TECHNICAL DOCUMENTATION

# C:\Users\Corinna\Downloads\twitter.jpgArchitecture

## Overview

The application consists of three parts:

* The frontend is realised in PHP and HTML. It is responsible for handing user interactions.
* Database
* The backend is realised in Java and is responsible for fetching and evaluating tweets.

## Frontend

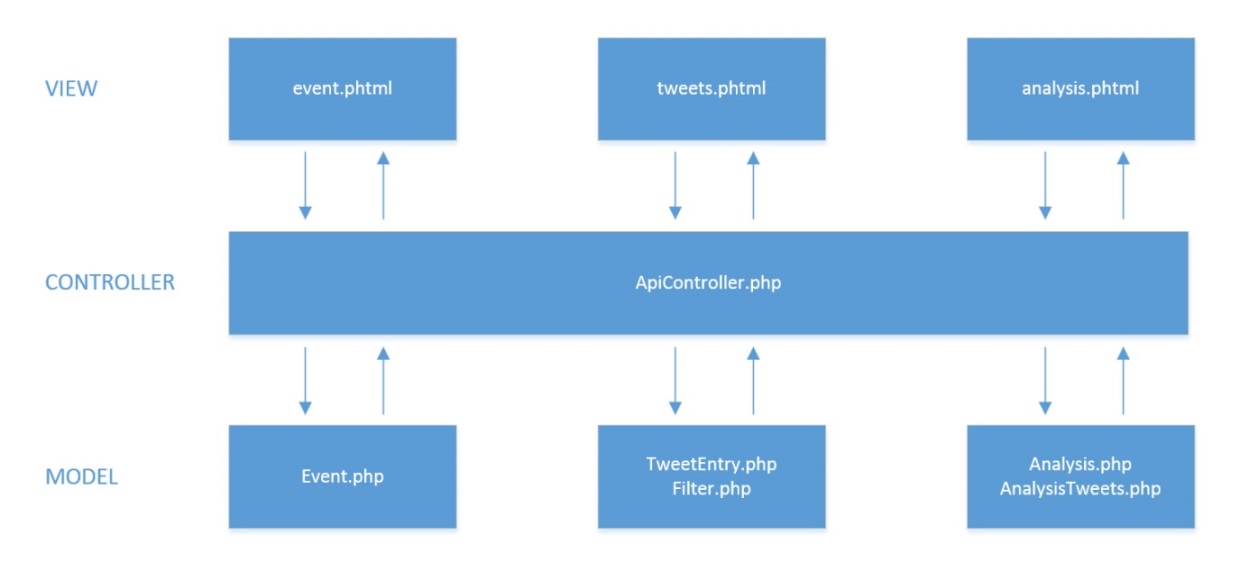
The structure of the frontend is a classic MVC architecture, realised with Zend framework (version 1).

The start of the application calls the index.php file which loads the Zend bootloader for the primary initialization of the project. All requests that are received via the web site are forwarded to the appropriate controller by the Zend framework. The controller is determined by the subfolder the application is currently in. All main menu items have a corresponding controller (e.g. map controller for [http://twitteranalyser.local/m](http://twitteranalyser.local/index)ap) The standard controller is the index controller (<http://twitteranalyser.local/index>).   
All menu subitems are actions in the corresponding controller. Every click on a subitem calls another action in the controller. The IndexAction is called whenever the controller is called.

The application uses Twitter-Bootstrap. This is significantly responsible for the design of the graphical user interface and the presentation (e.g. of the date selection)

##### **controllers**

The controller logic is split into five controllers:

* Index controller (main controller)
* Error controller
* API controller: This controller handles dialogs (e.g. creating of events) and multilingualism (translation of texts for the dialogs) via JSON strings
* Map controller
* Statistics controller

**views**

The phtml-files in „views“ are used for the displaying of the web page. They contain, besides the html code, also php code for the dynamic aspects of the page.  
All user interactions call an action in the controller that the view belongs to. For the views „analysis“, „event“ and „tweets“ it is the API controller.  
The view „event“ generates a list as an overview of all currently existing events. Possible user interaction are:

* Edit an existing event
* Create a new event
* Search for an event in the list
* Delete an event

The view “tweets“ generates a list of all collected tweets. Possible interactions are:

* Display tweets for a certain event
* Create or delete a filter
* Use a filter for a list of tweets of an event
* Start analysis
* Search for tweets containing certain words

The view “sentiment” shows a list of all entered words for the analysis. Word entries can be created, edited or deleted.

For the view of the map and the statistics, Google Maps and Google Visualisation are used for the presentation of the map and charts.

### apimanagement The dialogs are managed by the file „apimanagement.js“. All dialogs of the web application are generated by using EJS code that is loaded and displayed dynamically via JQuery (see „public“ folder; the EJS dialogs can be found in the „tpl“ folder). The api management sends JSON requests to the API controller. The controller sends responses or validation errors.

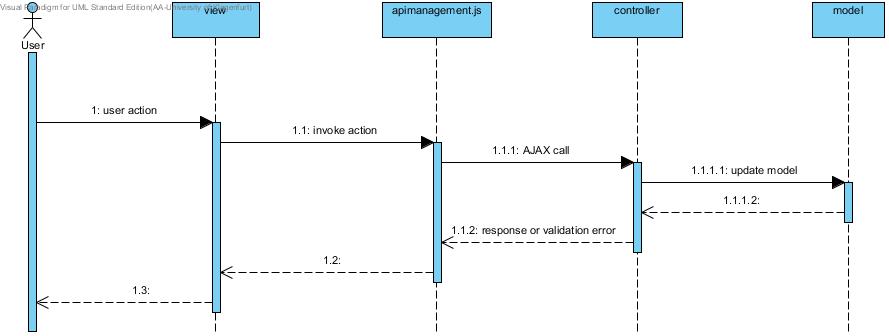
##### **languages**

The csv-files contain the texts that are displayed on the webpage.

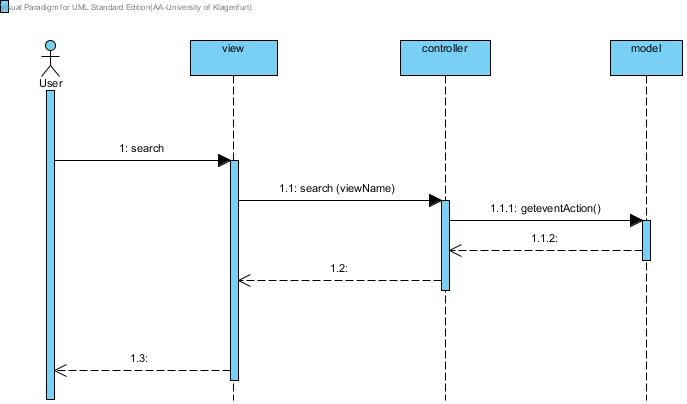
### models For all database tables that are relevant for the views exists a corresponding model.

##### **Control flow**

Most user actions cause changes on the database, which means creating, editing and deleting entries of a database table. These actions invoke the following control flow:

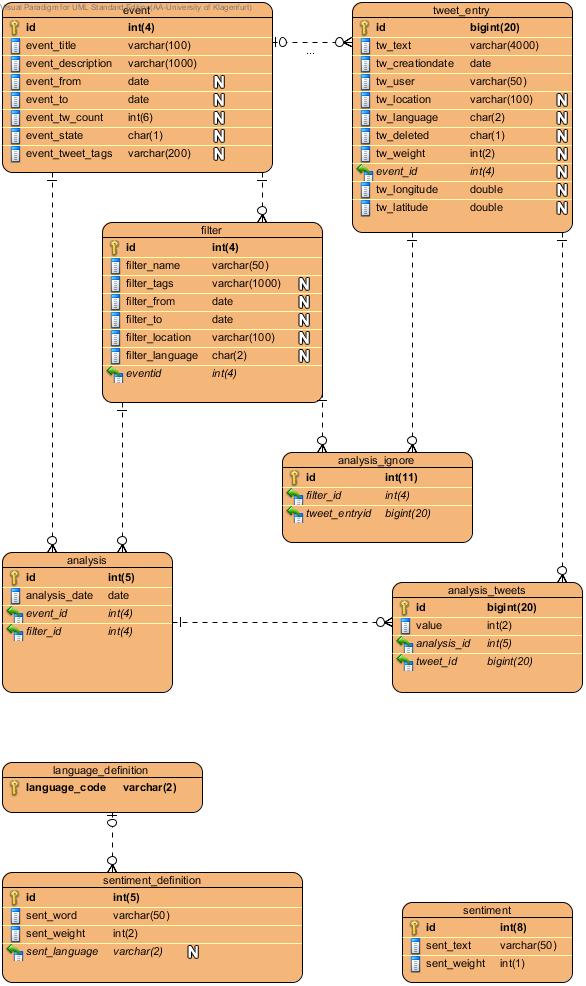


The action “search” invokes a different control flow:



## Database

The following Entity Relationship Diagram shows all tables of the database.



The table tweets\_entry contains all tweets that were collected for a certain event. These can be restricted by defining a filter. The table analysis\_tweets marks the subset of all collected tweets that are used in a certain analysis. It also contains the results of the sentiment analysis for each tweet.

The table sentiment contains keywords that determine if the text of a certain tweet has a positive or negative connotation.

##### **Table description**

The table event contains the information needed for each created event.

|  |  |  |
| --- | --- | --- |
| table event | details | description |
| id | AUTO\_INCREMENT | primary key |
| event\_title |  | title of the event |
| event\_description |  | short description of the event |
| event\_from | DEFAULT NULL | date to specify the beginning of the tweet collection |
| event\_to | DEFAULT NULL | date to specify the ending of the tweet collection |
| event\_tw\_count | DEFAULT NULL | specifies the number of tweets to collect |
| event\_state | DEFAULT '0' | 0: initial state  1: collection in progress  2: collection finished |
| event\_tweet\_tags | DEFAULT NULL | all tweet tags which should be considered, separated with a special delimiter |

The table tweet\_entry holds all collected tweets and their relevant information. Each tweet is assigned to an event because each collection of tweets is started for a certain event.

|  |  |  |
| --- | --- | --- |
| table tweet\_entry | details | description |
| id | AUTO\_INCREMENT | primary key |
| tw\_text |  | content/text of the tweet |
| tw\_creationdate |  | creation date of the tweet |
| tw\_user |  | user/author of the tweet |
| tw\_location | DEFAULT NULL | Location of the user/author |
| tw\_language | DEFAULT NULL | language of the tweet text |
| tw\_deleted | DEFAULT '0' | marks the tweet as deleted |
| tw\_weigth | DEFAULT NULL | result of the analysis |
| event\_id |  | foreign key, each tweet is assigned to a certain event |
| tw\_longitude | DEFAULT NULL |  |
| tw\_latitude | DEFAULT NULL |  |

The table filter contains attributes of tweets to restrict the tweets of a certain events that should be used in an analysis.

|  |  |  |
| --- | --- | --- |
| table filter | details | description |
| id | AUTO\_INCREMENT | primary key |
| filter\_name |  | shown name of the filter |
| filter\_tags | DEFAULT NULL | tags that should be contained in the tweet text |
| filter\_from | DEFAULT NULL | only tweets that where created after that date will be considered |
| filter\_to | DEFAULT NULL | only tweets that where created before that date will be considered |
| filter\_location | DEFAULT NULL | only tweets with this location will be considered |
| filter\_language | DEFAULT NULL | only tweets with this location will be considered |
| event\_id |  | filters are created for a certain event |

The table analysis contains informations about all performed analysis.

|  |  |  |
| --- | --- | --- |
| table analysis | details | description |
| id | AUTO\_INCREMENT | primary key |
| analysis\_date |  | creation date of the analysis |
| event\_id |  | foreign key, id of the event the analysis belongs to |
| filter\_id |  | foreign key, id of the used filter |

The entries in the table analysis\_tweets marks which of the collected tweets should be considered in the analysis. It also contains the results of the sentiment analysis.

|  |  |  |
| --- | --- | --- |
| table analysis\_tweets | details | description |
| id | AUTO\_INCREMENT | primary key |
| tweet\_id |  | foreign key, id of the considered tweet |
| analysis\_id |  | foreign key, id of the analysis the tweet belongs to |
| value |  | Degree of sentiment assigned to the tweets |

The table sentiment contains keywords that are used for the sentiment analysis. Each keyword is weighted concerning their influence on the grade of sentiment of a tweet.

|  |  |  |
| --- | --- | --- |
| table sentiment | details | description |
| id | AUTO\_INCREMENT | primary key |
| sent\_text |  | keyword for sentiment analysis |
| sent\_weight |  | weight of the keyword  range: [-5;5]  (-5 … negative, 5 … positive) |

The table language\_definition contains all languages that can be chosen when creating a new word for the analysis.

|  |  |  |
| --- | --- | --- |
| table language\_definition | details | description |
| language\_code |  | iso code of the language |

The table sentiment contains all words for the analysis that were created by the user.

|  |  |  |
| --- | --- | --- |
| table sentiment\_definition | details | description |
| id | AUTO\_INCREMENT | primary key |
| sent\_language |  | foreign key, language code |
| sent\_word |  | keyword for sentiment analysis |
| sent\_weight |  | weight of the keyword  range: [-5;5]  (-5 … negative, 5 … positive) |

The table analysis ignore determines which tweets are not used in the analysis for a certain filter.

|  |  |  |
| --- | --- | --- |
| table analysis\_ignore | details | description |
| id | AUTO\_INCREMENT | primary key |
| filter\_id |  | foreign key |
| tweet\_id |  | foreign key |

## Backend

##### **tweetloader**

The tweetloader uses Twitter4J (Java library for the Twitter API) to establish a connection to the public streaming endpoint of twitter.  
It forms a query for all events in the database, considering tags and specified dates of the events and collects the relevant tweets. The results of the query are fetched via the twitter stream. The relevant data fields of the resulting tweets are inserted into the database via the tweetdatabaseconnector.

The evaluation is done by reading the sentiment words and the tweets entries from the database. The texts of the tweets are compared with the sentiments word. Each tweet entry in the database gets the mean value of all sentiment words contained in the text of the tweets representing the weight of the tweet.

##### **tweetdatabaseconnector**

The connector handles requests from the tweetloader:

* Requests reading events, tweets and sentiment words
* Updates: event state and tweet weight
* Inserts of tweets