Lab 3

Asynchronous client-server communication

Authors:

Sahand Sadjadee

Alexander Kazen

Gustav Bylund

Per Jonsson

Tobias Jansson

Spring 2014

TDDD24 Web Programming and Interactivity

TDDI15 Advanced Programming and Interactivity on the WWW

http://www.ida.liu.se/~TDDD24/

Department of Computer and Information Science (IDA)

Linköping University

Sweden

1. Introduction

In this third lab, you will complete your Twidder web application by implementing the communication between the front-end, implemented in lab 1, and the back-end, implemented in lab 2.

You are required to develop step by step and implement each step according to the instructions. Once you are finished with each lab, you will present your work to your responsible lab assistant. For more specific information about the presentation and evaluation process of lab 3, please check section 6: Presentation and Evaluation. For more general information about the examination process, please check the course page.

Requirements

By the end of lab 3, the following **requirements** shall be met:

Functional

- There can only be one valid session at a time. It means once the user is logged in, the other possible session shall automatically be expired. In case of the application being opened on that expired session, in some other browser or environment, the welcome view shall be automatically displayed.

Non-Functional

- The server and client shall communicate asynchronously.
- HTTP and WebSocket requests are used for implementing one way and two way communication between the client and server.
- JSON shall be used as data exchange format.

2. Project folder

In step 0, you relay out the project folder according to the Flask recommended layout pattern.

3. Lab Instructions

Step 0: Packing the web application

In this section you will merge the client-side and server-side code into one project folder, named Twidder, based on the project layout pattern recommended for Flask based applications. All your HTML, CSS and Javascript files are considered as static. client.html is also required to be served by opening the web application root. This can be done by using send_static_file() function available in Flask object.

URLs	Filename
/ /static/client.html	client.html

Information about project folder layout
http://flask.pocoo.org/docs/patterns/packages/

Information about Flask API including send_static_file() function
http://flask.pocoo.org/docs/api/

Note: From now on, you need to open client.html via the Flask web server and its specified URL.

Step 1: Sending asynchronous HTTP requests by using XMLHttpRequest object

In this step you will replace all the calls to "serverstub.js", at client-side implemented in lab 1, with HTTP calls to counterpart server functions implemented in lab 2. For sending asynchronous HTTP requests to the server, you shall use XMLHttpRequest object provided by your browser built-in.

More information about XMLHttpRequest object
http://www.w3schools.com/ajax/ajax_xmlhttprequest_send.asp

W3C tutorial on sending HTTP requests using XMLHttpRequest
http://www.w3schools.com/ajax/ajax_xmlhttprequest_create.asp

Note: Unlike the functions provided by "serverstub.js", all the server functions return data in text-based JSON format which need to be parsed to Javascript objects at client-side.

More information about JSON.parse()

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/JSON/parse

Step 2: Changing to Gevent's WSGI Server

So far you have been using Flask development web server. In this step you will replace it with Gevent's WSGI Server. Web Server Gateway Interface is a specification between web servers and frameworks for Python language.

More information about Gevent and some other WSGI servers.
http://flask.pocoo.org/docs/deploying/wsgi-standalone/

Note: runserver.py is changed based on your new Gevent's WSGI server.

Step 3: Two way asynchronous communication using WebSocket protocol

In this step we are going to establish a two way communication between the server-side and client-side of the web application in a way that the server can send data back to the client without the client requested it. In this step your are required to implement the auto-logout functionality which works based on the idea that a user can only be logged in from one browser at the same time. if a user id tries to login from some other browser, it can be a private window as well, the first login is automatically logged out and the welcome view is displayed instead. The two way communication can be achieved by using WebSocket API available built-in in your web browser as part of HTML5 and Gevent-websocket module available at server-side.

More information about WebSocket protocol
http://www.websocket.org/

Writing WebSocket client applications
https://developer.mozilla.org/en-US/docs/WebSockets/Writing_WebSocket_client_applications

More information about Gevent-websocket
https://pypi.python.org/pypi/gevent-websocket/

4. Questions for consideration

1. XML is also used as another widely used data exchange format. please have a comparison between the two and pinpoint the differences and similarities. Would you still use JSON over XML or not?

2. Is it possible to have two way communication without using WebSocket protocol? Please elaborate your answer.

3. What is REST architectural style? Is our Twidder web application based on REST architecture? please elaborate your answer.

4. What does web application deployment mean? What pieces of information do you think a web server needs to run a web application?

5. Please mention and explain three real world functionalities which require two way client-server communication to be implemented. Is it possible to implement them without two way communication? how?

5. Presentation and Evaluation

Once you are finished with lab 3, you will present your work to your responsible lab assistant during a scheduled lab session. You may be asked about the details of your implementation individually.