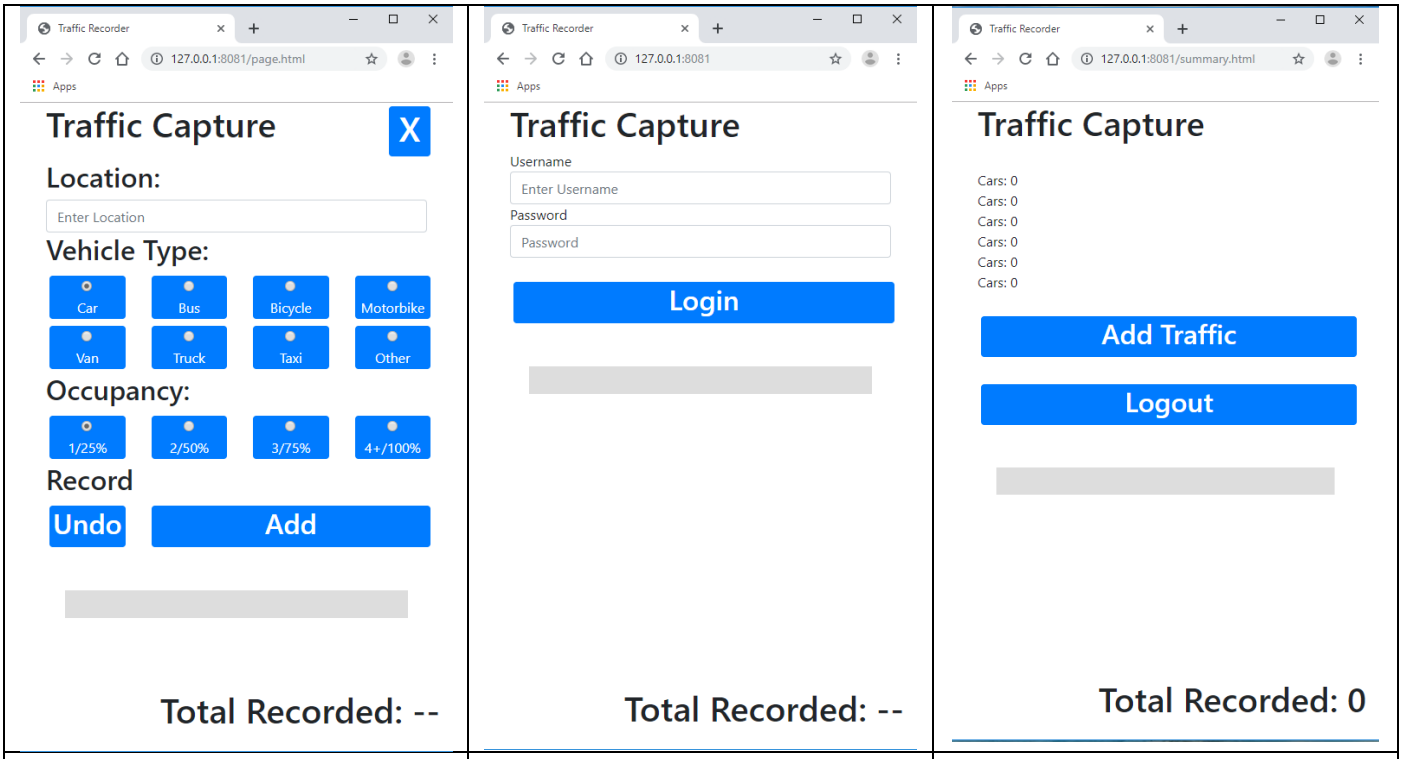
CM50267 Software Technologies for Data Science

Project: Traffic Capture Application

Specification



page.html	Index.html	summary.html

Figure 1: Web App Interface Pages.

Project Deadline: 10 December 2021, 8pm.
Fix Deadline: 17th December 2021, 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 the user 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. Nor will you need 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.

3. An initialised sqlite3 database schema.

You are required to:

1. Maintain the sqlite3 database that will 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.

This document describes the behaviour required of the app. You will be assessed against this specification. You will use python for program development and use SQL to interact with the sqlite database. You must write the SQL queries yourself and not rely on a library such as pandas to abstract it for you. The SQL database will be held in a file called traffic.db

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 *port*** 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. And ***port*** with an appropriate port number, e.g. 8081.

You are strongly 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.

When first run, the server will access the username 'test' to allow you to see the operation of the code. This hardcoded behaviour must be removed.

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. 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’ or ‘other’.

Occupancy is an integer in the range 1-4. The occupancy either represents the number of occupants of a vehicle as a number for everything except a bus (to a maximum of 4) 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 active entry that is part of the same login session.

4. back — Decides if the 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. The statistics for all vehicle types must be returned, even if they are zero. ‘0’ and not ‘zero’ should be returned.

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 JSON formatted response that contains a list of dir type responses

Action: refill

{“type”:”refill”, “where”:”…”, “what”:”…}

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*. 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 occupies the *what* entry.

Action: Redirect

{“type”:”redirect”, “where”:”…”}

The *redirect* action response indicates the client should load the page specified by the *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.

CSV File Return

Where when the browser requests files that begin with */statistics*, these are expected to be *.csv* files and there is place holder code to return the two expected files based on content generated by the server.

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. i.e. you do not need to concern yourself with LOCKs.

API

Your primary task in this project is to implement the API behaviour. This section provides the details of the API.

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 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.

User Login

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.~~ [Ignore this text] 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 if the password does not match or end any existing session and replace it with a new one if the password does match. 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. The magic session token should be unique for each session.

User Logout

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.

Traffic Adding

Extend the *handle_add_request()* function so that it records the vehicle in the database. The table used includes entries for the location, type, occupancy and time of the recording ~~as a minimum.~~ [Ingore this text.] 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.

Traffic Correction

Extend the *handle_undo_request()* function so that it records the need to undo the vehicle in the database. 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. Undoing applies only the current session. The undo must match the location, vehicle type and occupancy of a valid entry to be undone. An undo operation only undoes one valid add.

Online Summary

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.

Offline Traffic Summary

On the summary page there is an option to 'Download Traffic Summary File". If this option is selected, then the server should provide a summary of the traffic currently stored in the database for the most recent day on which traffic has been captured.

The summary is provided as a CSV file download.

Traffic from previous days should not be included in the summary. The traffic to include may have been recorded by multiple users. Any given user may have recorded traffic in more than one session.

The expected format of the summary is given in Figure 2. Any combination of location and vehicle type should appear only once in the output file. Entries where all the occupancy figures are zero must not be included. The output may appear in any order.

Location,Type,Occupancy1,Occupancy2,Occupancy3,Occupancy4

The file should have a header as shown.
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.
Occupancy1 is the number of that type that was recorded with occupancy 1/25%
Occupancy2 is the number of that type that was recorded with occupancy 2/50%
Occupancy3 is the number of that type that was recorded with occupancy 3/75%
Occupancy4 is the number of that type that was recorded with occupancy 4+/100%

For Example:
Location,Type,Occupancy1,Occupancy2,Occupancy3,Occupancy4
"main road",car,1,0,0,0
"ring road",bus,0,0,0,1
…

Figure 2. Task 6 Output CSV Format.

Worked Hours Summary

On the summary page there is an option to 'Download Worked HoursFile". If this option is selected, then the server should provide a summary of the hours that have been worked by all users for whom an entry exists in the users table of the database.

The summary is provided as a CSV file download.

The expected format of the summary is given in Figure 3. The output may appear in any order, but there should be exactly one entry per user, including any who have worked no hours. The date to use to calculate the summary is the last date on which someone logged out.

Username,Day,Week,Month

The file should have a header as shown.
User is one of the usernames in the database
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:
User, Day, Week, Month
test1,4.3,20.4,81.7
test2,6.5,32.4,100.7
…

Figure 3. Task 7 Output CSV Format.

Database Schema

The database schema is provided and contains three tables. Your application should maintain these tables. You cannot alter the definition of these tables or add additional tables as the database will not persist between tests. Your server must assume the database will exist when it starts and will be held in traffic.db. Your server should not delete any entries from the database.

Table: users

Access to the application is controlled by the username/password pairing. Each user is also assigned a unique integer ID that is used to identify their entries in the other tables. There should be no need to change this table.

userid	INTEGER PRIMARY KEY	A unique integer ID assigned to each user.
username	TEXT NOT NULL	The username of a user.
password	TEXT NOT NULL	An encoded version of the users password.

Table: session

Each user login session is recorded in the session table. This provides both the active sessions and a record of previous session.

sessionid	INTEGER PRIMARY KEY	A unique integer ID assigned to each session.
userid	INTEGER	The userid of the user to which this session belongs.
magic	TEXT NOT NULL	A magic session token used by the browser to identify the session in API calls via a cookie.
start	INTEGER	A unix timestamp indicating the start time of the session.
end	INTEGER	A unix timestamp indicating the end time of the session. A zero indicates the session is active.

The magic value should be a unique string that cannot be easily guessed for each session. It is equivalent to a per-session password and used to ensure only the correct user can access a session. The start and end time are recorded as unix timestamps (number of seconds from a fixed time). You can obtain the current timestamp using code similar to the following:

```
import time
now = int(time.time())
print(now)
```

When calculating how long a user has been logged on in order calculate their hours, you should subtract the start from the end time. Invalid login sessions are not recorded in the table.

Table: traffic

The traffic table records each addition or undo entered by the user of the app.

recordid	INTEGER PRMARY KEY	A unique, non-zero integer that identifies this record.
sessionid	INTEGER	The sessionid in which this record was added.
time	INTEGER	The unix timestamp of the time at which this record was added.
type	INTEGER	The type of vehicle recorded. {"car": 0, "van":1, "truck":2, "taxi":3, "other":4, "motorbike":5, "bicycle":6, "bus":7}
occupancy	INTEGER	A value between 0 and 4.
location	STRING NOT NULL	A string indicating the location of the record.
mode	INTEGER	Indicates the nature of the record. 1: An active add. 0: An undo for which their was a matching add.

		2: An add that has been undone.

The mode field specifies the type of record. When a valid traffic observation is made, a record with mode=1 is added to the table. If an undo is executed, the table should be searched for an add with mode=1 in the current session on which the type, occupancy and location match. If found, the mode of that add should be updated to 2 and new entry recording the undo with mode = 0 [a typo here has been corrected. It previously said 2 and should be 0] should be made. Invalid add or undo operations are not recorded in the table.

The Assessment Process

The total marks available for this assignment are 40.

The Test Suite

The majority of the marks for this assignment are assessed using a test suite. For each test in the test suite the following actions are carried out.

1. A copy of an sqlite3 database that contains the specified tables and a know state is copied into the working directory. It will be called traffic.db and this is the file that should be used by server.py.
2. The server.py Python program will be run with a port number argument.
3. A python test function will make a number of requests to the server and check the values returned.
4. The server program will be terminated.
5. Optionally, the traffic.db file may be checked to ensure it has been updated as expected.

To assist you in checking that your server operates correctly within the test environment, a version containing a subset of the tests is provided. The full test suite contains 36 tests, each worth one mark and covering the whole of the API.

Code Quality

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 or is hard to maintain. 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.

Bug Fix Exercise (up to 25% of corrected marks)

After you submit your code, the full marking script used to assess it will be released to you. You will have the opportunity to make corrections to your code to improve your mark. 25% of any improvement you make will be added to your final mark.

For example, if your initial submission scored 60% and you correct all errors, the updated version would score 100%. You would gain 25% of the 40% you have fixed. This would be an additional 10% added to the 60% you initially obtained. Your final mark would now be 70%.

Your re-submission will not be marked against an identical script. It will contain the same tests, but the data values used in the API calls or in the database will be different.

If your updated version scores less than your initial submission your mark will not be reduced. You will be given the initial mark.

Deliverables & Submission

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

server.py	Your updated version of server.py

This is the only file needed. Do not upload any other files and do not submit it within a zip or other file format.

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.
TrafficApp_Verifier_Subset.py	An example test script that interacts with the application server.
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.
db/*	A series of database files used by the test scripts. Initially, it contains only clean.db

KMC2021