Lab week 8 (ISYS2120 sem2 2024)

Welcome to week 8's lab. This week the lab lasts for **two hours** (and therefore, there is no private meeting of the group with tutor). We are going to focus on building your practical skills for applications that access a database, in particular, for a web-application written in Python with the flask library. In the lab you will need to follow a sequence of steps, which show you how to install and run the application, and also you will be looking at the code itself, to understand how the code causes the behaviour you observe. The code you use this week, is also the basis for Assignment 3, so it is really important to make sure that you have both practical skill and also understanding of the concepts. Also, some activities use the pgadmin4 client which was covered in week 3 lab; you should refresh your recall of that before coming to this week's lab!

*Important practical note: Any work on these tasks will need to access university machines for some services (computation and/or postgresql access), and for that, you will need to be **inside** the University network. If you are not performing this from a site on campus, you will need to use the **VPN** to connect to the University network, before trying any of these activities. See https://sydneyuni.service-now.com/selfservice/?id=kb_article&sys_id=836b51e6dbeb6010ea7d0793f3961901 for information about obtaining the VPN client if you do not already have it installed. Note also that whenever you do these activities on a provided machine in a university computer lab, a few aspects will be different than when you work on a personal laptop. Finally, we warn that there may be difficulties with the software and systems used, or with these instructions. Please speak to your tutor if any difficulties arise in following the instructions, and be prepared to be flexible if the tutor asks you to do something a little different. It is hard to support this large number of students all working at the same time.*

Reminder.
Tutors will manage time in their class, aiming to cover the most crucial material and help anyone who meets technical or conceptual challenges. Should students have any additional questions or need further clarification, we strongly encourage them to utilize the ED platform for further assistance.

Before the lab
Read through the instructions for this lab! If anything is unclear, post on Ed about it.

During the lab
For week 8, there are four activities, with details below:

- Tutor presents an overview of the concepts of database-backed applications and their security issues,
- Logging in, obtaining files, etc
- Setting up (and understanding) the data in the database server, using pgadmin4 client
- Installing the application on an application server, seeing it run, and exploring its code.

The instructions are given step-by-step, and you should try to follow them that way. You need to be working logged in on some client machine (either a lab desktop, or your own laptop, etc)

A. Tutor presents an overview
The tutor will give a quick refresh of some key aspects of the material needed for this lab, and in particular, they will try to fix any gaps or misunderstanding they saw commonly in the hand-written summaries.

B. Login and getting files

Log in to the machine you will be working on (if this is a lab machine, you login with your unikey and its corresponding password; if you choose to work on a personal laptop in the lab room, you will likely already be logged in to your laptop!). Download the materials needed for this week's (wk08) tutorial from Canvas. The materials are in two zip (compressed) archives, called isys2120_airline_data.zip and isys2120_airline_webapp.zip

The database schema and contents are in isys2120_airline_data.zip file. Extract/uncompress this file; you should get a folder which contains some sql files.

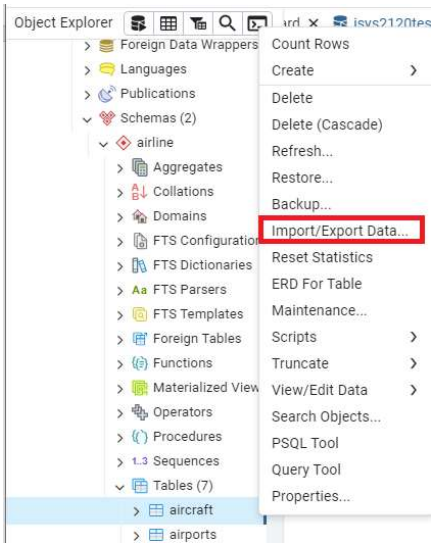
The webapp component is in isys2120_airline_webapp.zip. Unzip it into a folder of your choosing, make sure you know where it is located as it will be used for Assignment 3.

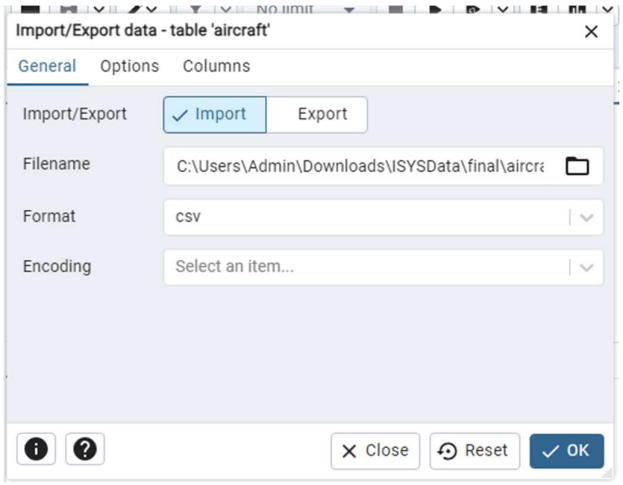
C. Interactive queries through pgadmin4

You should already have pgadmin4 set up in Lab 3. Use this to connect to your postgresql database (recall that this is on the sever whose hostname is **awsprddbs4836.shared.sydney.edu.au** and you have an account there with user name and password that was sent in email early in semester – these are different from your unikey etc). Use pgadmin4 to load the file called 01_schema.sql (which you found inside the extracted database zip file) into the query editor window, by clicking on the  icon and run it with the  button.

After running 01_schema.sql, right click on your database in the left pane and click on "Refresh" in the context menu. The airline schema should now appear. Next we'll import the template data into these tables, by navigating to the **airline** schema, then "Tables (7)", then for every table except for the users and userroles table, right click on that table and select "Import/Export Data...".

You should be presented with the following dialog:





After you have done this for the aircraft, airports, flights, passengers and tickets tables, all the data has now been imported into your database.

Now clear the contents of the query editor, and type SELECT * FROM users; Now run this query, and see the 2 rows appear in the output window.
You can experiment with other queries, to explore the data template, depending on the time you have, but this should feel very similar to working with Ed lessons.

D. Running a web app

For this activity, we will focus on the webapp component, which you extracted from isys2120_airline_webapp.zip This code when run, acts as a web server, responding to HTTP messages which can come from anywhere; to determine the correct response, the web server interacts with the data held in the postgresql database on the database server).

These instructions are for what we expect to be the most common situation: you will run the application code on a machine the university provides. To edit and inspect that code, one way is to use git so that you can do edits and read the files on your local machine, and keep this synched with the code on the web server machine. That is what we describe here.

There are several other ways to deploy and work with the webapp. In Appendix 1, we describe how you can look at and edit the code on the University-provided appserver machine, but without using git to keep things synched with local changes, Instead, the same webapp server machine can also support working with the code as a Jupyter notebook, accessed through a browser. In Appendix 2, we discuss using the desktop machine in the lab as the webapp server, since you can work there directly when in the lab.

To do the Assignment 3, you can do any of these, or if you wish, you can use your own laptop to run the app server as well as any Python development tools, and

also the browser that accesses the webapp! Whatever you do, it is crucial that you keep track of which machine is doing what parts of the work, and where you are logged in, etc.

You will be using git so you can work on the code locally on your machine (lab desktop or your own laptop), while the code is available to run on the app server (however if there are technical challenges with this, an alternative approach is available which your tutor can guide you on). For Windows users, please install git bash through here: https://gitforwindows.org/ For Mac/Linux/WSL users, git should already be installed in your environment.

Now, open a command window (Powershell or CMD for Windows, Terminal for Mac or XTerm for Linux or similar) and navigate to the folder you chose earlier and init the git repository.

```
cd /path/to/your/extracted/files
git init .
```

Next, connect to the app server machine at **soitpa10005.shared.sydney.edu.au** using SSH, and your unikey and password

```
ssh YOUR_UNIKEY@soitpa10005.shared.sydney.edu.au
```

We will set up a remote repository on this server as well as a read-only copy of the repository files. Run the following commands:

```
mkdir airline_webapp.git
mkdir airline_webapp
cd airline_webapp.git
git init . --bare
echo -e '#!/bin/bash\nGIT_WORK_TREE=~/airline_webapp git checkout -f' >
hooks/post-receive
chmod ug+x hooks/post-receive
```

You can now disconnect from the remote server using the “logout” command.

```
logout
```

Back on your local machine, add this as a remote:

```
git remote add \
origin YOUR_UNIKEY@soitpa10005.shared.sydney.edu.au:~/airline_webapp.git
```

And make an initial commit and push:

```
git add .
git commit -m "Initial commit"
git push --set-upstream origin master
```

You should now have a local working copy of the webapp template set up, as well as a remote mirror with a **read-only** copy of the template.

Connect again to the server, and navigate to the copy of the template and run the webapp with the following commands:

```
cd airline_webapp
python3 web_app.py
```

Instructions should now appear in your console output, indicating that you need to set your database account, as well as your port for the webapp.

Navigate to your local copy of the code, and open up “config.ini”. Set your database username and password as in Lab 3, also set the Flask port on the bottom to “10” followed by the last 3 digits of your SID (we will refer to this value as YOURPORT below)

We will now sync your changes to the server. Run the following commands on your local machine:

```
git add .
git commit -m "Change config"
git push
```

These commands will be what you run to push your changes to the server. Feel free to choose a meaningful commit message in your future pushes.

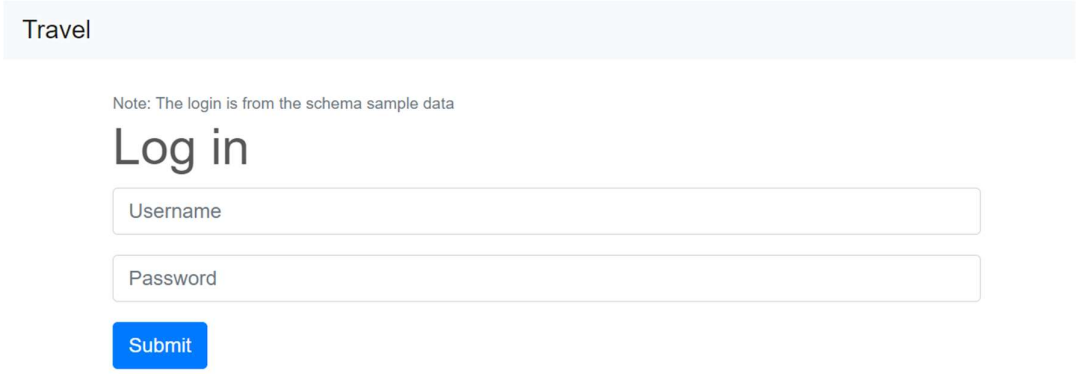
Try running the webserver again, you should now be able to access the webapp.

```
python3 web_app.py
```

You can open http://soitpa10005.shared.sydney.edu.au:YOURPORT on a web browser to browse the GUI.
To halt the server, just press Ctrl+C to terminate the process in PuTTY or the ssh terminal [It is crucial that you do this before ending the connection]

Note that you can also run the webapp locally by running the same app on your local machine.

You should now see the following page:



To login to the webapp, you need to pretend to be one of the application accounts, which are given by the “users” table.

First, let’s log in as a normal user (login details user/password). This will get you a “Welcome <name>” page, with a bar at the top with functionality to execute.

Now, click on the View/Manage Users menu on the top bar, then List users. You

should be able to see a list of all the users.

Take a look at the address bar for the page, and open routes.py. Navigate to the list_users() function on line 107. Have a look at how the simple template is rendered by inspecting the code as well as by looking at the file list_users.html in the templates folder.

Open up the file database.py and look at the list_users() function there. What SQL query is being executed here? Try running the same query on pgadmin4, and check that you get the same answer.

Navigate to the address bar of your brower, and add '/user' to the end of the address. Press enter, and you should be at the same page with the only user being displayed being John. Try a few times with different usernames. Repeat what we did before for the Users functionality and work out how the page is being served by the webapp. Specifically, find which function is being called in routes.py and which in database.py, and also which template is being rendered. After that, try running the same query on pgadmin4. Does the data in the webapp display match the output on pgadmin4?

Move on now to the User Stats button on the top bar. Again, try to work out how this functionality works.

Log out of the web application (do not log out of the accounts on the server, or on the lab machine/.laptop!) and log back in to the web app as an admin (login details admin/admin). You should see a different interface now that you are an admin. Navigate to the dropdown and try clicking on the 'Add User' link, and create a user. Now, log out of the web app and try logging in as the new user you just created. Query the 'users' table in pgadmin4, and see how it has changed, now that you created this user. Log out again from the web app, and log back in as an admin again.

Now, try using the update functionality to change the name of the user you just created. List all the users and click on the edit button. Check on pgadmin4 that this had changed the table. Have a look at the terminal the webapp is running on. What's the output on the terminal?

Next, delete the new user you created. Again, check which SQL query was executed, and how it was achieved through a template, routes.py and database.py.

Try the user stats functionality next. Execute the same query through pgadmin4.

After that, try searching for the user 'admin' through the search box. Again, trace through the template html file, routes.py and database.py to see how the functionality was achieved.

Recall what we have discussed in class regarding security, in particular SQL injection. Try typing a single apostrophe into any textbox you find. Does it cause an error? Try this with other symbols on your keyboard. How does this represent the possibility or absence of a SQL injection? Why do you think SQL injection is possible/prevented in the code? Make sure to take a close look at the functions in database.py, especially the

dictfetchone function.

If you haven't already, now look at the Search Field textbox in the User Search page. Does the possibility of SQL injection exist here? (Advanced) Try to craft a SQL injection that will reveal the passwords of all users, while logged in, but not as an admin.

Answer (copy and paste to reveal): [redacted]

Now, take a few minutes to think about how you can change the code and template to prevent this SQL injection.

Fred Foolish suggests that changing the input box from a text box to a drop down in the html will solve the SQL injection bug. Why is this not enough to defend against Malicious Mallory?

Now, try to come up with an alternative defence against this SQL attack. (Multiple answers are possible here)

DO NOT LEAVE THIS LAB without ending the flask processing on the app server (type Control+C in the ssh or PuTTY terminal.) You can then close the ssh or PuTTY terminal itself.

After the lab session

- Before the end of **Friday** (September 20) look over the week 9 lecture slides on Canvas, and use them to prepare your hand-written half-page summary9 and upload it to Canvas
- Before the end of **Sunday** September 22, complete SQL Tasks 12, 13, 14
- Look at the Assignment 3 ibnstructions, and get together with your group (the people you have been meeting with when you have 20 minutes with tutor; this usually combine stwo smaller subgroups doing Asst2)
- Prepare yourself for the week 9 lab, when you will do a SQL Online Quiz which is worth 5% of the unit. You can try a sample quiz on Ed. This lab is also 2 hours long.

Appendix 1. Running on Jupyter Lab on the webapp server machine

If you are having difficulties or technical issues with git, you can instead follow the below alternative instructions to use Jupyter Lab instead to work with your code on the app server machine.

We will be using a Jupyter notebook server to run a web application from Python code. You can connect to the server via a web browser: open the URL
https://soitpa10005.shared.sydney.edu.au:10000/hub/login

You will be prompted with a Jupyter login screen. Use your normal unikey and password for this service.

Use the main page of the Jupyter server to "upload" the compressed isys2120_airline_webapp.zip

You then need to login to the machine soitpa10005.shared.sydney.edu.au (it's the Jupyter server, but we will use it for editing and then running the web app too). Use a terminal or command window on your device or lab machine, to execute ssh abcd1234@ soitpa10005.shared.sydney.edu.au (replace abcd1234 by your own Unikey). At the prompt for password, give your Unikey password. Alternatively, use PuTTY to connect to soitpa10005.shared.sydney.edu.au on port 22.

Once logged in to your account on the server, use "ls" (to list contents of the directory you are in) and "cd" commands (to change to a subdirectory) to get to the place where you put the file isys2120_airline_webapp.zip
Now uncompress this, by running unzip isys2120_airline_webapp.zip. That will create a directory with all the sql and python contents (just like you found on your lab machine or laptop in A above).
You should now see a directory when you look with the Jupyter browser. Here, you now need to edit the 'config.ini' file. While you can use the Jupyter browser to edit any '.py' file by clicking on it, for other files, you must click on the tick box first, then select 'Edit' from the menu options. In editing the config.ini file, set username and password for the postgresql account as sent in email, and for port, use 10 followed by the last 3 digits of your student id. Then click 'File'->'Save'

Now you will move back to your still open shell/terminal. Navigate to the 'code' folder. Do 'ls' to see the same files as listed in the Jupyter browser.
To begin your flask application, you should run 'python3 web_app.py' [*Very important*: do this in the ssh or PuTTY terminal; do not try this from Jupyter interface]

When you run the command, the Flask webserver goes online on whichever port you specified (that is, port 10-followed-by-last-3-digits-of-your-sid).
You can open http://soitpa10005.shared.sydney.edu.au:YOURPORT on a web browser to browse the GUI.
To halt the server, just press Ctrl+C to terminate the process in PuTTY or the ssh terminal [It is crucial that you do this before ending the connection]

Appendix 2. Running the webapp code on Lab Computers

If the app server machine is currently unavailable to you, and you are in a lab on campus, you can use the university lab desktop computer instead. From the login screen, select the restart option. Wait for the computer to boot up to the boot manager and select the linux option. You can then log in with your unikey and password as per normal.

Now, you can follow the procedure as per normal, except you will have to use the editors available on the lab machine instead of the editor built into Jupyter Lab.

Furthermore, you should connect to http://localhost:YOUR_PORT instead of the ucpu link, once you have the server running.

Appendix 3. Extra resources

Flask tutorial
https://www.youtube.com/playlist?list=PL-osiE80TeTs4UjLw5MM6OjgkjFeUxCYH

DB-API tutorial
https://youtu.be/DlNIXC9SaF4?si=JQzN1E79YaS-qOt7&t=599