysis named after the associated swiping of images. This module contains a photo directive which displays the photos being shown in the interest analysis, as well as the like/dislike buttons and progress bar. Furthermore, it contains a service that manages retrieving images and sending (dis)likes to the back end API. The photo directive is governed by a controller which obtains images from the service and routes them into the photo directive.

- **user**
  The user module consists of the model, view, controller and service of the user detail screen. This module contains a controller that displays the input fields of the user profile. This includes:

  - The name of the user;
  - The gender of the user;
  - The birth date of the user;
  - The submission button.

  Furthermore, this module contains a service that manages retrieving and storing user info in the back end API.

3.2 Back end

- **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 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.

Chapter 4
Build procedure

4.1 Front end

The TravelMatch app can be built and deployed by following the below procedure.

4.1.1 Prerequisites

1. The build PC is prepared for building on Android or iOS.
2. NPM is installed.
3. Git is installed.

4.1.2 Build process

1. Clone the TravelMatch Git repository.
2. Open a console window with admin/superuser privileges and go to the `src` folder:
    - `cd src`
3. Create the output directory:
    - `mkdir www`
4. Use NPM to install `gulp`, `Bower`, `Cordova` and `Ionic`:
    - `npm install gulp bower cordova ionic`
5. Install `karma-cli` globally.
    - `npm install karma-cli -g`
6. Install all development dependencies:
    - `npm install`
7. Add Android and/or iOS as Cordova platforms. Note that adding iOS is only possible on PC running OS X.
    - `cordova platform add android`
    - `cordova platform add ios`
8. Install all app dependencies:
    - `gulp cook`
9. Building the app for either Android or iOS:
    - `gulp android`
    - `gulp ios`

4.2 Back end

The TravelMatch server can be built and deployed by following the below procedure.

4.2.1 Prerequisites

1. Ubuntu 14.04 LTS or a compatible version is running on the server.
2. Python 2.7.6 or a compatible version is installed on the server.

4.2.2 Building server

1. Install Django via `pip` with following command:
   - `sudo python get-pip.py`
   - `git clone git://github.com/django/django.git django-trunk`
   - `sudo pip install -e django-trunk/`
   - `sudo pip install djangorestframework`
2. Install Mailgun.
   - `sudo pip install -e git://github.com/mailgun/mailgun.py.git#egg=pymailgun`
3. Install related Django packages.
   - `sudo pip install django_facebook`
4. Make migrations for the database in the ~/TravelMatch/server/travelmatch folder.
   - `python manage.py make migrations`
5. Set up the database. The database has the SQLite engine as its default configuration. This configuration can be changed in the `settings.py` file. The `DATABASES` variable must be set according to the Django tutorial.[34] An example of a MySQL configuration may be found below.

```
DATABASES = {
        'default': {
                'ENGINE': 'django.db.backends.mysql',
                'NAME': 'DB_NAME',
                'USER': 'DB_USER',
                'PASSWORD': 'DB_PASSWORD',
                'HOST': 'localhost',
                'PORT': '3306',
        }
}
```

6. Initialize the database.
   - `python manage.py migrate`
7. Start the server.
   - `python manage.py runserver 0.0.0.0:80`

Appendix A
Front end documentation

A.1 Documentation template

Each object in the front end documentation is documented with the following template:

Name of object

Description of the object.

Methods (if applicable)

- **nameOfMethod(parameterName)**

  This method is private. (if applicable)

  Description of the method.

  Parameters

Param	Type	Details
parameterName	Type of the parameter	Description of the parameter

  Returns (if applicable)

return type	Description of the return value

Properties (if applicable)

- **nameOfProperty**

  This property is private. (if applicable)

  Description of the property.

  Type: `type of the property`

A.2 app.about

The `app.about` module contains all templates and controllers that pertain to the about screen of the app.

A.2.1 AboutCtrl

The AngularJS controller for the about screen.

A.2.2 AnalyticsService

Provides methods for recording events for analytics purposes.

Methods

- **getDevice()**

  Gets all device information available

  Returns

Object	device device information

- **sendEvent(eventName, category, data)**

  Sends the specified event with the specified category to the analytics, with the specified data. If the user is authenticated, the user info is also sent.

  Parameters

Param	Type	Details
eventName	string	The name of the event.
category	string	The name of the category of the event.
data	string	The data to send.

A.3 app.config

The `app.config` module is a special module that contains global settings and configurations for the entire app. This module exposes a number of constants as well as special services.

A.3.1 HttpInterceptor

A factory that acts as an interceptor for the AngularJS $http service. It extends the $http service with the following features:

- A `Content-Type: application/json` HTTP header is automatically added to all HTTP requests.
- If a user is authenticated, their authentication token is automatically added to the `Authorization` HTTP header.
- All `string` parameters for HTTP GET requests are automatically encoded to URL-safe Base64 encoding as per RFC 4648. This is a regular Base64 encoding with the following differences:
  - - is used in place of +
  - '_' is used in place of /
  - All trailing = are trimmed.

Methods

- **encode(s)**

  This method is private.

  Encodes the given string with URL-safe Base64 encoding as per RFC 4648.

  Parameters

Param	Type	Details
s	string	The string to encode.

Returns

string	The string encoded with URL-safe Base64 encoding as per RFC 4648.

- **encodeAll(obj)**

  This method is private.

  Encodes all properties with type `string` in the provided object with URL-safe Base64 encoding as per RFC 4648. Properties that do not have the type `string` are ignored. Furthermore, this method does not support recursion; a `string` in an `Object` in the provided `Object`, for instance, will not be encoded.

  This method does not return anything; rather, the provided object is modified in place.

  Parameters

Param	Type	Details
obj	Object	The object to encode all strings in.

- **request(config)**

  Modifies the provided HTTP configuration object, applying the features as listed above. This HTTP configuration object follows the same format as the `config` parameter in the $http AngularJS service.

  Parameters

Param	Type	Details
config	Object	The HTTP configuration object to modify.

Returns

Object	The modified HTTP configuration object.

A.3.2 BACK_BUTTON

Defines the behavior of the back button for each routing state. Each property in the object has the following format:

- key - The internal routing name for the current state.
- value - The internal routing name for the state to transition to when the back button is pressed.

Type: `Object`

A.3.3 BASE_URL

Defines the base URL of the back end REST API. Changes to this constant affect all TravelMatch back end API calls throughout the entire app. API calls to third party APIs, such as Facebook, are not affected.

Type: `string`

A.3.4 DEBUG_URL

Defines the URL of the debugging server to use. Changes to this constant affect all `DebugService` calls.

Type: `string`

A.3.5 SHOW_SWIPE_DEBUG

Whether or not to show swipe debug overlay.

Type: `boolean`

A.3.6 STATE_NAMES

Defines the names of each routing state, to be shown in the header and menu. Each property in the object has the following format:

- key - The internal routing name for the state.
- value - A $translate ID for the name of the state.

Type: `Object`

A.3.7 USE_DEBUG

Whether or not to enable debugging functionality.

Type: `boolean`

A.3.8 USE_FRONT

Whether or not to include the front screen.

Type: `boolean`

A.3.9 AuthService

Manages user authentication data, including the storage and retrieval of tokens. User info is stored in local storage as an object with the following properties:

- `Identity` - The string that represents the user's identity.
- `Token` - The authentication token of the user.

Methods

- **existsUserInfo()**

  Checks if user info exists for one or more users.

  Returns

boolean	true if user info exists and is not null; otherwise, false.

- **getUserInfo()**

  Retrieves all stored user info.

  Returns

Object	The user info stored in local storage, or null if no user info was found.

- **isAuthenticated(id)**

  Checks whether a particular user is authenticated.

  Parameters

Param	Type	Details
id	string	The identity of the user to compare against.

  Returns

boolean	true if the given user is authenticated; otherwise, false.

- **logout()**

  Logs the user out and removes all user info. Internally, this is simply an alias for `removeUserInfo()`.

- **removeUserInfo()**

  Removes all user info.

- **setUserInfo(user, token)**

  Stores the given user info.

  Parameters

Param	Type	Details
user	string	The string that represents the identity of the user.
token	string	The authentication token of the user.

A.4 app.debug

The `app.debug` module contains all services that pertain to debugging of the app.

A.4.1 DebugService

Provides methods for remote debugging.

Methods

- **debug(log)**

  Posts the specified object to the debug server, if and only if `USE_DEBUG` is set to `true`.

  Parameters

Param	Type	Details
log	Object	The object to log.

  Returns

HttpPromise	A promise of the post. If `USE_DEBUG` is set to false, this method instead returns a promise that resolves immediately.

A.5 app.details

The `app.details` module contains all templates and controllers that pertain to the vacation details screen of the app.

A.5.1 DetailCtrl

The AngularJS controller for the vacation details screen.

Methods

- **checkForm()**

  This method is private.

  Validates the vacation details form to check if all fields are filled in correctly. If this is not the case, an error pop-up is shown.

  Returns

boolean	`true` if all fields are valid; otherwise, `false`.

- **init()**

  This method is private.

  Initializes this controller. Functionality of this method includes:

  – Displaying a pop-up with instructions on the first time ever opening this screen.
  – Retrieving the vacation details from the back end server.
  – Entering the vacation details of the last vacation into the model.
  – Checking the "Surprise me!" checkbox if the budget equals 0.
  – Showing an error pop-up if retrieval of the vacation details failed.

- **showInfo()**

  Shows a popup with instructions for the user.

- **submit()**

  This method is private.

  Validates and posts the vacation details to the back end API. If this is successful, the user is transferred to the interest analysis screen; otherwise, an error pop-up is shown.

Properties

- **vacation**

  Represents the model of the vacation details input fields.

  Object properties:

  – `start_date` - The `Date` of departure for the vacation.
  – `end_date` - The `Date` of return for the vacation.

  - `start_date_extend` - The amount of days that the `start_date` may be off.
  - `end_date_extend` - The amount of days that the `end_date` may be off.
  - `persons_adults` - The number of adults.
  - `persons_children` - The number of children.
  - `vac_id` - The vacation details ID for the current vacation details.
  - `noBudget` - Whether the user has checked `Surprise me!`
  - `budget` - The maximum budget of the user. If set to 0, the budget is not be taken into account.

Type: `Object`

A.5.2 VacationDetailsService

Provides methods for getting and setting the vacation details from the back end. The last vacation details received from the back end server are cached until new vacation details are obtained.

Methods

- **apiToVac()**

  This method is private.

  Converts an object received from the back end server API to a vacation details object.

- **clear()**

  Clears the cached set of vacation details.

- **createVacation(vac)**

  Creates new vacation details in the back end server with the specified parameters. The promise returned by this method is resolved if the vacation details were successfully created; otherwise, it is rejected with the translation ID of the error that occurred.

  If the vacation details were created successfully, they are added to the cached set of vacation details.

  Parameters

Param	Type	Details
vac	Object	The vacation details. This object has the following properties: – `name` - The name of the vacation details. – `startDate` - The `Date` of departure. – `startRange` - The number of days that the `startDate` may be off. – `endDate` - The `Date` of return. – `endRange` - The number of days that the `endDate` may be off. – `adults` - The number of adults. – `children` - The number of children. – `budget` - The budget for the vacation. If set to 0, the budget is not taken into account.

Returns

Promise	A promise of the vacation details creation.

- **currentVacations()**

  Gets the last set of vacation details that were retrieved or sent to the back end server.

  Returns

Array.<Object>	The set of vacation details, or an empty array if no vacation details are available. Each object has the following properties: – `id` - The ID of the vacation details. – `name` - The name of the vacation details. – `swipes` - The number of swipes left before a new recommendation can be obtained. – `startDate` - The `Date` of departure. – `startRange` - The number of days that the `startDate` may be off. – `endDate` - The `Date` of return. – `endRange` - The number of days that the `endDate` may be off. – `adults` - The number of adults. – `children` - The number of children. – `budget` - The budget for the vacation. If set to 0, the budget is not taken into account.

- **deleteVacation(id)**

  Deletes the vacation details with the specified ID in the back end server. The promise returned by this method is resolved if the vacation details were deleted successfully; otherwise, it is rejected with the translation ID of the error that occurred.

  If the vacation details with the specified ID exist in the set of cached vacation details, then they are deleted from the set.

  Parameters

Param	Type	Details
id	number	The ID of the vacation details to delete.

  Returns

Promise	A promise of the vacation details deletion.

- **findVacation(id)**

  This method is private.

  Finds the index of the vacation details with the specified ID in the cached set of vacation details.

  Parameters

Param	Type	Details
id	number	The ID of the vacation details to find.

Returns

number	The index of the vacation details in the cached set of vacation details, or `-1` if the vacation details were not found.

- **getLatestVacation()**

  Retrieves the ID of the last saved vacation details from the back end server. The promise returned by this method is resolved if the ID of the last saved vacation details were retrieved successfully; otherwise, it is rejected with the translation ID of the error that occurred.

  Returns

Promise	A promise of the last saved vacation details retrieval.

- **getVacations()**

  Retrieves the set of vacation details of the current user from the back end server. Additionally, this method also retrieves the latest vacation details that were saved. The promise returned by this method is resolved if both retrievals succeeded; otherwise, it is rejected with the translation ID of the error that occurred.

  Returns

Promise	A promise of the vacation details retrieval.

- **latestVacation()**

  Returns the vacation details that were last saved, or `null` if the last saved vacation details have not been retrieved from the back end server yet.

  Returns

Object	The last saved vacation, with the following properties, or `null`:

- **setVacations(vacs)**

  This method is private.

  Sets the cached set of vacation details to the specified set of vacation details.

  Parameters

Param	Type	Details
vacs	Array.<Object>	The set of vacation details.

- **updateVacation(id, vac)**

  Updates the vacation details with the specified ID in the back end server with the specified parameters. The promise returned by this method is resolved if the vacation details were successfully updated; otherwise, it is rejected with the translation ID of the error that occurred.

  If the vacation details with the specified ID exist in the set of cached vacation details, then they are updated in the set.

Parameters

Param	Type	Details
id	number	The ID of the vacation details to update.
vac	Object	The vacation details. This object has the following properties: – `name` *(optional)* - The name of the vacation details. – `startDate` *(optional)* - The `Date` of departure. – `startRange` *(optional)* - The number of days that the `startDate` may be off. – `endDate` *(optional)* - The `Date` of return. – `endRange` *(optional)* - The number of days that the `endDate` may be off. – `adults` *(optional)* - The number of adults. – `children` *(optional)* - The number of children. – `budget` *(optional)* - The budget for the vacation. If set to 0, the budget is not taken into account.

Returns

Promise	A promise of the vacation details update.

- **vacToApi()**

  This method is private.

  Converts the specified vacation details to the object format requested by the back end server API.

Properties

- **latestVacationId**

  This property is private. Use `latestVacation().vac_id`.

  The ID of the last vacation details that were saved that was last retrieved from the back end server, or `null` if the last vacation details were never retrieved.

  Type: `number`

- **vacations**

  This property is private. Use `currentVacations()`.

  The last set of vacation details retrieved from the back end server.

  Type: `Array.<Object>`

A.6 app.front

The `app.front` module contains all templates and controllers that pertain to the front screen.

A.6.1 FrontCtrl

The AngularJS controller for the front screen.

Methods

- **checkUser()**

  Checks if user is already logged in, and if so, forwards the user to the vacation details screen. This method is called upon initialization of the controller.

- **continueWithoutAccount()**

  Sends the device ID to the back end server to continue as guest. If the device ID is undefined, or guest account authentication failed, an error pop-up is shown. Otherwise, the user obtains a guest account and is forwarded to the vacation details screen.

- **fbLogin()**

  Performs a Facebook login.

A.7 app.hotel.overview

The `app.hotel.overview` module contains all templates and controllers that pertain to the hotel overview screen.

A.7.1 HotelOverviewCtrl

The AngularJS controller for the hotel overview screen.

Methods

- **furtherAnalysis()**

  Redirects the user to the interest analysis screen.

- **init()**

  Initializes this controller. Functionality of this method includes:

  – Entering the last retrieved recommendations into the model.
  – Retrieving a set of recommendations if no cached recommendations are currently available.
  – Showing an error pop-up if retrieval of recommendations failed.

- **openInBrowser(link)**

  Opens the specified link in the device's standard external browser.

  Parameters

Param	Type	Details
link	string	The link to open.

- **showAdvice(index)**

  Displays the recommendation at the specified index in the model. If this is the first time that the recommendation is opened, the event is also recorded in the analytics.

  Parameters

Param	Type	Details
index	number	The index of the recommendation.

- **showDescription(event)**

  Expands the description of the clicked trip offer, and closes any other open descriptions.

  Parameters

Param	Type	Details
event	Event	The $event object received from `ngClick`.

Properties

- **recommendations**

  The model for the trip offers shown in the hotel overview screen. Each object in the array has the following properties:

  - `location` - The location of the recommendation, with the following properties:
    - `city_name` - The name of the city.
    - `region_name` - The name of the region.
    - `country_name` - The name of the country.
  - `offers` - The trip offers associated with the recommendation, with the following properties:
    - `offer_id` - The ID of the trip offer.
    - `name` - The name of the trip offer.
    - `description` - The description of the trip offer.
    - `price` - The price of the trip offer, in euro cents.
    - `link` - The affiliate link to book the trip offer.
    - `image` - The URL for an image of the trip offer.
    - `hotel_stars` - The Hotelstars rating for the trip offer.
    - `min_people` - The minimum number of people for the trip offer.
    - `dept_date` - The departure date for the trip offer.
    - `duration_days` - The duration, in days, for the trip offer.
    - `user_rating` - The user rating for the trip offer.

  Type: `Array.<Object>`

- **selected**

  The index of the currently selected recommendation in the `recommendations` property.

  Type: `number`

A.7.2 HotelService

Provides methods for getting and setting the hotel recommendations and hotel info from the back end. The last recommendations received from the back end server are cached until new recommendations are obtained.

Methods

- **deleteRecommendation(recId)**

  Deletes the cached recommendation at the specified index in the back end server. The promise returned by this method is resolved if the deletion was successful; otherwise, it is rejected with the translation ID of the error that occurred.

  Note: If the deletion is successful, the recommendation is also deleted from the cached recommendations. This causes the index of all recommendations that follow it to be updated!

  Parameters

Param	Type	Details
recId	number	The index of the recommendation to delete.

  Returns

Promise	A promise of the recommendation deletion.

- **getNewRecommendations(vacId, n)**

  Gets new recommendations from the back end server and caches them in the hotel service. The promise returned by this method is resolved if the hotel recommendations were successfully retrieved; otherwise, it is rejected with the translation ID of the error if the retrieval failed.

  Parameters

Param	Type	Details
vacId	number	The ID of the vacation details to take into account.
n	number	The number of recommendations to retrieve.

  Returns

Promise	A promise of the recommendations retrieval.

- **lastRecommendations()**

  Gets the last recommendations that were retrieved from the back end server.

  Returns

Array.<Object>	The recommendations.

- **loadRecommendation(locId, renew)**

Loads a recommendation for the specified location from the back end server. An optional parameter `renew` indicates whether the trip offers should be renewered, or the previous trip offers should be returned. The promise returned by this object is resolved if the retrieval was successful; otherwise, it is rejected with the translation ID of the error that occurred.

Parameters

Param	Type	Details
locId	number	The ID of the location to get new trip offers for.
renew	boolean	Whether to renew the trip offers. *(default: false)*

Returns

Promise	A promise of the trip offers retrieval.

- **recommendationCount()**

  Gets the number of recommendations currently cached in the HotelService.

  Returns

number	The number of recommendations.

- **saveRecommendation(recId)**

  Saves the cached recommendation at the specified index in the back end server. The promise returned by this object is resolved if the storage was successful; otherwise, it is rejected with the translation ID of the error that occurred.

  Parameters

Param	Type	Details
recId	number	The index of the recommendation to save.

Returns

Promise	A promise of the recommendation storage.

- **setRecommendations(recs)**

  This method is private.

  Sets the cached recommendations to the specified recommendations and returns a boolean that indicates whether the storage succeeded or failed.

  Parameters

Param	Type	Details
recs	Array.<Object>	The recommendations.

Returns

boolean	`true` if the storage succeeded; otherwise, `false`.

- **validateOffers(offers)**

  This method is private.

  Validates the specified trip offers, checking if the trip offers match the specification.

  Parameters

Param	Type	Details
offers	Array.<Object>	The array of offers.

Returns

boolean	`true` if the offers are valid; otherwise, `false`.

- **validateRecommendations(recs, expectedCount)**

  This method is private.

  Validates the specified recommendations, checking if the recommendations match the specification.

  Parameters

Param	Type	Details
recs	Array.<Object>	The array of recommendations.
expectedCount	number	The minimum number of recommendations expected. *(default: 0)*

Returns

boolean	`true` if the recommendations are valid; otherwise, `false`.

Properties

- **recommendations**

  This property is private. Use `lastRecommendations()`.

  An array of the last recommendations retrieved from the back end server. Each object has the following properties:

  – `location` - The location of the recommendation, with the following properties:
    * `city_name` - The name of the city.
    * `region_name` - The name of the region.
    * `country_name` - The name of the country.

  - `offers` - The trip offers associated with the recommendation, with the following properties:
    * `offer_id` - The ID of the trip offer.
    * `name` - The name of the trip offer.
    * `description` - The description of the trip offer.
    * `price` - The price of the trip offer, in euro cents.
    * `link` - The affiliate link to book the trip offer.
    * `image` - The URL for an image of the trip offer.
    * `hotel_stars` - The Hotelstars rating for the trip offer.
    * `min_people` - The minimum number of people for the trip offer.
    * `dept_date` - The departure date for the trip offer.
    * `duration_days` - The duration, in days, for the trip offer.
    * `user_rating` - The user rating for the trip offer.

  Type: `Array.<Object>`

A.8 app.login

The `app.login` module contains all templates, controllers and services that pertain to the login screen of the app.

A.8.1 LoginCtrl

The AngularJS controller for the login screen. The login screen currently supports logging in with e-mail address and password, or by connecting with Facebook.

Methods

- **fbLogin()**

  Performs a Facebook login.

- **init()**

  This method is private.

  Initializes this controller. Functionality of this method includes:

  - Setting the value of the e-mail field to the last e-mail address used for successful registration.

- **login()**

  Performs a login with the current values of the e-mail address and password input fields. If the login fails, an error pop-up is shown; otherwise, a success pop-up is shown and afterwards the user is transferred to the vacation details screen. This method also clears the password input field.

Properties

- **data**

  Represents the model of the credentials input fields.

  Object properties:

  - `username` - The value of the e-mail address input field.
  - `password` - The value of the password input field.

  Type: `Object`

A.8.2 facebookButton

A Facebook login button that can be used to authenticate TravelMatch with Facebook. If clicked, the user is asked to authorize the TravelMatch app and an account is made in the back end server. After this, the user is logged in.

Usage

as attribute

```
<ANY facebook-button>
   ...
</ANY>
```

Directive info

- This directive creates new scope.

Methods

- **login()**

  Performs a login via Facebook. If the Facebook SDK has not been loaded, then an error pop-up is shown instead.

  The login function is called in the Facebook SDK, which opens Facebook in a new window. Here, the user is asked to log in to Facebook if not already logged in, and then is asked to authorize the TravelMatch app to use their account.

  Successful Facebook authorization is followed by an API call to the back end server to create a new user in the database with the Facebook ID obtained from the Facebook API, and/or obtain the existing TravelMatch authentication token for that user.

  If both Facebook authorization and TravelMatch registration/login are successful, a success pop-up is shown and afterwards the user is transferred to the vacation details screen. If either failed, an error pop-up is shown.

  This method requests the following Facebook permissions:

  - `public_profile` - Included by default in any Facebook app authorization request. Needed to obtain the app-specific Facebook ID as well as the Facebook authentication token, which is verified in the back end to confirm the authorization.

  Returns

Promise	A promise of the Facebook login.

A.8.3 LoginService

Provides methods for logging in to TravelMatch accounts in the back end.

Methods

- **loginUser(id, token, use)**

  Performs a login action in the back end API using the specified authentication provider, with the specified identity string and token.

  Facebook Login with this method is only availabld if the Facebook ID and token are known beforehand.

Parameters

Param	Type	Details
id	string	The string that represents the identity of the user, i.e. the user's e-mail address or Facebook app-specific ID.
token	string	A token that provides authentication of the user, i.e. the user's password or Facebook authentication token.
use	string	The authentication provider to use. This parameter can have the following values: – `email` - Use a combination of e-mail address and password to log in. – `facebook` - Use a combination of Facebook app-specific ID and authentication token to log in. **This value is deprecated.**

Returns

Promise	A promise of the user login action that is resolved if the login was successful or rejected with the translation ID of the error if the login failed.

A.9 app.module

The `app.module` module contains all templates and controllers that pertain to the vacation details screen of the app.

A.10 app.navigation

The `app.navigation` module contains all templates and directives that pertain to the header bar, menu sidebar and navigation of the app.

A.10.1 MainCtrl

The AngularJS controller for the persistent elements of the app, which includes the header bar and menu sidebar. This controller adds functions to the scope that allow any element that inherits from it to open or close the menu sidebar at will.

Methods

- **goBack()**

  Sends the user back to the state taken from `BACK_BUTTON`. If the current state has no back button state, then the user will be sent to the previous state.

- **hideMenu()**

  Hides the menu sidebar, if it is currently open.

- **isMenuOpen()**

  Checks whether the menu sidebar is currently open or closed.

  Returns

boolean	`true` if the menu sidebar is currently open; `false` if the menu sidebar is currently closed.

- **showMenu()**

  Shows the menu sidebar, if it is currently closed.

- **toggleMenu()**

  Toggles the display of the menu sidebar: the menu sidebar is opened if it is currently closed, or closed if it is currently open.

A.10.2 tmHeader

A persistent header bar that is shown at the top of every screen in the app. This header shows the title of the current state of the app, the menu button, and a back button if the current state supports it. The header bar is hidden upon a transition to a full screen state, and adds a toggle button to open or close it in such a state.

Usage

as element:

```
<tm-header>
</tm-header>
```

Methods

- **addStateListeners()**

  This method is private.

  Adds a listener to the `$stateChangeStart` event in the `$rootScope` that is fired upon starting a transition to a different state. This listener prepares the header for the transition to that state.

- **getTitle()**

  Gets the title of the current state. This title is taken from `STATE_NAMES`.

  Returns

string	The title of the current state.

- **init()**

This method is private.

Initializes this directive. Functionality of this method includes:

- Registering the native back button.
- Adding state listeners.
- Preparing for the current state.

- **isFullScreenState()**

  Checks whether the current state is a full screen state. Only the `tm.main.swipe` state is a full screen state, so this method simply compares the name of the current state to that.

  Returns

boolean	`true` if the current state is a full screen state; otherwise, `false`.

- **isHeaderlessState()**

  Checks whether the current state is a headerless state. Only the `tm.front` state is a full screen state, so this method simply compares the name of the current state to that.

  Returns

boolean	`true` if the current state is a headerless state; otherwise, `false`.

- **prepareState(toState)**

  This method is private.

  Prepares the header for a transition to the specified state. If the state is a full screen state, then the header is hidden and the toggle button displayed. Also, the back button is displayed if the state has a back button state defined.

  Parameters

Param	Type	Details
toState	string	The state that the app is about to transition to.

- **registerBackButton()**

  This method is private.

  This method registers the function of the native back button on devices that support it, such as Android-based devices. If the native back button is pressed, the app transitions to the back button state as defined in BACK_BUTTON. If no back button state has been defined for the current state, then the back button exits the app.

- **toggleHeader()**

  Toggles the display of the header: hiding it if the header is visible, or showing it if the header is not visible. Additionally, if the menu sidebar is open, this method also closes it.

A.10.3 tmLoginTabs

The tabs shown at the top of the login and registration screens, that allow the user to switch between the two screens. This directive provides no further content.

Usage

as element:

```
<tm-login-tabs>
</tm-login-tabs>
```

A.10.4 tmMenu

A persistent menu sidebar that slides in from the right, with a set of options that can change depending on whether the user is authenticated or not.

Usage

as element:

```
<tm-menu>
</tm-menu>
```

Methods

- **isLoggedIn()**

  Checks whether the user is currently authenticated or not.

  Returns

boolean	`true` if the user is authenticated; otherwise, `false`.

- **logout()**

  Hides the menu sidebar, removes all authentication data of the current user and shows a pop-up message notifying the user that they have been logged out. Afterwards, the user is transferred to the front screen, or to the login screen if `USE_FRONT` is `false`.

Properties

- **states**

  An array of menu option objects that can be shown in the menu sidebar. The menu options are shown in the same order that they are defined in this array. Each menu option object has the following properties:

  – `state` - The state to transition to when this menu option is pressed.
  – `name` - The translation ID for the name to show on the menu option. If this name is `undefined`, then the translation ID is taken from STATE_NAMES instead.
  – `needAuth` - Controls when the menu option is displayed based on whether the user is authenticated or not. This property can take one of the following values:
    * `true` - The menu option is only displayed when the user is authenticated.
    * `false` - The menu option is only displayed when the user is not authenticated.
    * `undefined` - The menu option is always displayed, regardless of whether the user is authenticated or not.
  – `click` - The name of the function to call when the menu option is selected. This function must be defined in the scope of this directive. If this property is undefined, then the menu sidebar is hidden when the menu option is selected.

  Type: `Array.<Object>`

A.10.5 tmMenuButton

The menu button used in the header bar. This directive provides no further content.

Usage

as element:

```
<tm-menu-button>
</tm-menu-button>
```

A.11 app.registration

The `app.registration` module contains all templates, controllers and services that pertain to the registration screen of the app.

A.11.1 RegistrationCtrl

The AngularJS controller for the registration screen. The registration screen currently supports registering with an e-mail address and password.

Methods

- **register()**

  Performs an account registration with the current values of the e-mail address and both password input fields. If the registration fails, an error pop-up is shown. Otherwise, a success pop-up is shown telling the user to check their e-mail for the activation link, and afterwards the user is transferred to the login screen. This method also clears the password input fields.

Properties

- **data**

  Represents the model of the credentials input fields.

  Object properties:

  - `username` - The value of the e-mail address input field.
  - `password` - The value of the first password input field.
  - `password2` - The value of the second password input field.

  Type: `Object`

A.11.2 RegistrationService

Provides methods for registering TravelMatch accounts in the back end.

Methods

- **FBRegister(fbuser, fbtoken)**

  Registers a new user in the back end server with the specified Facebook app-specific ID and Facebook authentication token. The promise returned by this method is resolved if the registration was successful, or rejected with the translation ID of the error that occurred if the registration failed.

  Parameters

Param	Type	Details
fbuser	string	The Facebook app-specific ID of the user.
fbtoken	string	The Facebook authentication token of the user.

Returns

Promise	A promise of the registration.

- **lastEmail()**

  Returns the last e-mail address used for successful registration in this app session.

  Returns

string	The e-mail address, or `null` if no e-mail registration succeeded in this session.

- **registerUser(email, pw, pw2)**

  Registers a new user in the back end server with the specified e-mail address and password. The promise returned by this method is resolved if the registration was successful, or rejected with the translation ID of the error that occurred if the registration failed. After registration, the user must click an activation link in their e-mail before they may login.

  Parameters

Param	Type	Details
email	string	The e-mail address of the user.
pw	string	The desired password of the user.
pw2	string	A repeat of the desired password, for validation purposes.

Returns

Promise	A promise of the registration.

Properties

- **email**

  This property is private. Use email().

  The last e-mail address used for successful registration in this session.

  Type: `string`

A.12 app.swipe

The `app.swipe` module contains all templates, controllers, services and directives that pertain to the interest analysis screen of the app.

A.12.1 SwipeCtrl

The AngularJS controller for the interest analysis screen.

Methods

- **checkNeedsMore()**

  This method is private.

  Checks whether the image buffer contains enough images to finish the current interest analysis without having to retrieve additional images. Additionally, this method updates $scope.needsMore.

  Returns

boolean	`true` if more images are needed; otherwise, `false`.

- **imageError(data)**

  This method is private.

  Displays the specified error message in a pop-up, then transfers the user to the vacation details screen.

  Parameters

Param	Type	Details
data	string	The translation ID of the error message.

- **imageSuccess(images)**

  This method is private.

  Receives images from the ImageService and stores them in the image buffer in the model. If no images are supplied, an error pop-up is shown instead.

  Parameters

Param	Type	Details
images	Array.<Object>	An array of image objects, with the following properties: – `id` - The ID of the image. – `url` - The URL of the image. – `width` - The width of the image, in pixels. – `height` - The height of the image, in pixels.

- **init()**

  Initializes this controller. Functionality of this method includes:

  - Redirecting the user to the vacation details screen if the vacation details are unavailable.
  - Retrieving the initial set of images.
  - Setting the `limit` of the current interest analysis.

- **onSwipe(imageId, choice)**

  Posts a (dis)like for the specified image to the back end and retrieves the next image.

  Parameters

Param	Type	Details
imageId	number	The image ID of the image that was (dis)liked.
choice	boolean	`true` to post a like; `false` to post a dislike.

Properties

- **currentProgress**

  The current progress of the interest analysis.

  Type: `number`

- **images**

  An array of image objects to use as the model for the swipeable images, with the following properties:

  - `id` - The ID of the image.
  - `url` - The URL of the image.
  - `width` - The width of the image, in pixels.
  - `height` - The height of the image, in pixels.

  Type: `Array.<Object>`

- **isDone**

Whether the current interest analysis is done. This property is updated whenever checkNeedsMore is called.

Type: `boolean`

- **limit**

  The amount of images shown per interest analysis.

  Type: `number`

- **needsMore**

  Whether the current interest analysis requires more images to be loaded from the server in order to finish. This property is updated whenever checkNeedsMore is called.

  Type: `boolean`

A.12.2 tmPhoto

A container for the swiping interface in the interest analysis screen. This directive is powered by an image buffer from which the images to swipe are drawn. The top image in the buffer can be dragged to the left or right to indicate a dislike or like respectively. This directive also generates buttons on the bottom that can be used to (dis)like an image. Upon (dis)liking an image, a callback function can be called.

Usage

as element:

```
<tm-photo
       images="{Array.<Object>}"
       onchoice="{expression}"
       needsmore="{boolean}"
       isdone="{boolean}">
</tm-photo>
```

Directive info

- This directive creates new scope.

Param	Type	Details
images	Array.<Object>	An array of image objects to use as the model for the swipeable images, with the following properties: • `id` - The ID of the image. • `url` - The URL of the image. • `width` - The width of the image, in pixels. • `height` - The height of the image, in pixels.

Param	Type	Details
onchoice	expression	The callback expression to execute in the controller when the user (dis)likes an image. This function takes the following arguments: • `image_id` - The image ID of the image that was (dis)liked. • `choice` - `true` if the image was liked; `false` if the image was disliked.
needsmore	boolean	A boolean that indicates whether more pictures will be supplied.
isdone	boolean	A boolean that indicates whether all pictures have been displayed.

Parameters

Methods

- **finished()**

  This method is private.

  Called when interest analysis is complete. Shows calculating image and text.
- **nextImage()**

  Sets a timer of 300ms to remove the current swipe image from the image buffer and enable the next one.
- **onDrag()**

  This method is private.

  Called on every step of a dragging motion on the swipe image.
- **onDragStart()**

  This method is private.

  Initializes a dragging motion on the swipe image.
- **onDragStop()**

  This method is private.

  Ends a dragging motion on the swipe image. If the image was dragged beyond 1/4th of the screen or was dragged at a delta of 1/100th of the screen, it is swiped to the left or right depending on its direction. If neither event occurred or both occurred in opposite directions, the swipe image moves back to its initial position.
- **refresh()**

  Refreshes the current swipe image and background image with the first two images currently in the image buffer.

- **swipe(choice)**

  Moves the current swipe image off the screen and retrieves the next image. If `onchoice` is defined, it is called with the image ID of the first image in the image buffer and the `choice` parameter.

  If `choice` is `true` then the image is moved to the right; otherwise, it is moved to the left.

  Parameters

Param	Type	Details
choice	boolean	`true` to indicate a like; `false` to indicate a dislike.

A.12.3 ImageService

Provides methods for receiving images from the back end and recording choices in the back end.

Methods

- **acceptImages(data)**

  This method is private.

  Accepts images from the back end server and filters out all invalid images. For every image, the optimal size for this device's screen is selected. This method selects the largest size that is smaller than this device's resolution; this is calculated by multiplying width by height. If no suitable size could be found, the smallest possible size is selected instead.

  Parameters

Param	Type	Details
data	Array.<Object>	An array of image objects with multiple size objects, obtained from the back end API.

  Returns

Array.<Object>	An array of the chosen image objects, with the following properties: – `id` - The ID of the image. – `url` - The URL of the chosen image size. – `width` - The width of the chosen image size, in pixels. – `height` - The height of the chosen image size, in pixels. If no valid images were found, an empty array is returned.

- **get(n)**

  Retrieves the specified number of images. The promise returned by this method is resolved if the retrieval was successful or rejected with the translation ID of the error if the retrieval failed. If the promise resolves, an object with following properties is passed:

  - `id` - The ID of the image.
  - `url` - The URL of the chosen image size.
  - `width` - The width of the chosen image size, in pixels.
  - `height` - The height of the chosen image size, in pixels.

Parameters

Param	Type	Details
n *(optional)*	number	The amount of images to retrieve. Must be at least 1 and at most 100. If out of range, the closest value within range is chosen instead. *(default: 1)*

Returns

Promise	A promise of the image retrieval action that

- **httpError(deferred, data)**

  This method is private.

  Rejects the provided deferred object with the matching translation ID of an error message based on the HTTP status code received from the back end.

  Parameters

Param	Type	Details
deferred	Object	The deferred object received from the caller.
data	Object	The data received from the back end.

- **httpSuccess(deferred, data, parser)**

  This method is private.

  Parses the provided data with the provided parser function, then resolves the provided deferred object with the parsed data.

  Parameters

Param	Type	Details
deferred	Object	The deferred object received from the caller.
data	Object	The data received from the back end.
parser	function	The parser function to be used to parse the data received from the back end.

- **initial()**

Retrieves the initial set of images for the specified vacation details.

Equivalent to calling `get(5)`.

Returns

Promise	A promise of the image retrieval action that is resolved if the retrieval was successful or rejected with the translation ID of the error if the retrieval failed.

- **next(imageId, like, isLast)**

  Posts the (dis)like choice of the current image to the back end and optionally retrieves the new image. The promise returned by this method is resolved if the action was successful or rejected with the translation ID of the error if the retrieval failed.

  Parameters

Param	Type	Details
imageId	number	The ID of the image that was (dis)liked.
like	boolean	`true` if the user liked the specified image; otherwise, `false`.
isLast *(optional)*	boolean	If `true`, this method will not retrieve any new images. *(default: false)*

  Returns

Promise	A promise of the choice post and image retrieval action.

A.13 app.user.details

The `app.user.details` module contains all templates, controllers and services that pertain to the user details screen.

A.13.1 UserDetailCtrl

The AngularJS controller for the user details screen.

Methods

- **save()**

  Posts the current values of the user data input fields to the back end. If the post fails, an error pop-up is shown; otherwise, a success pop-up is shown and afterwards the user is transferred to the interest analysis screen.

Properties

- **info**

The model for the user info input fields, with the following properties:

- `name` - `string` - The name of the user.
- `gender` - `string` - The gender of the user. Can be one of the following values:
  - `none`
  - `male`
  - `female`
- `birthday` - `Date` - The birth date of the user.

Type: `Object`

A.13.2 UserDetailsService

Provides methods for getting and setting the user info from the back end.

Methods

- **get()**

  Gets the user info from the back server. The promise returned by this method is resolved if the retrieval was successful or rejected with the translation ID of the error if the retrieval failed. If the promise resolves, an object with the following properties is passed:

  - `name` - `string` - The name of the user.
  - `gender` - `string` - The gender of the user. Can be one of the following values:
    - `none`
    - `male`
    - `female`
  - `birthday` - `Date` - The birth date of the user.

  Returns

Promise	A promise of the user info get action.

- **put(info)**

  Puts the user data on the back end service. The promise returned by this method is resolved if the storage was successful or rejected with the translation ID of the error if the storage failed.

  Parameters

Param	Type	Details
info	Object	A user info object with the following optional properties: – `name` - `string` - The name of the user. – `gender` - `string` - The gender of the user. Can be one of the following values: ∗ `none` ∗ `male` ∗ `female` – `birthday` - `Date` - The birth date of the user.

Returns

Promise	A promise of the user info put action.

Appendix B
Back end documentation

B.1 Documentation template

Each object in the back end documentation is documented with the following template:

Name of class

A description of the class, with an inheritance diagram.

Public Member Functions (if applicable)

- def **functionName** (self, parameterName)
  Description of the function.

Private Member Functions (if applicable)

- def **functionName** (self, parameterName)
  Description of the function.

Private Attributes (if applicable)

- **_attribute_name**

Constructor & Destructor Documentation

def __init__ (self) A description of the constructor.

Member Function Documentation (if applicable)

def _function_name (self, parameter_name, optional_parameter_name = *default_value* **)** [private] A description of the function.
Parameters (if applicable)

parameter_↩ name	A description of the parameter.
<parameter_↩ name>	The type of the parameter.
optional_↩ parameter_↩ name	A description of the optional parameter.
<optional_↩ parameter_↩ name>	The type of the parameter.

default_value	A description of the default value. Default: the default value.

Returns (if applicable)

The return type

Member Data Documentation (if applicable)

_data_name `[private]` The documentation for this class was generated from the following file:

- directory/**file.py**

B.2 affiliate

`affiliate` is a namespace that contains classes, variables and functions that relate to the affiliate networks components. This namespace mainly acts as a container for several other namespaces, namely those relating to the affiliate parsers, the models, serializers and views.

Namespaces

- **affiliate_parser**
- **models**
- **serializer**
- **tradetracker**
- **views**

B.3 affiliate.affiliate_parser

`affiliate.affiliate_parser` is a namespace that contains classes, variables and functions that relate to affiliate network parsers. This namespace contains an abstract parser, which can be extended to parse feeds from any supported affiliate network. Extenders should define a `parser_name` and set up an entry mapping in the constructor to adapt the base parser to a specific feed.

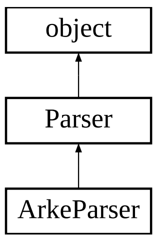
Classes

- class **Parser**

  This class represents a parser object in affiliate.

B.3.1 Parser

This class represents a parser object in affiliate.
Inheritance diagram for Parser:



Public Member Functions

- def __**init**__ (self)

  Initialize all required dictionaries and lists.
- def **process_single** (self, feed_url)

  Processes a single url.

Private Member Functions

- def **_store_entry** (self)

  Stores all information inside the dictionaries inside the model.
- def **_find_and_add_all_xml_attributes** (self, xml_keys_required, entry)

  Finds the required xml variables using the xml-attribute syntax in the entry and stores their attribute them in the dictionary.
- def **_find_and_add_all_xml_elements** (self, xml_keys_required, entry)

  Finds the required xml variables using the xml-elements syntax in the entry and stores their attribute them in the dictionary.
- def **_get_root** (self, feed_url)

  Opens a URL and returns the root of its elementTree.
- def **_add_entry**

  Adds an entry to the dictionaries of this object.
- def **_check_for_discard** (self)

  Checks if the current entry should be discarded.
- def **_get_correct_attribute_value** (self, model_field_name)

  Gets the attribute value corresponding to the given model_field_name.
- def **_get_correct_date_format** (self, date)

  Make the format of the date as Django accepts.
- def **_is_rep_ok** (self)

  Checks if all must have model variables have been initialized.

Private Attributes

- **_model_variables**
- **_must_have_model_variables**
- **_model_to_attributes**
- **_xml_to_model**

Detailed Description

This class represents a parser object in affiliate.

Constructor & Destructor Documentation

def __init__ (self) Initialize all required dictionaries and lists.

Member Function Documentation

def _add_entry (self, model_field_name, xml_name, not_found_value = *None* **)** `[private]`

Adds an entry to the dictionaries of this object.

The variables given in xml_name will be sought for in the url and stored in the database at the given model_field variable.

Parameters

model_field_↩ name	The name of the model field variable.
<model_field↩ _name>	basestring
xml_name	The EXACT xml name inside the description. Input 'None' if value needs not to be fetched from the XML feed.
<xml_name>	basestring
not_found_↩ value	Default value which is stored in the model when not found. Default: None.

Returns

Void

def _check_for_discard (self) `[private]`

Checks if the current entry should be discarded.

This happens when a must have model variable was None (i.e. not found and no default).

Returns

Boolean whether the entry should be sicarded.

def _find_and_add_all_xml_attributes (self, xml_keys_required, entry) `[private]`

Finds the required xml variables using the xml-attribute syntax in the entry and stores their attribute them in the dictionary.

Parameters

xml_keys_↩ required	All XML keys whereof the attributes needs to be added.
<xml_keys_↩ required>	String[]
entry	An entry of the xml file which needs to be checked.
<entry>	eTree

def _find_and_add_all_xml_elements (self, xml_keys_required, entry) `[private]`

Finds the required xml variables using the xml-elements syntax in the entry and stores their attribute them in the dictionary.

Parameters

xml_keys_↩ required	All XML keys whereof the attributes needs to be added.
<xml_keys_↩ required>	String[]
entry	An entry of the xml file which needs to be checked.

<entry>	eTree
	
	`Finds all needed xml keys in the XML elements syntax and adds them to`
	`the attributes ADT.`
	
	`:param xml_keys_required: All XML keys whereof the attributes needs to be added.`
	`:param entry: The entry to be checked.`
	`:return: Void`

def _get_correct_attribute_value (self, model_field_name) `[private]`
Gets the attribute value corresponding to the given model_field_name.
Contains extra input checking for inputs in the wrong format.
Parameters

model_field_↩ name	The variable name of the model_field_variable.
<model_field↩ _name>	basestring

Returns

The corresponding attribute value of the model_field_name

```
Gets the attribute value corresponding to the given model_field_name.

:param model_field_name: The name of the model field variable.
:return: The corresponding attribute value.
```

def _get_correct_date_format (self, date) `[private]`
Make the format of the date as Django accepts.
Parameters

date	The date or datetime to be formalized in a correct format.
<date>	basestring

Returns

The date in a correct format.

```
Make the format of the date as Django wants.

:param date: The date (or datetime) to be formalized in a correct format.
:return: Returns date in the correct Django format.
```

def _get_root (self, feed_url) `[private]`
Opens a URL and returns the root of its elementTree.
Parameters

feed_url	The feed url which needs to be opened
<feed_url>	URL

Returns

Root of the elementtree of the feed_url.

```
Opens a URL and returns the root
```

def _is_rep_ok (self) `[private]`
Checks if all must have model variables have been initialized.

def _store_entry (self) `[private]`
Stores all information inside the dictionaries inside the model.
Returns

A boolean whether it is was a success or failure.

def process_single (self, feed_url) Processes a single url.
The required data will be retrieved from the URL and stored in the database.

Parameters

feed_url	The feed url which needs to be parsed.
<feed_url>	URL

Returns

Void

```
Processes a single feed and makes a database entry.
:param feed_url: The feed url to be processed.
:return: Void
```

Member Data Documentation

_model_to_attributes [private]

_model_variables [private]

_must_have_model_variables [private]

_xml_to_model [private]

The documentation for this class was generated from the following file:

- travelmatch/affiliate/**affiliate_parser.py**

B.4 affiliate.models

`affiliate.models` is a namespace that contains classes, variables and functions that relate to the affiliate network models. This namespace contains all the models for the retrieval and storage of affiliate feeds, as well as the feeds and parsers themselves.

Classes

- class **AbstractParserModel**

  The abstract of a parser which can process a URL.
- class **AffiliateFeed**

  A feed consists of the url feed an a parser to process the url.
- class **ArkeParserModel**

  The specific parser for Arke.
- class **Trip**

  Represents a trip users can book.

Functions

- def **parse_the_feed** (sender, instance=None, args, kwargs)

  This method pass the feed to the parser.

Function Documentation

def affiliate.models.parse_the_feed (sender, instance = *None***, args, kwargs)** This method pass the feed to the parser.

Parameters

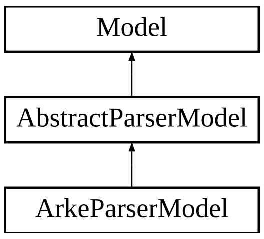
instance	The feed that needs to be parsed
<*instance*>	object
sender	not used
args	not used
kwargs	not used

Precondition

instance is not None : instance being parsed

B.4.1 AbstractParserModel

The abstract of a parser which can process a URL.
Inheritance diagram for AbstractParserModel:



Public Member Functions

- def **process_single** (self, url)
  Processes the given URL and stores the info inside the URL into the database.
- def **__unicode__** (self)
  This method make sure the object is stored and retrieved in certain format:
- def **name** (self)
  Returns the name of the parser.

Static Public Attributes

- tuple **parser_id** = models.AutoField(primary_key=True)

Private Member Functions

- def **_get_parser** (self)

Detailed Description

The abstract of a parser which can process a URL.

Member Function Documentation

def __unicode__ (self) This method make sure the object is stored and retrieved in certain format:

Returns

object in this format: u'Parser '+str(self.name())

def _get_parser (self) `[private]`

def name (self) Returns the name of the parser.

Returns

Name of the parser.

def process_single (self, url) Processes the given URL and stores the info inside the URL into the database.
Parameters

url	The URL to be processed.
<*url*>	URL

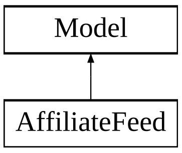
Member Data Documentation

tuple parser_id = models.AutoField(primary_key=True) `[static]`
The documentation for this class was generated from the following file:

- travelmatch/affiliate/**models.py**

B.4.2 AffiliateFeed

A feed consists of the url feed an a parser to process the url.
Inheritance diagram for AffiliateFeed:



Classes

- class **Meta**

  This enforce the combined super keys.

Public Member Functions

- def **parse** (self)

  Processes the contents of the URL and stores it inside the database.

Static Public Attributes

- tuple **url** = models.URLField(max_length=2048)
- tuple **parser** = models.ForeignKey(**ArkeParserModel**, null=False)
- tuple **unique_together** = (("url", "**parser**"),)

Detailed Description

A feed consists of the url feed an a parser to process the url.

Member Function Documentation

def parse (self) Processes the contents of the URL and stores it inside the database.

Member Data Documentation

tuple parser = models.ForeignKey(ArkeParserModel, null=False) [static]

tuple unique_together = ((”url”, ”parser”),) [static]

tuple url = models.URLField(max_length=2048) [static]

The documentation for this class was generated from the following file:

- travelmatch/affiliate/**models.py**

B.4.3 AffiliateFeed.Meta

This enforce the combined super keys.

Detailed Description

This enforce the combined super keys.

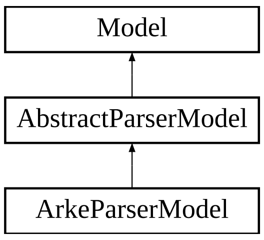
The documentation for this class was generated from the following file:

- travelmatch/affiliate/**models.py**

B.4.4 ArkeParserModel

The specific parser for Arke.

Inheritance diagram for ArkeParserModel:



Private Member Functions

- def **_get_parser** (self)

Additional Inherited Members

Detailed Description

The specific parser for Arke.

Member Function Documentation

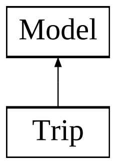
def _get_parser (self) [private]

The documentation for this class was generated from the following file:

- travelmatch/affiliate/**models.py**

B.4.5 Trip

Represents a trip users can book.
Inheritance diagram for Trip:



Public Member Functions

- def **get_must_fields** (self)
  Returns the must-have field names as defined by our client.
- def **get_fields** (self)
  Returns the field names of the model as a list.
- def **convert_to_trip_offer** (self)
  This function convert an location to an Trip Offer object.

Static Public Attributes

- tuple **name** = models.CharField(max_length=64)
- tuple **description** = models.TextField()
- tuple **city** = models.TextField()
- tuple **region** = models.TextField(null=True)
- tuple **country** = models.TextField(null=True)
- tuple **hotel_stars** = models.IntegerField(null=True)
- tuple **price** = models.FloatField()
- tuple **link** = models.URLField()
- tuple **image** = models.URLField()
- tuple **min_nr_people** = models.IntegerField(null=True)
- tuple **departure_date** = models.DateField()
- tuple **duration** = models.IntegerField()
- tuple **with_flight** = models.TextField()
- tuple **user_rating** = models.FloatField(null=True)
- tuple **created_on** = models.DateTimeField(auto_now_add=True)

Detailed Description

Represents a trip users can book.
Parameters

name	The name of the accommodation.
<name>	String
description	A description of the destination.
<description>	String

city	The city of the trip.
<city>	String
region	The region of the trip.
<region>	String
country	The country of the trip.
<country>	String
hotel_stars	The amount of stars the accommodation has
<hotel_stars>	Integer
price	The price of the whole trip.
<price>	Float
link	An affiliate link to book the trip.
<link>	URL
image	An image of the trip.
<image>	Image
min_nr_people	(Minimum) number of people for the trip.
<min_nr_↩ people>	Integer
departure_date	The date of departure
<departure_↩ date>	Date
duration	The duration of the trip.
<duration>	Integer
with_flight	Whether a flight is included with the trip.
<with_flight>	Boolean
user_rating	A user rating of the hotel
<user_rating>	Float

Member Function Documentation

def convert_to_trip_offer (self) This function convert an location to an Trip Offer object.

Returns

whether the conversion worked or not my_offer.save(): The new Trip Offer object saved

def get_fields (self) Returns the field names of the model as a list.

Returns

All field names of the model.

def get_must_fields (self) Returns the must-have field names as defined by our client.

Returns

All must-have field names of the model.

Member Data Documentation

tuple city = models.TextField() `[static]`

tuple country = models.TextField(null=True) `[static]`

tuple created_on = models.DateTimeField(auto_now_add=True) `[static]`

tuple departure_date = models.DateField() `[static]`

tuple description = models.TextField() `[static]`

tuple duration = models.IntegerField() `[static]`

tuple hotel_stars = models.IntegerField(null=True) `[static]`

tuple image = models.URLField() `[static]`

tuple link = models.URLField() `[static]`

tuple min_nr_people = models.IntegerField(null=True) `[static]`

tuple name = models.CharField(max_length=64) `[static]`

tuple price = models.FloatField() `[static]`

tuple region = models.TextField(null=True) `[static]`

tuple user_rating = models.FloatField(null=True) `[static]`

tuple with_flight = models.TextField() `[static]`

The documentation for this class was generated from the following file:

- travelmatch/affiliate/**models.py**

B.5 affiliate.serializer

`affiliate.serializer` is a namespace that contains classes, variables and functions that relate to the affiliate network serializers. This namespace contains Django serializers, which verify and modify the input and output of data in the model.

B.6 affiliate.tradetracker

`affiliate.tradetracker` is a namespace that contains classes, variables and functions that relate to TradeTracker-specific affiliate network parsers. This namespace contains namespaces with classes that extend the abstract parser in the `affiliate.affiliate_parser` namespace to parse feeds from TradeTracker.

Namespaces

- **arke_parser**

B.7 affiliate.tradetracker.arke_parser

`affiliate.tradetracker.arke_parser` is a namespace that contains classes, variables and functions that relate to ArkeFly-specific affiliate network parsers. This namespace contains the concrete parser for ArkeFly from the TradeTracker affiliate network that extends the abstract parser in the `affiliate.affiliate_parser` namespace.

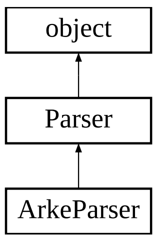
Classes

- class **ArkeParser**

  This is the specific parser for the Arke feed of TradeTracker.

B.7.1 ArkeParser

This is the specific parser for the Arke feed of TradeTracker.
Inheritance diagram for ArkeParser:



Public Member Functions

- def **__init__** (self)

  Initializes all variables which needs to be found in the XML files.

Static Public Attributes

- string **parser_name** = "Arke"

Detailed Description

This is the specific parser for the Arke feed of TradeTracker.

Constructor & Destructor Documentation

def __init__ (self) Initializes all variables which needs to be found in the XML files.
and stored inside the database.

Member Data Documentation

string parser_name = "Arke" `[static]`

The documentation for this class was generated from the following file:

- travelmatch/affiliate/tradetracker/**arke_parser.py**

B.8 affiliate.views

`affiliate.views` is a namespace that contains classes, variables and functions that relate to the affiliate network views. This namespace would contain Django view controllers for managing the affiliate network feeds; however, as no Django view controllers are required, this namespace is left empty.

B.9 ai

`ai` is a namespace that contains classes, variables and functions that relate to the artificial intelligence components. This namespace mainly acts as a container for several other namespaces, namely those relating to the entropy calculation, the recommender system, the models, serializers and views.

Namespaces

- **entropy**
- **models**
- **recommender_system**
- **serializers**
- **views**

B.10 ai.entropy

`ai.entropy` is a namespace that contains classes, variables and functions that relate to the entropy calculation. This namespace contains the functions which retrieves the next images based on a self-defined entropy value of that image. These images are then shown to the user in the interest analysis to maximize the information gain for the Travel DNA. It also stores a blacklist from the images which should not be recommended as well as keeps track of the entropy score of the user.

Functions

- def **calculate_next_images** (entropy_input, number_of_images_to_load, vacation)

  The function that calclulates the next n recommended images to send to the user from some travel↩ DNA.
- def **_initialize_vacation_tags** (vacation)

  Initializes all vacation tags to 0.
- def **_get_entropy_data** (vacation)

  Returns the entropy data that is stored inside vacation tag.
- def **_add_to_blacklist** (image_object, vacation)

  Adds an image to the blacklist.
- def **_purge_blacklist** (vacation)

  Removes all elements from the blacklist.
- def **get_first_n_random** (number_of_images_to_load, vacation)

  Returns n random images which are not in the blacklist.

- def **_get_best_tag** (user_entropy_data)
  Returns the tag with the lowest score >> the best priority.
- def **_get_best_image** (tag_id, forbidden_images)
  Returns the best image object with the highest potential score for this tag >> the highest tag_val sum.
- def **_get_best_tags_on_total_score** (tag_dict)
  Returns a list of tags with the lowest total potential score.
- def **_get_images_sorted_on_best_tag_val** (given_tag_id, forbidden_images)
  Returns a list of images based on the highest value for the given tag id.
- def **_get_best_tags_on_priority** (tag_id_list)
  Returns a list of tags with the lowest priority value (= best priority).
- def **_get_best_image_on_max_sum_values** (best_images_list)
  Returns a list of images with the largest sum value for its tags.

Function Documentation

def ai.entropy._add_to_blacklist (image_object, vacation) `[private]`
Adds an image to the blacklist.
Parameters

image_object	The image object to add
vacation	The vacation id of the vacation
<vacation>	int

def ai.entropy._get_best_image (tag_id, forbidden_images) `[private]`
Returns the best image object with the highest potential score for this tag >> the highest tag_val sum.
Parameters

tag_id	The tag_id to try to maximize on
forbidden_↩ images	the input image lists

Returns

best_image The image with the highest entropy potential

def ai.entropy._get_best_image_on_max_sum_values (best_images_list) `[private]`
Returns a list of images with the largest sum value for its tags.
Parameters

best_images_↩ list	A list of Image objects to filter upon

Returns

A list of Image objects with the highest sum for their tag_values

def ai.entropy._get_best_tag (user_entropy_data) `[private]`
Returns the tag with the lowest score >> the best priority.

Parameters

user_entropy←↩ _data	A dictionary consisting of {img_id: img_total_pot_value}

Returns

tag_id of the best tag

def ai.entropy._get_best_tags_on_priority (tag_id_list) [private]

Returns a list of tags with the lowest priority value (= best priority).

Parameters

tag_id_list	A list of tag_ids

Returns

List of tag_ids with the lowest priority value.

def ai.entropy._get_best_tags_on_total_score (tag_dict) [private]

Returns a list of tags with the lowest total potential score.

Parameters

tag_dict	A dictionary containing {tag_id: total_potential_score} for all tag_ids

Returns

Dictionary containing top 1 {tag_id: total_potential_score} with the possible lowest total_←↩ potential_score (returns multiple if tied).

def ai.entropy._get_entropy_data (vacation) [private]

Returns the entropy data that is stored inside vacation tag.

Parameters

vacation	The vacation id of the vacation
<vacation>	int

Returns

Dictionary with {tag_id: total_pot_value} for all tag_id in Tag.objects.all()

def ai.entropy._get_images_sorted_on_best_tag_val (given_tag_id, forbidden_images) [private]

Returns a list of images based on the highest value for the given tag id.

Parameters

given_tag_id	The tag_id to search an image on
forbidden_←↩ images	the input image lists

Returns

List of image objects

def ai.entropy._initialize_vacation_tags (vacation) [private]

Initializes all vacation tags to 0.

Parameters

vacation	The vacation id of the vacation
<vacation>	int

def ai.entropy._purge_blacklist (vacation) [private]

Removes all elements from the blacklist.

Parameters

vacation	The vacation id of the vacation
<vacation>	int

def ai.entropy.calculate_next_images (entropy_input, number_of_images_to_load, vacation) The function that calclulates the next n recommended images to send to the user from some travelDNA.

It bases this calculation on the entropy loss for each image. The user_input consists of a array of likings and disliking with the according tagvalues. So it consists of [{'like': True, 0: val0, 1: val1, 2: val2, .., n: valn}, ...] where the 'like' key is a boolean that represents a liking. The 0 to n keys are the id's of all n active tags in the database with their corresponding values for the image that was liked/disliked. If no value was given for a tag the value is set to 0.

Precondition

0 <= val <= 100 for all values
1 <= n <= 100
all active tag_ids in the database are in the dictonary

Parameters

entropy_input	[{'like': True, 0: val0, 1: val1, 2: val2, .., n: valn}, ...,]
number_of_↩ images_to_load	the number of images that are requested
<number_of_↩ images_to_↩ load>	int
vacation	the input vacation
<vacation>	VacationDetails

Returns

an array of image objects of length n. When it cannot find n images, less (or zero) image objects can be returned

def ai.entropy.get_first_n_random (number_of_images_to_load, vacation) Returns n random images which are not in the blacklist.

Parameters

number_of_↩ images_to_load	the number of images that are requested

<number_of_↩ images_to_↩ load>	int
vacation	The vacation id of the iser
vacation	int

Returns

List of 5 random non-duplicate image objects.

B.11 ai.models

`ai.models` is a namespace that contains classes, variables and functions that relate to the artificial intelligence models. This namespace contains all Django models related to artificial intelligence, which includes swipe images and their attributes, locations, all type of tags, Travel DNAs, and data required for AI calcuation such as the image and location blacklists.

Classes

- class **ImageBlacklistItem**
  This class is for the image blacklist item object.
- class **ImageDimension**
  This represents all the image dimensions database support.
- class **ImageTag**
  This represents the tag for images.
- class **Location**
  The locations the Travelmatch supports.
- class **LocationBlacklistItem**
  This class is for the lovation blacklist item object.
- class **LocationTag**
  This represents the tag for locations.
- class **SwipeImage**
  The images user relieve to swipe.
- class **Tag**
  This class represents the tag object either for images or locations.
- class **TravelDNA**
  This stores the Travel DNA of the user.
- class **TripOffer**
  This represents the offer of trip.
- class **VacationTag**
  This represesnts the vacation tag object.

Functions

- def **get_max_abs_price** (budget)
  Does the budget filtering.
- def **create**
  < activation status, for versioning
- def **__unicode__** (self)
  Returns the SwipeImage object in certain format.

- def **get_file_loc** (self, image_dimension)

  Return the file location of this image for each dimension to get the abs path (/var/www/media/swipe↩ _images/test-photo-1_1080x1920.jpg) you use MEDIA_ROOT + get_file_loc().
- def **get_abs_file_loc** (self, image_dimension)
- def **get_file_url** (self, image_dimension)
- def **get_all_file_dimension_instaces_in_folder** (self)

  This method returns all the instance within the iamge folder.
- def **remove_all_file_instances** (self)

  This method remove all swipe image file in the folder os.remove(file) all the files in the swipe image folder gets removed.
- def **force_create_file_instances** (self, image_dimensions)

  Create all the different versions of an image with the specified image dimensions.
- def **update_file_instances** (self, image_dimensions)

  This method update all the swipe image files and adapt them to the given image_dimensions.
- def **has_all_file_instances** (self, image_dimensions)

  This method check whether the swipe image was converted to image_dimensions or not.
- def **create_json_response** (self)

  Creates the json response in a python dict as specified in the API (GET: /user/swipe)
- def **img_html_tag** (self)

  returning a string containing image's resolution and url
- def **create_missing_tags** (self, new_tags)

  Creates any missing tag values.
- def **activate_image** (self)

  This function activate the swipe image self.active=True: activation of the swipe image.
- def **deactivate_image** (self)

  This function deactivate the swipe image self.active=False: deactivation of the swipe image.
- def **update_images_dimension** (sender, instance=None, args, kwargs)

  This function adapts the SwipeImage object into the dimensions in the database.
- def **create_missing_tags** (sender, instance=None, args, kwargs)

  This function creates the needed LocationTags when creating a new location.

Variables

- tuple **img_id** = models.AutoField(primary_key=True)
- tuple **created** = models.DateTimeField(auto_now_add=True)

  < id of the image (primary key) (integer)
- tuple **uploaded_by** = models.ForeignKey(User)

  < date image being uploaded (integer)
- tuple **original_filename** = models.ImageField(upload_to='swipe_images')

  < the admin user added the image (admin user id)
- tuple **active** = models.BooleanField(null=False, default=True)

  < the image (ImageField)

Function Documentation

def ai.models.__unicode__ (self) Returns the SwipeImage object in certain format.

Returns

Tag u'Image s: s' % (self.img_id, self.original_filename)

def ai.models.activate_image (self) This function activate the swipe image self.active=True: activation of the swipe image.

def ai.models.create (self, force_insert = *False*, **force_update =** *False*, **using =** *None*, **update_fields =** *None* **)** < activation status, for versioning
creating a new tag tuple (not update, just create)

Precondition

the tuple does not exist int e database

Postcondition

tuple updated

Exceptions

IntegrityError	if the was precondition violated

Returns

Void new Tag object

def ai.models.create_json_response (self) Creates the json response in a python dict as specified in the API (GET: /user/swipe)

Returns

json: json object containing image dimensions and size

def ai.models.create_missing_tags (self, new_tags) Creates any missing tag values.
Parameters

new_tags	all the Tag objects to create if needed

def ai.models.create_missing_tags (sender, instance = *None*, **args, kwargs)** This function creates the needed LocationTags when creating a new location.

Precondition

None

Parameters

sender	
instance	input images
args	not used
kwargs	not used Location: Location.create_missing_tags

def ai.models.deactivate_image (self) This function deactivate the swipe image self.↩ active=False: deactivation of the swipe image.

def ai.models.force_create_file_instances (self, image_dimensions) Create all the different versions of an image with the specified image dimensions.

Parameters

image_↩ dimensions	ImageDimensions

Returns

image: image with image_dimensions

def ai.models.get_abs_file_loc (self, image_dimension)

See also

get_file_loc

def ai.models.get_all_file_dimension_instaces_in_folder (self) This method returns all the instance within the iamge folder.

Returns

[SwipeImages]: all swipe images in the image folder

def ai.models.get_file_loc (self, image_dimension) Return the file location of this image for each dimension to get the abs path (/var/www/media/swipe_images/test-photo-1_1080x1920.jpg) you use MEDIA_ROOT + get_file_loc().

To get the abs url use BASE_URL+MEDIA_URL + get_file_loc().

Returns

string: filename + _ + dimension.to_x() + .jpg

def ai.models.get_file_url (self, image_dimension)

See also

get_file_loc

def ai.models.get_max_abs_price (budget) Does the budget filtering.

Parameters

budget	the budget saved in the location details (in cents) (int)

Returns

the budget margin for which offers can be retrieved (in euros)

def ai.models.has_all_file_instances (self, image_dimensions) This method check whether the swipe image was converted to image_dimensions or not.

Parameters

image_↩ dimensions	the image dimensions you wish to check
<image_↩ dimensions>	[ImageDimension]

Returns

boolean_value whether the swipe image was converted to image_dimensions or not

def ai.models.img_html_tag (self) returning a string containing image's resolution and url

Returns

string: '' % (url, resolution[0].width, resolution[0].height)

def ai.models.remove_all_file_instances (self) This method remove all swipe image file in the folder os.remove(file) all the files in the swipe image folder gets removed.

def ai.models.update_file_instances (self, image_dimensions) This method update all the swipe image files and adapt them to the given image_dimensions.

Parameters

image_↩ dimensions	the image dimensions you wish to have
<image_↩ dimensions>;	[ImageDimension]

Returns

Boolean whether the images are updated or not SwipeImages updated according to the image dimensions

def ai.models.update_images_dimension (sender, instance = $None$**, args, kwargs)** This function adapts the SwipeImage object into the dimensions in the database.

Precondition

None

Parameters

sender	
instance	input images
args	not used
kwargs	not used SwipeImage: SwipeImage.update_images_dimension

Variable Documentation

active = models.BooleanField(null=False, default=True) < the image (ImageField)

tuple created = models.DateTimeField(auto_now_add=True) < id of the image (primary key) (integer)

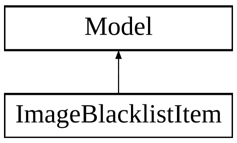
tuple img_id = models.AutoField(primary_key=True)

tuple original_filename = models.ImageField(upload_to='swipe_images') < the admin user added the image (admin user id)

tuple uploaded_by = models.ForeignKey(User) < date image being uploaded (integer)

B.11.1 ImageBlacklistItem

This class is for the image blacklist item object.
Inheritance diagram for ImageBlacklistItem:



Classes

- class **Meta**

    < date the tag is added (Datefield)

Public Member Functions

- def **create**

    creating a new tag tuple (not update, just create)
- def **remove** (self)

    This function removes object from the database the BlacklistItem with vac && img is deleted.

Static Public Attributes

- tuple **img** = models.ForeignKey(**SwipeImage**, null=False)
- tuple **vac** = models.ForeignKey('appusers.VacationDetail', null=False)

    < the image of the image blacklist object (SwipeImage)
- tuple **created_on** = models.DateTimeField(auto_now_add=True)

    < the vacation of the image blacklist object (VacationDetails)

Detailed Description

This class is for the image blacklist item object.

Member Function Documentation

def create (self, force_insert = *False***, force_update =** *False***, using =** *None***, update↩ _fields =** *None* **)** creating a new tag tuple (not update, just create)

Precondition

the tuple does not exist int e database

Postcondition

tuple updated

Exceptions

IntegrityError	if the was precondition violated

Returns

Void new Tag object

def remove (self) This function removes object from the database the BlacklistItem with vac && img is deleted.

Member Data Documentation

tuple created_on = models.DateTimeField(auto_now_add=True) `[static]`
< the vacation of the image blacklist object (VacationDetails)

tuple img = models.ForeignKey(SwipeImage, null=False) `[static]`

tuple vac = models.ForeignKey('appusers.VacationDetail', null=False) `[static]`
< the image of the image blacklist object (SwipeImage)
The documentation for this class was generated from the following file:

- travelmatch/ai/**models.py**

B.11.2 ImageBlacklistItem.Meta

< date the tag is added (Datefield)

Static Public Attributes

- tuple **unique_together** = ((”img”, ”**vac**”),)

Detailed Description

< date the tag is added (Datefield)
This makes (img && vacation) a super key

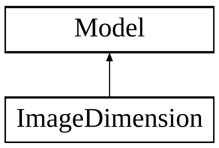
Member Data Documentation

tuple unique_together = ((”img”, ”vac”),) `[static]`
The documentation for this class was generated from the following file:

- travelmatch/ai/**models.py**

B.11.3 ImageDimension

This represents all the image dimensions database support.
Inheritance diagram for ImageDimension:



Classes

- class **Meta**

  < This is the height of the image (int)

Public Member Functions

- def **create**

  creating a new tag tuple (not update, just create)
- def **__unicode__** (self)

  This returns the ImageDimensions with certain format in a string.
- def **to_x** (self)

  This returns the ImageDimensions with certain format in a string.

Static Public Member Functions

- def **get_all** ()

  This returns all the ImageDimensions objects in the database.

Static Public Attributes

- tuple **width** = models.IntegerField(null=False, blank=False)
- tuple **height** = models.IntegerField(null=False, blank=False)

  < This is the width of the image (int)

Detailed Description

This represents all the image dimensions database support.

Member Function Documentation

def __unicode__ (self) This returns the ImageDimensions with certain format in a string.

Returns

string: u''+self.to_x()

def create (self, force_insert = *False***, force_update =** *False***, using =** *None***, update_fields =** *None* **)** creating a new tag tuple (not update, just create)

Precondition

the tuple does not exist int e database

Postcondition

tuple updated

Exceptions

IntegrityError	if the was precondition violated

Returns

Void new Tag object

def get_all ( ) `[static]`
This returns all the ImageDimensions objects in the database.

Returns

all ImageDimensions objects

def to_x (self) This returns the ImageDimensions with certain format in a string.

Returns

string: str(self.width) + "x" + str(self.height)

Member Data Documentation

tuple height = models.IntegerField(null=False, blank=False) `[static]`
< This is the width of the image (int)

tuple width = models.IntegerField(null=False, blank=False) `[static]`
The documentation for this class was generated from the following file:

- travelmatch/ai/**models.py**

B.11.4 ImageDimension.Meta

< This is the height of the image (int)

Static Public Attributes

- tuple **unique_together** = (("width", "**height**"),)

Detailed Description

< This is the height of the image (int)
This makes (width&&height) a super key

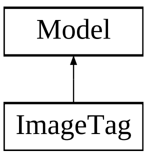
Member Data Documentation

tuple unique_together = (("width", "height"),) `[static]`
The documentation for this class was generated from the following file:

- travelmatch/ai/**models.py**

B.11.5 ImageTag

This represents the tag for images.
Inheritance diagram for ImageTag:



Classes

- class **Meta**
  < the value of the image tag (int)

Public Member Functions

- def **create**

  creating a new tag tuple (not update, just create)
- def **save** (self, args, kwargs)

  This method override the save function from django and ensures that every object is unique.

Public Attributes

- **pk**

Static Public Attributes

- tuple **img** = models.ForeignKey(**SwipeImage**, null=False)
- tuple **tag** = models.ForeignKey(**Tag**, null=False)

  < the image of the image tag (SwipeImage)
- tuple **value** = models.IntegerField(null=False, blank=True, default=0)

  < the tag of the image tag (Tag)
- tuple **unique_together** = (("img", "**tag**"),)

Detailed Description

This represents the tag for images.

Member Function Documentation

def create (self, force_insert = *False***, force_update =** *False***, using =** *None***, update↩ _fields =** *None* **)** creating a new tag tuple (not update, just create)

Precondition

> the tuple does not exist int e database

Postcondition

> tuple updated

Exceptions

IntegrityError	if the was precondition violated

Returns

> Void new Tag object

def save (self, args, kwargs) This method override the save function from django and ensures that every object is unique.

Member Data Documentation

tuple img = models.ForeignKey(SwipeImage, null=False) `[static]`

pk

tuple tag = models.ForeignKey(Tag, null=False) `[static]`
< the image of the image tag (SwipeImage)

tuple unique_together = ((”img”, ”tag”),) `[static]`

tuple value = models.IntegerField(null=False, blank=True, default=0) `[static]`
< the tag of the image tag (Tag)
The documentation for this class was generated from the following file:

- travelmatch/ai/**models.py**

B.11.6 ImageTag.Meta

< the value of the image tag (int)

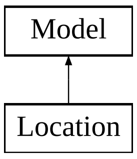
Detailed Description

< the value of the image tag (int)
This makes (img && tag) a super key
The documentation for this class was generated from the following file:

- travelmatch/ai/**models.py**

B.11.7 Location

The locations the Travelmatch supports.
Inheritance diagram for Location:



Public Member Functions

- def **create**

  < versioned or not for versioning purposes (boolean)
- def **__unicode__** (self)

  This function enforce certain format on object output return object with certain format u'Location s: s' % (self.loc_id, self.city_name)
- def **generate_hotels_offer** (self, vac)

  Generates hotel id's for the location overview.
- def **create_missing_tags** (self, tags)

  Creates any missing tag values.
- def **activate_location** (self)

  This function activate the location self.active=True: activation of the swipe image.
- def **deactivate_location** (self)

  This function deactivate the location self.active=False: deactivation of the swipe image.

Public Attributes

- **active**

Static Public Attributes

- tuple **loc_id** = models.AutoField(primary_key=True)
- tuple **city_name** = models.CharField(max_length=64, null=False, blank=False)

  < id(primary key) of the location (integer)
- tuple **country_name** = models.CharField(max_length=64, null=False, blank=True, default="")

  < the name of the city of the location (string)
- tuple **region_name** = models.CharField(max_length=64, null=False, blank=True, default="")

  < the country of the location (string)
- tuple **active** = models.BooleanField(null=False, default=True)

  < the region of the location (string)

Detailed Description

The locations the Travelmatch supports.

Member Function Documentation

def __unicode__ (self) This function enforce certain format on object output return object with certain format u'Location s: s' % (self.loc_id, self.city_name)

def activate_location (self) This function activate the location self.active=True: activation of the swipe image.

def create (self, force_insert = *False***, force_update =** *False***, using =** *None***, update↩ _fields =** *None* **)** < versioned or not for versioning purposes (boolean)
creating a new tag tuple (not update, just create)

Precondition

the tuple does not exist int e database

Postcondition

tuple updated

Exceptions

IntegrityError	if the was precondition violated

Returns

Void new Tag object

def create_missing_tags (self, tags) Creates any missing tag values.
Parameters

tags	all the Tag objects to create if needed

def deactivate_location (self) This function deactivate the location self.active=False: deactivation of the swipe image.

def generate_hotels_offer (self, vac) Generates hotel id's for the location overview.
Parameters

vac	The vacationdetails for wich the hotel overview must be

Returns

list: an array of hotelOffer instances

Member Data Documentation

tuple active = models.BooleanField(null=False, default=True) `[static]`
< the region of the location (string)

active

tuple city_name = models.CharField(max_length=64, null=False, blank=False) `[static]`
< id(primary key) of the location (integer)

tuple country_name = models.CharField(max_length=64, null=False, blank=True, default="") `[static]`
< the name of the city of the location (string)

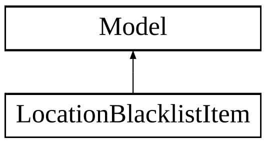
tuple loc_id = models.AutoField(primary_key=True) `[static]`

tuple region_name = models.CharField(max_length=64, null=False, blank=True, default="") `[static]`
< the country of the location (string)
The documentation for this class was generated from the following file:

- travelmatch/ai/**models.py**

B.11.8 LocationBlacklistItem

This class is for the lovation blacklist item object.
Inheritance diagram for LocationBlacklistItem:



Classes

- class **Meta**
  < the vacation of the image blacklist object (VacationDetails)

Public Member Functions

- def **create**
  creating a new tag tuple (not update, just create)
- def **remove** (self)
  This function removes object from the database the LocationBlacklistItem with vac && img is deleted.

Static Public Attributes

- tuple **loc** = models.ForeignKey(**Location**, null=False)
- tuple **vac** = models.ForeignKey('appusers.VacationDetail', null=False)

Detailed Description

This class is for the lovation blacklist item object.

Member Function Documentation

def create (self, force_insert = *False*, **force_update =** *False*, **using =** *None*, **update↩ _fields =** *None* **)** creating a new tag tuple (not update, just create)

Precondition

the tuple does not exist int e database

Postcondition

tuple updated

Exceptions

IntegrityError	if the was precondition violated

Returns

Void new object

def remove (self) This function removes object from the database the LocationBlacklistItem with vac && img is deleted.

Member Data Documentation

tuple loc = models.ForeignKey(Location, null=False) `[static]`

tuple vac = models.ForeignKey('appusers.VacationDetail', null=False) `[static]`
The documentation for this class was generated from the following file:

- travelmatch/ai/**models.py**

B.11.9 LocationBlacklistItem.Meta

< the vacation of the image blacklist object (VacationDetails)

Static Public Attributes

- tuple **unique_together** = (("loc", "**vac**"),)

Detailed Description

< the vacation of the image blacklist object (VacationDetails)
This makes (img && vacation) a super key

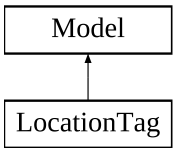
Member Data Documentation

tuple unique_together = ((”loc”, ”vac”),) `[static]`
The documentation for this class was generated from the following file:

- travelmatch/ai/**models.py**

B.11.10 LocationTag

This represents the tag for locations.
Inheritance diagram for LocationTag:



Classes

- class **Meta**
  This makes (tag_id && loc_id) a super key.

Public Member Functions

- def **create**
  creating a new tag tuple (not update, just create)
- def **save** (self, args, kwargs)
  This method override the save function from django and ensures that every object is unique.

Static Public Member Functions

- def **put** (**tag_id**, **loc_id**, **value**, **initial_value**)
  This save the new location tag, or upgrade it if it exist.

Public Attributes

- **pk**
- **value**

Static Public Attributes

- tuple **tag_id** = models.ForeignKey(**Tag**, null=False)
- tuple **loc_id** = models.ForeignKey(**Location**, null=False)
  < tag of location tag (Tag)
- tuple **value** = models.IntegerField(null=False, blank=True, default=0)
  < location of location tag (Location)
- tuple **initial_value** = models.IntegerField(null=False, default=0)
  < value of the location tag (integer)

- tuple **last_modified_by** = models.ForeignKey(User, null=True)
- tuple **unique_together** = ((”tag_id”, ”**loc_id**”),)

Detailed Description

This represents the tag for locations.

Member Function Documentation

def create (self, force_insert = *False***, force_update =** *False***, using =** *None***, update↩ _fields =** *None* **)** creating a new tag tuple (not update, just create)

Precondition

the tuple does not exist int e database

Postcondition

tuple updated

Exceptions

IntegrityError	if the was precondition violated

Returns

Void new Tag object

def put (tag_id, loc_id, value, initial_value) `[static]`
This save the new location tag, or upgrade it if it exist.
Parameters

tag_id	the id of the input tag
<tag_id>	int
loc_id	the input id of the location
<loc_id>	int
value	the input value of the tag
<value>	int
initial_value	the initial value of the input tag
<initial_↩ value>	int

Returns

new tag with the parameters iff. the tag does not exist, else update and return the exist tag new Tag object

def save (self, args, kwargs) This method override the save function from django and ensures that every object is unique.

Member Data Documentation

tuple initial_value = models.IntegerField(null=False, default=0) `[static]`
< value of the location tag (integer)

tuple last_modified_by = models.ForeignKey(User, null=True) `[static]`

tuple loc_id = models.ForeignKey(Location, null=False) `[static]`
< tag of location tag (Tag)

pk

tuple tag_id = models.ForeignKey(Tag, null=False) `[static]`

tuple unique_together = ((”tag_id”, ”loc_id”),) `[static]`

tuple value = models.IntegerField(null=False, blank=True, default=0) `[static]`
< location of location tag (Location)

value The documentation for this class was generated from the following file:

- travelmatch/ai/**models.py**

B.11.11 LocationTag.Meta

This makes (tag_id && loc_id) a super key.

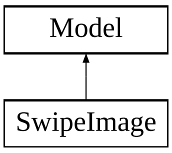
Detailed Description

This makes (tag_id && loc_id) a super key.
The documentation for this class was generated from the following file:

- travelmatch/ai/**models.py**

B.11.12 SwipeImage

The images user relieve to swipe.
Inheritance diagram for SwipeImage:



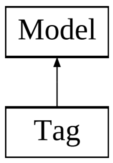
Detailed Description

The images user relieve to swipe.
The documentation for this class was generated from the following file:

- travelmatch/ai/**models.py**

B.11.13 Tag

This class represents the tag object either for images or locations.
Inheritance diagram for Tag:



Classes

- class **Meta**

  < *prioritizeing the tags for undecided cases, lower number has a higher priority (integer)*

Public Member Functions

- def **__unicode__** (self)

  < *the index for ordering/sorting tag objects in the database*
- def **create**

  creating a new tag tuple (not update, just create)

Static Public Attributes

- tuple **tag_id** = models.AutoField(primary_key=True)
- tuple **name** = models.CharField(max_length=256, null=False, blank=False)

  < *the id(primary key) of the tag (integer)*
- tuple **created_on** = models.DateTimeField(auto_now_add=True)

  < *the name of the tag (string)*
- tuple **created_by** = models.ForeignKey(User)

  < *date the tag is added (Datefield)*
- tuple **active** = models.BooleanField(null=False, default=False)

  < *django admin users (integer)*
- tuple **priority** = models.IntegerField(null=False, default=100)

  < *versioned or not for versioning purposes (boolean)*
- string **verbose_name** = "Tag"
- tuple **ordering** = ('**created_on**',)

  < *human readable name*

Detailed Description

This class represents the tag object either for images or locations.

Member Function Documentation

def __unicode__ (self) < the index for ordering/sorting tag objects in the database
representation for the Tag object

Returns

Tag object with specific format: "u'Tag tag_id, name, created_on"

def create (self, force_insert = *False***, force_update =** *False***, using =** *None***, update↩_fields =** *None* **)** creating a new tag tuple (not update, just create)

Precondition

> the tuple does not exist int e database

Postcondition

> tuple updated

Exceptions

IntegrityError	if the was precondition violated

Returns

> Void new Tag object

Member Data Documentation

tuple active = models.BooleanField(null=False, default=False) `[static]`
< django admin users (integer)

tuple created_by = models.ForeignKey(User) `[static]`
< date the tag is added (Datefield)

tuple created_on = models.DateTimeField(auto_now_add=True) `[static]`
< the name of the tag (string)

tuple name = models.CharField(max_length=256, null=False, blank=False) `[static]`
< the id(primary key) of the tag (integer)

tuple ordering = ('created_on',) `[static]`
< human readable name

tuple priority = models.IntegerField(null=False, default=100) `[static]`
< versioned or not for versioning purposes (boolean)

tuple tag_id = models.AutoField(primary_key=True) `[static]`

string verbose_name = "Tag" `[static]`
The documentation for this class was generated from the following file:

- travelmatch/ai/**models.py**

B.11.14 Tag.Meta

< prioritizeing the tags for undecided cases, lower number has a higher priority (integer)

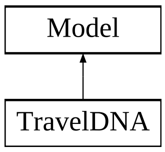
Detailed Description

< prioritizeing the tags for undecided cases, lower number has a higher priority (integer)
Meta specify the human-readable name and ordering of the code
The documentation for this class was generated from the following file:

- travelmatch/ai/**models.py**

B.11.15 TravelDNA

This stores the Travel DNA of the user.
Inheritance diagram for TravelDNA:



Classes

- class **Meta**

  < boolean value see if the user likes it or not

Public Member Functions

- def **create**

  creating a new tag tuple (not update, just create)
- def **display_img_filename** (self)

  This is for returning the image file name attribute.
- def **user** (self)

  This is for returning the user of the vacation.
- def **display_img_id** (self)

  This is for returning the img_id of the image.
- def **vac_id** (self)

  This is for returning the vac_id of the vacation.
- def **display_vac** (self)

  This is for calling the display function on the vacation from the Vacation class in AppUser.
- def **display_user** (self)

  This is for calling the display function on the vacation from the AppUser class in AppUser.

Static Public Attributes

- tuple **img** = models.ForeignKey(**SwipeImage**, null=False)
- tuple **vacation** = models.ForeignKey('appusers.VacationDetail', null=False)

  < the image of the TravelDNA object (SwipeImage)
- tuple **like** = models.BooleanField(null=False)

  < the vacation of the TravelDNA object (VacationDetails)

Detailed Description

This stores the Travel DNA of the user.

Member Function Documentation

def create (self, force_insert = `False`**, force_update =** `False`**, using =** `None`**, update↩_fields =** `None` **)** creating a new tag tuple (not update, just create)

Precondition

the tuple does not exist int e database

Postcondition

tuple updated

Exceptions

IntegrityError	if the was precondition violated

Returns

Void new Tag object

def display_img_filename (self) This is for returning the image file name attribute.

Precondition

: None

Postcondition

print the image file name

Returns

the image file name

def display_img_id (self) This is for returning the img_id of the image.

Precondition

None

Postcondition

None

Returns

the img_id of the image

def display_user (self) This is for calling the display function on the vacation from the AppUser class in AppUser.

Precondition

None

Postcondition

None

Returns

display() for the user of the vacation of the travelDNA

def display_vac (self) This is for calling the display function on the vacation from the Vacation class in AppUser.

Precondition

  None

Postcondition

  None

Returns

  display() for the vacation of the travelDNA

def user (self) This is for returning the user of the vacation.

Precondition

  None

Postcondition

  None

Returns

  user related to the vacation

def vac_id (self) This is for returning the vac_id of the vacation.

Precondition

  None

Postcondition

  None

Returns

  the vac_id of the vacation

Member Data Documentation

tuple img = models.ForeignKey(SwipeImage, null=False) `[static]`

tuple like = models.BooleanField(null=False) `[static]`
< the vacation of the TravelDNA object (VacationDetails)

tuple vacation = models.ForeignKey('appusers.VacationDetail', null=False) `[static]`
< the image of the TravelDNA object (SwipeImage)
The documentation for this class was generated from the following file:

- travelmatch/ai/**models.py**

B.11.16 TravelDNA.Meta

< boolean value see if the user likes it or not

Static Public Attributes

- tuple **unique_together** = (("img", "**vacation**"),)

Detailed Description

< boolean value see if the user likes it or not
This makes (img && vacation) a super key

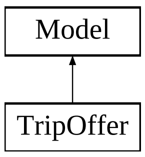
Member Data Documentation

tuple unique_together = (("img", "vacation"),) `[static]`
The documentation for this class was generated from the following file:

- travelmatch/ai/**models.py**

B.11.17 TripOffer

This represents the offer of trip.
Inheritance diagram for TripOffer:



Classes

- class **Meta**
  This makes a lable for this TripOffer class.

Public Member Functions

- def **create**
  creating a new tag tuple (not update, just create)
- def **generate_json_response** (self)
  this function generates json response from TripOffSerilaizer return the json response data

Static Public Attributes

- tuple **offer_id** = models.AutoField(primary_key=True)
- tuple **loc** = models.ForeignKey(**Location**)
- tuple **name** = models.CharField(max_length=64, null=False, blank=False)
- tuple **description** = models.TextField()
- tuple **hotel_stars** = models.IntegerField(null=True, blank=True, default=None)
- tuple **price** = models.FloatField(null=False, blank=False)
- tuple **link** = models.URLField(null=False, blank=False)
- tuple **image** = models.URLField(null=False, blank=False)
- tuple **min_people** = models.IntegerField(null=True, blank=True, default=None)

- tuple **dept_date** = models.DateField(null=False, blank=False)
- tuple **duration_days** = models.IntegerField(null=False, blank=False)
- tuple **with_flight** = models.BooleanField(null=False, blank=False, default=False)
- tuple **user_rating** = models.FloatField(null=True, blank=True, default=None)
- tuple **priority** = models.IntegerField(null=False, blank=True, default=100)

Detailed Description

This represents the offer of trip.

Member Function Documentation

def create (self, force_insert = *False***, force_update =** *False***, using =** *None***, update↩ _fields =** *None* **)** creating a new tag tuple (not update, just create)

Precondition

the tuple does not exist int e database

Postcondition

tuple updated

Exceptions

IntegrityError	if the was precondition violated

Returns

Void new Tag object

def generate_json_response (self) this function generates json response from TripOffSerilaizer return the json response data

Member Data Documentation

tuple dept_date = models.DateField(null=False, blank=False) `[static]`

tuple description = models.TextField() `[static]`

tuple duration_days = models.IntegerField(null=False, blank=False) `[static]`

tuple hotel_stars = models.IntegerField(null=True, blank=True, default=None) `[static]`

tuple image = models.URLField(null=False, blank=False) `[static]`

tuple link = models.URLField(null=False, blank=False) `[static]`

tuple loc = models.ForeignKey(Location) `[static]`

tuple min_people = models.IntegerField(null=True, blank=True, default=None) `[static]`

tuple name = models.CharField(max_length=64, null=False, blank=False) `[static]`

tuple offer_id = models.AutoField(primary_key=True) `[static]`

tuple price = models.FloatField(null=False, blank=False) `[static]`

tuple priority = models.IntegerField(null=False, blank=True, default=100) `[static]`

tuple user_rating = models.FloatField(null=True, blank=True, default=None) `[static]`

tuple with_flight = models.BooleanField(null=False, blank=False, default=False) `[static]`
The documentation for this class was generated from the following file:

- travelmatch/ai/**models.py**

B.11.18 TripOffer.Meta

This makes a lable for this TripOffer class.

Static Public Attributes

- string **app_label** = 'affiliate'

Detailed Description

This makes a lable for this TripOffer class.

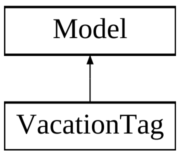
Member Data Documentation

string app_label = 'affiliate' `[static]`
The documentation for this class was generated from the following file:

- travelmatch/ai/**models.py**

B.11.19 VacationTag

This represnts the vacation tag object.
Inheritance diagram for VacationTag:



Classes

- class **Meta**

  This makes (img && vacation) a super key.

Public Member Functions

- def **remove** (self)

  This function removes object from the database the LocationBlacklistItem with vac && img is deleted.

Static Public Attributes

- tuple **vac** = models.ForeignKey('appusers.VacationDetail', null=False)
- tuple **tag** = models.ForeignKey(**Tag**, null=False)

  < the vacation of the image blacklist object (VacationDetails)
- tuple **sum_value** = models.IntegerField(null=False, blank=True, default=0)

Detailed Description

This represnts the vacation tag object.

Member Function Documentation

def remove (self) This function removes object from the database the LocationBlacklistItem with vac && img is deleted.

Member Data Documentation

tuple sum_value = models.IntegerField(null=False, blank=True, default=0) `[static]`

tuple tag = models.ForeignKey(Tag, null=False) `[static]`
< the vacation of the image blacklist object (VacationDetails)

tuple vac = models.ForeignKey('appusers.VacationDetail', null=False) `[static]`
The documentation for this class was generated from the following file:

- travelmatch/ai/**models.py**

B.11.20 VacationTag.Meta

This makes (img && vacation) a super key.

Static Public Attributes

- tuple **unique_together** = ((”vac”, ”**tag**”),)

Detailed Description

This makes (img && vacation) a super key.

Member Data Documentation

tuple unique_together = ((”vac”, ”tag”),) `[static]`

The documentation for this class was generated from the following file:

- travelmatch/ai/**models.py**

B.12 ai.recommender_system

`ai.recommender_system` is a namespace that contains classes, variables and functions that relate to the recommender system. This namespace contains functions for recommending locations given an input set of likes and dislikes from the user. The recommendations are calculated using a Vector Space Model, where each tag is a dimension in an n-dimensional graph and the similarity is calculated using the cosine similarity function.

Functions

- def **calc_recommendations** (user_input_set, n, vacation)

  The function that calclulates the n reccomended locations from some travelDNA The user_input consists of a array of likings and disliking with the according tagvalues.
- def **calculate_travel_dna** (user_input_set)

  Calculates the travelDNA given a user_input_set.
- def **_get_user_dict** (user_travel_dna)

  Returns a dictionary which can be used to get the best match.
- def **_normalize_my_tags** (tag_val_list)

  Returns a normalized user tag score for the tags.
- def **_get_best_match** (user_travel_dna, n, forbidden_locations)

  Gets the city that is the closest to the user_travel_dna using the cosine similiarity as measure.
- def **_get_city_matrix** (forbidden_locations)

  Normalizes all image_values inside the database.
- def **_normalize_function** (image_value)

  Normalization function of the city matrix.
- def **_cosine_similarity** (user_dict, city_dict)

  Computes the cosine similarity of the first dicitonary to the second dictionary.

Function Documentation

def ai.recommender_system._cosine_similarity (user_dict, city_dict) `[private]`

Computes the cosine similarity of the first dicitonary to the second dictionary.

Uses an optimized method for increased speed. All values of the tag_values should be 0 <= v <= 100

Parameters

user_dict	A dictionary containing user tag_ids with their values.
<user_dict>	dictionary
city_dict	A dictionary containing city tag_ids with their values.
<city_dict>	dictionary

Returns

Float with the cosine similarity.

def ai.recommender_system._get_best_match (user_travel_dna, n, forbidden_locations) `[private]`

Gets the city that is the closest to the user_travel_dna using the cosine similiarity as measure.

Parameters

user_travel_dna	The TravelDNA of the user
<*user_travel↩ _dna*>	Dictionary
n	The number of locations needed
<*n*>	int
forbidden_↩ locations	the input location lists
<*forbidden_↩ locations*>	[Location]

Returns

A list of city_ids which are the best match.

def ai.recommender_system._get_city_matrix (forbidden_locations) `[private]`

Normalizes all image_values inside the database.

Returns

city_matrix_dict with normalized values, {city_id: {image_id: image_val}} example: {1: {1: 0.5, 2: 0.7}, 2: {1: 0.9, 2: 0.65}}

def ai.recommender_system._get_user_dict (user_travel_dna) `[private]`

Returns a dictionary which can be used to get the best match.

Parameters

user_travel_dna	The TravelDNA of the user
<*user_travel↩ _dna*>	Dictionary

Returns

A dictionary where {tag_id: tag_val_norm}

def ai.recommender_system._normalize_function (image_value) `[private]`

Normalization function of the city matrix.
Currently 0..100 –> 0..1

Parameters

image_value	The image value to be normalized
<*image_↩ value*>	int

Returns

Returns a normalized float image_value from 0..1

def ai.recommender_system._normalize_my_tags (tag_val_list) `[private]`

Returns a normalized user tag score for the tags.

Parameters

tag_val_list	A list of two items: [current user score, total potential score]
<*tag_val_list*>	int[]

Returns

Normalized current_score / total_potential_score if v[1] > 0 else 0

def ai.recommender_system.calc_recommendations (user_input_set, n, vacation) The function that calclulates the n reccomended locations from some travelDNA The user_input consists of a array of likings and disliking with the according tagvalues.

So it consists of [{'like': True, 0: val0, 1: val1, 2: val2, .., n: valn}, ...] where the 'like' key is a boolean that represents a liking. The 0 to n keys are the id's of all n active tags in the database with their corresponding values for the image that was liked/disliked. If no value was given for a tag the value is set to 0.

Precondition

0 <= val <= 100 for all values
all tag_id's are in the dictonary
n == 1 || n == 2

Postcondition

len(returned) == n || len(returned) == 0

Parameters

user_input_set	[{'like': True, 0: val0, 1: val1, 2: val2, .., n: valn}, ...,]
<*user_input*_↩ *set*>	dictionary
n	The amount of locations that have to be recommended.
<*n*>	n
vacation	the input vacation
<*vacation*>	VacationDetails

Returns

List of loc_id of length n or empty array when the AI fails

def ai.recommender_system.calculate_travel_dna (user_input_set) Calculates the travelDNA given a user_input_set.

Precondition

0 <= tag_value <= 100

Parameters

user_input_set	[{'like': True, '0': 30, '1': 10}, ..., {'like': False, '4': 0, '33': 100}]
<*user_input*_↩ *set*>	dictionary

Returns

The TravelDNA of a user where TravelDNA = ['0': [user_val, total_val], ..., '33': [user_val, total_val]]

B.13 ai.serializers

`ai.serializer` is a namespace that contains classes, variables and functions that relate to the AI serializers. This namespace contains Django serializers, which verify and modify the input and output of data in the model.

Classes

- class **LocationSerializer**

  This makes a django serializer object for the Location class.
- class **Meta**

  This creates the serializer with specific model and fields.
- class **TripOfferSerializer**

  This is for trip offer serializer.

Functions

- def **validate_city_name** (self, value)

  this method returns validated city_name
- def **validate_country_name** (self, value)

  this method returns validated city_name
- def **validate_region_name** (self, value)

  this method returns validated city_name

Function Documentation

def ai.serializers.validate_city_name (self, value) this method returns validated city_name

Parameters

value	the input city_name
<*value*>	string

Precondition

len(value) == 0

Exceptions

ValidationError	if precondition is violated

Returns

value: city_name.lower(), it is case insensitive

def ai.serializers.validate_country_name (self, value) this method returns validated city_name

Parameters

value	the input country_name
<*value*>	string

Precondition

len(value) == 0

Exceptions

ValidationError	if precondition is violated

Returns

value: country_name.lower(), it is case insensitive

def ai.serializers.validate_region_name (self, value) this method returns validated city_name

Parameters

value	the input region_name (string)
<*value*>	string

Precondition

len(value) == 0

Exceptions

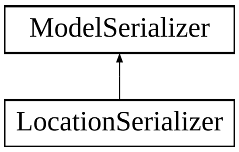
ValidationError	if precondition is violated

Returns

value: region_name.lower(), it is case insensitive

B.13.1 LocationSerializer

This makes a django serializer object for the Location class.
Inheritance diagram for LocationSerializer:



Detailed Description

This makes a django serializer object for the Location class.
The documentation for this class was generated from the following file:

- travelmatch/ai/**serializers.py**

B.13.2 Meta

This creates the serializer with specific model and fields.

Static Public Attributes

- **model** = Location
- list **fields** = ['loc_id', 'city_name', 'country_name', 'region_name']
- tuple **fields**

Detailed Description

This creates the serializer with specific model and fields.
This create a model and a fields for the serializer.

Member Data Documentation

list fields = ['loc_id', 'city_name', 'country_name', 'region_name'] `[static]`

tuple fields `[static]`
Initial value:

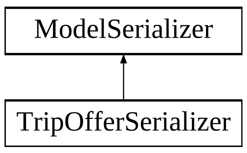
```
= ('offer_id', 'name', 'description', 'hotel_stars', 'price', 'link',
                 'image', 'min_people', 'dept_date', 'duration_days', 'user_rating', )
```

model = Location `[static]`
The documentation for this class was generated from the following file:

- travelmatch/ai/**serializers.py**

B.13.3 TripOfferSerializer

This is for trip offer serializer.
Inheritance diagram for TripOfferSerializer:



Detailed Description

This is for trip offer serializer.
The documentation for this class was generated from the following file:

- travelmatch/ai/**serializers.py**

B.14 ai.views

`ai.views` is a namespace that contains classes, variables and functions that relate to the artificial intelligence views. This namespace would contain Django view controllers for managing the artificial intelligence data; however, as no Django view controllers are required, this namespace is left empty.

B.15 appusers

`appusers` is a namespace that contains classes, variables and functions that relate to the user management components. This namespace mainly acts as a container for several other namespaces, namely those relating to authentication, Mailgun integration, the models, serializers and views.

Namespaces

- **authentication**
- **mailgun**
- **models**
- **serializers**
- **views**

B.16 appusers.authentication

`appusers.authentication` is a namespace that contains classes, variables and functions that relate to user authentication. This namespace contains a class that implements the abstract `JSONWebTokenAuthentication` class from the JSON Web Token library, to provide a custom authentication function that authenticates TravelMatch-specific credentials.

Classes

- class **MyJSONWebTokenAuthenticator**
  inherits authentication mechanism from JSONWebTokenAuthentication

Variables

- **jwt_payload_handler** = api_settings.JWT_PAYLOAD_HANDLER
- **jwt_encode_handler** = api_settings.JWT_ENCODE_HANDLER
- **jwt_decode_handler** = api_settings.JWT_DECODE_HANDLER
- **jwt_get_user_id_from_payload** = api_settings.JWT_PAYLOAD_GET_USER_ID_HANDLER

Variable Documentation

jwt_decode_handler = api_settings.JWT_DECODE_HANDLER

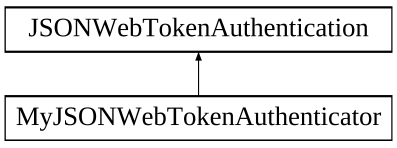
jwt_encode_handler = api_settings.JWT_ENCODE_HANDLER

jwt_get_user_id_from_payload = api_settings.JWT_PAYLOAD_GET_USER_ID_HANDLER

jwt_payload_handler = api_settings.JWT_PAYLOAD_HANDLER

B.16.1 MyJSONWebTokenAuthenticator

inherits authentication mechanism from JSONWebTokenAuthentication
Inheritance diagram for MyJSONWebTokenAuthenticator:



Public Member Functions

- def **authenticate_credentials** (self, payload)
  Returns an active user that matches the payload's user id and email.

Detailed Description

inherits authentication mechanism from JSONWebTokenAuthentication

Member Function Documentation

def authenticate_credentials (self, payload) Returns an active user that matches the payload's user id and email.

Parameters

payload	The input user
<payload>	object

Precondition

payload is a valid user with valid content

Returns

user: The correlated user object

Exceptions

AuthenticationFailed	if the precondition failed

The documentation for this class was generated from the following file:

- travelmatch/appusers/**authentication.py**

B.17 appusers.mailgun

`appusers.mailgun` is a namespace that contains classes, variables and functions that relate to Mailgun integration. This namespace contains function to send e-mails to TravelMatch users on behalf of TravelMatch, via Mailgun. A built-in function is also provided to send confirmation e-mails for newly registered users.

Functions

- def **send_to_mailgun** (**sender**, to, **subject**, htmltext, plaintext)
  The actual sending of the message with desired variables.
- def **create_htmltext** (to, userid, key)
  Creating the email in html.
- def **create_plaintext** (to, userid, key)
  Creating the email in plaintext.
- def **send_confirmation_message** (to, userid, key)
  This function can be called from anywhere else in the server to have an email created_on and sent with:

Variables

- string **sender** = "noreply@gotravelmatch.com"
- string **subject** = "Confirm your new TravelMatch account"
- string **mailgun_api_base_url** = "https://api.mailgun.net/v3/gotravelmatch.com/messages"
- string **mailgun_api_key** = "key-07bd4dbc95e6f5d62192e1d4d6a7ace5"
- string **AUTH_LINK_PATTERN** = BASE_URL+API_URL+"/user/auth?userid={!s}&key={!s}"

Function Documentation

def appusers.mailgun.create_htmltext (to, userid, key) Creating the email in html.

Parameters

to	the receivers' email address
<*to*>	[email]
userid	the userid generated on the server for that new user
<*userid*>	int
key	key generated to verify that the user indeed received the email on the specified address
<*key*>	string

Returns

plaintext: the html text email with correct address, receivers and content

def appusers.mailgun.create_plaintext (to, userid, key) Creating the email in plaintext.
Parameters

to	the receivers' email address
<*to*>	[email]
userid	the userid generated on the server for that new user
<*userid*>	int
key	key generated to verify that the user indeed received the email on the specified address
<*key*>	string

Returns

plaintext: the plain text email with correct address, receivers and content

def appusers.mailgun.send_confirmation_message (to, userid, key) This function can be called from anywhere else in the server to have an email created_on and sent with:
Parameters

to	the receivers' email address
<*to*>	[email]
userid	the userid generated on the server for that new user
<*userid*>	int
key	key generated to verify that the user indeed received the email on the specified address
<*key*>	string : the email address specified by the user to which the email is sent

def appusers.mailgun.send_to_mailgun (sender, to, subject, htmltext, plaintext) The actual sending of the message with desired variables.
Parameters

sender	email of the sender
<*sender*>	string
to	email of the receiver
<*to*>	string

htmltext	the htmltext to be sent
<htmltext>	string
plaintext	the plain text to be sent
<plaintext>	string
subject	subject of the email
<subject>	string : email sent with input parameters

Variable Documentation

string AUTH_LINK_PATTERN = BASE_URL+API_URL+"/user/auth?userid={!s}&key={!s}"

string mailgun_api_base_url = "https://api.mailgun.net/v3/gotravelmatch.com/messages"

string mailgun_api_key = "key-07bd4dbc95e6f5d62192e1d4d6a7ace5"

string sender = "noreply@gotravelmatch.com"

string subject = "Confirm your new TravelMatch account"

B.18 appusers.models

`appusers.models` is a namespace that contains classes, variables and functions that relate to the artificial intelligence models. This namespace contains all Django models related to TravelMatch users, including various types of user credentials, vacation details, pending activations, and saved trip offers.

Classes

- class **AppUser**

  This is the class represents the application user of the application.
- class **FBAppUser**

  This represents the facebook users of the application users.
- class **GuestAppUser**

  This represents the guest user of the application.
- class **MailAppUser**

  This represents the mail user of the application user.
- class **PendingActivation**

  This represents the mail users yet to be activated.
- class **SavedLocation**

  This is the model for the trip saved locations.
- class **TripList**

  This model represents the list of trips the users have.
- class **TripListEntry**

  This is the model for the trip list entries.
- class **VacationDetail**

  This represents the detail of a certain vacation.

Functions

- def **encode**

  This function encode wit pbkdf2 method the password.
- def **verify** (entered, encoded)

  This function compares entered password with encoded password.
- def **default_start_date** ()

  This function sets the default start date return now()+timedelta(days=5): the 5 days after current day.
- def **default_end_date** ()

  This function sets the default end date return now() + timedelta(days=5+7): the 5+7 days after current day.

Function Documentation

def appusers.models.default_end_date ( ) This function sets the default end date return now() + timedelta(days=5+7): the 5+7 days after current day.

def appusers.models.default_start_date ( ) This function sets the default start date return now()+timedelta(days=5): the 5 days after current day.

def appusers.models.encode (password, salt = *None***, iterations =** *10000* **)** This function encode wit pbkdf2 method the password.
Parameters

password	the input password
<password>	string
salt	the salt of the pbkdf2
<salt>	string
iterations	iterations of pbkdf2
<iterations>	int

Precondition

None encode the password

Returns

hashed passowrd

def appusers.models.verify (entered, encoded) This function compares entered password with encoded password.
Parameters

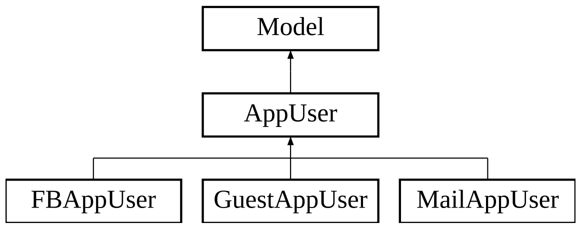
entered	entered password
<entered>	string
encoded	encoded password
<entered>	string

Returns

True if encode(entered)==encoded

B.18.1 AppUser

This is the class represents the application user of the application.
Inheritance diagram for AppUser:



Classes

- class **Meta**

  This makes the humam-readable name of the table, and the means to sort it.

Public Member Functions

- def **is_authenticated** (self)

  this set the authenticated to true
- def **__unicode__** (self)

  This returns the AppUser object in certain format in a string.
- def **display** (self)

  This display the user and it's user id.
- def **create**

  creating a new tuple (not update, just create)
- def **latest_vac** (self)

  This returns the latest vacation of the app user.

Static Public Attributes

- tuple **user_id** = models.AutoField(primary_key=True)
- tuple **name** = models.CharField(max_length=500, null=False, default="")
- tuple **gender** = models.CharField(max_length=20, null=False, default="none")
- tuple **birthday** = models.DateField(null=False, default=date(1, 1, 1))
- tuple **activation** = models.BooleanField(default=True)

Detailed Description

This is the class represents the application user of the application.

Member Function Documentation

def __unicode__ (self) This returns the AppUser object in certain format in a string.

Returns

string: u's s' % (self.user_id, self.name)

def create (self, force_insert = *False***, force_update =** *False***, using =** *None***, update↩ _fields =** *None* **)** creating a new tuple (not update, just create)

Precondition

the tuple does not exist int e database

Postcondition

tuple updated

Exceptions

IntegrityError	if the was precondition violated

Returns

Void new Tag object

def display (self) This display the user and it's user id.

Returns

string: "AppUser "+str(self.user_id)

def is_authenticated (self) this set the authenticated to true

Returns

True

def latest_vac (self) This returns the latest vacation of the app user.

Precondition

vacs.last() exist

Returns

vacs.last(): the latest vacation from the query

Exceptions

error	if the precondition is violated

Member Data Documentation

tuple activation = models.BooleanField(default=True) `[static]`

tuple birthday = models.DateField(null=False, default=date(1, 1, 1)) `[static]`

tuple gender = models.CharField(max_length=20, null=False, default="none") `[static]`

tuple name = models.CharField(max_length=500, null=False, default="") `[static]`

tuple user_id = models.AutoField(primary_key=True) `[static]`
The documentation for this class was generated from the following file:

- travelmatch/appusers/**models.py**

B.18.2 AppUser.Meta

This makes the humam-readable name of the table, and the means to sort it.

Static Public Attributes

- string **verbose_name** = "App User"
- tuple **ordering** = ('**user_id**',)

    < the human readable name is "App User"

Detailed Description

This makes the humam-readable name of the table, and the means to sort it.

Member Data Documentation

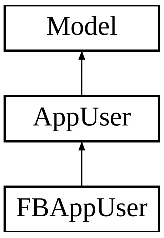
tuple ordering = ('user_id',) `[static]`
< the human readable name is "App User"

string verbose_name = "App User" `[static]`
The documentation for this class was generated from the following file:

- travelmatch/appusers/**models.py**

B.18.3 FBAppUser

This represents the facebook users of the application users.
Inheritance diagram for FBAppUser:



Public Member Functions

- def **display** (self)

    Returns the facebook user with certain format in a string "@return FBUser "+str(self.user_id)+" "+str(self.fbid)

Static Public Attributes

- tuple **fbid** = models.CharField(max_length=256, blank=False)

Detailed Description

This represents the facebook users of the application users.

Member Function Documentation

def display (self) Returns the facebook user with certain format in a string "@return FBUser "+str(self.user_id)+" "+str(self.fbid)

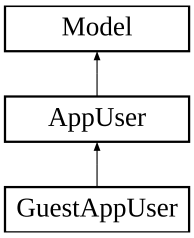
Member Data Documentation

tuple fbid = models.CharField(max_length=256, blank=False) `[static]`
The documentation for this class was generated from the following file:

- travelmatch/appusers/**models.py**

B.18.4 GuestAppUser

This represents the guest user of the application.
Inheritance diagram for GuestAppUser:



Public Member Functions

- def **display** (self)

  Returns the guest user with certain format in a string "@return "GuestAppUser "+str(self.user_id)+" "+str(self.timestamp)

Static Public Attributes

- tuple **device_id** = models.CharField(max_length=250, blank=False, primary_key=True)
- tuple **timestamp** = models.DateTimeField(auto_now=True)

Detailed Description

This represents the guest user of the application.

Member Function Documentation

def display (self) Returns the guest user with certain format in a string "@return "GuestAppUser "+str(self.user_id)+" "+str(self.timestamp)

Member Data Documentation

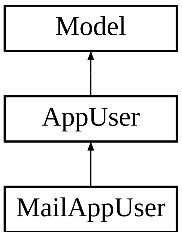
tuple device_id = models.CharField(max_length=250, blank=False, primary_key=True) `[static]`

tuple timestamp = models.DateTimeField(auto_now=True) `[static]`
The documentation for this class was generated from the following file:

- travelmatch/appusers/**models.py**

B.18.5 MailAppUser

This represents the mail user of the application user.
Inheritance diagram for MailAppUser:



Public Member Functions

- def **send_activation** (self)

  < email of the user (string)
- def **display** (self)

  Return the Mail user in certain format.

Public Attributes

- **activation**

Static Public Attributes

- tuple **password** = models.CharField(max_length=256, blank=False)
- tuple **email** = models.CharField(max_length=256, blank=False)

  < hashed password of the user (string)

Detailed Description

This represents the mail user of the application user.

Member Function Documentation

def display (self) Return the Mail user in certain format.

Returns

"MailUser "+str(self.user_id)+" "+str(self.email)

def send_activation (self) < email of the user (string)
This function send the activation key to the mail user and removes old pending activations

Precondition

None

Postcondition

sent the email

Returns

Void

Member Data Documentation

activation

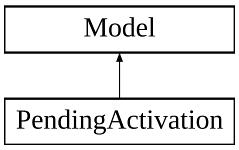
tuple email = models.CharField(max_length=256, blank=False) `[static]`
< hashed password of the user (string)

tuple password = models.CharField(max_length=256, blank=False) `[static]`
The documentation for this class was generated from the following file:

- travelmatch/appusers/**models.py**

B.18.6 PendingActivation

This represents the mail users yet to be activated.
Inheritance diagram for PendingActivation:



Classes

- class **Meta**
  This set the default ordering of PendingActivations to DESC timestemp.

Public Member Functions

- def **__unicode__** (self)
  Returns the PendingActivation with certain format in a string.

Static Public Member Functions

- def **id_generator**
  This method generates the id.

Static Public Attributes

- tuple **user** = models.ForeignKey(**MailAppUser**)
- tuple **timestamp** = models.DateTimeField(auto_now_add=True)
- tuple **key** = models.CharField(max_length=255, null=False)

Detailed Description

This represents the mail users yet to be activated.

Member Function Documentation

def __unicode__ (self) Returns the PendingActivation with certain format in a string.

Returns

: u's: user s -> s' % (self.timestamp, self.user, self.key)

def id_generator (size = *20*, **chars =** *string.ascii_uppercase + string.digits* **)** `[static]`

This method generates the id.

Parameters

size	the size of the id. default value is 20 (int)
<*size*>	int
chars	the id will be generated using the characters from this, default all cases letters (list)
<*chars*>	list

Returns

string: ''.join(random.choice(chars) for _ in range(size))

Member Data Documentation

tuple key = models.CharField(max_length=255, null=False) `[static]`

tuple timestamp = models.DateTimeField(auto_now_add=True) `[static]`

tuple user = models.ForeignKey(MailAppUser) `[static]`

The documentation for this class was generated from the following file:

- travelmatch/appusers/**models.py**

B.18.7 PendingActivation.Meta

This set the default ordering of PendingActivations to DESC timestemp.

Static Public Attributes

- tuple **ordering** = ('**-timestamp**',)

Detailed Description

This set the default ordering of PendingActivations to DESC timestemp.

Member Data Documentation

tuple ordering = ('-timestamp',) `[static]`

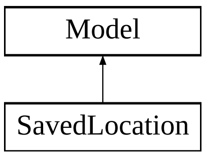
The documentation for this class was generated from the following file:

- travelmatch/appusers/**models.py**

B.18.8 SavedLocation

This is the model for the trip saved locations.

Inheritance diagram for SavedLocation:



Classes

- class **Meta**

  This class enforce loc_list&&user as a super key.

Static Public Attributes

- tuple **loc** = models.ForeignKey('ai.Location')
- tuple **user** = models.ForeignKey(**AppUser**)
- tuple **loc_list** = models.ForeignKey(**TripList**)

Detailed Description

This is the model for the trip saved locations.

Member Data Documentation

tuple loc = models.ForeignKey('ai.Location') `[static]`

tuple loc_list = models.ForeignKey(TripList) `[static]`

tuple user = models.ForeignKey(AppUser) `[static]`

The documentation for this class was generated from the following file:

- travelmatch/appusers/**models.py**

B.18.9 SavedLocation.Meta

This class enforce loc_list&&user as a super key.

Static Public Attributes

- tuple **unique_together** = ((”loc_list”, ”**user**”),)

Detailed Description

This class enforce loc_list&&user as a super key.

Member Data Documentation

tuple unique_together = ((”loc_list”, ”user”),) `[static]`

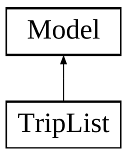
The documentation for this class was generated from the following file:

- travelmatch/appusers/**models.py**

B.18.10 TripList

This model represents the list of trips the users have.

Inheritance diagram for TripList:



Static Public Member Functions

- def **create_trip_list** (offer_ids)

Static Public Attributes

- tuple **trip_list_id** = models.AutoField(primary_key=True)

Detailed Description

This model represents the list of trips the users have.

Member Function Documentation

def create_trip_list (offer_ids) `[static]`

Exceptions

ObjectDoesNotExist	is one of the loc_ids doesn't exists anymore

Returns

a new TripList instance from the db

Member Data Documentation

tuple trip_list_id = models.AutoField(primary_key=True) `[static]`

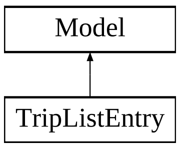
The documentation for this class was generated from the following file:

- travelmatch/appusers/**models.py**

B.18.11 TripListEntry

This is the model for the trip list entries.

Inheritance diagram for TripListEntry:



Public Member Functions

- def **generate_api_response** (self)

  < *this is no foreign key to keep it even if the hotels disappear*

Static Public Attributes

- tuple **trip_list** = models.ForeignKey(**TripList**, null=False)
- tuple **cached_name** = models.CharField(null=False, max_length=64)

  < *list of trips*
- tuple **trip_offer_id** = models.IntegerField(null=True)

  < *see TripOffer.name*

Detailed Description

This is the model for the trip list entries.

Member Function Documentation

def generate_api_response (self) < this is no foreign key to keep it even if the hotels disappear
This function returns a trip list as an api response

Returns

ser.data: the serialized TripOffer object

Member Data Documentation

tuple cached_name = models.CharField(null=False, max_length=64) `[static]`
< list of trips

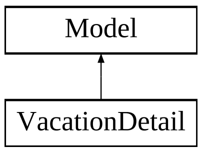
tuple trip_list = models.ForeignKey(TripList, null=False) `[static]`

tuple trip_offer_id = models.IntegerField(null=True) `[static]`
< see TripOffer.name
The documentation for this class was generated from the following file:

- travelmatch/appusers/**models.py**

B.18.12 VacationDetail

This represents the detail of a certain vacation.
Inheritance diagram for VacationDetail:



Classes

- class **Meta**

  < timestamp of the last added TravelDNA (datetime)

Public Member Functions

- def **__unicode__** (self)

  this method returns the VacationDetail object in a string whenever we need to return a VacationDetail object
- def **create**

  creating a new tag tuple (not update, just create)
- def **display_user** (self)

  display the user
- def **display** (self)

  display the user
- def **persons_total** (self)

  The total persons.

Static Public Attributes

- tuple **internal_id** = models.AutoField(primary_key=True)
- tuple **user** = models.ForeignKey(**AppUser**, null=False)

  < *This offers the internal id of the model (int) not in the db desigh*
- tuple **vac_id** = models.PositiveSmallIntegerField(null=False, default=0)

  < *The user involved in the vacation (user)*
- tuple **vac_name** = models.CharField(max_length=256, null=False, blank=True, default="")

  < *id of the vacation(int) the default vacation has id 0*
- tuple **start_date** = models.DateField(null=False, blank=False, default=**default_start_date**())

  < *name of the vation, (string) "" means the default vacation*
- tuple **start_date_extend** = models.SmallIntegerField(null=False, default=0)

  < *date vacation starts (date)*
- tuple **end_date** = models.DateField(null=False, blank=False, default=**default_end_date**())

  < *the extend margin of the start date (int)*
- tuple **end_date_extend** = models.SmallIntegerField(null=False, default=0)

  < *date vacation ends (date)*
- tuple **persons_adults** = models.PositiveSmallIntegerField(null=False, blank=False, default=1)

  < *extend margin of the end date (int)*
- tuple **persons_children** = models.PositiveSmallIntegerField(null=False, default=0)

  < *number of adult persons involved (int)*
- tuple **budget** = models.PositiveIntegerField(null=False, default=0)

  < *number of children involved (int)*
- tuple **last_modified** = models.DateTimeField(null=False, blank=False, auto_now=True)

  < *buget of the vacation, in cents (int)*
- tuple **unique_together** = (("user", "**vac_id**"),)

Detailed Description

This represents the detail of a certain vacation.

Member Function Documentation

def __unicode__ (self) this method returns the VacationDetail object in a string whenever we need to return a VacationDetail object

Returns

u's: s'%(self.user, self.vac_name) out put string with certain format

def create (self, force_insert = *False***, force_update =** *False***, using =** *None***, update↩ _fields =** *None* **)** creating a new tag tuple (not update, just create)

Precondition

the tuple does not exist int e database

Postcondition

tuple updated

Exceptions

IntegrityError	if the was precondition violated

Returns

Void new Tag object

def display (self) display the user

Precondition

user.vac_name == vac_name

Returns

name of the vacation if it exist, else return "Niet gedefineerd"

def display_user (self) display the user

Returns

name (name of the user)

def persons_total (self) The total persons.

Returns

persons_adults + persons_children

Member Data Documentation

tuple budget = models.PositiveIntegerField(null=False, default=0) `[static]`
< number of children involved (int)

tuple end_date = models.DateField(null=False, blank=False, default=default_end_date()) `[static]`
< the extend margin of the start date (int)

tuple end_date_extend = models.SmallIntegerField(null=False, default=0) `[static]`
< date vacation ends (date)

tuple internal_id = models.AutoField(primary_key=True) `[static]`

tuple last_modified = models.DateTimeField(null=False, blank=False, auto_now=True) `[static]`
< buget of the vacation, in cents (int)

tuple persons_adults = models.PositiveSmallIntegerField(null=False, blank=False, default=1) `[static]`
< extend margin of the end date (int)

tuple persons_children = models.PositiveSmallIntegerField(null=False, default=0) `[static]`
< number of adult persons involved (int)

tuple start_date = models.DateField(null=False, blank=False, default=default_start_date()) `[static]`
< name of the vation, (string) "" means the default vacation

tuple start_date_extend = models.SmallIntegerField(null=False, default=0) `[static]`
< date vacation starts (date)

tuple unique_together = (("user", "vac_id"),) `[static]`

tuple user = models.ForeignKey(AppUser, null=False) `[static]`
< This offers the internal id of the model (int) not in the db desigh

tuple vac_id = models.PositiveSmallIntegerField(null=False, default=0) `[static]`
< The user involved in the vacation (user)

tuple vac_name = models.CharField(max_length=256, null=False, blank=True, default="") `[static]`
< id of the vacation(int) the default vacation has id 0
The documentation for this class was generated from the following file:

- travelmatch/appusers/**models.py**

B.18.13 VacationDetail.Meta

< timestamp of the last added TravelDNA (datetime)

Detailed Description

< timestamp of the last added TravelDNA (datetime)
This makes (user && vac_id) a super key
The documentation for this class was generated from the following file:

- travelmatch/appusers/**models.py**

B.19 appusers.serializers

`appusers.serializer` is a namespace that contains classes, variables and functions that relate to the AI serializers. This namespace contains Django serializers, which verify and modify the input and output of data in the model.

Classes

- class **FBUserSerializer**
  This makes a django serializer object for the FBAppUser class.
- class **GuestAccountSerializer**
  This makes a django serializer object for the GuestAppUser class.
- class **MailUserSerializer**
  This makes a django serializer object for the MailUser class.
- class **Meta**
  This creates the serializer with specific model and fields.
- class **UserSerializer**

This makes a django serializer object for the AppUser class.

- class **VacationDetailsSerializer**

  This makes a django serializer object for the VacationDetail class.

Functions

- def **validate_email** (self, value)

  this method returns validated email
- def **validate_password** (self, value)

  this method returns validated password
- def **validate_fbid** (self, value)

  this method returns validated fbid
- def **validate_device_id** (self, value)

  this method returns validated device_id
- def **validate_user** (self, value)

  this method returns validated user
- def **validate_vac_id** (self, value)

  this method returns validated vac_id
- def **validate_start_date** (self, value)

  this method returns validated
- def **validate_end_date** (self, value)

  this method returns validated end_date
- def **validate_persons_adults** (self, value)

  this method returns validated persons
- def **validate_vac_name** (self, value)

  this method returns validated vac_name
- def **validate_budget** (self, value)

  this method returns validated budget

Variables

- list **VAC_FIELDS**

Function Documentation

def appusers.serializers.validate_budget (self, value) this method returns validated budget

Parameters

value	the input budget
<*value*>	int

Precondition

value < 0

Exceptions

ValidationError	if precondition is violated

Returns

value: validated budget

def appusers.serializers.validate_device_id (self, value) this method returns validated device_id

Parameters

value	the input device_id
<*value*>	string

Returns

fbid: facebook id and it is validated

def appusers.serializers.validate_email (self, value) this method returns validated email

Parameters

value	the input city_name
<*value*>	string

Precondition

len(value) == 0

Exceptions

ValidationError	if precondition is violated

Returns

value: mail.lower(), it is case insensitive

def appusers.serializers.validate_end_date (self, value) this method returns validated end_date

Parameters

value	the input end_date (date)
<*value*>	DateField

Returns

end_date and it is validated

def appusers.serializers.validate_fbid (self, value) this method returns validated fbid

Parameters

value	the input fbid
<*value*>	string

Returns

fbid: facebook id and it is validated

def appusers.serializers.validate_password (self, value) this method returns validated password

Parameters

value	the input password
<*value*>	string

Precondition

len(value) == 0

Exceptions

ValidationError	if precondition is violated

Returns

value: password and it is validated

def appusers.serializers.validate_persons_adults (self, value) this method returns validated persons
Parameters

value	the personsNI
<*value*>	int

Returns

value: persons and it is validated

def appusers.serializers.validate_start_date (self, value) this method returns validated
Parameters

value	the input start_date
<*value*>	DateField

Returns

value: start_date and it is validated

def appusers.serializers.validate_user (self, value) this method returns validated user
Parameters

value	the input user (User object)
<*value*>	AppUser

Returns

user and it is validated

def appusers.serializers.validate_vac_id (self, value) this method returns validated vac_id
Parameters

value	the input vac_id
<*value*>	VacationDetail

Returns

value: vac_id and it is validated

def appusers.serializers.validate_vac_name (self, value) this method returns validated vac_name

Parameters

value	the input vac_name
<value>	string

Precondition

value != null

Exceptions

ValidateError	if precondition is violated

Returns

value: validated vacation name

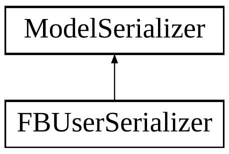
Variable Documentation

list VAC_FIELDS Initial value:

```
= ['user', 'vac_id', 'vac_name', 'start_date', 'start_date_extend', 'end_date', 'end_date_extend',
                  'persons_adults', 'persons_children', 'budget']
```

B.19.1 FBUserSerializer

This makes a django serializer object for the FBAppUser class.
Inheritance diagram for FBUserSerializer:



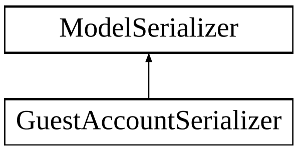
Detailed Description

This makes a django serializer object for the FBAppUser class.
The documentation for this class was generated from the following file:

- travelmatch/appusers/**serializers.py**

B.19.2 GuestAccountSerializer

This makes a django serializer object for the GuestAppUser class.
Inheritance diagram for GuestAccountSerializer:



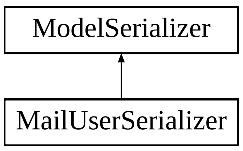
Detailed Description

This makes a django serializer object for the GuestAppUser class.
The documentation for this class was generated from the following file:

- travelmatch/appusers/**serializers.py**

B.19.3 MailUserSerializer

This makes a django serializer object for the MailUser class.
Inheritance diagram for MailUserSerializer:



Detailed Description

This makes a django serializer object for the MailUser class.
The documentation for this class was generated from the following file:

- travelmatch/appusers/**serializers.py**

B.19.4 Meta

This creates the serializer with specific model and fields.

Static Public Attributes

- **model** = **AppUser**
- tuple **fields** = ('name', 'gender', 'birthday')
- list **fields** = ['email', 'password']
- **fields** = **VAC_FIELDS**

Detailed Description

This creates the serializer with specific model and fields.

Member Data Documentation

tuple fields = ('name', 'gender', 'birthday') `[static]`

list fields = ['email', 'password'] `[static]`

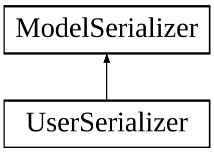
fields = VAC_FIELDS `[static]`

model = AppUser `[static]`
The documentation for this class was generated from the following file:

- travelmatch/appusers/**serializers.py**

B.19.5 UserSerializer

This makes a django serializer object for the AppUser class.
Inheritance diagram for UserSerializer:



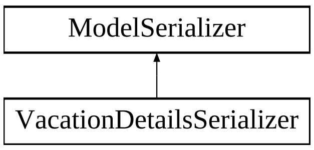
Detailed Description

This makes a django serializer object for the AppUser class.
The documentation for this class was generated from the following file:

- travelmatch/appusers/**serializers.py**

B.19.6 VacationDetailsSerializer

This makes a django serializer object for the VacationDetail class.
Inheritance diagram for VacationDetailsSerializer:



Detailed Description

This makes a django serializer object for the VacationDetail class.
The documentation for this class was generated from the following file:

- travelmatch/appusers/**serializers.py**

B.20 appusers.views

`appusers.views` is a namespace that contains classes, variables and functions that relate to the artificial intelligence views. This namespace contain Django controllers for the TravelMatch API, as well as mappings from REST endpoints to Django functions.

Classes

- class **APIError**

  default class that just returns an error
- class **APIRecommendation**

  This class queries existing VacationDetail to see its recommendation.
- class **APIUser**

  API for log in related issues, this class is used whenever a url(api_pattern.format('/user') is called.
- class **APIUserAuth**

  API for MailUser authentication, when url(api_pattern.format('/user/auth'), APIUserAuth.as_view()) is called.

- class **APIUserHotel**

  This class gets the hotel user use for certain trip offer.
- class **APIUserLocation**

  This is for get and post SaveLocation from AppUser.
- class **APIUserLogin**

  API for log in, when url(api_pattern.format('/user/login'), APIUserLogin.as_view()), is called.
- class **APIUserMe**

  API for retrieve and modify ones profile.
- class **APIUserMyLocation**

  This is the api for the location of an user.
- class **APIUserMyLocationAll**

  API for getting all of user's locations.
- class **APIUserMyLocationAllValues**

  This is the API to get all the value from a user's locations.
- class **APIUserSwipe**

  The API for the swipe operations carried out by users.
- class **APIUserVacationDetails**

  API for user's vacation details.
- class **APIUserVacationDetailsAllValues**

  This is the API for query all the VacationDetail object from a certain user.
- class **APIUserVacationDetailsLatest**

  This is the API for query all the VacationDetail object from a certain user.
- class **JSONResponse**

  An HttpResponse that renders its content into JSON.

Functions

- def **__init__** (self, data, kwargs)

  Initializing the data.
- def **create_token_response_data** (user)

  This method is for creating a Json Web Token for the user.
- def **decode_string** (encoded)

  Decoding the data from b64.
- def **encode_string** (s)

  Encoding the data into b64.
- def **create_encoding_err_resp** (field)

  This is for creating a http response if the field is not properly encoded.
- def **post**

  < *The permission of the /user: is allowed by everyone*
- def **delete** (self, request)

  This method delet a user.
- def **get** (self, request)

  < *The permission of the /user: is allowed by everyone*
- def **post** (self, request)

  < *The permission of the /user: is allowed by everyone*
- def **update_field** (key, data, obj)

  This is the method to update the fields in an object.

- def **put** (self, request)
  This method update the field requested to be updated.
- def **create_vacation_response_data** (vac)
  This function create a response for VacationDetail object.
- def **is_unique_vac_name** (vac_name, vac_id, vacs)
  This method checks if certain vacation exist in the given list of vacations and its uniqueness.
- def **update_vacation** (vac, data)
  Function that update and maintains the VactionDetail object if it exist.
- def **record_swipe** (vacation, img, like)
  This method record a swipe operation carried out by user.
- def **create_new_images_response** (n, user, vac)
  This is the method that calls the AI to get a new set of images based on the entropy calculation.

Variables

- tuple **logger** = logging.getLogger(__name__)
- string **DEFAULT_VACATION_NAME** = "Vakantie"
- tuple **permission_classes** = (AllowAny,)

Function Documentation

def appusers.views.__init__ (self, data, kwargs) Initializing the data.

```
An HttpResponse that renders its content into JSON.
```

Precondition

None

Postcondition

None

Returns

None

def appusers.views.create_encoding_err_resp (field) This is for creating a http response if the field is not properly encoded.
Parameters

field	the field
<*field*>	field, django field

Precondition

field is not encoded

Returns

http response with the field, message that it is not encoded, and 400 http statues code

def appusers.views.create_new_images_response (n, user, vac) This is the method that calls the AI to get a new set of images based on the entropy calculation.

Parameters

n	amount of SwipeImages needed
<*n*>	int
user	the AppUser that needs SwipeImages
<*user*>	AppUser
vac	input vacation details
<*vac*>	VacationDetails

Precondition

n lager than 0

Returns

a list of SwipeImages

def appusers.views.create_token_response_data (user) This method is for creating a Json Web Token for the user.

Parameters

user	the user you need to create token for
<*user*>	AppUser

Precondition

None

Postcondition

None

Returns

response_data the JWT containing the user information

def appusers.views.create_vacation_response_data (vac) This function create a response for VacationDetail object.

Parameters

vac	the input VacationDetail object
<*vac*>	VacationDetail

Precondition

the input is a VacationDetail object

Returns

response_data with the vac as http response, but without vac.user

def appusers.views.decode_string (encoded) Decoding the data from b64.

Parameters

encoded	the string you wish to decode
<*encoded*>	string

Precondition

None

Postcondition

None

Returns

decoded string from encoded

def delete (self, request) This method delet a user.
This method deletes a location.
This is the method for deleting VacationDetail objects.
Parameters

request	html request
<*request*>	html request

Returns

Response: response with corresponding http status code : request.user.delete(): if user exist, remove the user from the database

Parameters

request	the incoming http request
<*request*>	HttpRequest

Precondition

None

Returns

response and http status code

Parameters

request	the incoming http request
<*request*>	HttpRequest

Precondition

the vacation exist, the loc_id exist

Returns

Response: string message with whether get operations is successful with http status code

def appusers.views.encode_string (s) Encoding the data into b64.

Parameters

s	the sting you wish to encode
<*s*>	string

Precondition

None

Postcondition

None

Returns

encoded string from s

def get (self, request) < The permission of the /user: is allowed by everyone
This method returns all the hotels and its id for a TripOffer.
This method queries for Locations.
This method queries for Location.
This method queries for Savelocation.
This method queries for recommendation.
This method queries for n images for certain vacation.
This is the function that query all the VacationDetail object from a certain user.
This is the method for querying and returning VacationDetail object.
This methods fetch and returns the user data.
This is the function that sends the authentication token
Parameters

request	the incoming http request
<*request*>	HttpRequest

Precondition

it is from a MailUser and the information is correct None

Returns

response with authenticity token

Exceptions

def appusers.views.is_unique_vac_name (vac_name, vac_id, vacs) This method checks if certain vacation exist in the given list of vacations and its uniqueness.
Parameters

vac_name	name of the vacation
<*vac_name*>	string

vac_id	id of the vacation
<*vac_id*>	int
vacs	the list of vacation to be checked
<*vacs*>	[VacationDetails]

Returns

True if vacation in vac and it is unique, False otherwise

def appusers.views.post (self, request, format = *None* **)** < The permission of the /user: is allowed by everyone

This methods post the data from the request check it and stores them in the database

Parameters

request	the incoming http request
<*request*>	HttpRequest
format	the format
<*format*>	format

Precondition

the user information sent via request is either a MailUser or a FB user

Returns

access token

Exceptions

error	if precondition is violated

def post (self, request) < The permission of the /user: is allowed by everyone

This method storing for location.

This method records a swipe and then provide a new image to swipe.

This method creates a new VacationDetail Object from.

This returns an access token for log in

Parameters

request	the incoming http request
<*request*>	HttpRequest

Precondition

the request is a valid request with correct criteria for a FBUser or a MailUser

Postcondition

None

Returns

response with access token

Exceptions

error	and corresponding http status code if the precondition is violated

Parameters

request	the incoming http request
<*request*>	HttpRequest

Precondition

None

Postcondition

the VacationDetail updated if it exist beforehand

Returns

http response with the updated data if it exist beforehand, message with http status code otherwise

Parameters

request	the incoming http request
<*request*>	HttpRequest

Precondition

request.data.image and the like exist, and there are more than 'n' images in the request.data to swipe, image exist : new swipe recorded

Returns

Response: string message with whether input satisfy the preconditions, the response with http status code

Parameters

request	the incoming http request
<*request*>	HttpRequest

Precondition

request.data['loc_id'] exist along with the request.data['user'] : new SaveLocation saved

Returns

Response: serialized message with SaveLocation object if it exist, and the response with http status code

def put (self, request) This method update the field requested to be updated.
This function puts the requested data JSONResponse and returns it.

Parameters

request	the incoming http request
<*request*>	HttpRequest

Precondition

None

Postcondition

None : the data that needs to be modified, if it exist

Returns

response with corresponding http status code

Parameters

request	http request
<*request*>	http request

Returns

JSONResponse with corresponding http status code

def appusers.views.record_swipe (vacation, img, like) This method record a swipe operation carried out by user.

Parameters

vacation	the vacation that being recorded
<*vacation*>	VacationDetail
img	the swipe image involved
<*img*>	SwipeImage
like	whether user liked the image or not
<*like*>	boolean
img	the input swipe image
<*img*>	SwipeImage

Precondition

there are no more than 1 TravelDNA with the same vacation && img the new swipe stored

Exceptions

IntegrityError	if precondition is violated

Returns

swipe: the input swipe with vaction, img, and like

def appusers.views.update_field (key, data, obj) This is the method to update the fields in an object.

Parameters

key	the keys with update of the object
<*key*>	abstract data type
data	the input data
<*data*>	data
obj	the object
<*obj*>	object

Precondition

None

Returns

modified the updated data

def appusers.views.update_vacation (vac, data) Function that update and maintains the VactionDetail object if it exist.

Parameters

vac	the VacationDetail object that needs to be updated
<*vac*>	VacationDetail
data	the new data for the update
<*data*>	data

Precondition

the related VacationDetail exists

Postcondition

the VacationDetail object up to date

Returns

response with up to date vac, or an error message with http status code

Variable Documentation

string DEFAULT_VACATION_NAME = "Vakantie"

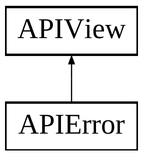
tuple logger = logging.getLogger(__name__)

tuple permission_classes = (AllowAny,)

B.20.1 APIError

default class that just returns an error

Inheritance diagram for APIError:



Detailed Description

default class that just returns an error

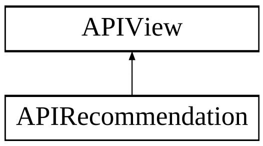
The documentation for this class was generated from the following file:

- travelmatch/appusers/**views.py**

B.20.2 APIRecommendation

This class queries existing VacationDetail to see its recommendation.

Inheritance diagram for APIRecommendation:



Detailed Description

This class queries existing VacationDetail to see its recommendation.

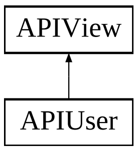
The documentation for this class was generated from the following file:

- travelmatch/appusers/**views.py**

B.20.3 APIUser

API for log in related issues, this class is used whenever a url(api_pattern.format('/user') is called.

Inheritance diagram for APIUser:



Detailed Description

API for log in related issues, this class is used whenever a url(api_pattern.format('/user') is called.

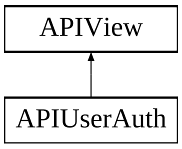
The documentation for this class was generated from the following file:

- travelmatch/appusers/**views.py**

B.20.4 APIUserAuth

API for MailUser authentication, when url(api_pattern.format('/user/auth'), APIUserAuth.as_view()) is called.

Inheritance diagram for APIUserAuth:



Detailed Description

API for MailUser authentication, when url(api_pattern.format('/user/auth'), APIUserAuth.as_view()) is called.

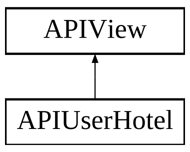
The documentation for this class was generated from the following file:

- travelmatch/appusers/**views.py**

B.20.5 APIUserHotel

This class gets the hotel user use for certain trip offer.

Inheritance diagram for APIUserHotel:



Detailed Description

This class gets the hotel user use for certain trip offer.

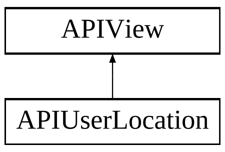
The documentation for this class was generated from the following file:

- travelmatch/appusers/**views.py**

B.20.6 APIUserLocation

This is for get and post SaveLocation from AppUser.

Inheritance diagram for APIUserLocation:



Detailed Description

This is for get and post SaveLocation from AppUser.

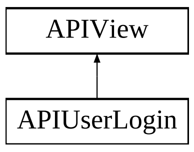
The documentation for this class was generated from the following file:

- travelmatch/appusers/**views.py**

B.20.7 APIUserLogin

API for log in, when url(api_pattern.format('/user/login'), APIUserLogin.as_view()), is called.

Inheritance diagram for APIUserLogin:



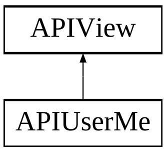
Detailed Description

API for log in, when url(api_pattern.format('/user/login'), APIUserLogin.as_view()), is called.
The documentation for this class was generated from the following file:

- travelmatch/appusers/**views.py**

B.20.8 APIUserMe

API for retrieve and modify ones profile.
Inheritance diagram for APIUserMe:



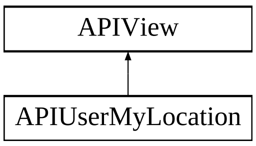
Detailed Description

API for retrieve and modify ones profile.
The documentation for this class was generated from the following file:

- travelmatch/appusers/**views.py**

B.20.9 APIUserMyLocation

This is the api for the location of an user.
Inheritance diagram for APIUserMyLocation:



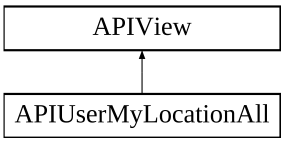
Detailed Description

This is the api for the location of an user.
The documentation for this class was generated from the following file:

- travelmatch/appusers/**views.py**

B.20.10 APIUserMyLocationAll

API for getting all of user's locations.
Inheritance diagram for APIUserMyLocationAll:



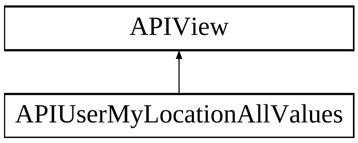
Detailed Description

API for getting all of user's locations.
The documentation for this class was generated from the following file:

- travelmatch/appusers/**views.py**

B.20.11 APIUserMyLocationAllValues

This is the API to get all the value from a user's locations.
Inheritance diagram for APIUserMyLocationAllValues:



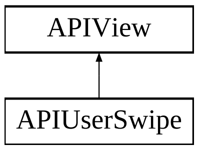
Detailed Description

This is the API to get all the value from a user's locations.
The documentation for this class was generated from the following file:

- travelmatch/appusers/**views.py**

B.20.12 APIUserSwipe

The API for the swipe operations carried out by users.
Inheritance diagram for APIUserSwipe:



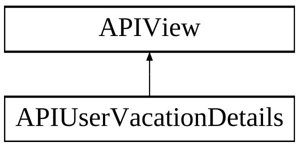
Detailed Description

The API for the swipe operations carried out by users.
The documentation for this class was generated from the following file:

- travelmatch/appusers/**views.py**

B.20.13 APIUserVacationDetails

API for user's vacation details.
Inheritance diagram for APIUserVacationDetails:



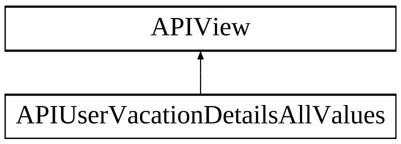
Detailed Description

API for user's vacation details.
The documentation for this class was generated from the following file:

- travelmatch/appusers/**views.py**

B.20.14 APIUserVacationDetailsAllValues

This is the API for query all the VacationDetail object from a certain user.
Inheritance diagram for APIUserVacationDetailsAllValues:



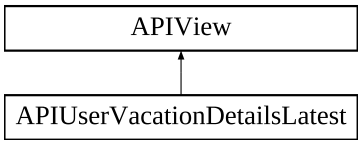
Detailed Description

This is the API for query all the VacationDetail object from a certain user.
The documentation for this class was generated from the following file:

- travelmatch/appusers/**views.py**

B.20.15 APIUserVacationDetailsLatest

This is the API for query all the VacationDetail object from a certain user.
Inheritance diagram for APIUserVacationDetailsLatest:



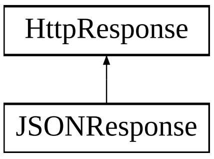
Detailed Description

This is the API for query all the VacationDetail object from a certain user.
The documentation for this class was generated from the following file:

- travelmatch/appusers/**views.py**

B.20.16 JSONResponse

An HttpResponse that renders its content into JSON.
Inheritance diagram for JSONResponse:



Detailed Description

An HttpResponse that renders its content into JSON.

The documentation for this class was generated from the following file:

- travelmatch/appusers/**views.py**