help2day.org System Documentation

Webapp

The webapp is setup in the style of a node.js application.

Application Logic

The application is implemented in files located in the `app` folder. This folder contains the following javascript files.

1. communication.js contains communication functions for mail transmission as well as a daemon function for transmission of notifications resulting from adding new help requests.
2. helpers.js contains general supportive functions e.g. for encryption and decryption
3. image_handler.js contains functions for image resizing and storage
4. impactgraph.js contains functions for the SIG API
5. impactgraphperiodic.js contains the kiosk animation
6. login.js contains the login code as well as registration functions
7. periodic.js contains some helper functions plus the integration of the communication periodic part
8. webapp.js contains the core application logic and API of the webapp

Configuration

The main configuration code is located in the `config` folder. This folder contains the following javascript files.

1. auth.js contains a helper for the facebook authentication (currently not active)
2. configuration.js contains the most of the configuration constants.
3. database.js contains the setup of the connection to the database.
4. model.js contains some old artifacts of the bookshelf module (no longer actively used)
5. passport.js contains the integration code for passport for local login, facebook login and anonymous login

Static Web Files

The static web files are located in the `public` folder. This folder contains the following files.

1. Several javascript and css files for semantic-ui, dropzone, jquery ...
2. Font files
3. images
4. CSS files
5. Logo files

Email Template files

The application uses eMail templates from the node.js module **email-templates**. The template files are located in the folder `templates`.

HTML Template files

The application uses HTML templates from the node.js module **handlebars**. The template files are located in the folder `views`.

Integration Code

The main file is `server.js` in the root folder. This file contains the setup and integration of the application.

API

The API of the Social Impact Graph allows the feeding of external data sources into a session of the graph. There are plans for many different actions to be supported by the API. At this point in time only the creation of events is documented.

Uploading an image

This call uploads an image file to the server and returns an image reference usable for a create event call.

- **URL**

  /sig/event/image

- **Method:**

  `POST`

- **URL Params**

  This call has URL params for authentication.

  Required:

  `ks_key=[string]` the session key for the social impact graph

  `ks_api_key=[string]` the api key for the social impact graph

- **Data Params**

  The server expects the image file to be transmitted as a multipart payload. The content shall have the name `file`.

- **Success Response:**

  On successfull upload the server responds with a general status field and with

  - **Code:** 200
    Content: `{ status : 'ok', se_image_ref : '<image ref string>' }`

- **Error Response:**

  - **Code:** 401 UNAUTHORIZED
    Content: `{ status : "nok" }`

- **Sample Call:**

```
var imageFormData = {
    file: fs.createReadStream(image),
    ks_api_key: apiKey,
    ks_key: key
};

request.post( { url:'http://localhost:8082/sig/event/image', formData: imageFormData}, function optionalCallback ( er
```

```
    if (err) {
      return console.error('upload failed:', err);
    }

    console.log('server responded with:', body );
}
```

- **Notes:**

  The API key has to be acquired from the operator of the platform. In a future version this key can be obtained via a login API.

Create an Event

- **URL**

  /sig/event

- **Method:**

  `POST`

- **URL Params**

  This call has URL params for authentication.

  Required:

  `ks_key=[string]` the session key for the social impact graph

  `ks_api_key=[string]` the api key for the social impact graph

  `se_sender_id=[integer]` the id of the sender (currently not used, can be 0 but has to be set)

  `se_title=[string]` the title of the event

  `se_message=[string]` the message of the event (can include HTML)

  `se_image_ref=[string]` the image reference string returned from the image upload call

  `se_long=[string]` the longitude of the position where the event took place in decimal

  `se_lat=[string]` the latitude of the position where the event took place in decimal

- **Data Params**

  The URL parameters can also be transmitted as data parameters.

- **Success Response:**

  On successfull upload the server responds with a general status field and with

  - **Code:** 200
    Content: `{ status : 'ok', se_id : <id of event> }`

- **Error Response:**

  - **Code:** 401 UNAUTHORIZED
    Content: `{ status : "nok" }`

- **Sample Call:**

```
var formData = {
    se_sender_id:0,
    se_title:"New event for the social impact graph",
    se_message: "This is a new event to be displayed on the social impact graph in session yxz",
    se_long: 16.0,
    se_lat: 48.0,
    ks_api_key: "...........",
    ks_key: "xyz"
```

```
};

formData.se_image_ref= bodyJson.se_image_ref;

console.log('sending form data:', formData );

request.post( { url:'http://localhost:8082/sig/event', formData: formData}, function optionalCallback ( err, http

    if (err) {
        return console.error('upload failed:', err);
    }

    console.log('Server responded with:', body);

});
```

- **Notes:**

  The API key has to be acquired from the operator of the platform. In a future version this key can be obtained via a login API.

CRM System

This needs to be done. However, the structure is pretty much the same as for the webapp.