Declaration on Plagiarism

Assignment Submission Form

This form must be filled in and completed by the student(s) submitting an assignment

Name(s): Alex Randles & Kealan Thompson
Programme: CASE4
Module Code: CA4006
Assignment Title: Assignment 2: A Room Booking System
Submission Date: 12/04/19
Module Coordinator: Dr Rob Brennan & Dr Martin Crane

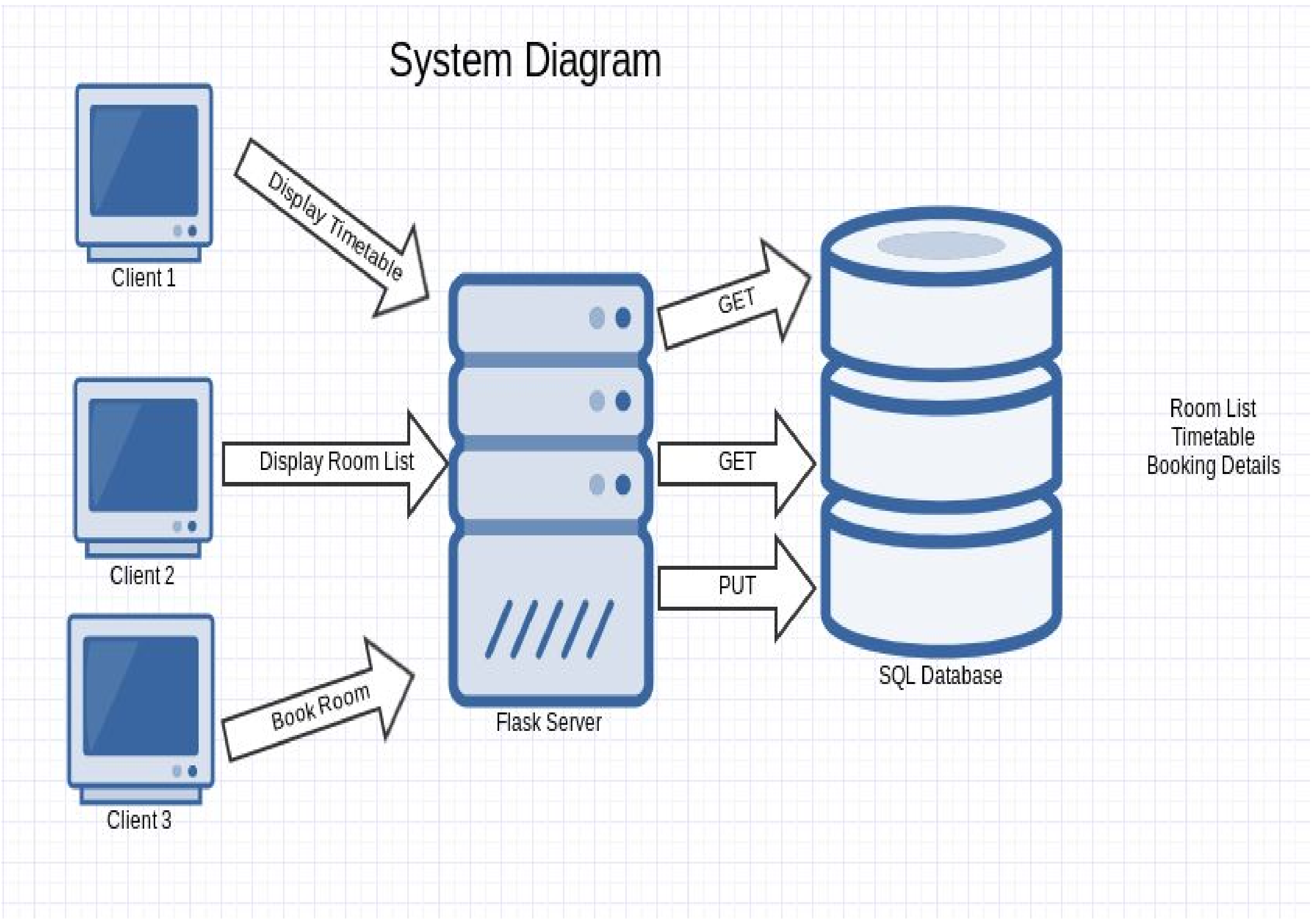
Name(s): Alex Randles & Kealan Thompson Date: 10/04/19

Running the program

- **Locally (Preferred)**
    - pip install -r requirements.txt
    - python server.py
    - http://127.0.0.1:5000
- **Remotely:** https://ca4006-assignment2.appspot.com

System Design

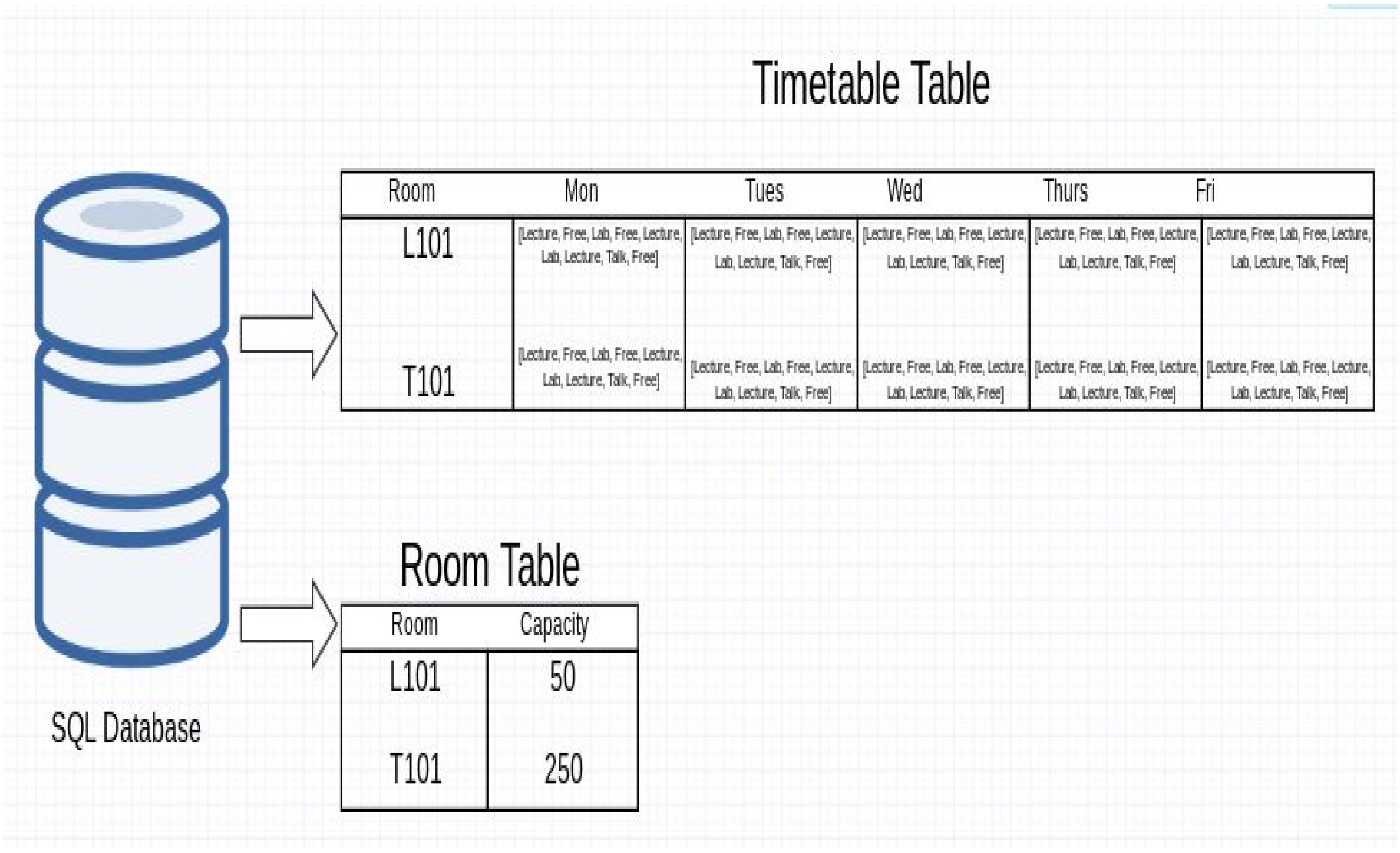
- **Programming Language:** Python
- **Architectural design:** REST API
- **Libraries:** Python standard Library, Flask micro framework library for designing the API framework and requests library for making request to the API
- **Database:** Lightweight SQLite database to store information relating to the system components: Room's, Timetables and bookings.
- **Supported methods:** GET, POST and PUT.



Database Design

- Room Table
  - Stores information about rooms the system manages.
  - Room name is the primary key.
  - Rooms can be easily added through the “/timetable/room/add” URI.

- Timetable Table
  - Stores information about the timetable of each room.
  - Room name is the primary key.
  - Contains a column for each day.
  - These columns contain an ordered list of each event in the room starting with event at 08:00 at the head of the list and the event at 17:00 at the tail of the list.
  - The values in this list are initially set to “Free” as there is no event booked in.

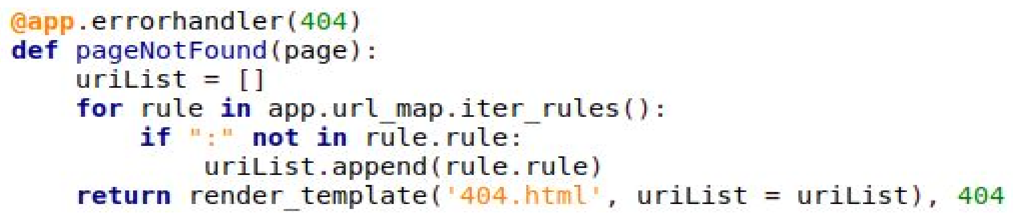


Room	Mon	Tues	Wed	Thurs	Fri
L101	[Lecture, Free, Lab, Free, Lecture, Lab, Lecture, Talk, Free]	[Lecture, Free, Lab, Free, Lecture, Lab, Lecture, Talk, Free]	[Lecture, Free, Lab, Free, Lecture, Lab, Lecture, Talk, Free]	[Lecture, Free, Lab, Free, Lecture, Lab, Lecture, Talk, Free]	[Lecture, Free, Lab, Free, Lecture, Lab, Lecture, Talk, Free]
T101	[Lecture, Free, Lab, Free, Lecture, Lab, Lecture, Talk, Free]	[Lecture, Free, Lab, Free, Lecture, Lab, Lecture, Talk, Free]	[Lecture, Free, Lab, Free, Lecture, Lab, Lecture, Talk, Free]	[Lecture, Free, Lab, Free, Lecture, Lab, Lecture, Talk, Free]	[Lecture, Free, Lab, Free, Lecture, Lab, Lecture, Talk, Free]

Room	Capacity
L101	50
T101	250

Client Failure Handling

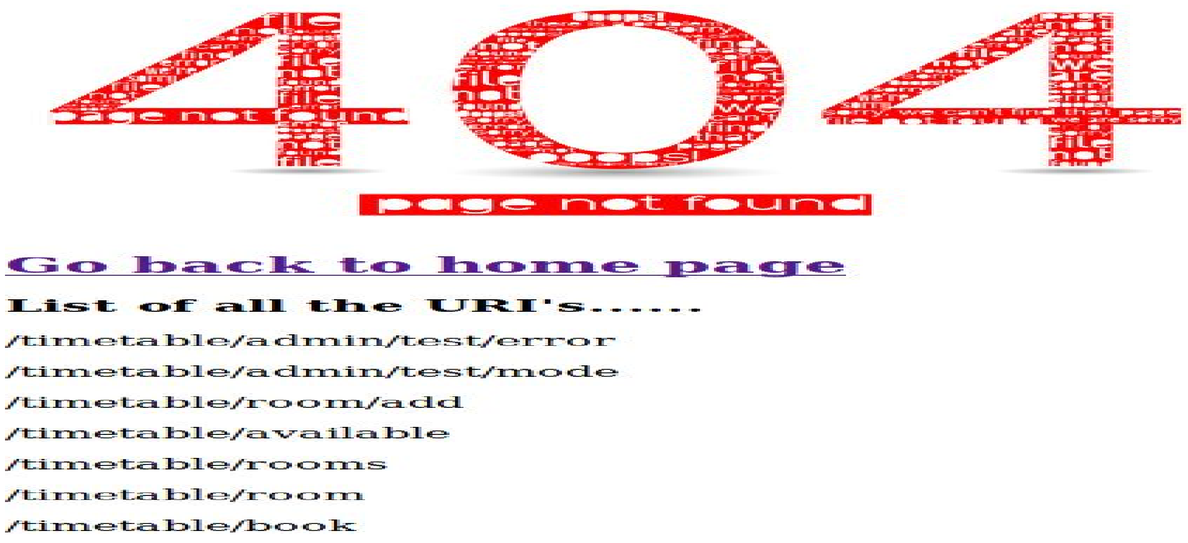
- How errors are handled?



```
@app.errorhandler(404)
def pageNotFound(page):
    uriList = []
    for rule in app.url_map.iter_rules():
        if ":" not in rule.rule:
            uriList.append(rule.rule)
    return render_template('404.html', uriList = uriList), 404
```

There is an error handler for flask which allows you to return custom responses to different kinds of errors.

- 404 - Not found
  - If a client tries to request a resource that doesn't exist they will be redirected to the API's custom 404 page.
  - This page will direct them back to the root URI and provide them a list of all the URI's the API offers



Go back to home page

List of all the URI's......

/timetable/admin/test/error
/timetable/admin/test/mode
/timetable/room/add
/timetable/available
/timetable/rooms
/timetable/room
/timetable/book

- 500 - Internal Server Fault
  - This will generate a similar page to a 404 error but it will also retry the request that caused the fault to occur.

- 400 - Bad request
  - This will generate a similar page informing them of which fault has occurred and instructing to go back to the root to retry the request.
- Other Errors
  - We also deal a wide variety of other errors such as connections errors, retrieval errors or input errors e.g entering a string for capacity or trying to add a room which exists.

Concurrency Strategies

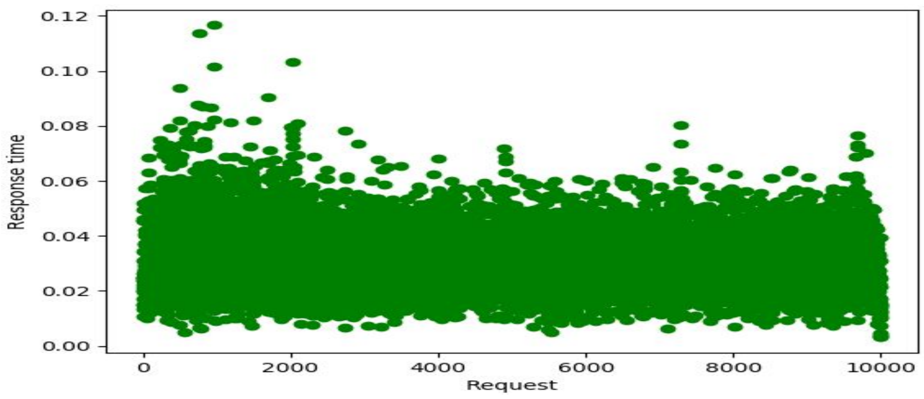
- **Flask:** By default, flask runs in single threaded mode but we've configured are flask service to run in multi-threaded configuration.

```
app.run(host="127.0.0.1", port = "5000", debug = True, threaded=True)
```

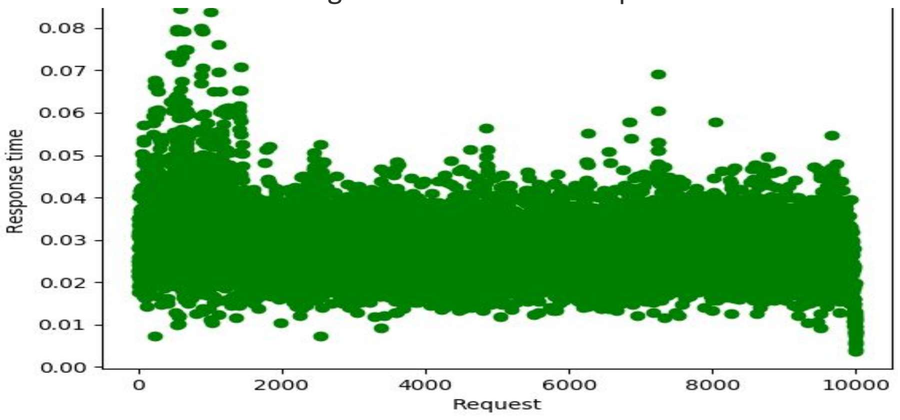
- **Database:** SQLite allows multiple connections to one database but we had to ensure the same connection was not used by each user as this would lock the database when a write action is taking place. When a user tries to update the database by booking a room or adding a room they will receive the lock to the database which will ensure that corrupt data is not created. We used a 'with' statement to create a different connection for each user which ensures concurrent access is allowed to the system.

```
with sqlite3.connect(database) as connection:
    cursor = connection.cursor()
    result = cursor.execute("SELECT * FROM rooms").fetchall()
```

- **Testing:** We wanted to ensure that the concurrency strategies we implemented within our system worked effectively.
- We created a script called "test_script.py" which would generated a number of request from a number of different clients simultaneously and then use the data from the responses to plot a graph.
- Each dot on the graph indicates a request and the color green indicates a successful request was made. The y-axis is the time taken for a request to be completed and the x-axis is the number of request.
- **Test case 1:** Unthreaded configuration with 1000 request from 10 clients.

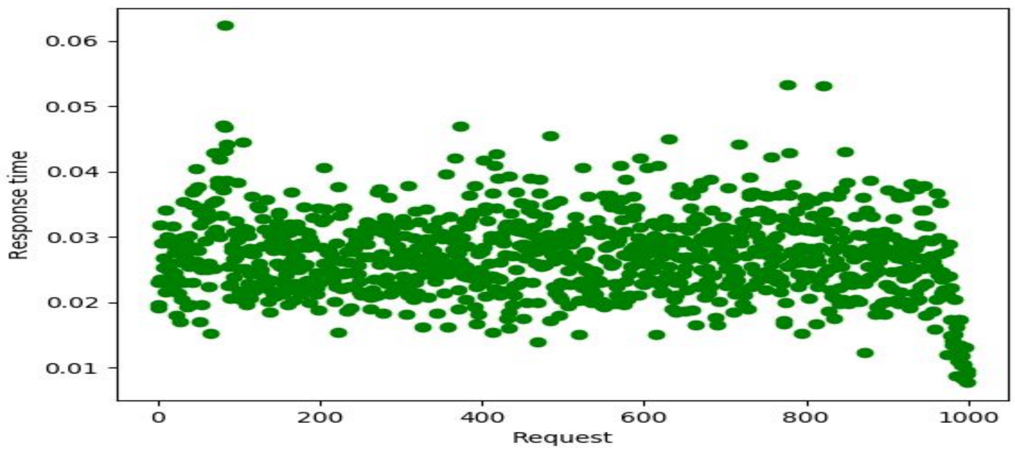


- **Test case 2:** Threaded configuration with 1000 request from 10 clients.

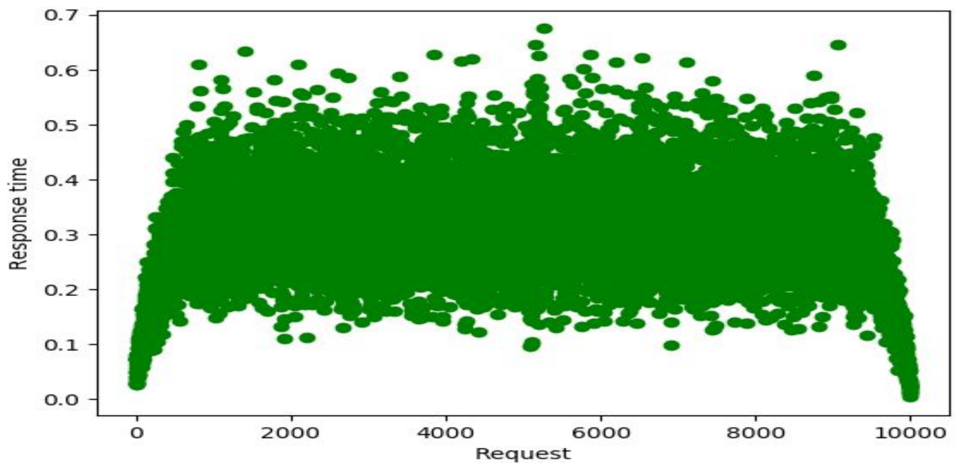


Scalability Study

- We created a script to generate simultaneous request to our API.
- Any failed request i.e didn't receive 200 status code back will be a red dot.
- **Test 1.1:** 100 GET request from 10 clients



- **Test 1.2** 100 GET request from 100 clients



- **Findings:**
    - **Test 1**
        - Average response time is 0.0295173773766
        - All status code 200 received back
    - **Test 2**
        - Average response time is 0.301873926902
        - All status code 200 received back
    - Difference: 0.2723565495254

- **Conclusion:** The response time did increase slightly more for each request when the number of clients increased as expected but all responses were green meaning they were successful (200 status code) which shows the reliability of the service.