



Lab 03

Asynchronous Client-Server Communication

TDDD97/732A56

Web Programming

Department of Computer Science

Linköping University

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 *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 sessions 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 are used for implementing one way and two way communications between the client and server.
- JSON shall be used as data exchange format.

Project folder

In step 0, you re-lay out the project folder according to the *Flask* recommended layout pattern.

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()` available in the `Flask` object.

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

Information about project folder layout

<https://flask.palletsprojects.com/en/1.1.x/patterns/packages/>

Information about Flask API including the `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 asynchronous HTTP calls to counterpart server functions implemented in lab 2.

NOTE

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

More information about XMLHttpRequest object

https://www.w3schools.com/xml/dom_http.asp

W3C tutorial on sending HTTP requests using XMLHttpRequest

http://www.w3schools.com/xml/xml_http.asp

More information about `JSON.parse()`

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

Step 2: Changing to an external Server

So far you have been using the Flask development web server. In this step you will replace it with the following external alternative which is a real-world production server.

Gunicorn

<https://flask.palletsprojects.com/en/1.1.x/deploying/wsgi-standalone/>

NOTE

Gunicorn workers do not share memory which means the global dictionary which you use for implementing websocket mapping may not be accessible from different requests. Please note that each worker is a separate process which will have the specified number of threads in the command. You can do one of the following:

1. Use only one worker(enough for the course but not in reality when you have better hardware and higher number of users). You still need multiple threads, for example 100(**recommended**).

```
gunicorn -b 0.0.0.0:5000 --workers 1 --threads 100 server:app
```

2. OR use a manager as following to share memory among workers/processes(**advanced**).

<https://medium.com/@jgleeee/sharing-data-across-workers-in-a-gunicorn-flask-application-2ad698591875>

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 requesting it. In this step you 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 flask-sock module available at server-side.

NOTE

The websocket connection shall be only established once the user logs in. If the page is refreshed while the user is logged in then the websocket connection needs to be re-established.

NOTE

The logout notification to the older client shall not be broadcasted and only be sent to the target client.

NOTE

Tokens shall be used for identifying the user requesting to create a websocket connection.

Websocket, flask-sockets, support for Flask 2.X

<https://blog.miguelgrinberg.com/post/add-a-websocket-route-to-your-flask-2-x-application>

More information about WebSocket protocol

<http://www.websocket.org/>

Writing WebSocket client applications

https://developer.mozilla.org/en-US/docs/Web/API/WebSockets_API/Writing_WebSocket_client_applications

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?

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.