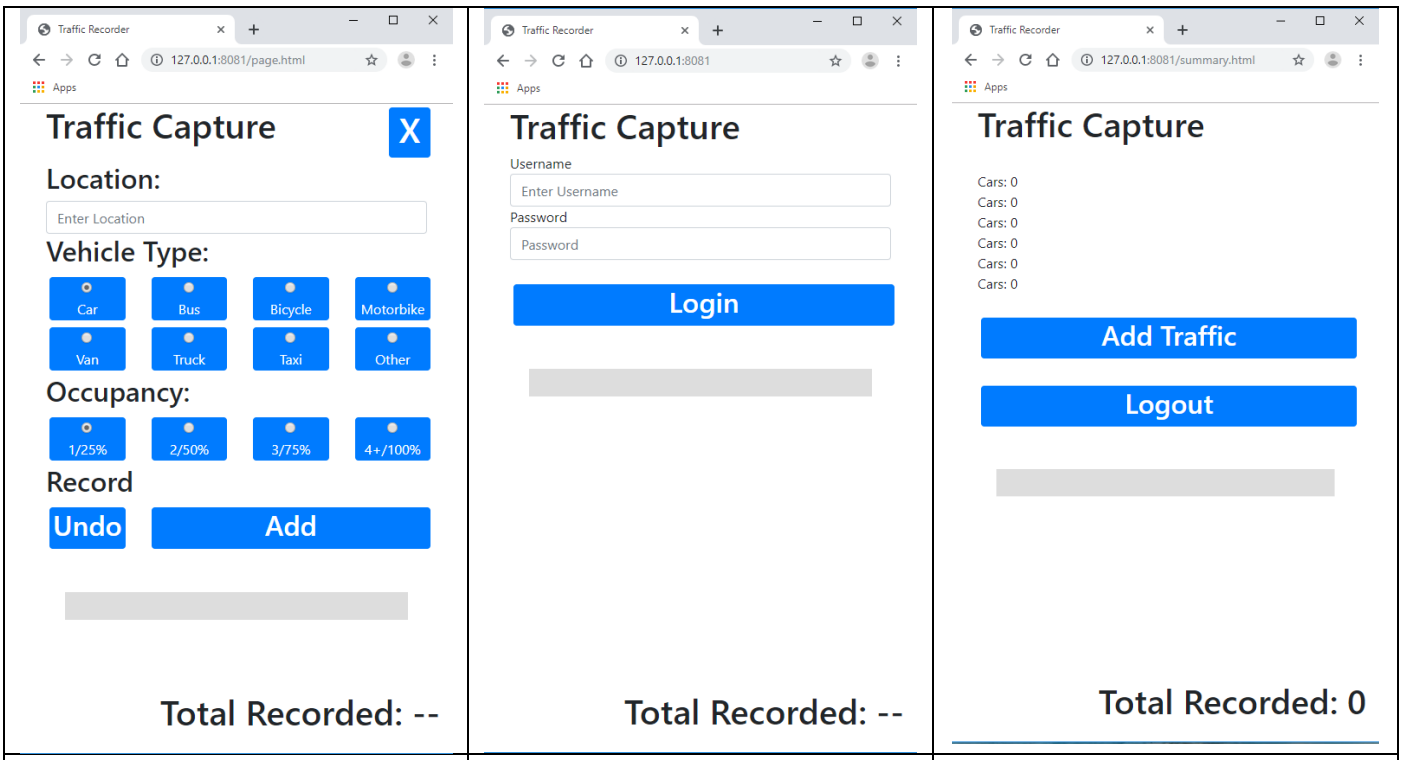
CM50267 Software Technologies for Data Science

Project: Traffic Capture Application

+ Lab 6: Code Review/Testing

Specification



page.html	Index.html	summary.html

Figure 1: Web App Interface Pages.

Project Deadline: 4th December 2020, 8pm.
Lab 6 Deadline: 13th December 2020, 8pm.
Submit online via Moodle.

Project: Traffic Recording Application

Introduction

In this coursework you will undertake the implementation of a simple traffic capturing application. The application is a web-based system that consists of two parts. The web server written in python and a browser based client that presents an interface.

You are provided with:

1. A complete browser based front-end for the app that makes use of HTML, CSS and Javascript. You will not need to modify this code. Or to understand how it operates, other than how it interacts with the server. This is described in the section Web Application Structure and will be covered in a lecture.

2. A skeleton backend framework for the app written in Python that provides the core web server functionality.

You are required to:

1. Create a suitable sqlite3 database to hold all the required data.

2. Extend the skeleton code to complete the app functionality by adding code as required. Where it is expected you will extend/modify the code is indicated by the use of '##' in comments. You may alter the code elsewhere if you wish.

3. Create two additional programs able to extract summarised data from the database as CSV files. Along with two support programs to allow these to be tested.

The coursework has been broken into a series of tasks. Marks have been allocated to each task. This document describes the behaviour required of the app and the support programs. Tasks will be assessed against this specification. Partial marks may be awarded for functionality present in incomplete tasks. You will use python for program development and using SQL to interact with an sqlite database. You should write the SQL queries yourself and not rely on a library such as pandas to abstract it for you.

Web Application Structure

This web application requires the python based webserver to be running and for this to be accessed by a browser. Your task is to develop the skeleton server into a functional capture application. You should run the server by executing **python server.py** from the command prompt while in the directory containing the support files. Replace **python** with whatever command is required on the system you have chosen to use for development. You are advised not to use Jupyter Notebook for this process. Start a web browser. Chrome and Edge have been demonstrated to work, most other modern browsers should also function. Access *http://127.0.0.1:8081/* using the browser. This will access the locally running server and display the index.html page that allows a user to login.

Where the browser requests html, css or javascript files, these will be returned by the existing code.

When the requested file is *'/action'* the parameter *'command'* is examined and if it is valid, an appropriate handler function is invoked. Where you are expected to modify the code of these handler functions.

The following commands are supported:

1. login — Expects username and password, validates these to generate a session token (magic).

2. add — Expects location, type and occupancy and records this in the traffic database.

   Location is a free form string.

Type is one of the predefined vehicle types ‘car’, ‘bus’, ‘bicycle’, ‘motorbike’, ‘van’, ‘truck’, ‘taxi’ and other.

Occupancy is an integer in the range 1-4. And either represents the number of occupants of a vehicle as a number for everything except a bus and as an approximate percentage for a bus.

3. undo — Expects the same as add and is used to correct mistakes by removing a matching entry.

   A matching entry is one where the location, type and occupancy are the same for an entry that is part of the same login session.

4. back — Decides if the in app back button should go to the login page or summary page.

   No parameters. If a user is logged in then the user should be redirected to the summary page, otherwise they should be redirected to the login page.

5. summary — Provides the summary statistics. No parameters.

6. logout — Logs the user out of the current session, ending it. No parameters.

Note than in addition to the parameters required by the listed commands, all *‘/action’* requests with have a *randn* parameter. This is used to prevent unwanted browser caching behaviour and need not be considered further. You can ignore it.

On completion of the action request, the server must provide an XML formatted response that contains the outer tags *<response>…</response>* that contains one or more of the following *<action>…</action>* entries.

Action: refill

<action>
<type>refill</type>
<where>…</where>
<what>…</what>
</action>

The *refill* action response allows the server to supply text that should be placed in various places within the current page. The location is identified by *<where>target</where>*. The skeleton server demonstrates the use of these by updating the available fields with placeholders. Location *message* is the gray box shown in the pages of Figure 1. *total* is the number shown at the base of each page. And the entries *sum_ ** are the various count values shown on the summary page.

The content should be base64 encoded to avoid character escaping issues and occupies the *<what>…</what>* field.

Action: Redirect

<action>
<type>redirect</type>
<where>…</where>
</action>

The *redirect* action response indicates the client should load the page specified by the *<where>…</where>* entry. It is used for the *back* command and where a session ends or is invalid.

Helper functions are provided to generate these actions and their use is demonstrated in the skeleton code, including the use of base64 encoding.

Multiple Users

This application is a prototype. A production version would use a more complete implementation of SQL to provide the database. Your application will need to support multiple users being logged in simultaneously, but you may assume that individual requests to your server will be serialised and you do not need to address mutually exclusive access within your SQL requests.

Tasks

The marks allocated to each task are shown. The task description provides the expected behaviour the software must have to achieve those marks. Partial credit within a task may be awarded.

Task 1 Setup (2 Marks)

Create an SQL database using *sqlite* that will be used to hold all data needed by the application. No data should be hardcoded into the application or held between server requests within the application. The sole exception to this is the vehicle type and occupancy level definitions. You are advised to consider all the tasks in designing the database as later tasks may require columns in tables initially created for the earlier tasks and for these to be filled in as part of the behaviour of the earlier task.

Task 2 Parameter Handling (1 Marks)

Validating that all parts of an incoming request are present is an important first step in a web application backend. For example, currently if a user leaves the username or password fields blank, an error will occur within the Python program as the parameter variable in which the incoming values are stored is not populated when the input is blank. For this task you will need to add appropriate handling for this case. As an example of what is required, the *parameters['command']* handling already deals with the same issue.

You also need to defend against deliberate or accidentally bad inputs. What is known as an 'injection' attack. This applies to all of the tasks and you risk not getting all the marks for task 3 thru 7 if they do not defend themselves against bad inputs. Note that the scripted tests allow input to be generated that is not in the form the front-end client may usually generate. This is part of what you need to defend against.

Task 3 User Login (4 Marks)

In the skeleton code, only one user 'test' is supported and they can login with any non-empty password. This is dealt with in *handle_login_request()*. Add an appropriate table in the database that includes the entries: username and password. Both should be strings. The password should be hashed for security rather than stored in plaintext. You may find hashlib a useful python module for this task. Pre-populate the database with 10 users (test1, test2,…) with corresponding passwords (password1,password2,…). Allow a user to be logged in only once at any given time. An attempt to login while already logged in should be rejected with a suitable error message. A magic session token should be generated that will be passed to the client via cookies and will be returned with each /action request. This will be used to validate access to other actions. A reminder that in this and other tasks you should defend against malicious inputs. Multiple users may be logged in at once.

Task 4 User Logout (1 Marks)

When a user chooses to log out, the session should be ended. A record should be kept in the database of the start and end times of each user session. Extend *handle_logout_request()* to support this. This action should result in the user being re-directed to the login page.

Task 5 Traffic Adding (4 Marks)

Extend the *handle_add_request()* function so that it records the vehicle in the database. The table used should include entries for the location, type, occupancy and time of the recording as a minimum. It should also be possible to identify which user created the record and which other records they created during the same login session. The response must include an update of 'message' and 'total', even if the input is invalid or a user is not logged in.

Task 6 Traffic Correction (4 Marks)

Extend the *handle_undo_request()* function so that it records the need to undo the vehicle in the database. The table used should include entries for the location, type, occupancy and time of the recording. It should also be possible to identify which user created the record and which other records they created during the same session. You can

choose to have add and undo records in the same or different tables. Undo entries should not cause add entries to be removed only prevent them being counted in the statistics generated. Vehicles must exist to be undone. The response must include an update of 'message' and 'total' even if the input is invalid or a user is not logged in.

Task 7 Online Summary (4 Marks)

Extend the handle_summary_request() function so that it returns the correct traffic statistics for the current session rather than the current sum_* placeholder value in the code.

Task 8 Offline Summary (4 Marks)

Create a program (separate from the web app) called task8_out.py that is able to query the database and indicated the number of each type of vehicle and occupancy at each location during a period specified by a start date and time and an end date and time. Invalid dates and times should be intentionally rejected. The output should be delivered as a csv file with a row per location with the format as described in Figure 1.

Create a second program (separate from the web app) called task8_in.py that consumes a csv file containing entries of the form described in Figure 2 and updates your database to include these traffic observation records. The updated records should be correctly reflected when you run the first program of this task. You do not need to maintain user/session details beyond what is required to make this task possible for your given database schema. They will not be extracted. You can, for example, assume that all entries were made by the same user in a single session.

The structure of the database should be the same as that used in the web app.

```
Location,Type,Occ1,Occ2,Occ3,Occ4

Location is a string containing one or more lowercase letters, the digits 0 thru 9 and spaces. [0-9a-z ]+
Type is one of car,bus,taxi,bicycle,motorbike,van,truck,other.
Occ1 is the number of that type that was recorded with occupancy 1/25%
Occ2 is the number of that type that was recorded with occupancy 2/50%
Occ3 is the number of that type that was recorded with occupancy 3/75%
Occ4 is the number of that type that was recorded with occupancy 4+/100%

For Example:
main road,car,1,0,0,0
ring road,bus,0,0,0,1
```

Figure 1. Task 8 Output CSV Format.

```
DateTime,Mode,Location,Type,Occupancy

DateTime is in the form yyyymmddhhmm.
Mode is either add or undo, this is equivalent to the corresponding user action. These will be in order and there
will be a corresponding add for each undo.
Location is a string containing one or more lowercase letters, the digits 0 thru 9 and spaces. [0-9a-z ]+
Type is one of car,bus,taxi,bicycle,motorbike,van,truck,other.
Occupany has the value 1-4

For Example
201906011243,add,main road,car,1
201906011243,add,main road,car,2
201906011244,undo,main road,car,1
201906021255,add,ring road,bus,4
```

Figure 2: Task 8 Input CSV format.

Task 9 User Hours (4 Marks)

Create a third python program (separate from the web app) called task9_out.py that is able to query the database and indicate the total hours each user was involved in counting in order to allow pay to be determined. It should allow a date to be provided in the form yyyymmdd and provide the totals for that date, the week ending on that date and the preceding month ending on that date. The output should be delivered as a csv file with a row per user of the format described in Figure 3.

Create a fourth python program (separate from the web app) called task9_in.py that is able to read a CSV file of the form described in Figure 4 and updates your database to reflect the user counting sessions specified.

The structure of the database should be the same as that used in the web app.

User,Day,Week,Month

User is one of the predefined usernames.
Day is the number of hours on the given date, with one decimal place rounded up.
Week is the number of hours in the preceding week including the date, with one decimal place rounded up.
Month is the number of hours in the preceding month including the date with one decimal place rounded up.
(A month means up to but not including the same day of the month from the preceding month.)

For Example:
test1,4.3,20.4,81.7
test1,6.5,32.4,100.7

Figure 3. Task 9 Output CSV Format.

User,DateTime,Mode

User is a one of the 10 predefined usernames
DateTime is in the form yyyymmddhhmm.
Mode is either login or logout

For Example
test1,201906011243,login
test2,201906011543,login
test1,201906011455,logout
test2,201906011800,logout

Figure 4: Task 9 Input CSV format.

Task 10 Code Quality (4 Marks)

Lint tools are designed to assess the overall quality of code. pylint performs this task for the python language. It checks things like variable naming conventions, unused variables and various other practices that can lead to developing code that contains errors. Use the pylint program to assess the standard of your code. For each point above 4 that pylint gives you, you will receive a mark, up to a limit of 4 marks.

The test will be performed as part of lab 6. You will be asked to apply it to the server.py and the task 8 and 9 code, but only the score from server.py will be used to determine the mark.

Task 11 Report (8 Marks)

Write a brief report that describes the application you have written. It should include:

Details the structure of your database.

A description of how your code manages the add and undo feature of the application.

The errors and malicious input behaviour it seeks to prevent.

It should be no longer that four pages in length.

Lab 6 Review Exercise (10 Marks)

Lab 6 is run in combination with the project. The purpose of this Lab is to give you experience in a phase of development that is frequently sacrificed to meet other priorities, testing and quality control.

When you submit your work to moodle, a copy will be made available to one of your fellow students. You will receive details of another students work. An 'Application Acceptance Test' form will be released. You must complete the tasks on that form using the code you have received and then upload the completed form to the Lab 6 Assignment on moodle prior to the deadline for the Lab.

If you complete this process, you will receive full marks for the lab. If you fail to complete the tasks you will receive partial credit for the tasks you have completed. You should also upload a test report for your own code.

The testing results will be available to the member of staff that marks your work. But they are advisory only they will not map directly to your final mark for the project tasks.

If any student's code is not tested by another student, it will be tested by the unit leader/tutors. This will result in a delay to the marking but will not impact the mark achieved.

The tests will include:

1. Running pylint on the server code.
2. Running a test script of the style found in TrafficApp_Test.ipynb. The script will have more tests and will not be released until after the project deadline.
3. Running the in and then out program for task 8 using a supplied test file and checking against an expected output.
4. Running the in and then out program for task 9 using a supplied test file and checking against an expected output.

Deliverables & Submission

You should provide the following by the project submission deadline, uploaded to moodle.

1. A code.zip containing:
   a. server.py — Your updated version of server.py
   b. task8_out.py — Your first task 8 program
   c. task8_in.py — Your second task 8 program
   d. task9_out.py — Your first task 9 program
   e. task9_in.py — Your second task 9 program
   f. initial_database.db — A sqlite3 database file that contains the tables needed by your app, with the login table pre-populated with the 10 specified users.
   g. instructions.txt — Details of how to run the task 8 and 9 programs from the command line.

   Note this is the part of your project you will send to another student and you should expect to receive. Please avoid including your username,details in any of the files uploaded.

2. A database initialising script containing SQL commands that will create the tables needed by the server and populates it with the users specified in task 3. i.e. a script that creates initial_database.db. We do not expect to run this and will not do so if you fail to provide initial_database.db

3. A brief report that describes the application you have written. Detailing the structure of the code you've added. The functionality it supports and the errors/malicious input behaviour it seeks to prevent.

For lab 6, you should upload the following to moodle by the lab deadline.

1. The completed acceptance forms.

Supplied Files

The following files are provided for the project phase:

server.py	The Python server code. This is the file to which you will add code.
index.html	The frontend main page (html). You should not need to modify this.
page.html	The frontend second page (html). You should not need to modify this.
summary.html	The frontend error page (html). You should not need to modify this.
css/*	The frontend cascading style sheet for the html pages (css). Uses bootstrap. You should not need to modify this.
js/*	The frontend javascript code makes AJAX requests to the server and process the responses. You should not need to modify this.
TrafficApp_Test.ipynb	An example test script that interacts with the application server.

KMC2020