Training Record Application

1 Introduction

In this coursework you will undertake the implementation of a simple training record 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 via the API (Application Programmer Interface). This is described in the section Web Application Structure.

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.

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 database.db in the current working directory.

2 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 training recording 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 load the index.html page. As you are not yet logged in to the app, the page is expected to re-direct to the login page.

Where the browser requests html, css or javascript files, these will be returned by the pre- existing code. You will not need to modify this code. It is contained within the do_GET() method of the http server object.

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 complete the code of these handler functions.

When the request file is '/create/…' or '/class/…' the existing code will load a matching HTML file and that file will invoke the '/action' API as required.

3 API

Your primary task in this project is to implement the backend 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.

You also need to defend against inputs that are deliberately or accidentally bad. What are known as 'injection' attacks. This applies to all of the tasks, and you risk not getting all the marks if your code does not defend against bad inputs. Note that the while the supplied front-end is intended to be well behaved and does not deliberately provide bad input, the same is not true of the scripted tests that will be used for assessment. These 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.

3.1 Command Requests

Command requests are delivered to the browser as POST requests. This means that in addition to any parameters in the request URL, a block of data is also delivered. The client (e.g. front-end) provides any additional parameters needed by the server as a JSON object in this block. Once each action has completed the server needs to provide a JSON formatted response that allows the client to update the output.

The following commands are supported:

3.1.1 *login*

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

{"username":"…", "password":"…"}

You must support the login of the users defined in the database. This is dealt with in *handle_login_request().* An attempt to login should be rejected with a suitable error message if the password does not match. If the password does match, all existing sessions for this user should be ended and be replaced with a new one. 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, for each user.

The client will record the returned session userid and token and supply this with all remaining command requests. A successful login should return a redirect response to index.html. An unsuccessful login should return an appropriate message response.

In all other future '/action' requests, the token must be validated and the command must only be successfully executed when a valid user is identified by a valid session token. Where a valid token is not present or does not indicate a valid user a redirect response to the */login.html* page should be returned for the request.

3.1.2 *logout*

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

When the cookie input matches an existing session, the session should be ended. This is done by deleting the session record from the session table. A *redirect* response to the /logout.html page should be returned. Where the cookie input does not match an existing session a *redirect* response to the /login.html page should be returned.

3.1.3 *get_my_skills*

No parameters expected.

This request should return a set of skill responses for the logged in user. For any skills for which the user is an attendee in a class, the list should begin with one entry for each skill for which the user is marked as 'passed' as an attendee. This should be followed by an entry for each skill for which the user is marked as 'enrolled' and the start time/date has passed and marked as 'pending'. This should be followed by the most recent entry for each skill for which the user is marked as 'failed' and does not have an entry for the same skill marked as 'passed' or 'pending'. Entries recorded as 'cancelled' or 'removed' should not be included in the response.

Where a user has an entry in the trainer table for a skill, that skill should not return 'passed', but instead return 'trainer'.

A suitable message response should also be included.

If the user is not logged in, only a redirect to the login page must be returned.

3.1.4 *get_upcoming*

This request must return a class response for each class for which the start time and date has not passed. These should be sent in timestamp order, soonest first. The action should be as follows:

1. Entries for which the user is the trainer should have an action of 'edit'.
2. Where the user has enrolled in the class and the start time/date has not passed they should have the option to 'leave'.
3. Where the user is not currently enrolled in the class and has not been removed from it, they are permitted to 'join' this class. Provided that they do not already attending a class with the same skill where the state is 'passed' or 'enrolled'. This includes if they have previously cancelled.
4. If the user is already signed up to another class with the same skill as this class should show as 'unavailable'.

A suitable message response must also be sent.

If the user is not logged in, only a redirect to the login page must be returned.

3.1.5 *join_class*

Parameter object: {"id": *classid*}

A user may join a class, so long as there is space and the class is not 'unavailable' to them. In which case the class size will be increased by one. Note that the current class size is not recorded in the database, only the maximum class size. The current size must be calculated using the attendee table in the database.

The may also only join a class when they are not already recorded as being an attendee for a class for the same skill, when the state is 'passed' or 'enrolled.' They may not join this specific class if they have been removed from it. But are permitted to join another class for the same skill.

An updated class response must be returned if the join is successful.

A suitable message response must also be sent.

If the user is not logged in, only a redirect to the login page should be returned.

3.1.6 *leave_class*

Parameter object: {"id": *classid*}

A user may leave a class in which they are ‘enrolled’ if the start time/date has not passed. In which case an updated class response should be returned.

A suitable message response should also be sent.

If the user is not logged in, only a redirect to the login page should be returned.

3.1.7 *get_class*

Parameter object: {”id”: *classid*}

If the user is not the trainer for the class an error message should be returned.

If the user is the trainer of the class a class response and all required attendee responses should be sent.

A suitable message response should also be sent.

If the user is not logged in, only a redirect to the login page should be returned.

3.1.8 *update_attendee*

Parameter object: {“id”:*attendeeid*, “state”:”*pass/fail/remove*”}

If the user is the trainer for the class indicated by the attendeeid, and the start date/time has passed and the state is ‘pass’ or ‘fail’, the state should be updated to ‘passed’ or ‘failed’ respectively. An attendee response will be required to reflect the new state.

If the user is the trainer for the class indicated by the attendeeid, and the start date/time has not passed and the state is ‘remove’, the state should be updated to ‘removed’. An attendee response will be required to reflect the new state. As will a class response.

A message response indicating success or failure must also be returned.

If the user is not logged in, only a redirect to the login page must be returned.

3.1.9 *cancel_class*

Parameter object: {”id”: *classid*}

If the user is the trainer for the class indicated and the start time/date has not passed, the class and all attendees currently shown as ‘enrolled’ should be marked as ‘cancelled’. The max of the class should be set to zero to indicate a cancelled class. Updated class and attendee responses must be returned. Only for those attendees whose status has changed.

A suitable message response must also be sent.

If the user is not logged in, a redirect to the login page must be returned.

3.1.10 *create_class*

Parameter object: {“id”: *skillid*, “note”: “…”, “max”:*max*, “day”:*day*, “month”:*month*, “year”:*year*, “hour”:*hour*, “minute”:*minute*}

If the user is a trainer for the specified skill, then they are able to create a new class. The components of the date and time are given as integers and need to be checked for validity. E.g. the 30th of February is not a valid date. And 03:82 is not a valid time. And must be beyond the current time and date. The maximum class size must be in the range 1 to 10. Classes must only be created for valid skills.

If there is an error in setting up a class, a suitable message response must be returned.

If the class is successfully created, a redirect response should be returned to '/class/*classid*' where *classid* is the newly created database entry for the class.

If the user is not logged in, only a redirect to the login page must be returned.

3.2 Available Responses

The following responses are understood by the web client and form part of the API. The supplied server code includes pre-build functions to generate these.

The general pattern of use is:

```
response = []
response.append(build_response_...(…))
response.append(build_response_...(…))
…
```

Response: redirect

{"type":"redirect", "where":"…"}

The *redirect* response indicates the client should load the page specified by the *where* entry. It is used when the server wishes to change the page displayed.

Response: message

{"type":"message", "code":…, "message":"…"}

Code	Label	Meaning
0	Success	The action requested was successfully completed.
100-199	Missing Parameter	An expected parameter was not present in the request or the request was otherwise malformed. The action could not be successfully completed
200-299	Bad Parameter	An expected parameter was present, but not valid. The action could not be successfully completed.

Table 1: Message Codes

A message response tells the frontend if the requested action was successful or if not indicates the nature of the problem using an integer code value. It also returns a human readable text message. The supplied frontend displays these when a response is processed. Where you are required by this specification to return a message, you should use codes according to Table 1. Where a range of codes are given, you are free to choose any value in that range. This may assist you in debugging.

Response: class

{"type":"class","id":…, "name":"…", "trainer":"…", "notes":"…", "when":…, "size":.., "max":, "action":"…" }

This response is provided to the index.html and class pages. It contains the general details about a class. The action will control the button/label options presented to the user.

id	The classid of the record of a user signed up to a class.
name	The full name of skill being taught in this class.
trainer	The full name of the trainer for this class.

notes	A note added by the trainer when the class was created.
when	The start time/date of the class. (Unix timestamp.)
size	The current number of attendees in the class (those not cancelled or removed.)
max	The maximum permitted number of attendees or the class.
action	“join” – This user is permitted to join this class. This includes if they have previously left, but not if they were removed. “leave” – The user has enrolled in the class and is permitted to leave it. You cannot leave a class after it has started. “pending” – The user is enrolled in the class, and the start time/date has passed. “cancel” – The user is the trainer and permitted to cancel the class. “edit” – The user the trainer and permitted to edit the class. “cancelled” – A class has been cancelled by the trainer or the current user has been removed from the class and is not permitted to join the class. You cannot re-join a class once it has started.

Response: attendee

{“type”:”attendee”, “id”:…, “name”:…,”action”:”…”}

The frontend will ask for these when it wishes to display a class details page, ‘/class/…’. The fields should contain the following information. They will also be supplied by the server when a currently displayed ‘/class/...’ page is updated.

id	The attendeeid of the record of a user signed up to a class.
name	The full name of the user signed up to the class
state	“remove” – The user is marked as enrolled in the class and the start time/date has not passed. The trainer will be permitted to remove this student from the class. “update” – The user is marked as enrolled in the class and the start time/date has passed. The trainer will be permitted to update the state to passed or failed. “passed” – The user is marked as passed in a class of this skill. This takes precedence over other options. “failed” – The user is marked as failed in a class, and there is no other enrolled or passed entry. “cancelled” – The user is marked as cancelled or removed in the attendee table.

Response:skill

{“type”:”skill”, “id”:…, “name”:”…”, “trainer”:”…”, “gained”:…, “state”:”…”}

The frontend will ask for these when it wishes to display a user skills page. The fields should contain the following information.

id	The skillid of the skill.
name	The full name of the skill with skillid.
trainer	The full name of the trainer who trained the user in this skill.
gained	The date on which the skill was gained. This is the date of the class. (Unix timestamp.)
state	“trainer” – This user has an entry in the trainer table for this skill. “scheduled” – The user is marked as enrolled in a class and the start time/date has not passed.

	"pending" – The user is marked as enrolled in a class and the start time/date has passed. "passed" – The user is marked as passed in a class of this skill. This takes precedence over other options. "failed" – The user is marked as failed in a class, and there is no other enrolled or passed entry.

3.3 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.

4 Database Schema

The database schema is provided and contains five 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 should assume the database exists when it starts and will be in database.db in the current working directory.

4.1 Table: users

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

Table 2: users sql table

To simplify the development and debugging process, the password will not be encrypted. This is bad practice and is only being done to allow visibility of the passwords in the supplied databases to reduce development effort of the coursework. If you ever need to write such code in the future, please don't do this.

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. You must not change the contents of this table during start-up, doing so will likely result in failures during testing.

4.2 Table: 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.

Table 3: session sql table

Each user login session is recorded in the session table described in Figure 3. This records all active sessions. 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. A given user may have multiple active sessions.

Note, the last statement can be true even if your server never allows two login sessions in the server as the database may have entries created by other database activity outside the server.

4.3 Table: skill

skillid	INTEGER PRMARY KEY	A unique, non-zero integer that identifies this record.
name	STRING NOT NULL	The name of the skill.

Table 4: skill sql table

The skill table provides a mapping between skill names and a unique integer identifier for the skill. You must not change the contents of this table during start-up, doing so will likely result in failures during testing.

4.4 Table: class

classid	INTEGER PRMARY KEY	A unique, non-zero integer that identifies this training session.
trainerid	INTEGER	The userid of the user that will run this class. The trainer.
skillid	INTEGER	The type of skill the class will teach.
start	INTEGER	Time in seconds since January 1, 1970 00:00:00 UTC when instance began.
max	INTEGER	The maximum number of users who are enrolled in the class. Cancelled/removed students will not be included in the count. Set to zero to cancel class.
note	STRING NOT NULL	An optional comment added by the trainer.

Table 5: class sql table

The details of a class set-up by a trainer.

4.5 Table: attendee

attendeeid	INTEGER PRMARY KEY	A unique, non-zero integer that identifies this attendance of a class by a user.
userid	INTEGER	The userid of the user.
classid	INTEGER	The classid into which the user is enrolled.
status	INTEGER	The status of this user in the class. 0 - “enrolled”, when a user is first added to a class. 1 - “passed”, after the class ends, the trainer can set. 2 - “ failed”, after the class ends, the trainer can set. 3 - “cancelled”, when a user has left the class. 4 - “removed”, when a user was removed. They may not rejoin.

Table 7: attendee sql table

The status of a user in a class.

4.6 Table: trainer

trainerid	INTEGER	A userid of someone allowed to teach this skill
skillid	INTEGER	A skill a trainer is allowed to teach.

Table 6: trainer sql table

This table holds a record of which skills a user is permitted to teach. You must not change the contents of this table during start-up, doing so will likely result in failures during testing. Note that this table does not have a primary key, it is used only to link the user and skill tables together for those users who are allowed to run classes.

5 The Assessment Process

The total marks available for this assignment are 100%.

5.1 The Test Suite

This assignment will initially be assessed using a regression test suite that replaces the web front-end. 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 database.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 several requests to the server and check the values returned.
4. The server program will be terminated.
5. Optionally, the database.db file may be checked to ensure it has been updated as expected.

The full test suite contains a series of tests, providing coverage of the API. The tests all have equal weight and account for 90% of the overall mark.

To assist you in checking that your server operates correctly within this testing framework a version of the regression test suite has been provided that contains a sample test. This will allow you to ensure you have not made changes to the way the server is expected to behave that will make it incompatible with the regression suite. You are encouraged to test your code with this method as well as using the supplied front-end. You are free to add additional tests to the test script but these will not form part of the assessment and you are not expected to submit them.

5.2 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 2.5% mark, up to a limit of 10% marks. The pylint score is out of 10. You only need to achieve 8 or more to get the maximum marks.

6 Deliverables & Submission

This course work has two deadlines.

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

server.py	Your version of server.py

This is the only file needed. Do not upload any other files and you must not submit it within a zip or other file format. You cannot spread your code across multiple files.

After the initial deadline, a copy of the regression test suite will be released. At the same time as the lecturers are running you code against the suite, you will have the opportunity to make any corrections require to your code.

You may then upload the corrected version of your code moodle. This second version of the code will be run against a second, similar, test suite. Where similar means the test functionality will be the same, but the data may be different. If your new regression score is larger than the old one, you will gain 20% of the difference. For example, if your initial submission scored 65% and your revised submission scored 90% then you would gain 20% of 25% or 5% giving you a final mark of 70% for the tests.

7 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 your code.	
pages/*	All the html pages delivered by the server live in this directory.	
	index.html	The frontend main page (html). You should not modify this. When logged in it is intended to show upcoming training classes.
	skills.html	A frontend page (html) that shows the skills for which a user has or is due to get. You should not modify this.
	class.html	A frontend page (html) that shows the details of a class. You should not modify this.
	create.html	A frontend page (html) that shows the class creation form. You should not modify this.
	login.html	The frontend login page (html). You should not modify this.
	logout.html	The frontend logout page (html). You should not modify this.
	menu.html	The frontend menu page (html). You should not modify this.
css/*	The frontend cascading style sheet for the html pages (css). Uses bootstrap. You should not modify this.	
js/*	The frontend javascript code makes AJAX requests to the server and process the responses. You should not modify this.	
db/*	A place where databases can be stored. One of these should be copied to database.db in the parent directory for use. Do not access these files directly from this location in your server.	
	database.db	This version here contains initial sqlite3 tables for reference.
	just_users.db	This version contains the tables plus some users.

8 Imperfect skeleton code

As we will cover in the lectures, software can contain bugs. The code supplied has been tested, but that testing cannot be exhaustive. It is possible the code contains bugs. If you find a bug in the server code, it's your job to fix it. A bug is defined as any behaviour that does not match the specification. You are not expected to address bugs in the front-end code. The front-end code is only intended as an example of how your server may be used and can be a useful debug tool during development. But front-end code is not used as part of the testing procedure and does not form part of the specification.

9 Plagiarism

The work you submit for assessment must be your own. The nature of this task is such that you are not likely to find any existing solutions on the web. You can discuss your work with your peers; however, you must be careful that this does not become collusion. This means that you can discuss the concepts involved but should not be sharing code or co-developing.

Your submitted code will be run through a tool that can perform pairwise comparison of all submissions to detect sharing of code. This tool does more than compare text. It understands the Python language and is

able to detect similarity even when superficial (to the computer) changes such as re-naming variables or re-formatting/re-ordering of code is performed. Any code flagged as similar will then be reviewed by the lecturer and where appropriate reported to the Director of Studies.