jamk.fi

Twitter manager

GO API intended for communication with Gnu Social API

Jaroslav Janiga

Bachelor’s thesis
May 2015

Software Engineering
School of Technology, Communication and Transport

JYVÄSKYLÄN AMMATTIKORKEAKOULU
JAMK UNIVERSITY OF APPLIED SCIENCES

JYVÄSKYLÄN AMMATTIKORKEAKOULU
JAMK UNIVERSITY OF APPLIED SCIENCES

Description

Author(s) Janiga, Jaroslav	Type of publication Bachelor's thesis	Date 01.05.2015
		Language of publication: English
	Number of pages	Permission for web publication: x

Title of publication
Twitter manager
GO API intended for communication with GNU social API

Software Engineering

Tutor(s)
Salmikangas Esa

Assigned by
Silokunnas Marko

Abstract

The bachelor's thesis had two goals. The first one was to create API for GO programming language that communicates with GNU Social API. The second one was to create an application that uses the mentioned API. The main purpose of creating the application was to reduce the time spent with the login and logout of the accounts by the members in the white team.

The application is intended for training purposes in JYVSECTEC project at JAMK University of Applied Sciences. JYVSECTEC stands for Jyväskylä Security Technology. The project focuses on development and maintenance of cyber security infrastructure (RGCE Realistic Global Cyber Environment). It enables research, development and training exercises for their co-operation network. The application is to be used during the cyber security exercises by the white team that will be responsible for making the exercise more interesting.

The application runs in any web browser. Thus, the front-end of the application was created by using technologies as HTML, CSS and some JavaScript. For better maintenance of the code some JavaScript framework was used. The framework is called React and was developed by Facebook.

GO API is intended for communication with GNU social, one of the social networks, and can be used for communication with Twitter, which is possible because GNU Social API is based on the Twitter API although there are some very simple differences.

Keywords/tags
GO, GNU Social, Twitter, framework

Miscellaneous

Contents

Figures

Tables

Abbreviations

JYVSECTEC	Jyväskylä Security Technology
HTML	HyperText Markup Language
CSS	Cascading Style Sheets
REST	Representational State Transfer
API	Application Programming Interface
GNU	GNU’s Not Unix!
FR	functional requirement
NFR	nonfunctional requirement

1 Introduction

Nowadays, communication software is used every day and in every part of the world. No matter whether it is used only in small networks such as intranet or worldwide. Everybody wants to have a connection with their family, friends and colleagues. Therefore, people very often use some social networks (Facebook, Twitter) and various applications (Skype, TeamViewer, Messenger) that provide various forms of communication (voice call, video call, chat).

One of the purposes for using communication software is the need to talk about some work related matters, which is the reason for writing this bachelor thesis. The idea of creating new communication software arose in JYVSECTEC project.

JYVSECTEC is dedicated to development and maintenance of a cyber-security infrastructure RGCE (Realistic Global Cyber Environment). It enables research, development and training for their co-operate network.

JYVSECTEC-project is implemented by the Institute of Information Technology at JAMK University of Applied Sciences in Jyväskylä. It started in September 2011. The goal of the project is to improve the awareness of the meaning of security, risk management and maintenance of security.

In JYVSECTEC project some interesting cyber security exercises are organized for students at JAMK University of Applied Sciences and companies as well. The members participating in the exercises are divided into a number of teams and have to fulfill some specified tasks. There is a team called the “white team” that has to manage many accounts in order to liven up the scenario of the exercises. Currently the members of the white team have to log in and out of the accounts on the website, which takes their precious time. Therefore, there is a requirement for creation of an application that provides a communication interface for more than one signed user.

Thus, the most important points in this thesis, are:

- Creating GO packages that establish communication between GNU Social server and application interface running in a web browser.
- Creating a user interface that allows to users basic functions as sign in, sign out, tweet and remove tweet.
- The application is packed into *.rpm* package and it is possible to install it under Linux.

2 Analysis

2.1 Requirements

Twitter manager is an application that will help to manage more accounts created in GNU Social/StatusNet simultaneously. It will be used during cyber security exercises in JYVSECTEC. It is intended for the “white team” that is responsible for livening up the scenario of the exercises and needs to discuss tasks at hand.

The list of basic requirements for back-end is as follows:

Table 1. The back-end requirements

Requirement ID	Requirement description
NFR 1	The program has to be packed as a DEB package and also as RPM package as well (preferably using FPM).
NFR 2	The application has to be coded in GO language
FR 1	It has to be possible to add and remove users in the application (in this case add and remove a user is taken as login and logout).
FR 2	It has to be possible to add and remove tweets from the timeline of the user
FR 3	The application has to contain API for authorization by the OAuth 1.0 Authorization protocol that is used in GNU Social/Statusnet.
FR 4	Communication with GNU Social will use the REST API

The list of basic requirements for front-end is presented below:

Table 2. The front-end requirements.

Requirement ID	Requirement description
1.	User interface will be displayed in a web browser.
2.	The twitter manager doesn’t have to be a single page application.
3.	The design should use CSS framework Bootstrap that is also used in other JYVSECTEC projects.
4.	JavaScript should be used for front-end logic.
5.	The act of tweeting on behalf of some user should be intuitive.
6.	A list of users should be displayed, in which all logged users will be shown.
7.	The UI should display all discussions.
8.	The user should be able to see whether the tweet was sent or not.

2.2 Designed solution

As mentioned above, back-end of the application was coded in GO. It was divided into four packages. Every package had its own functionality. The Application was divided into smaller and better understandable components. The most problematic part was to create package for authorization via OAuth 1.0, and secondly the part related to GNU Social REST API.

Packages in GO (back-end of the application):

1. Oauthlogin – It contains all necessary features related to OAuth 1.0 authorization protocol. This is package that includes all the features to establish a communication between an application, written in GO, and any server that allows to use OAuth 1.0 as authorization protocol.
2. Gnusocial – That is a package that contains GO API for work with GNU Social REST API. All functions related to user logging and tweeting are placed there.
3. Decoder – Small package that includes some additional functions that are intended for parsing data from responses or requests and other stuff.
4. Twittermgr – It contains the main part of the application. All business logic is placed in this package. The twitter manager that will be running between GNU Social server and web browser is placed there.

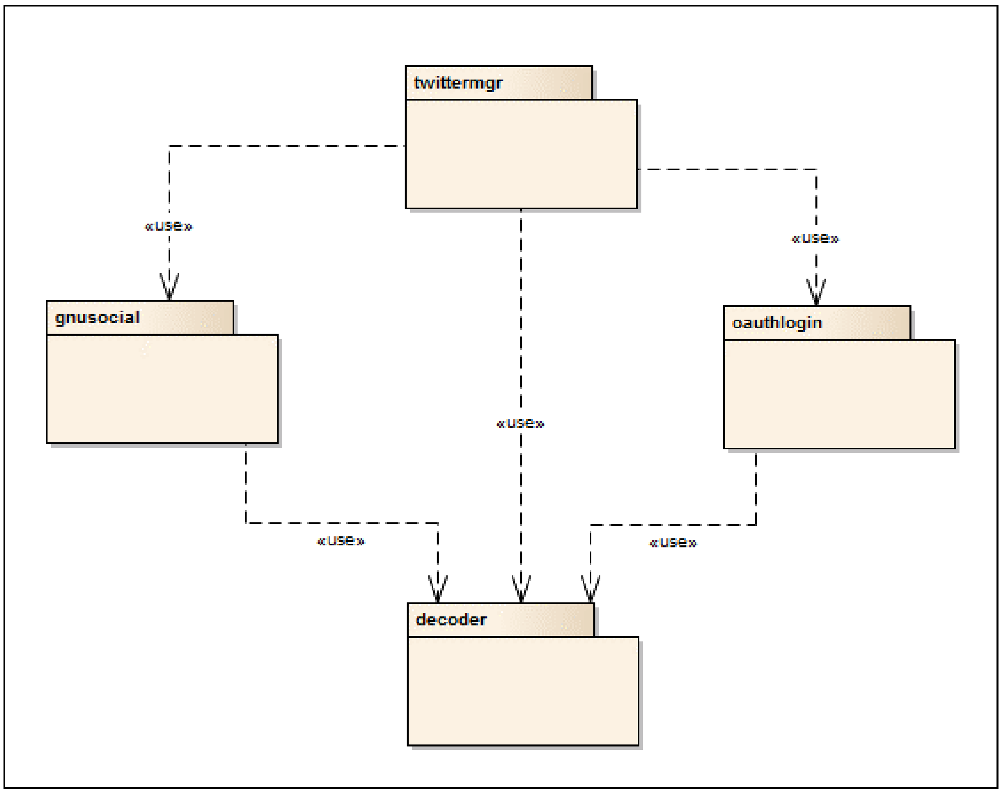


Figure 1 Package diagram of back-end of the application.

The front-end of the Twitter manager is composed of more technologies. The UI of the application has to run in a web browser. Therefore, HTML (HyperText Markup Language) will be used that is intended for creating websites. For design CSS (Cascading Style Sheet language) is usually used. As mentioned in requirements, Bootstrap CSS framework was used for the design. JavaScript was used for basic logic that is executed in web browser. In order to improve the maintaining of code and performance of the application, one of JavaScript frameworks called React was used. The framework was developed by Facebook. Getting data was to be realized by Ajax that allows to work with the data asynchronously. Basic interactions between a user and Twitter manager are shown in Figure 2.

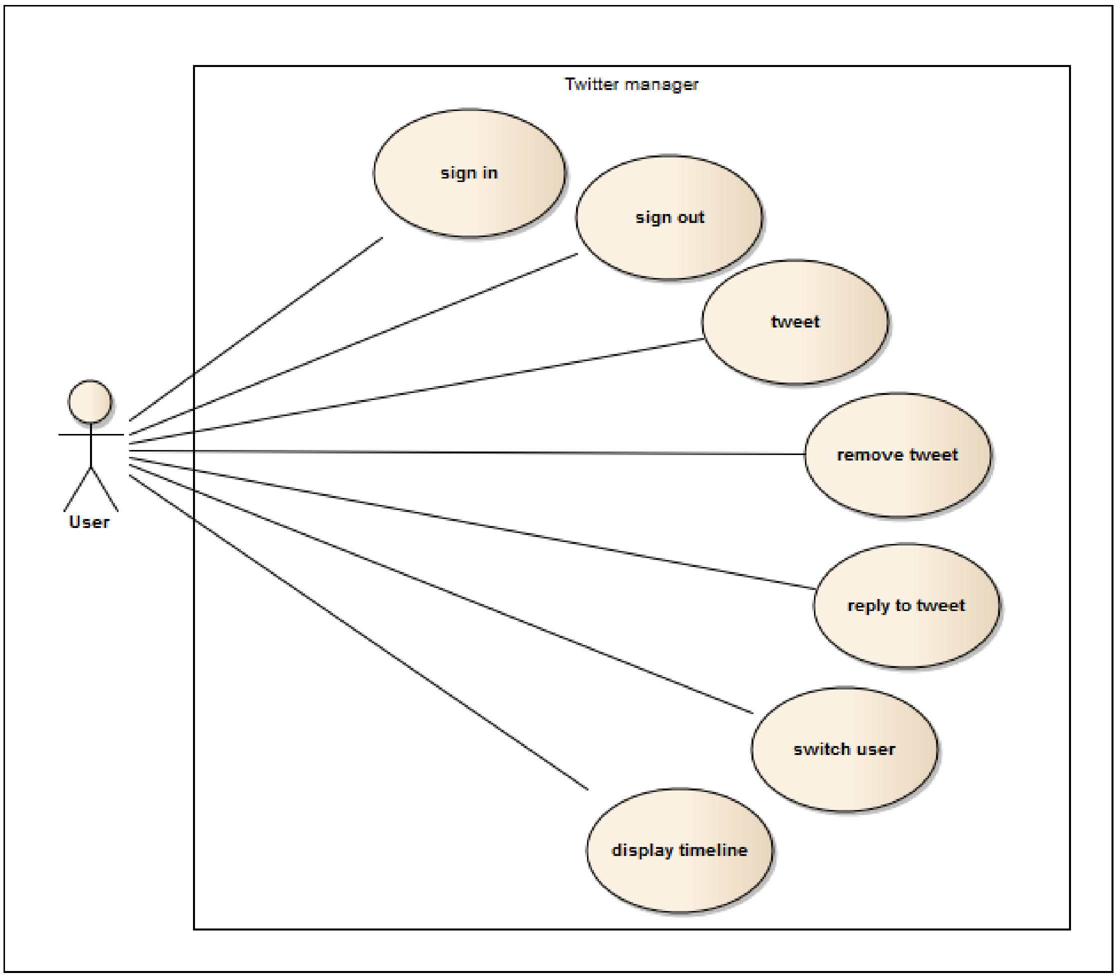


Figure 2 Use Case Diagram of Twitter manager.

3 Used technologies

This part includes all technologies related to topic of this bachelor’s thesis. A short description of used technologies during creation of the bachelor’s thesis is devote to this chapter. Every technology mentioned bellow contributed to the fulfillment of the goals of the thesis.

3.1 Back-end

3.1.1 Client-server architecture

This architecture consists of two parts. The one part includes the client’s systems and the other part includes the server’s systems, and they communicate together over the computer network. When the client-server model is used as a basic architecture of any software system, the software has to be divided into the abovementioned parts. Both parts of the software can be running in the same computer. By using this architecture, many clients can be connected to the server at the same time. (Client–server model, 2015)

When client is connected to the server, the client’s task is just to require the services of the server. For example, if there is any web application, then a client can be considered to be some web browser. (Client–server model, 2015)

The server is the part that waits for requests of the clients and after arrival of any request, it has to send a response to the client who sent the request. Of course, the type of the data in request depends on the type of the server. For example, if the server is a web server then the client can ask for some websites. If the server is a file server and is used as a remote storage (e.g. it can be a cloud), then the client requires any files from the server. Its task is waiting and responding to requests and to be responsible for sharing resources. (Client–server model, 2015)

Client and server send messages that are called requests (from client) and responses (from server). If they want to communicate, computers have to have a common language. It ensures that both of them know what to expect from the other side. The language and rules needed for communication are defined in communication protocols that operate in the application layer. Of course the server can implement some API for a specific content format. The API is an abstract layer for resources such as databases or custom application. (Client–server model, 2015)

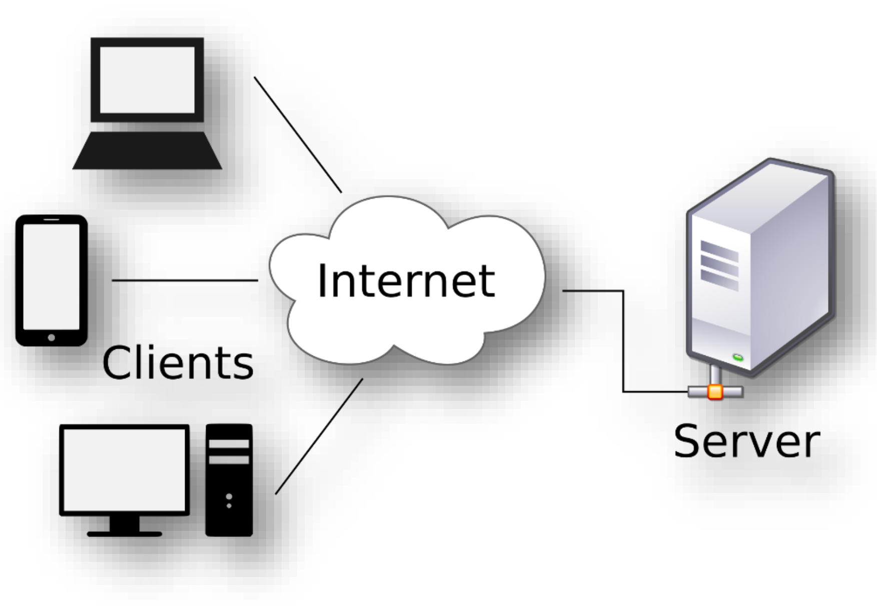


Figure 3 Client-server architecture

3.1.2 OAuth

It is an open standard used for secured communication. It provides client applications a secure delegated access to resources. Thanks the OAuth, the client application can get access to resources on behalf of the resource owner. OAuth is a supplement to client-server architecture, and it adds third role to the model. The role is called resource owner. In OAuth model the client is not resource owner, but only third party application that acts on behalf of resource owner. OAuth allows to client to require resources of a user and in addition, it allows to server to verify not only the resource owner but also the client that require the access to the resources. OAuth 1.0 authorization protocol is shown in the Figure 4. (E. Hammer-Lahav, 2015)

History

OAuth was first mentioned in November 2006 when Blaine Cook was developed Twitter OpenID and Ma.gnolia needed OpenID for Authorization Dashboards Widgets. Cook, Chris Messina, Larry Half and David Recordon met together to discuss the usage of OpenID in Twitter and Ma.gnolia API. They concluded that there is no open standard for API access. In April 2007 OAuth discussion group was created and DeWitt Clinton was interested in the project and expressed his support in the effort to create OAuth open standard. On 4th December 2007 final proposal for OAuth core version 1.0 was released. In April 2010 the OAuth authorization protocol 1.0 as RFC 5849 was published. Nowadays, there are the next versions of OAuth as OAuth 1.0a or the latest version 2.0. (Oauth, 2015)

OAuth authorization process consists of several steps as follows:

1. The client application sends the signed request for temporary credentials. If the server supports OAuth 1.0, then temporary *token* and *token secret* response arrive. (E. Hammer-Lahav, 2015)
2. The client has to send a signed authorization request to server that will send response, in which it will require resource owner authentication. (E. Hammer-Lahav, 2015)
3. When the resource owner sends its credentials and the server verifies them then the server sends *verifier* back to client. (E. Hammer-Lahav, 2015)
4. The client sends a signed request that has to include the *verifier* as well, and then the client obtains *token* and *token secret* that have to be used whenever the client requires some resources from the server. (E. Hammer-Lahav, 2015)

For better understanding how this works, there is short example below.

In order to obtain the access to the user resources, the application has to be registered on the server, which stores the resources. After registration of the client application on the server, *consumer key* and *consumer secret* that are necessary for signing of requests are generated. There are three important links for communication:

- Temporary Credential Request URI: https:// test.jamk.fi /initiate
- Resource Owner Authorization URI: https:// test.jamk.fi /authorize
- Token Request URI: https:// test.jamk.fi /token

The first step is to send a request in the following form:

```
POST /initiate HTTP/1.1
    Host: test.jamk.fi
    Authorization: OAuth realm="Album",
       oauth_consumer_key="abcd152abcd152",
       oauth_signature_method="HMAC-SHA1",
       oauth_timestamp="137151200",
       oauth_nonce="lkjfk",
       oauth_callback="http%3A%2F%2F test.jamk.fi %2Fready",
       oauth_signature="encodedSignature"
```

A short explanation of parameters is included in the Authorization header:

1. *oauth_consumer_key* – It is a consumer key generated by server for registered app. (E. Hammer-Lahav, 2015)
2. *oauth_signature_method* – Method that is used for creation of signature. (E. Hammer-Lahav, 2015)
3. *oauth_time_stamp* – Every sent request has to contain its own time stamp. It is used for security as a way of preventing compromised requests to be sent again. (E. Hammer-Lahav, 2015)
4. *Oauth_nonce* – This parameter of authorization header is used for the same purpose as time stamp. (E. Hammer-Lahav, 2015)
5. *oauth_callback* – A link to client application intended for redirecting after authorization. (E. Hammer-Lahav, 2015)
6. *oauth_signature* – A generated signature that is always generated for every request. (E. Hammer-Lahav, 2015)

After sending this request, the server verifies the request, and if it is alright, then the server sends back the *oauth_token* and *oauth_token_secret*, which are placed into the response body. The form of the response body is following:

```
oauth_token=tempTok1&oauth_token_secret=tempTok1Secret&oauth_callback_confi
rmed=true
```

After obtaining temporary credentials, the request intended for authorization can be sent. This request is much simpler than previous request because this request does not need to include authorization header. The request is shown below. (E. Hammer-Lahav, 2015)

```
https:// test.jamk.fi /authorize?oauth_token=tempTok1
```

When the authorization is successfully done, it requires user to grant access to the resource that is, in this case, test.jamk.fi. Then the user-agent is redirected to the callback link that was specified in the authorization header of the first request. The response has the following form:

```
http://test.jamk.fi/ready?oauth_token=tempTok1&oauth_verifier=tempTok1Verifier
```

When the client obtains verifier, then it can require a set of token credentials. This is again somewhat more complicated request because it has to contain the authorization header.

```
POST /token HTTP/1.1
    Host: test.jamk.fi
    Authorization: OAuth realm="Album",
      oauth_consumer_key=" abcd152abcd152",
      oauth_token="tempTok1",
      oauth_signature_method="HMAC-SHA1",
      oauth_timestamp="137151201",
      oauth_nonce="asjdkj",
      oauth_verifier="tempTok1Verifier",
      oauth_signature="encodedSignature"
```

After successful verification of the request above, the server sends a set of token credentials that have to be used for getting resources from test.jamk.fi. An example of the request, which asks for a resources from the test.jamk.fi, is shown below.

```
GET /album?file=vacation.jpg&size=original HTTP/1.1
    Host: test.jamk.fi
    Authorization: OAuth realm="Album",
       oauth_consumer_key=" abcd152abcd152",
       oauth_token="tok1",
       oauth_signature_method="HMAC-SHA1",
       oauth_timestamp="137151202",
       oauth_nonce="ufjlkg",
       oauth_signature="encodedSignature"
```

A very important part of the request, which asks for resources, is signature. Therefore, the last part of this section will be de devoted to the creation of the signature.

Signature creation:

This is probably the most difficult part of the OAuth protocol. The signature is created by encoding a base signature string that consists of all sent parameters of a request, request URL without the part that includes parameters and sending method (GET, POST…). There are two steps that have to be fulfilled during the creation of signature. The first step is creating a base signature string and the second step is encoding of the string by a method that is filled in the authorization header. Next short explanation, how to create the base signature string. To join all parameters with their values into one string. For explanation a request is used as illustrated bellow. (E. Hammer-Lahav, 2015)

```
GET /album?file=test.jpg&size=original HTTP/1.1
    Host: test.jamk.fi
    Authorization: OAuth realm="",
       oauth_consumer_key="abcd152abcd153",
       oauth_token="tok1",
       oauth_signature_method="HMAC-SHA1",
       oauth_timestamp="137151202",
       oauth_nonce=" ufjlkg"
```

Before putting them together into string they have to fulfill following rules:

- Every name of a parameter and its value have to be percent encoded. (E. Hammer-Lahav, 2015)
- All parameters are alphabetically sorted by name. If a situation occurs, in which there are more parameters with the same name, then they are ordered by value. (E. Hammer-Lahav, 2015)
- After that, the parameter's names and their values are joined by "=". (E. Hammer-Lahav, 2015)
- When every parameter has assigned value, all these small strings are joined by "&". (E. Hammer-Lahav, 2015)
- After that, the whole string has to be percent encoded again. (E. Hammer-Lahav, 2015)

When all rules mentioned above are fulfilled, then a string like this as follows is illustrated.

```
file%3Dtest.jpeg%26oauth_consumer_key%3Dabcd152abcd153oauth_nonce%3D
ufjlkg%26oauth_signature_method%3DHMACSHA1%26oauth_timestamp%3D13715120
2%26 oauth_token%3Dtok1%3Doriginal
```

When all parameters are joined, it is time to merge the string of parameters with request method and with a part of the URL that does not contain parameters. Of course, the URL is encoded by percent-encoding as well. Then all base signature string looks as the following string below: (E. Hammer-Lahav, 2015)

```
GET&http%3A%2F%2Ftest.jamk.fi/album&file%3Dtest.jpeg%26oauth_consumer_key%3
Dabcd152abcd153oauth_nonce%3Dufjlkg%26oauth_signature_method%3DHMACSHA1
%26oauth_timestamp%3D137151202%26oauth_token%3Dtok1%3Doriginal
```

Now it has to be hashed by the method that is mentioned in authorization request and the signature is done.

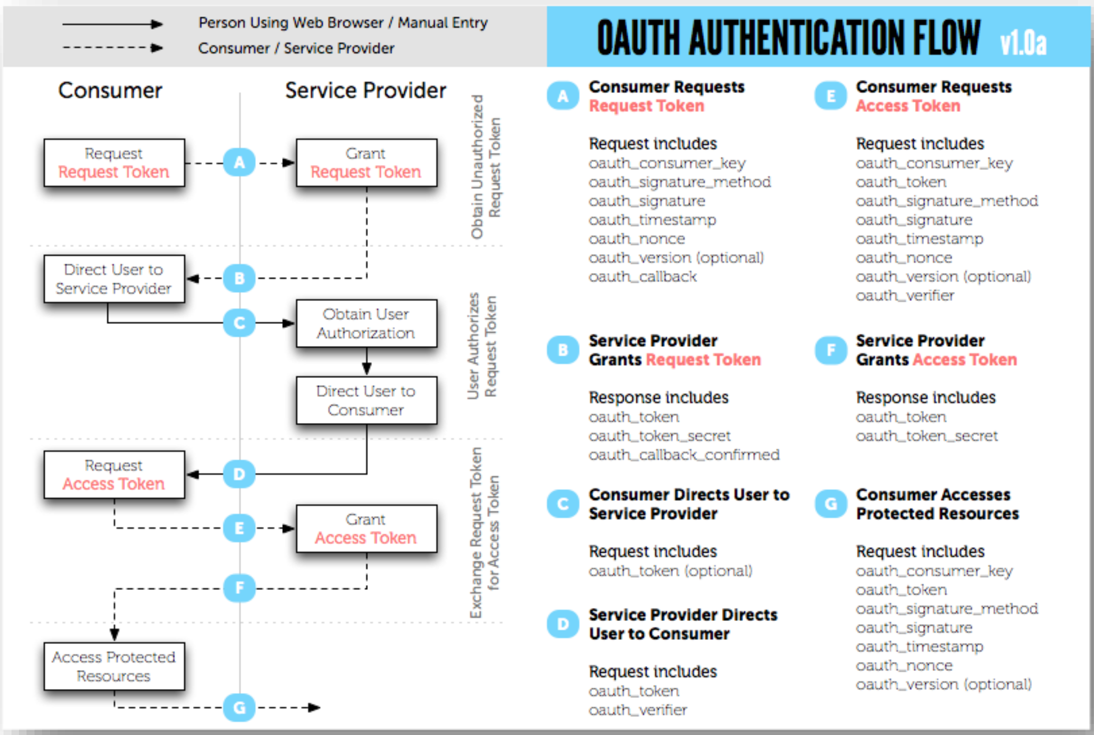


Figure 4 OAuth Authorization protocol process.

GNU Social is a newly created social network, the base of which is created according to UNIX philosophy that states “the small programs do small jobs”. Therefore, it is possible to install GNU Social on some small server. Of course, it has to be mentioned, the GNU Social belongs to free open source projects. (Matt Lee, 2015)

The whole core of GNU Social is coded in PHP. If a user likes to install GNU social on their own server, there are few requirements. For running of the server Apache or another HTTP server are necessary to install. PHP 5 and some components related to PHP 5 have to be installed. Of course, GNU Social is a social network and it needs some database server to store the data. There are two possibilities. The first one is

MySQL 5 or the second one Maria DB. Which one will be used, is up to the user who decides to install GNU Social on their own server.

When all requirements are accomplished, it is necessary to set up some basic properties in configuration file and the server is ready to use.

3.1.3 REST API

REST abbreviation stands for Representation State Transfer. It is one of the software architecture that consists of many guidelines and best practices. This architecture is very useful for creating scalable web services. The most used uniform interface in REST is HTTP. The data in REST are typically transferred as JSON or XML. There are some constraints that apply to the components of REST. (Representational state transfer, 2015)

REST is based on client – server architecture (Representational state transfer, 2015)

Stateless – It means that server does not contain any state of the client. Any session state, which could be available, is held on the client. (Representational state transfer, 2015)

Cacheable – Client can cache any response from server. The responses can be cacheable explicitly, implicitly or negotiated. In the future, the cache ability can improve performance and scalability. (Representational state transfer, 2015)

Layered system – In this case, it means that a client cannot exactly say, whether it is directly connected to a server or not. It improves scalability of the system. (Representational state transfer, 2015)

Code on demand – This one is an optional constraint. It means that the server can transfer some logic to client, on which the logic is to be executed. For example, it could be any JavaScript or Java applet. (Representational state transfer, 2015)

3.1.4 GO language

This project is developed by Google. The first version of GO was released in March 2012. It does not belong to object-oriented languages although it allows programmers to create small structures that have a type *struct* and also to define some methods for them however, it is not enough the GO was included in a group of object-oriented languages. Go belongs to languages that include a garbage-collector. (Google, 2015)

Programing in Go can be slightly difficult because there are only some plugins for developing so far. There is no IDE that includes direct support for GO programming language yet. Nevertheless, some of the plugins are quite good for work with GO. Under Linux, there is a possibility to install the GO plugin for VIM editor. Syntax highlighting works as well as autocomplete. Of course, it can be a bit complicated to set it for users who are not very familiar with Linux. There are also other alternatives. A very useful plugin is developed for IntelliJ IDEA. The plugin provides support for syntax highlighting, autocomplete, and for debugging too. There are also some plugins for other tools, for example Notepad++, Eclipse, Atom or LiteIDE. Maybe in the future there will be direct support for GO. (Google, 2015)

GO has some interesting features:

The formal grammar in Go uses semicolons; however, they can be omitted. It is necessary to follow two rules to omit the semicolons.

a) Semicolon is automatically added at the end of non-blank line, if the line’s final token is an identifier, integer, floating-point, imaginary, rune, string or one of the few keywords used in GO such as break, continue, fallthrough or return, or one of the operators ++, --,),], or }. (Google, 2015)
b) If a complex statement occupies one line then a semicolon may be omitted before closing of the statement by “)” or “}”. (Google, 2015)

GO as well as C or C++ use pointers. It is necessary to be very careful here because GO allows to define functions for structures and also for pointers to the structures.

There is one important issue, which everyone should remember. If a function of a structure is called as a function of an instance of the structure, then all changes, which will be realized within the instance, will be realized within a copy of the instance. It means, if some attribute of the instance was changed during processing the function, after processing, the instance will be in the same state as before the execution of the function. However, if the function is called as a function of a pointer to the instance, after execution of the function, all changes which were made within the instance will be done. (Google, 2015)

One very interesting matter however, also very dangerous is shadowing. It occurs when any variable declared in certain scope has the same name as another variable that is declared in outer scope. In GO, this effect can occur within some function that includes nested scopes.

Go belongs to programing languages that allow functions to have more than one return value. It means, if somebody needs, for some reason, the function to return two or more values, then the function just needs to have defined types of return values that are separated by comma. (Google, 2015)

3.1.5 RPM and DEB packages

RPM, its original name Red hat package manager and DEB are installation packages for Linux.

DEB packages are primary intended for Debian and its distributions. It is binary package that usually contains configuration files, executable files, some man documentation pages and other documentation. Primarily, these packages are unpacked by dpkg. (Fernandez-Sanguino, 2015)

RPM packages are primarily used by Red Hat and Fedora. They can contain an arbitrary set of files. Usually, they contain mainly binary files. Then they are called binary RPMs. They may also contain some source files. Then they are called source RPMs. (RPM Package Manager, 2015)

3.2 Front-end

This part is devoted to user interface (UI) and technologies that are used for its creation. In this chapter mainly technologies useful for creating webpages are mentioned because UI should be a kind of webpage.

3.2.1 HTML, CSS, Bootstrap

HTML stands for HyperText Markup Language. This language is used for definition of webpage structure. Every HTML document should begin with DTD – document type declaration that helps web browser to define the rendering mode. (HTML Introduction, 2015)

CSS is abbreviation for Cascading Style Sheets. This language is intended for creating website design. The main reason, for which the CSS was developed, is separation of document content from document presentation. Separation of content and design can improve accessibility and flexibility. After separation, more HTML files can share the same CSS file etc. (CSS Tutorial, 2015)

Bootstrap is a framework based on HTML, CSS and JavaScript. It is intended for developing responsive design of websites and web applications. Its original name was Twitter Blueprint. Bootstrap consists of smaller components. From version 3.0 on it supports basic philosophy for designing of mobile applications. Because Bootstrap belongs to open source projects, developers can adapt the core according to their wishes. (Twitter, 2015)

3.2.2 JavaScript, Ajax, JQuery, React

JavaScript is a dynamic programing language that is usually used in webpage development. It is the part that interacts with the user, control the web browser, allows asynchronous communication and can modify the document that is displayed. (JavaScript Tutorial, 2015)

Ajax is a shortened form for asynchronous JavaScript and XML. This is not a programming language. It is just a technique that allows web applications to obtain data from the server asynchronously. Ajax merges more technologies together: HTML, CSS, DOM, and JavaScript. DOM is used for dynamic interaction with data and their dynamic display.

JQuery is a cross-platform JavaScript library that contains many useful functions intended for selecting DOM elements from displayed website, creating beautiful animations and developing Ajax applications. It is the most used JavaScript library.

React is a JavaScript framework developed by Facebook and Instagram. It is intended for building large applications that work with data, which changes over time. There are two basic ideas, in order to reach what React is intended for. (Facebook Inc., 2015)

1. Simple – Simple expresses a look of application in given point in time and React manages all data updates. (Facebook Inc., 2015)
2. Declarative – When data changes, React updates only part that was changed. (Facebook Inc., 2015)

Very important part in React is how to think in React. There are few rules for correct approach.

The UI has to be divided into component hierarchy. It is necessary to look at the whole UI and very carefully divide it into smaller components that can be nested. For this part single responsibility principle could be very useful, which states that each component should do only one thing. (Facebook Inc., 2015)

After dividing UI to components, it’s good to create static version of the UI in React. By creating this static version, states and props must not be used at all because they are intended for the interactive model. (Facebook Inc., 2015)

It is necessary to identify the minimal but complete UI state representation. It must be identified what is state and what is not. There are three questions, to which it is necessary to answer in searching for states.

a) Is it passed from parent via props? If so, then it probably is not a state.
b) Does it change over time? If not, it probably is not a state.
c) Can it be computed based on props in component? If so, it is not a state. (Facebook Inc., 2015)

It has to be identified, where the state should be placed. This could be a very confusing part for people who first time use React. Therefore, it is wise to follow these steps:

a) It is necessary to identify all components that render something based on the state.
b) It is necessary to find the component that owns the state.
c) Also the component higher in hierarchy that could own the state should be found.
d) If it is not possible to find a state owner, it is good to create a new one, that will be placed in hierarchy above and it will be the owner of the state.

Simply put, all states should be placed close to the higher component. (Facebook Inc., 2015)

Add inverse dataflow. After all steps mentioned before, the states have to be sent down the hierarchy by props. If child components are to do something after click or should have another function that can change the state, it is necessary to define the function in the component, in which the state is placed. After that, the function (callback) is sent as one of the props down the hierarchy and can be assigned to some event or called in some function. (Facebook Inc., 2015)

4 Implementation

This part includes a detailed description of the created packages. It is divided into two basic parts. The first part is devoted to back-end implementation and that part contains a description of the packages and their contents. The second part includes the description of front-end of the application and a short description of Twitter manager functionality.

4.1 Back-end

4.1.1 OAuth package – oauthlogin

This Package includes basic features needed for authorization by OAuth 1.0. It contains two small *structs* called *Config* and *Credentials*. There all necessary links and settings are placed.

Config contains these attributes:

- *Host* – It is URL to GNU Social main page
- *ReqTokenUrl* – This is link intended for getting temporary credentials from GNU Social.
- *AuthorizeUrl* – Address for starting authorization process.
- *AccessTokeUrl* – Address for getting credentials for third party application.
- *ConsumerId* – This is an ID that is generated by server for the application that is registered by user and will be used for resources processing.
- *ConsumerSecret* – This one is also generated by server for registered third party application and it is used for signing the requests.
- *SignatureMethod* – it says what kind of method is used for creation of request signature.
- *Version* – It is version of the OAuth that is used for authorization.
- *Callback* – This contains link to application, to which user will be redirected after successful authentication.

Config features

String () – this method writes to the standard output all information stored in an instance of the *Config* structure. It is very useful for debugging and can be used for displaying information loaded and used at the runtime of the application.

LoadConfig (confPath string) – this method loads data from JSON file. A parameter is a string that contains the path directly to the configuration file. Configuration data are placed in the configuration file in order to allow user to change and set up them. For example, *host* of GNU Social and others.

Credentials – contains only three attributes:

- *Token* – This token is used during authorization process and also during getting resources.
- *TokenSecret* – Also used during authorization and getting resources. Primarily, it is used for creating signatures.
- *Verification* – It is used to verify credentials over authorization process.

The main functions that are intended to establish communication

*RequireRequestToken (conf * Config) (*http.Response,error)* – This functions is used for getting temporary credentials from the GNU Social server. By using this function, it is only necessary to provide pointer to *Config* structure as a parameter. All needed things, which have to be done before sending request, are automated. If all is successfully done, it returns a response that contains temporary credentials (*token* and *tokenSecret*) and no error. Otherwise, it returns no response and some error.

*Authorize (oc *Config, credentials *Credentials) (*http.Response, error)* – This one is second step in authorization process. It is necessary to provide pointer to *Config* and pointer to *Credentials* as parameters. If all is in order, it returns a response that should contain form for user authentication and no error. If not, it returns no response and any error.

*SendVerification (oc *Config, credentials *Credentials) (*http.Response, error)* – After successful user authentication, verification code will be sent to callback address that is specified in configuration file. The verification code has to be placed in *Credentials*. Then this function can be called that sends the verification code to server and after verification by the server, it should receive a response that includes credentials for an access to resources. If all is in order, it returns the mentioned response and no error. If not, it returns no response and some error.

*RequireResources(oc *Config, credentials *Credentials, srcUrl, method string) (*http.Response, error)* – This function requires resources from the server. There are some parameters that have to be provided. The first two parameters include credentials and necessary things intended to create signature. The third parameter is *srcURL*. It is a link to resources that are required. If *srcUrl* contains some parameters then they have to be percent-encoded. The last parameter states, what method will be used for sending request (POST, GET…).

GenerateSignature(method, reqUrl, consumerSecret, tokenSecret string, authHeaderMap map*[string]string) string* – This method is used for generating signature. It needs many parameters.

- *Method* – Method of sending request (POST, GET…).
- *reqUrl* – Address of the resource.
- *consumerSecret* – This one is used as a key for encoding signature.
- *tokenSecret* - It is used as a key for encoding signature.
- *authHeaderMap* – Parameter includes necessary parameters intended for signature. The parameters are obtained from configuration file.

CreateBaseStringForSignature(method, reqUrl string, authHeaderMap map[string]string) string – It creates the base signature string as is mentioned in Chapter 3.1.2 describing OAuth 1.0. A meaning of the parameters is the same as it is described in previous method.

Hmacsha1(baseString, key string) string – This function creates the hash of the *baseString*. It is named according to the algorithm used for hashing (HMAC-SHA1).

GenerateNonce() string and *GenerateTimeStamp() string* are small functions for generating *nonce* and *timestamp* mentioned in Chapter 3.1.2 describing OAuth 1.0.

4.1.2 GNU Social API – gnusocial package

This package contains functions that use GNU Social REST API and OAuth 1.0 for logging. This API is developed for the needs of JYVSECTEC. It means, only significant features and properties are implemented that are currently needed. Of course, because it is intended to communicate with GNU Social REST API, all functions depend on the REST API and are also limited by the API.

Description of implemented features:

There are two small structures that represent the user profile and tweet in GNU Social, and one small structure that contains user credentials and configuration things needed for communication. Of course, they do not contain all attributes that GNU Social provides. Currently, there are only five attributes needed so far.

Structure *User* has got the following attributes:

- *Id* – It is ID of registered user in GNU Social.
- *Name* – Name of the user that is displayed in GNU Social.
- *Screen_name* – This is name that is specified during registration process.
- *Profile_image_url* – It is a link to user profile picture.
- *Friends_count* – That says, how many friends the user has got.

Structure *Tweet* contains these attributes:

- *Id* – Identification number of tweet.
- *Text* – Included text written by user who created the tweet.
- *Created_at* – It is a date on which the tweet was created.

- *In_reply_to_status_id* – If the tweet is in reply to another tweet, then this parameter is filled in. Otherwise, it is empty string.
- *In_reply_to_user_id* – Similar to previous attribute. But it is the identification number of the user that created the tweet that is commented.
- *In_reply_to_screen_name* – Screen name of the user that is filled in previous attribute.
- *User* – It contains information about owner of the tweet.
- Favorited – It is used to inform currently logged user whether the tweet is liked by the user or not.
- *Repeated* – Also intended to inform currently logged user, but this time, it is related to repeating the tweet.

Small structure *Conn* contains only two following parameters described in Chapter 4.1.1 describing *oauthlogin* package:

- OauthConf – pointer to *Config* structure in *oauthlogin* package
- OauthCred – pointer to *Credentials* as well placed in *oauthlogin* package.

*StartAuthentication(conn *Conn) (*http.Response, error)* – This function starts authorization by OAuth 1.0. It has only one parameter *Conn* that contains all necessary things for authorization process. This function returns response that contains a form for user authentication and no error, only if everything was successfully otherwise it returns no response and any error.

*FinishAuthentication(conn *Conn, authReq *http.Request) bool* – This method is called for finishing Authorization process. It needs two parameters:

- *conn* – It contains parameters for Authorization.
- *authReq* – Request that contains the verification code from GNU Social.

When the function is successfully finished, credentials needed for getting resources should be stored in *conn* parameter, and the function returns *true*. If something happened over processing, credentials will not be placed in *conn* and *false* will be returned.

*GetUserProfile(conn *Conn) (string, error)* – This one is intended to get information about a user that is currently logged in. Parameter *conn* contains all needed parameters for identifying the user. It returns a string that contains JSON document corresponding to *User* structure that is mentioned in this chapter. If something happens over processing, it returns an empty string and some error.

*SendTweet(conn *Conn, status string, id int) bool* – This feature has two possible uses. The first is to send new tweet to GNU Social server, and the second is to send tweet in reply to another. This is decided by parameter *id* that specifies whether it is a new tweet or a reply to another. If *id* is less than 0, then the function sends a new tweet. Otherwise, it sends a tweet in reply to the tweet, the identification number of which is specified as *id* parameter. Parameter *conn* was mentioned above. It returns *true*, if all was processed correctly. Otherwise it returns false.

*GetTweetById (conn *Conn, id int) (string, error)* – It is intended for getting one tweet from GNU Social, the identification number of which is specified as parameter *id*. If all is in order, it returns a string containing JSON document corresponding to *Tweet* structure that is described in this chapter. Otherwise, it returns an empty string and some error.

*RemoveTweet (conn *Conn, id int) bool* – This feature removes tweets. Each call of the function can remove only one tweet, identification number of which is specified as a parameter *id*. If a tweet will be removed, it returns *true*. Otherwise it returns *false*.

*RetweetTheTweet (conn *Conn, id int) bool* – It retweet the tweet that is specified as parameter *id*. Also, if everything is alright, it returns *true* otherwise *false*.

*AddFavorite (conn *Conn, id int) bool* – User in GNU Social can mark a tweet as favorite, when they like the tweet. This function adds the tweet to the user’s favorite tweets. It only needs the identification number of the tweet. It returns *true*, if all will be done correctly, otherwise false.

*RemoveFavorite (conn *Conn, id int) bool* – Every tweet marked as favorite, can be also unmarked. It is necessary to provide the identification number of the tweet in this function and it will be done. After finishing that, it will return *true*, if tweet was unmarked, otherwise false.

*GetListOfFavorites (conn *Conn, count int) (string,error)* – Function that returns tweets that were favorite of a user who is currently logged in. Parameter *count* limits the amount of tweets to be returned. It can return the number of tweets between 0 and 200 at once. If the count is not defined, then it returns 20 tweets as a default amount. The tweets are placed in string as JSON array and every tweet corresponds to *Tweet* structure that is described in this chapter. If something happens over processing, then it returns empty string and any error

*PublicTimeline(conn *Conn, count int) (string,error)* – It returns the most recent tweets from all posted tweets. If the count is less than 0, it returns 20 tweets. It can return amount of tweets between 0 and 200 at once. All tweets are placed in the string as JSON array. It returns tweets and no error, if all is successfully done. Otherwise it returns an empty string and some error.

*HomeTimeline(conn *Conn, count int) (string,error)* – This feature is similar to the previous one. Although, this does not return recent tweets from all tweets, but recent tweets from all tweets related to the currently logged user. It means, there are tweets that state, for example, the user added something to their favorite tweets. Also, there can be tweets that state, the user retweeted something. It is slightly different from getting the public timeline described above because the public timeline returns only new tweets and the replies to tweets. At once it can return from 0 to 200 tweets and all are placed in a string as a JSON array. If something fails over processing, it returns an empty string and any error. Otherwise it returns a string of tweets and no error.

*GetTimeline (conn *Conn, urlToTimeline string) (string,error)* – This method arose only because of code optimization. It contains code that is used in the two previous functions. If all is in order after processing, it returns a string including tweets as a JSON array and no error. Otherwise, it returns an empty string and some error.

ParseTweets(r io.Reader) ([]Tweet,error) – This one is intended to parse tweets that are placed in Reader structure as a JSON. All tweets, which are placed there, are returned in an array. Every tweet placed in the array has the same structure as *Tweet* mentioned in this chapter. If something happens during the processing of the tweets, the function returns an empty array and some error.

EncodeTweetsToJson(tweets []Tweet) (string, error) - It makes the opposite of the previous function. This has as a parameter an array of tweets and encodes them to JSON string. Of course, if something fails during the process, it returns an empty string and some error.

4.1.3 Twitter manager and supplement package.

The First, small package is to be mentioned that contains some additional functions. Its name is *decoder*. As the name indicates, the functions inside are mainly devoted to parsing data. The second one is to be mentioned Twitter manager and basic introduction about how it works. The last one is the package *twittermgr* that contains a runnable application.

Functions included in *decoder*

ParseString(text, sep1, sep2 string) map*[string]string* – This function is intended to parse any string, which is specified as parameter *text*, by two separators that are specified as parameters *sep1* and *sep2*. The first step is to divide *text* into n parts by *sep1*. After that, each part which arose by the first dividing is divided into n parts by *sep2*. Then strings which arose after the second dividing are placed into a map. Although, only the first two strings of every part are placed to the map, because one is used as a key and other one as a value. This function is not absolutely general, because it is designed mainly for parsing parameters that are placed in HTTP requests. After all, if no error occurred, it returns the map of parameters and no error. Otherwise, it returns an empty map and some error.

*ParseResponseBody (res *http.Response) (*map *[string] string, error)* – It is based on the previous function. This is less general because separators are exactly specified inside the function. Those separators are “&” and the second one is “=”. It parses parameters from the HTTP response and provides them in a map. If all is successfully done, it returns map and no error. Otherwise it returns an empty map and some error.

*GetVerificationFromResponse (res *http.Response) string* – This one arose based on the *Callback* parameter that is mentioned in Chapter 3.1.2 related to OAuth 1.0. This function is used only if the callback parameter is set to *oob*. In that case, the verification code is placed inside the body of the response and has to be cut out. It is tested and prepared to the future.

*GetVerificationCodeFromReq (req *http.Request)* map *[string] string* – This feature is also based on *Callback* however, in this case, the value is any address and the verification code is placed in URL as a one of the parameters. It returns a map of parameters including the verification code and no error or an empty map and some error.

Twitter manager

The application is designed for running under Linux. It runs as a small proxy server on localhost. The default port is set to 4000 however, it can be changed by user before starting. All the configuration parameters necessary for running, are to be loaded from configuration file. At runtime it gets requests from the user interface, which are described in the next chapter devoted to front-end (4.2). It answers only to requests specified in the package called *twittermgr*. If it gets any request that is not specified it ignores that. The application stores logged users at runtime. The only thing that can be provided without user authentication is public timeline. The functions intended for users as like, unlike, tweet, retweet will not be done, until any user is logged in and chosen in user interface.

***Twittermgr* package**

LoggedUser – This structure represents the user that is currently logged in Twitter manager and contains basic information about the user and their credentials intended for GNU Social. The attributes are as follows:

- *Cred* – It is pointer to structure *Conn* that is declared in package gnusocial.
- *UserProf* - This is pointer to structure *User* declared in gnusocial too.

Handler takes care of all requests that are delivered to Twitter manager. It contains the following attributes:

- *GnusConn* – Pointer to structure *Conn*. It is used as a template for user authentication.
- *Users* – It is an array of *LoggedUsers*. Every user is stored in this array after authentication.
- *NewUser* – When authentication starts, in this attribute is stored every information about the user. After authorization the user is moved to *Users* and this reference is set to *nil*.

Functions defined for *Handler* structure

*(h *Handler) UsersInJason() string* – This function returns a string that contains all information about logged users. If no user is logged in, it returns a string containing only an empty array. Otherwise, the string that contains JSON array of users is returned.

*(h * Handler)getUserById(id int) *LoggedUser* – This one returns a user, whose identification number is defined as a parameter *id*. If the user is not placed in the attribute *Users,* then it returns *nil.*

*(h * Handler) removeLoggedUser(id int) bool* – When handler gets the request for logging the user out, this method is used for removing the user from logged users. If

the user is not placed in *Users*, then it returns false. Otherwise, it removes the user and returns *true*.

*(h *Handler) ServeHTTP(wr http.ResponseWriter, req *http.Request)* – This one is the largest method. It takes care of catching requests and their processing. Here are the registered requests and under each of them is a brief example:

"/getPublicTimeline" – This is requirement for tweets that are available on the public timeline. It can contain one parameter that is user identification number. If the parameter is included, then the tweets, which will be sent as a response, will be partially related to the user that asked for them. Otherwise it will return tweets without any relationship with user.

“http://localhost:4000/getPublicTimeline?id=1”

“http://localhost:4000/getPublicTimeline”

"/getHomeTimeline" – It requires tweets available on the home timeline. The response to this request depends on the logged user. The request has to contain the identification number of a user that requires records from home timeline. If the request does not contain user ID or contains a bad user ID, then the response includes only an empty array.

“http://localhost:4000/getHomeTimeline?id=1”

"/profile" – This is registered as a requirement for obtaining the user profile. It also has to include identification number of user that asks for its profile. If not, it will return only expression about a failure.

“http://localhost:4000/profile?id=1”

"/logInUser" – Requirement for starting user authentication. It does not need to include any parameters. For starting of user authorization it is not necessary to provide any information.

“http://localhost:4000/logInUser”

"/logOutUser" – Requirement for user logout. This request has to contain user identification number. Without that, it is not possible to log out any user.

“http://localhost:4000/logOutUser?id=1”

"/getLoggedUsers" – It requires information about all logged users. This does not need any parameter.

“http://localhost:4000/getLoggedUsers”

"/newTweet" – This is the request for sending a new tweet to GNU Social. It has to contain the identification number of the user that sends the tweet and also any text.

“http://localhost:4000/newTweet?id=1&text=any%20text”

"/like" – This one is a request for adding any tweet among favorite tweets of the user. It has to include some parameters. One is the identification number of a user who wants to add a tweet to their favorite tweets. Number two is the identification number of a tweet that is to be added to the user favorite tweets.

“http://localhost:4000/like?id=1&tweetId=45”

“/unlike” – it is very similar to “/like” request. Also, it has to contain the same attributes, however, it does the opposite.

“http://localhost:4000/unlike?id=1&tweetId=45”

“/removeTweet” – As the previous two functions, this one contains the same parameters. The identification number of tweet specifies, which tweet will be removed, and the user identification number specifies who wants to remove the tweet. Although, all parameters are correct, the tweet does not have to be removed. If user, who is specified by id is not owner of the tweet, then tweet won’t be removed.

“http://localhost:4000/removeTweet?id=1&tweetId=45”

“/retweet” – This is a requirement for retweeting. As the previous three functions, this one has the same parameters. User identification number states who wants to retweet, and tweet ID says, which tweet will be retweeted.

“http://localhost:4000/retweet?id=1&tweetId=45”

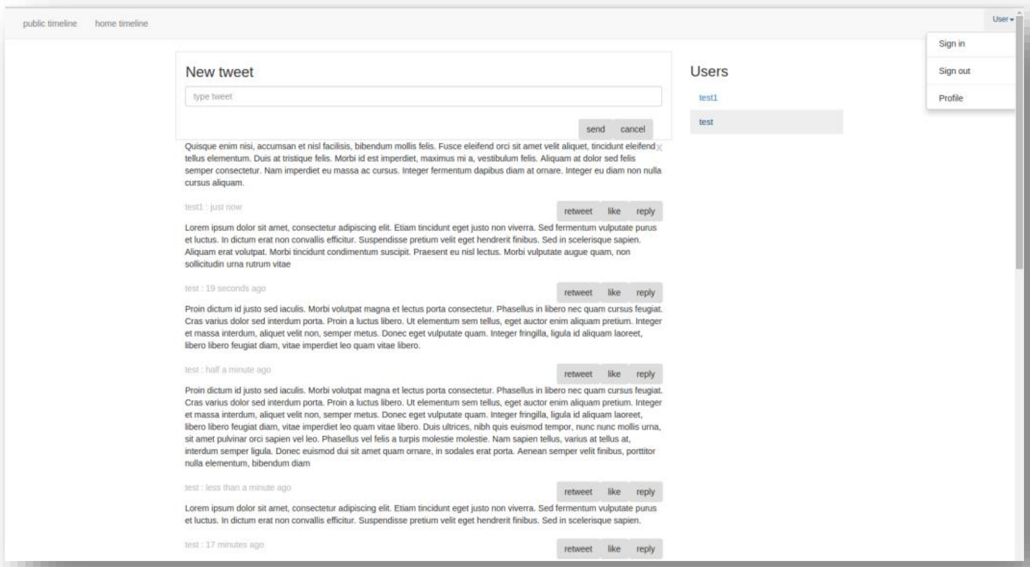
“/replyToTweet” – requirement that needs to contain the following three parameters.

- *id* – Identification of the user that wants to send a reply.
- *tweetId* – It is ID of the tweet that will include the reply.
- *text* – Text of the reply.

main () – is a function that executes Twitter manager.

4.2 Front-end

Significant role by creating the front-end of the application belongs to React framework. Thanks to that framework, all view is divided into more parts that are joined together. Because the design was not the main goal of the bachelor’s thesis, it does not look very good. It will be redesigned in the future. The nature of the front-end is to create a user interface runnable in web browser that will fulfill all needed functions. The interface has to support tweeting and changing users at runtime. Also, it should be designed in order to ensure intuitive work for users. The basic appearance of the application is displayed in the Figure 5.



Quisque enim nisi, accumsan et nisl facilisis, bibendum mollis felis. Fusce eleifend orci sit amet velit aliquet, tincidunt eleifend tellus elementum. Duis at tristique felis. Morbi id est imperdiet, maximus mi a, vestibulum felis. Aliquam at dolor sed felis semper consectetur. Nam imperdiet eu massa ac cursus. Integer fermentum dapibus diam at ornare. Integer eu diam non nulla cursus aliquam.

test1 : just now

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Etiam tincidunt eget justo non viverra. Sed fermentum vulputate purus et luctus. In dictum erat non convallis efficitur. Suspendisse pretium velit eget hendrerit finibus. Sed in scelerisque sapien. Aliquam erat volutpat. Morbi tincidunt condimentum suscipit. Praesent eu nisl lectus. Morbi vulputate augue quam, non sollicitudin urna rutrum vitae

test : 19 seconds ago

Proin dictum id justo sed iaculis. Morbi volutpat magna et lectus porta consectetur. Phasellus in libero nec quam cursus feugiat. Cras varius dolor sed interdum porta. Proin a luctus libero. Ut elementum sem tellus, eget auctor enim aliquam pretium. Integer et massa interdum, aliquet velit non, semper metus. Donec eget vulputate quam. Integer fringilla, ligula id aliquam laoreet, libero libero feugiat diam, vitae imperdiet leo quam vitae libero.

test : half a minute ago

Proin dictum id justo sed iaculis. Morbi volutpat magna et lectus porta consectetur. Phasellus in libero nec quam cursus feugiat. Cras varius dolor sed interdum porta. Proin a luctus libero. Ut elementum sem tellus, eget auctor enim aliquam pretium. Integer et massa interdum, aliquet velit non, semper metus. Donec eget vulputate quam. Integer fringilla, ligula id aliquam laoreet, libero libero feugiat diam, vitae imperdiet leo quam vitae libero. Duis ultrices, nibh quis euismod tempor, nunc nunc mollis urna, sit amet pulvinar orci sapien vel leo. Phasellus vel felis a turpis molestie molestie. Nam sapien tellus, varius at tellus at, interdum semper ligula. Donec euismod dui sit amet quam ornare, in sodales erat porta. Aenean semper velit finibus, porttitor nulla elementum, bibendum diam

test : less than a minute ago

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Etiam tincidunt eget justo non viverra. Sed fermentum vulputate purus et luctus. In dictum erat non convallis efficitur. Suspendisse pretium velit eget hendrerit finibus. Sed in scelerisque sapien.

test : 17 minutes ago

Figure 5 Basic View of the application.

4.3 GUI of Twitter manager and its functionality

As mentioned before, all views consist of few parts. Every part is designed as a small component. The list of the components contains these parts:

- navigation bar
- timeline
- new tweet
- tweet
- list of users

These five parts is contained in an element called “MainPage”. In the following part, each component is described with its functions.

4.3.1 Navigation bar

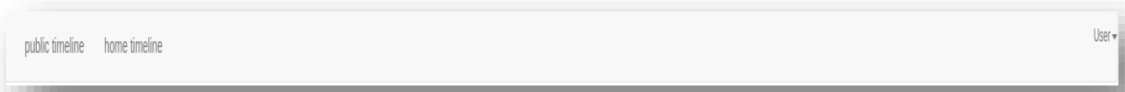


Figure 6 Navigation bar of main page

This navigation bar consists of three parts. The first part on the left side is button “Public timeline”. After clicking on this button, the application displays default amount of tweets in the middle of the main page that is called “timeline”. After scrolling down, it displays previously created tweets. “Home timeline” is a part that contains tweets that are in the public timeline and also tweets that are more related to the logged-in user. The tweets are displayed in the middle of the main page inside “timeline”. If no user is logged in or none is chosen in the list of users, it returns no tweets. The last part is drop-down menu “User” that is displayed in the Figure 7.

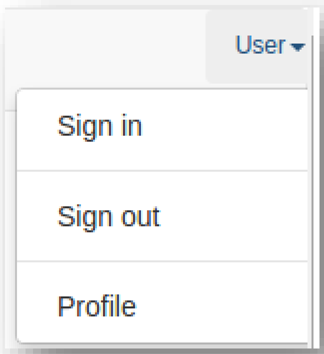


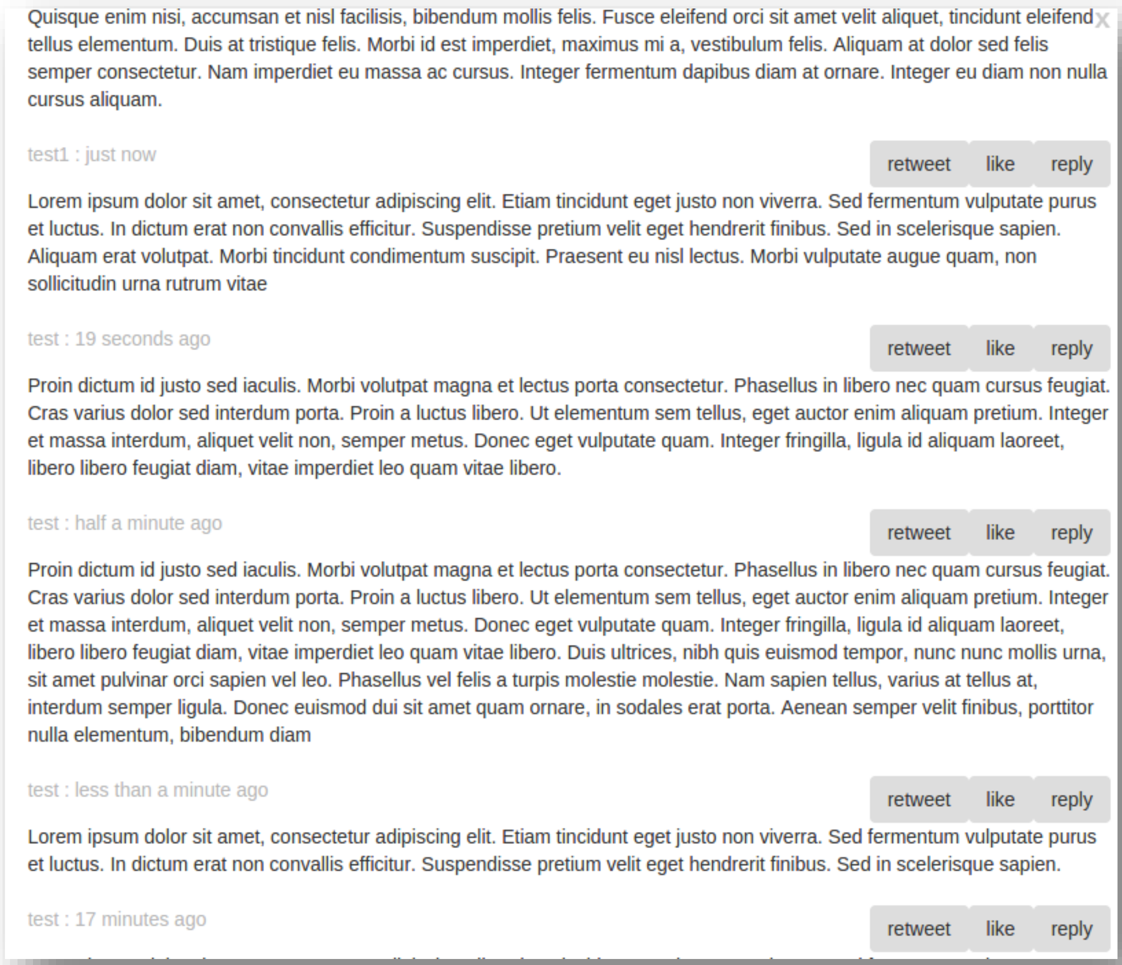
Figure 7 drop-down menu for user

As can be seen in the Figure 7, there are three options.

- “Sign in” will open a new tab that is intended for user authentication.
- “Sign out” will log out the user that is currently chosen in the list of users.
- “Profile” is used for getting some information about a user that is currently chosen in the list of users.

4.3.2 Timeline

This part is used for displaying tweets from GNU Social. It is updated always when the user clicks on the part “Public timeline” or “Home timeline” and also when the user sends a new tweet or any reply to tweet. The timeline is updated, even if the current user is changed. Outlook of the timeline is shown in the Figure 8.



Quisque enim nisi, accumsan et nisl facilisis, bibendum mollis felis. Fusce eleifend orci sit amet velit aliquet, tincidunt eleifend tellus elementum. Duis at tristique felis. Morbi id est imperdiet, maximus mi a, vestibulum felis. Aliquam at dolor sed felis semper consectetur. Nam imperdiet eu massa ac cursus. Integer fermentum dapibus diam at ornare. Integer eu diam non nulla cursus aliquam.

test1 : just now

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Etiam tincidunt eget justo non viverra. Sed fermentum vulputate purus et luctus. In dictum erat non convallis efficitur. Suspendisse pretium velit eget hendrerit finibus. Sed in scelerisque sapien. Aliquam erat volutpat. Morbi tincidunt condimentum suscipit. Praesent eu nisl lectus. Morbi vulputate augue quam, non sollicitudin urna rutrum vitae

test : 19 seconds ago

Proin dictum id justo sed iaculis. Morbi volutpat magna et lectus porta consectetur. Phasellus in libero nec quam cursus feugiat. Cras varius dolor sed interdum porta. Proin a luctus libero. Ut elementum sem tellus, eget auctor enim aliquam pretium. Integer et massa interdum, aliquet velit non, semper metus. Donec eget vulputate quam. Integer fringilla, ligula id aliquam laoreet, libero libero feugiat diam, vitae imperdiet leo quam vitae libero.

test : half a minute ago

Proin dictum id justo sed iaculis. Morbi volutpat magna et lectus porta consectetur. Phasellus in libero nec quam cursus feugiat. Cras varius dolor sed interdum porta. Proin a luctus libero. Ut elementum sem tellus, eget auctor enim aliquam pretium. Integer et massa interdum, aliquet velit non, semper metus. Donec eget vulputate quam. Integer fringilla, ligula id aliquam laoreet, libero libero feugiat diam, vitae imperdiet leo quam vitae libero. Duis ultrices, nibh quis euismod tempor, nunc nunc mollis urna, sit amet pulvinar orci sapien vel leo. Phasellus vel felis a turpis molestie molestie. Nam sapien tellus, varius at tellus at, interdum semper ligula. Donec euismod dui sit amet quam ornare, in sodales erat porta. Aenean semper velit finibus, porttitor nulla elementum, bibendum diam

test : less than a minute ago

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Etiam tincidunt eget justo non viverra. Sed fermentum vulputate purus et luctus. In dictum erat non convallis efficitur. Suspendisse pretium velit eget hendrerit finibus. Sed in scelerisque sapien.

test : 17 minutes ago

Figure 8 Timeline displaying some tweets

4.3.3 New tweet

This part is not only intended for sending a new tweet but also for replying to tweets. It contains a part for writing text and two buttons. One button sends the tweet and the next one cancels the tweet. It automatically forbids to send an empty tweet. If the tweet was sent correctly, it is shown in the timeline.

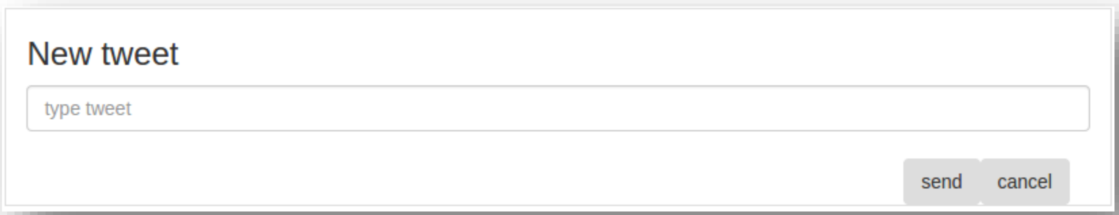
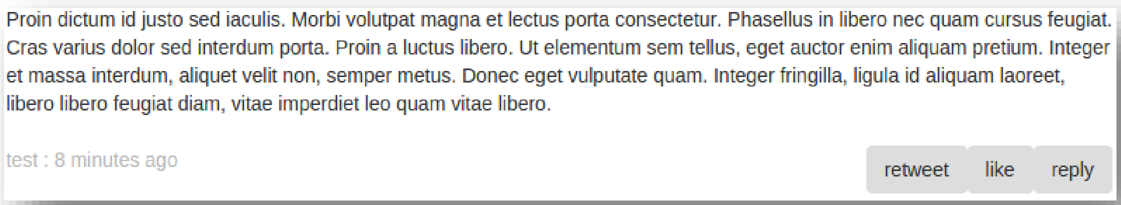


Figure 9 Form for sending new tweet and replying to tweets.

4.3.4 Tweet

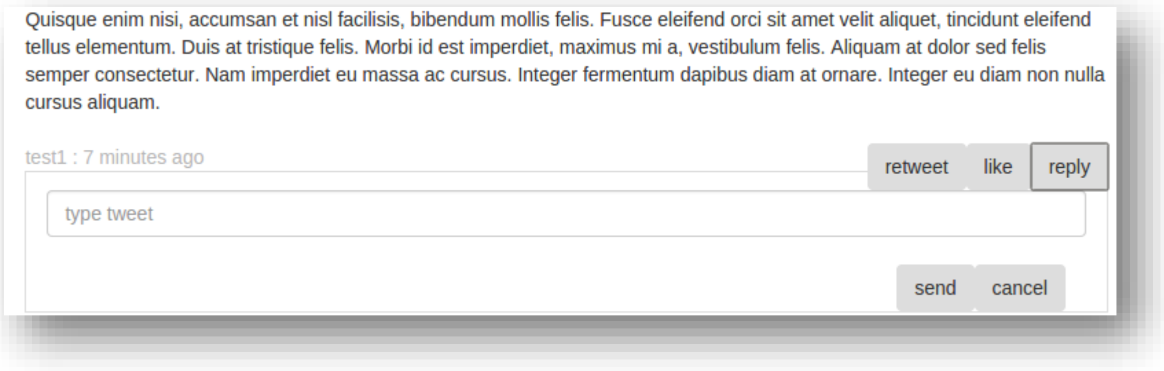
The next part is intended for displaying a tweet and making some actions that are related to the tweet. For the illustration the Figure 10 is added.


Proin dictum id justo sed iaculis. Morbi volutpat magna et lectus porta consectetur. Phasellus in libero nec quam cursus feugiat. Cras varius dolor sed interdum porta. Proin a luctus libero. Ut elementum sem tellus, eget auctor enim aliquam pretium. Integer et massa interdum, aliquet velit non, semper metus. Donec eget vulputate quam. Integer fringilla, ligula id aliquam laoreet, libero libero feugiat diam, vitae imperdiet leo quam vitae libero.

test : 8 minutes ago

Figure 10 Looks of the tweet in twitter manager

The tweet in the figure bellow has few parts. The one part contains the text of the tweet. In the right top corner is placed a small button for removing the tweet. Of course, it is hidden right now because only the user creating the tweet, can see the cross in the corner, which is because only the owner of the tweet can remove it. Under text, in the left down corner the user who made this tweet and time are shown as well as when it was published. In the right down corner three buttons are placed one is for retweeting the tweet that is called “retweet”. “like” is the button that adds the tweet to the user favorites. After that, it changes the text of the button to “unlike”. After clicking on it again, the tweet is removed from user favorites. The last one is “reply” which displays the part “new tweet” inside this part. Then it will change a view slightly, and its appearance is as follows in the Figure 11.


Quisque enim nisi, accumsan et nisl facilisis, bibendum mollis felis. Fusce eleifend orci sit amet velit aliquet, tincidunt eleifend tellus elementum. Duis at tristique felis. Morbi id est imperdiet, maximus mi a, vestibulum felis. Aliquam at dolor sed felis semper consectetur. Nam imperdiet eu massa ac cursus. Integer fermentum dapibus diam at ornare. Integer eu diam non nulla cursus aliquam.

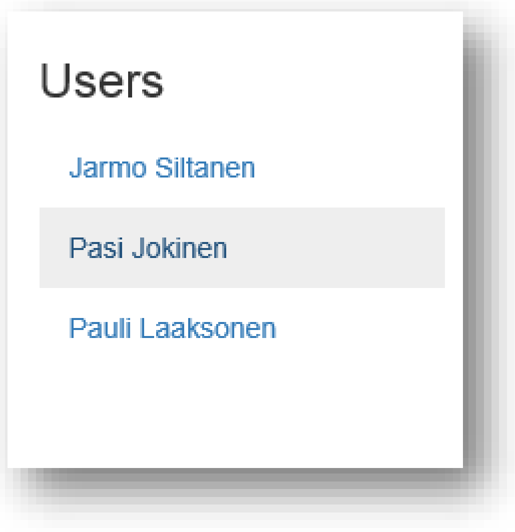
test1 : 7 minutes ago

Figure 11 Tweet with part for retweeting

The functionality is almost the same as mentioned in the section about new tweet (See page 37). The only difference is if the button “cancel” is clicked, then the form for retweeting is to be hidden again.

4.3.5 List of users

After successful user authentication, the user is shown in this part of the main page. It is a clickable list and always after click on any user, the user is set as the current one. If the current user is set and the user clicks on the “sign out” (it is mentioned in section navigation bar), the current user is immediately removed from the list of users. After that, no user is chosen and public timeline is refreshed. List of users is shown in the Figure 12.



Users

Jarmo Siltanen

Pasi Jokinen

Pauli Laaksonen

Figure 12 List of users

5 Testing

Testing is included in every kind of development and this application is no exception. Software testing is the part of the software development that provides the information about the quality of the software under test. Testing can provide an answer whether the requirements were met or not, whether the software responses are correct to all kinds of inputs or not, and also whether the performance is sufficient or not. Of course, testing can answer to many other questions and provide interesting information.

Currently, there are many testing methods and types. Static and dynamic testing, black-box testing or white-box testing belong to the popular test methods. Testing of the software is divided into four levels. The lowest level is unit testing. The second level is integration testing that is slightly more complicated. The third level is component interface testing and the last one, the highest and the most complicated is the system testing.

The tests of the Twitter manager are not a part of the bachelor's thesis, however, it is necessary to mention the testing phase. Every part of the application was tested during development in many different situations. Of course, many errors were caught and repaired, however, only one single person is not able to catch all errors and test all possible situations that can occur during the use of the application. Therefore, next two phases of testing will be written. The first phase will be the creation of the unit tests. Currently the unit test are well known. They are often used in many software companies. The unit test for this application will be created for front-end and back-end as well. Every package of the back-end will have its own test package that should cover 100 % of content of the tested package. The tests of the front-end will test every component created in React. The second phase will be a test after the deployment of the application on the server. This will be a part, in which many errors can be revealed. The first time, this test will be probably made by the employees of JYVSECTEC. After this testing and repairing all occurred errors the application will be ready to use during the cyber security exercises.

6 Conclusion and results

The objectives of the thesis were to create GO API for communication with GNU Social, GO API for OAuth 1.0 authorization protocol and the application was to be able to manage more than one account simultaneously. The application was developed according to the requirements mentioned in chapter 2.1 Requirements. GO API was divided into three packages and the application was placed in separate package.

The OAuth package (oauthlogin) contains API for communication with some server that uses OAuth 1.0 authorization protocol. Using this package, it should be possible to establish communication with any server that uses OAuth 1.0 authorization protocol.

The package for GNU Social contains many functions for communication with GNU Social server. Besides, the implemented GO API for GNU Social contains many additional functions that were not required in the first version. There are additional functions intended for retweeting, adding tweets to favorites, removing tweets from favorites, showing favorites and other functions. This package is directly dependent on the GNU Social REST API. If some change in the REST API related to the GO API in this package will be made, then it is necessary to make the changes in this package as well.

The back-end of the application is placed in the package called twittermgr. Many functions created for communication with GNU Social are used in Twitter manager and a few can be used in the future. All the functions necessary for the communication with the front-end part of the application have been implemented.

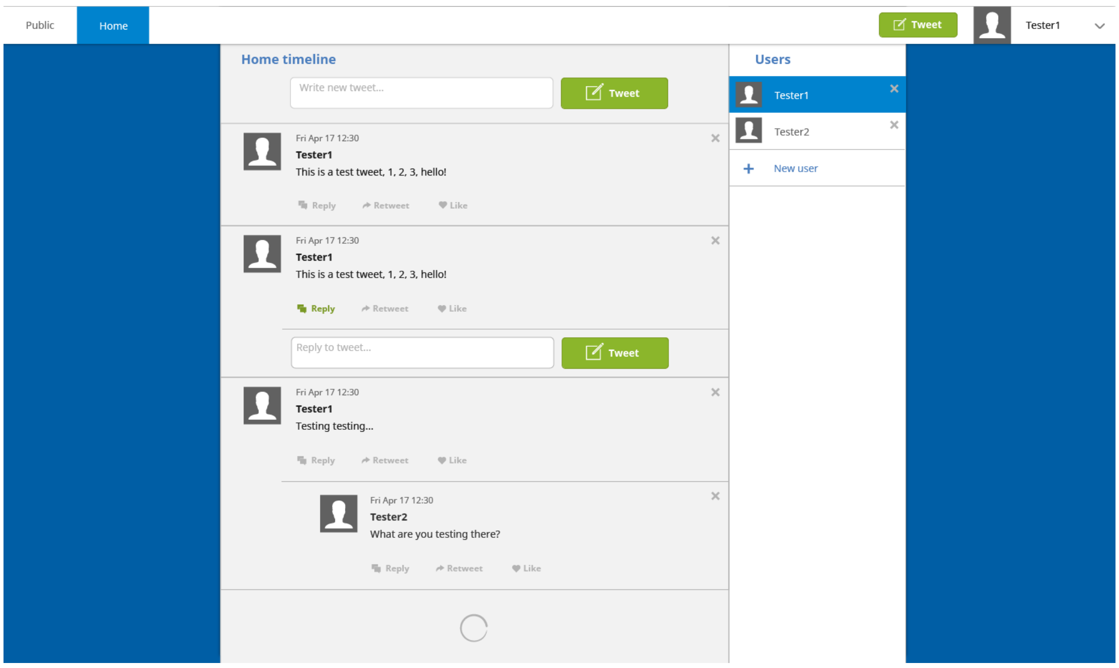
The front-end of Twitter manager consists of three files. It was created as a single webpage application so far. New webpages can be added in the future if necessary. The front-end of the application is designed according to the requirements mentioned in Table 2. Every front-end requirement was fulfilled in the first appearance of the application as well as in the new one. The new outlook of the application is shown in Figure 13. The new appearance was designed for the following reasons:

- to make the appearance of the application visually more pleasing to the eye
- to make the user interface more user friendly
- to make the functionality of the UI more intuitive

Only one back-end requirement was not met. The requirement is related to the creation of RPM and DEB packages. The packages were not created because a new appearance of the application was created. The appearance is almost finished and prepared for testing.

The application was tested during the development in many situations. Of course, it is necessary to make more tests in order to ensure the quality of the application. Therefore, in the near future the application will pass further tests. The application will be tested in two ways. The unit tests will be the first kind of tests. Secondly, it will be tested by the users who can provide some feedback. Then the application can be repaired and improved according to the feedback.

Then next step after the testing can be to add the groups to the application. The group could be used as a container for some topic. Dividing the topics into groups could ensure even higher clarity of the content.



Fri Apr 17 12:30
Tester1
This is a test tweet, 1, 2, 3, hello!

Fri Apr 17 12:30
Tester1
This is a test tweet, 1, 2, 3, hello!

Fri Apr 17 12:30
Tester1
Testing testing...

Fri Apr 17 12:30
Tester2
What are you testing there?

Figure 13 New design for Twitter Manager

References

Client–server model. Accessed on 2015, April 16. Retrieved from Client–server model: http://en.wikipedia.org/wiki/Client%E2%80%93server_model

CSS Tutorial. Accessed on 2015, April 24. Retrieved from CSS: http://www.w3schools.com/css/default.asp

E. Hammer-Lahav, E. Accessed on 2015, February 4. Retrieved from RFC 5849.

Facebook Inc. Accessed on 2015, March 20. Retrieved from Thinking in React: https://facebook.github.io/react/docs/thinking-in-react.html

Fernandez-Sanguino, J. Accessed on 2015, May 3. *The Debian GNU/Linux FAQ*. Retrieved from Basics of the Debian package management system: https://www.debian.org/doc/manuals/debian-faq/ch-pkg_basics.en.html

Fredrich, T. Accessed on 2015, March 29. (Pearson eCollage) Retrieved from What is the REST anyway?: http://www.restapitutorial.com/lessons/whatisrest.html

Google. Accessed on 2015, January 25. Retrieved from The GO Programing Language: https://golang.org/

HTML Introduction. Accessed on 2015, April 20. Retrieved from HTML: http://en.wikipedia.org/wiki/HTML

JavaScript Tutorial. Accessed on 2015, April 22. Retrieved from JavaScript: http://www.w3schools.com/js/

Matt Lee, M. N. Accessed on 2015, January 18. Retrieved from GNU Social: https://gnu.io/social/

Oauth. Accessed on 2015, March 3. Retrieved from OAuth: http://en.wikipedia.org/wiki/OAuth

Representational state transfer. Accessed on 2015, March 12. Retrieved from Representational state transfer: http://en.wikipedia.org/wiki/Representational_state_transfer

RPM Package Manager. Accessed on 2015, May 5. Retrieved from RPM package manager: http://en.wikipedia.org/wiki/RPM_Package_Manager

Twitter. Accessed on 2015, May 1. Retrieved from Bootstrap: http://getbootstrap.com/