1. Project description:

In the recent years, the KTN project has been a part of the Common project. However, as most groups have been doing it separately and are not using the implementation for the networking layer of their application, the KTN project will now stand on its own. However, there are many elements which can be re-used because the implementation should be language-agnostic. Hence, it could be modified and reused to fit the data models that are used in the common project (which will probably be implemented in Java).

In this project, you will create a *chat server* and *client* that communicate over a simple proprietary application-layer protocol on top of TCP.

The main purpose of this project is to implement a *protocol*, so it is not strictly necessary that you use a certain programming language. A protocol is by definition a design that enables different systems, languages or endpoints to speak together using a common, shared language. To test your implementation, we will simply check that the protocol is implemented correctly according to the requirements specification.

However, since the curriculum is based on python, and the language is used throughout the course, it will also be used in the skeleton code provided in this project. It is also very likely that the course staff will be able to provide the best help and assistance if you implement the project using python.

The goal of this new project is to have a practical and useful approach to network programming. Previously in this course, a lot of emphasis was put on very detailed and technical aspects of lower-level network programming. As it has become in the recent years, there are very good open source libraries for most of these tasks that does all the heavy lifting for us.

We believe that using and learning some of these libraries in this project will prove useful for you in the future as well as for your own programming projects.

Happy hacking!

2. Overall Requirements

You will implement a simple protocol for a command line interface (cli) chat client that communicates with a server backend. The protocol uses the JavaScript Object Notation (JSON) format. JSON offers a readable and human friendly format for serializing data and objects when sending the information over the web, between processes and / or between programming environments; as well as for storing / retrieving information. It has become a standard in modern web applications through its heavy use within application programming interfaces (API) by companies such as Twitter, Instagram and Facebook. For more information on the JSON format, you can refer to:

- http://en.wikipedia.org/wiki/JSON
- http://json.org/example
- http://json.org

The server must (evidently) be able to handle multiple connected clients at once.

3. Protocol and detailed requirements

3.1 Log in with username

- A username must be *unique*, meaning that it must not be already in use by another logged-in user.
- A username can *only* contain alphanumerical characters and underscores
- When logging in succesfully, the server should return the username, as well as the message backlog (all the messages so far in the session, would be

The client will send a message to the server in this form when attempting to log in:

```
{
'request': 'login',
'username': <username>
}
```

The server may respond in three different ways:

```
{
'response': 'login',
'username': <username>,
'messages': [<messages>]
}
```

or

```
{
'response': 'login',
'error': 'Invalid username!',
'username': <username>
}
```

or

```
{
'response': 'login',
'error': 'Name already taken!',
'username': <username>
}
```

3.2 Send a message to logged-in clients

The message is sent from a logged-in client, and then broadcasted from the server to all the logged-in clients (the easiest to do here is to send the message to the server and then send it to all clients, including sending it back to the client who sent it).

This is a somewhat special case, since it is a server “push” that is generated and sent to the client and not a response from a previous request (except for the client who sent it). But we can still use it in the same manner as before, even though it goes a little bit against the request-response scheme that we have used until now.

The client will send this message to the server:

```
{
'request': 'message',
'message': <themessage>
}
```

The server will respond with this message to all clients:

```
{
'response': 'message',
'message': <themessage>
}
```

or this message to only the client who sent the message:

```
{
'response': 'message',
'error': 'You are not logged in!',
}
```

NOTE:
Since a message can be sent out at any time, you can use this to for instance send out notifications about people logging in and out (if you want to). It is further not required that there is a link between the messages and the user sending it, but it should be easy to format the message like this: <username> said @ <timestamp>: <the message>.

3.3 Log out client

A logged-in client can log out by sending this message to the server:

```
{
'request': 'logout'
}
```

The server can respond in two different ways:

```
}
'response': 'logout',
'username': <username>
}
```

or

```
{
'response': 'logout',
'error': 'Not logged in!',
'username': <username>
}
```

3.2 Notes and further work

The requirements listed above are the *minimal* requirements necessary to get the project approved, and it will be tested against a working implementation, both from the client and the server side.

There are no requirements for persistence across sessions or server restarts, meaning that the chat and everything else can be in-memory during runtime. However, it is fairly easy to implement persistence using for instance mongodb or another non-relational database, and also very useful if you want to use the project, or parts of it in a real life or production setting (http://api.mongodb.org/python/2.6.3/installation.html).

Other functions that is not mandatory, but can be added:

- Moderators
  - Banning users
  - Removing messages
- Chat rooms
  - creating chat rooms
  - logging into chat rooms

4. Deadlines and deliverables

4.1 KTN1: (Project plan) Deliver by 03.03.14

You should deliver:
- Classes diagrams
- Sequence diagrams for different use cases like:
    - login
    - send message from client
    - logout
- A short textual description of your design

The goal here is for you to show us that you understand the task at hand and have plan for solving it.

4.2 KTN2: Deliver a working implementation by 24.03.14

Deliver:
- Your complete implemenation of the Chat Client and Server in a .zip file.
- Updated KTN1 so it matches the final implementation.
- You should also demonstrate your Chat for one of our student or teaching assistants at P15.