

Lab 2

Server-side Development using Python and SQL

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 second lab, you will use Python and SQL to implement the server-side of the final web application. You will use the Flask micro-framework, and the SQLite DBMS to handle web requests and data storage respectively.

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 2, please check section 5: Presentation and Evaluation. For more general information about the presentation and examination process, please check the course page.

Requirements

Functional:

By the end of lab 2, the following functional requirements shall be met:

- All server functions specified in the step 1 shall work exactly as specified.

Non-Functional:

By the end of lab 2, the following non-functional requirements shall be met:

- The server shall store all sensitive user data in a secure way. Passwords must be hashed.
- The server shall use an SQLite database to store all user data.
- All database calls shall be made in a Python module separate from the main server.
- The server shall use appropriate HTTP methods for all routes.
- All of the server functions shall be tested using telnet.

2. The Project Folder

You need to create a new project folder with an arbitrary name. Your project folder shall at least contain the following files by the end of lab 2:

1. server.py
2. database_helper.py
3. database.db
4. database.schema

server.py

This file shall contain all the server side remote procedures, implemented using Python language.

database_helper.py

This file will contain all the functions that access and control the database and shall contain some SQL scripts. This file will be used by the server to access the database.

database.db

This is a SQLite file which will contain your database. Your database is composed of different tables which in turn contain the actual data, such as users' personal information.

database.schema

This file shall contain the SQL script used to initialize the database. database_helper.py or SQLite3 front-end will use this file to create all the tables and insert the default data. This file should be completed and executed before implementing and running any of the server side procedures.

Note: depending on your implementation, you can accommodate more source files in your project folder.

3. Development tools

Like lab 1 you are free to use any text/code editor of your choice. You will also need the following tools to complete lab 2.

Python

In order to be able to install the packages you need to complete this lab you will need to create a virtual environment for Python, in which you can install additional modules. In this lab you are required to use python 2.7.

You can verify which version of Python you are running by executing the following command in a terminal:

```
python -V
```

The systems located in IDA computer halls run Python 2.6 by default, so you'll need to change it to python 2.7 by executing the following command in a terminal:

```
module add prog/python/2.7.3
```

For more info about IDAs module system, see the following documentation:

<http://www.ida.liu.se/local/students/initfiles/modules.en.shtml>.

Once you made sure that you are running the correct version of Python, you can proceed to create a new directory for the virtual environment. Python virtual environments are created using the command line tool `virtualenv`, which should already be installed if you are using the correct version of Python. You can learn how to use `virtualenv` in the documentation found at their website:

<http://www.virtualenv.org/en/latest/virtualenv.html#usage>

Use `virtualenv` to create a new virtual environment in a new directory. `virtualenv` will install a local version of Python found in "`<directory>/bin`".

`virtualenv` will also install `pip`, a tool for managing and installing Python packages. The documentation for `pip` can be found at <http://www.pip-installer.org/en/latest/usage.html#usage>.

You'll need to use `pip` in order to install Flask.

Once you are done you should have done the following:

- Verified that you're using Python 2.7
- Created a virtual environment

Note: From this point you will work with the executables available in the virtual environment.

Python language documentation

<http://docs.python.org/2/>

Flask

Flask is a lightweight web framework written in Python language. One of the good features of Flask framework is that it provides a built-in development web server and debugger. By using Flask you can write your back-end code in Python language and run it using the built-in development web server. Flask can be added as a module to your already created Python virtual environment using the `pip` command. The following links provide more information about Flask framework and how to install it.

Flask official website

<http://flask.pocoo.org/>

Information about Flask installation

<http://flask.pocoo.org/docs/installation/>

Once you are done you can use `pip freeze` command to verify that Flask has been installed successfully. Make sure that you always run your application using the executables in the virtual environment. If you use the global Python executable, any packages installed using `pip` will not be available.

SQLite

SQLite is a relational database management system which is now widely used for client-side and server-side purposes. It does not provide a separate process and lets you store your data directly in an individual file on disk using SQL language. the version 3 of SQLite is available on the lab systems and accessible via the command `sqlite3`. you can get information about how to use the SQLite3 front-end by executing `man sqlite3` on the lab systems.

SQLite official website

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

Information about SQL language

<http://www.w3schools.com/sql/default.asp>

Information about using SQLite3 in python programs

<http://docs.python.org/2/library/sqlite3.html>

Information about using SQLite3 with Flask framework

<http://flask.pocoo.org/docs/patterns/sqlite3/>

Telnet client

Telnet is a network protocol for bidirectional text exchange over network. By Using telnet client you can get connected to a web server and send http requests to it. Once your request is processed by the server, it returns a text-based http response to the telnet client which is displayed right after. The Telnet client is installed on our lab systems and your can access it by using the telnet command. you can get information about how to use the Telnet client by executing man telnet on the lab systems.

HTTP basics

In this lab, you are about to write the server-side functionalities for your Twidder application. This means you need to understand the basics of HTTP, the protocol used to transfer data in Web. There are a huge amount of resources available on-line, we suggest taking a look at the following link:

http://www3.ntu.edu.sg/home/ehchua/programming/webprogramming/HTTP_Basics.html.

Make sure you read especially the parts about the POST and GET methods, to be able to use each of them in place. It is important to have a good understanding about what an HTTP header is and what it contains.

3. Lab Instructions

The lab has been divided into the following three steps. After completing each server function, you are required to test it using telnet.

Step 0: Hello Flask!

In this first step you are required to install the Flask framework on your system according to the instructions available in section 2. Once it is done, you will write your first Flask application and test it by using the Telnet client according to instructions and references provided in section 2. For this step write a simple application that just returns the text “Hello World!”. An example of how to do this can be found in the documentation provided for Flask framework.

Step 1: Implementing the Twidder backend using Python and SQL

In this step you shall implement all the functions needed by your front-end application from lab 1. The server functions you create here are based on the functions available in “serverstub.js” provided in the previous lab. You will need to create an accessible route for each function, so it will be accessible by the client.

A list of the functions you shall implement, and a small description for each function:

The list is formatted as this:

```
function_name(arguments)
```

Description: A small description of what the function is intended to do.

Input: A description of what input the server is expecting.

Returned data: Either a description of what data the server should return, or a “-” if no special data is to be returned.

```
sign_in(email, password)
```

Description: Authenticates the username by the provided password.

Input: Two string values representing the username (email address) and password.

Returned data: A string value containing a randomly generated access token if the authentication is successful.

```
sign_up(email, password, firstname, familyname, gender, city, country)
```

Description: Registers a user in the database.

Input: Seven string values representing the following: email, password, firstname, familyname, gender, city and country.

Returned data: -

```
sign_out(token)
```

Description: Signs out a user from the system.

Input: A string containing the access token of the user requesting to sign out.

Returned data: -

```
change_password(token, old_password, new_password)
```

Description: Changes the password of the current user to a new one.

Input:

- token: A string containing the access token of the current user
- oldPassword: The old password of the current user
- newPassword: The new password

Returned data: -

```
get_user_data_by_token(token)
```

Description: Retrieves the stored data for the user whom the passed token is issued for. The currently signed in user can use this method to retrieve all its own information from the server.

Input: A string containing the access token of the current user.

Returned data: A user object containing the following fields:
email, firstname, familyname, gender, city and country.

```
get_user_data_by_email(token, email)
```

Description: Retrieves the stored data for the user specified by the passed email address.

Input:

- token: A string containing the access token of the current user
- email: The email address of the user to retrieve data for

Returned data: A user object containing the following fields:
email, firstname, familyname, gender, city and country.

```
get_user_messages_by_token(token)
```

Description: Retrieves the stored messages for the user whom the passed token is issued for. The currently signed in user can use this method to retrieve all its own messages from the server.

Input: A string containing the access token of the current user.

Returned data: An array containing all messages sent to the user.


```
get_user_messages_by_email(token, email)
```

Description: Retrieves the stored messages for the user specified by the passed email address.

Input:

- token: A string containing the access token of the current user
- email: The email address of the user to retrieve messages for

Returned data: An array containing all messages sent to the user.

```
post_message(token, message, email)
```

Description: Tries to post a message to the wall of the user specified by the email address.

Input:

- token: A string containing the access token of the current user
- message: The message to post
- email: The email address of the recipient

Returned data: -

Each function, when invoked through its route, is required to return a text string in a human readable format containing the following information:

1. A true or false value indicating if the function has been executed successfully or not.
2. A message describing the success or failure.
3. The requested return data for each procedure.

Note: You can also use error codes instead of true/false values. Remember, if you do you will need to upgrade your client side code by the end of lab 3.

One of the important tasks of the backend is to store information more or less permanently. In this step you need to use SQLite3/SQL and python data structures to store certain information at server side, for example information about signed up or signed in users. It's up to you to decide how to store each type of data but you need to defend your choice during presentation. If you are in doubt, discuss your choice of implementation with your lab assistant.

As you proceed, you need to test implemented procedures using the Telnet client. This tool gives you the possibility to call serverside procedures with arbitrary arguments and investigate the output.

Step 2: Final test using Telnet

In this final step, you shall simulate the following scenario by calling appropriate server side procedures via Telnet:

1. Sign up user A.
2. Sign up user B.
3. Sign in as user A.
4. Post a message on user B's wall.
5. Check all messages posted to user B's wall.
6. Sign out user A.
7. Sign in as user B.
8. Post a message on the signedin user's wall.
9. Check the messages belonging to the signed in user's wall.
10. Sign out user B.
11. In the terminal running the server, terminate it and restart it.
12. Sign in as user A again.

The reason for doing this is to test that your backend works as a whole and isn't just a collection of functions which cannot work together. During your presentation, you are required to repeat these steps.

4. Questions for consideration

1. As part of requirements you have been asked not to store the users' passwords in plain text. What security risks such action can cause? How can this problem be address programmatically?
2. As http requests and responses are text-based, information, can be easily intercepted and read by a third-party on the Internet. Please explain how this problem has been solved in real-world scenarios.
3. How can we use Flask for implementing multi-page web applications? Please explain how Flask templates can help on the way?
4. Please describe a Database Management System. How SQLite is different from other DBMSs?
5. Do you think the Telnet client is a good tool for testing server-side procedures? What are its possibles shortages?

5. Presentation and Evaluation

Once you are finished with lab 2, you will present your work to your responsible lab assistant during a scheduled lab session. The result of your work is evaluated based on what you have been required to do and how you have achieved it. You are also required to pick up three questions from the mentioned questions in Section 6 and bring a hard copy of your answers to the presentation session. Each answer must be between 100 and 150 words.