Mini Project - Quiz App

Mairi MacLeod: 1700231

CMP408 - System Internals and Cybersecurity

BSc(Hons) Ethical Hacking

14th January 2021

Contents

1 Introduction

This project is broken down into three components; a Flask web application, a Amazon Web Services (AWS) Relational Database Service (RDS) database and a Raspberry Pi Zero. For the web application Python based Flask was used as the author decided it would be the most straightforward for connecting to AWS and the Pi. AWS was chosen to store the database in order to reduce the amount stored on the Pi itself and to allow others with Pis to also do the quiz and compare their results. In regards to the Pi Zero, buttons were seen as the obvious choice for selecting answers and LEDs were just an aesthetic choice to make the application feel more interactive. The quiz itself being themed on Cybersecurity, specifically Linux internal security, relates it to the topic of this module. Security was also considered and best practices were used wherever possible throughout all aspects of this project.

1.1 Objectives

For this project the aim was to create a quiz application that used components on the Pi Zero to select answers. In order to achieve this aim the following objectives had to be met:

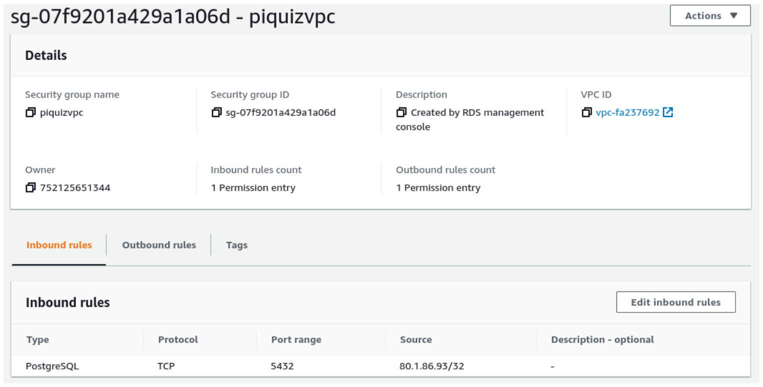
- Create web application
  - Allows user to enter a username at the start in order for their scores to be stored
  - Question answers to be displayed in a randomised order
  - High scores to be displayed at the end
- Get input from Pi buttons to answer questions
- Use coloured LEDs to display if answer is correct or not
- Create database on AWS to store high scores and questions

2 Procedure

2.1 Amazon Web Services Database

An RDS Postgresql instance was created in order to store users' names and scores. Due to the small amount of storage space on the Raspberry Pi, to have all the questions stored locally would make it difficult for the questions to be updated on all devices that wish to use the quiz therefore it was decided that they should also be stored in a database. The database is comprised of two tables; HighScores and Questions. A database helper script titled `createDB.py` was created to create, populate and delete data from the tables as appropriate.

The RDS instance requires both password and IAM authentication, this was deliberate as it allowed the author to access the database externally and provided an extra layer of security. In order to control access to the database a Virtual Private Cloud (VPC) was created that restricted ingress traffic. All data required for the authentication is imported from a separate python file, `envs.py` that is included in the gitignore, so that it would not be uploaded to GitHub and remain private.



sg-07f9201a429a1a06d - piquizvpc

Details

Security group name	Security group ID	Description	VPC ID
piquizvpc	sg-07f9201a429a1a06d	Created by RDS management console	vpc-fa237692

Owner	Inbound rules count	Outbound rules count
752125651344	1 Permission entry	1 Permission entry

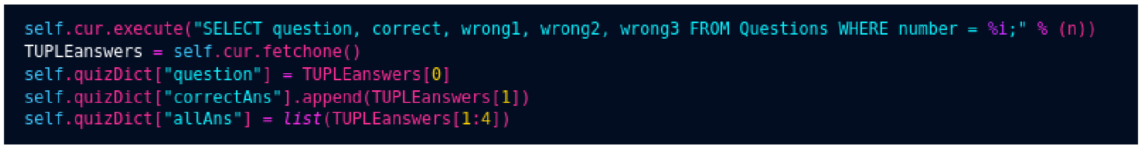
Inbound rules

Type	Protocol	Port range	Source	Description - optional
PostgreSQL	TCP	5432	80.1.86.93/32	-

Figure 1: VPC Rules for RDS

2.2 Flask Web App

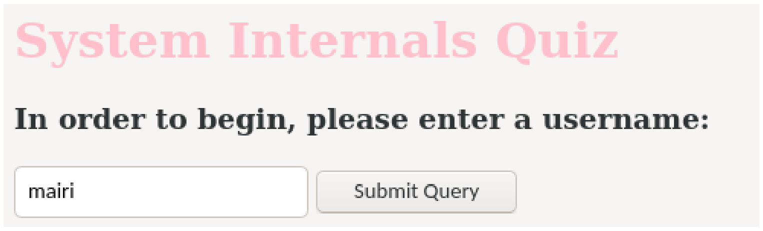
The main component of this project is the web application. For a clear diagram of the application life cycle see *Appendix 51*. It consists of a class, `QuizClass`, and four functions; `index`, `getUsername`, `quiz` and `highScores`. `QuizClass` gets a question and corresponding answers from the database, shuffles the answers' order and returns them. *(Figure 2)* A class was chosen for these functions as there are multiple that all pass data between each other so it was logical for them all to be combined in this way.



```
self.cur.execute("SELECT question, correct, wrong1, wrong2, wrong3 FROM Questions WHERE number = %i;" % (n))
TUPLEanswers = self.cur.fetchone()
self.quizDict["question"] = TUPLEanswers[0]
self.quizDict["correctAns"].append(TUPLEanswers[1])
self.quizDict["allAns"] = list(TUPLEanswers[1:4])
```

Figure 2: Questions and Answers Being Taken from Database

`Index()` renders the corresponding `index.html` page which contains a form for a user to enter their username. *(Figure 2* The username is then sent via a POST request to `getUsername()` which adds it to the `HighScores` table. Once the username is inserted into the table all of the LEDs are told to flash quickly and `quiz.html` is rendered.

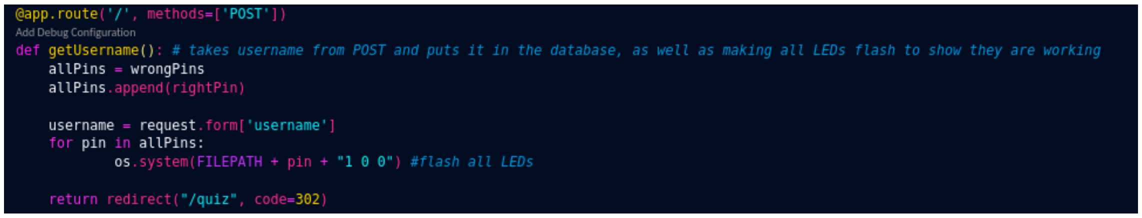


System Internals Quiz

In order to begin, please enter a username:

mairi

Figure 3: index.html



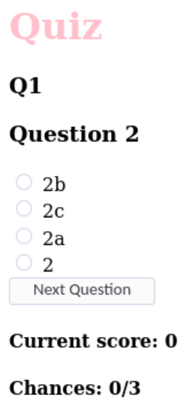
```
@app.route('/', methods=['POST'])
def getUsername(): # takes username from POST and puts it in the database, as well as making all LEDs flash to show they are working
    allPins = wrongPins
    allPins.append(rightPin)

    username = request.form['username']
    for pin in allPins:
            os.system(FILEPATH + pin + "1 0 0") #flash all LEDs

    return redirect("/quiz", code=302)
```

Figure 4: getUsername Function

This page gets a question and it is displayed with radio buttons so the user can choose an answer. Every time a question is answered it is checked against the correct answer and depending on the result the `right` or `wrong` item in the dictionary `score` is incremented.



Quiz

Q1

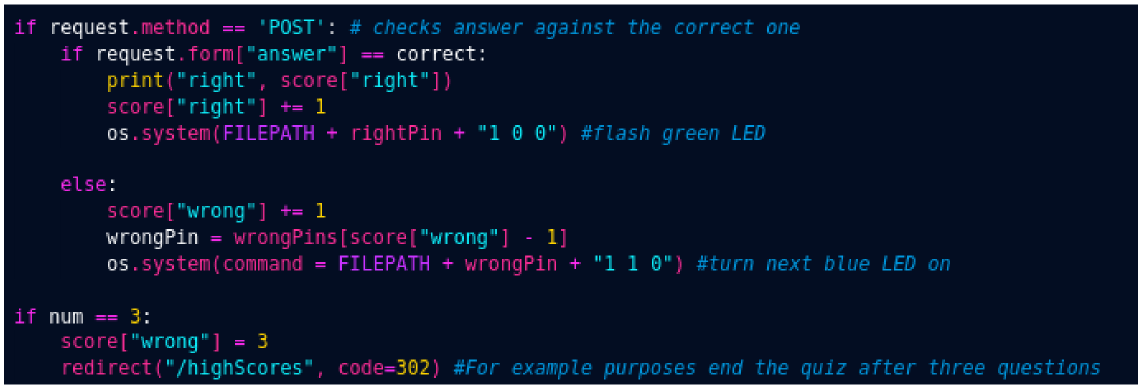
Question 2

- 2b
- 2c
- 2a
- 2

Current score: 0

Chances: 0/3

Figure 5: quiz.html



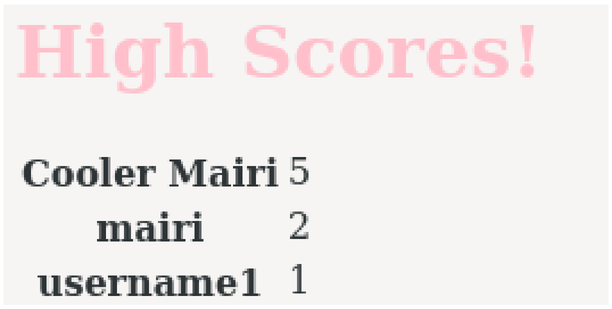
```
if request.method == 'POST': # checks answer against the correct one
    if request.form["answer"] == correct:
        print("right", score["right"])
        score["right"] += 1
        os.system(FILEPATH + rightPin + "1 0 0") #flash green LED

    else:
        score["wrong"] += 1
        wrongPin = wrongPins[score["wrong"] - 1]
        os.system(command = FILEPATH + wrongPin + "1 1 0") #turn next blue LED on

if num == 3:
    score["wrong"] = 3
    redirect("/highScores", code=302) #For example purposes end the quiz after three questions
```

Figure 6: Checking User Input

If `wrong` reaches three then the quiz was designed to end, the users score is displayed, inserted into the database and then they can go to the high scores page and view the top ten scores. However, in the final product even when wrong reaches three the quiz freezes and does not go to highScores or another question but the highScores page can be seen and renders correctly via the URL.



High Scores!

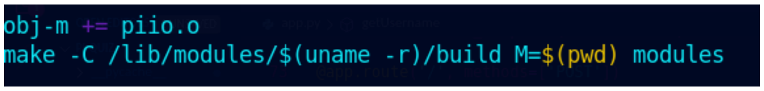
Cooler Mairi	5
mairi	2
username1	1

Figure 7: highScores.html

2.3 Pi Zero

Unfortunately due to various issues in development buttons were not implemented, however LEDs still were and they are used to display whether a user got an answer wrong or right. *(Figure 6)* When a user gets an answer correct the green LED flashes briefly but when they get one wrong one of the three blue LEDs is lit to inform them how many more they can get wrong before the quiz ends. To enable the Pi to interact with the LEDs on the breadboard a driver file, piio.c, was created. This was then compiled using a makefile, shown below, and the following comands were ran to create and load the kernel objects.

```
make -C /lib/modules/$ (uname -r)/build M=$(pwd) modules
sudo insmod piio.ko
gcc piio.c
```



```
obj-m += piio.o
make -C /lib/modules/$(uname -r)/build M=$(pwd) modules
```

Figure 8: Makefile

The flask application was then cloned from the authors GitHub repository, the env.py file copied over using `scp` and all of the python libraries installed using the command: `pip install -r requirements.txt` To interact with the Pi the author used the `library`. In the end there were some issues with interacting with the driver code and so none of the LEDs turn on throughout the application lifecycle.

3 Conclusion

Overall most of the features of this project were implemented according to the original plan. A flask web application takes in a username, displays quiz questions, accepts answers and displays high scores. Even if they do not work, on the Pi three blue and a green LED are used to display how many wrong answers the user has gotten along with a visual indicator of the answer being correct. AWS was successfully used to create two tables in an RDS database, which is accessed through the flask application and a database helper python script.

For the web application no sensitive data is passed between the pages although POST was still chosen over GET to display best practice in the case any private data is handled in the future. Since the code for this project is stored online on GitHub all data relating to authenticating and connecting to the RDS database is stored externally to the python script and imported in. Another security feature relating to the database was the implementation of a VPC to control the ingress traffic and only allow the authors Pi to connect to the database. A IAM user was considered and the author used their private AWS account to create the database for this reason, however this was not implemented in the final product.

3.1 Future Work

If given more time the author would first solve the errors in the flask application so that the scores increment appropriately and the LEDs turn on when told to. In terms of making it closer to the original design, some buttons to allow the user to use the Pi to input their answers would be implemented as well. Due to lack of available components the author was forced to use blue LEDs to display the amount of wrong answers, in future they would purchase red ones to swap them out.

4 References

Amazon Web Services (2020). *Amazon Relational Database Service (RDS)* [online] Available at: https://aws.amazon.com/rds/ (Accessed 9th December 2020)

Amazon Web Services Docs (2020)[1]. *Connecting to an Amazon DocumentDB Cluster from Outside an Amazon VPC - Amazon DocumentDB.* [online] Available at: https://docs.aws.amazon.com/documentdb/latest/developerguide/connect-from-outside-a-vpc.html (Accessed: 9th December 2020)

Amazon Web Services Docs (2020)[2]. *Connecting to your DB instance using IAM authentication and the AWS SDK for Python (Boto3) - Amazon Relational Database Service.* [online] Available at: https://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/UsingWithRDS.IAMDBAuth.Connecting.Python.html. (Accessed: 12th December 2020)

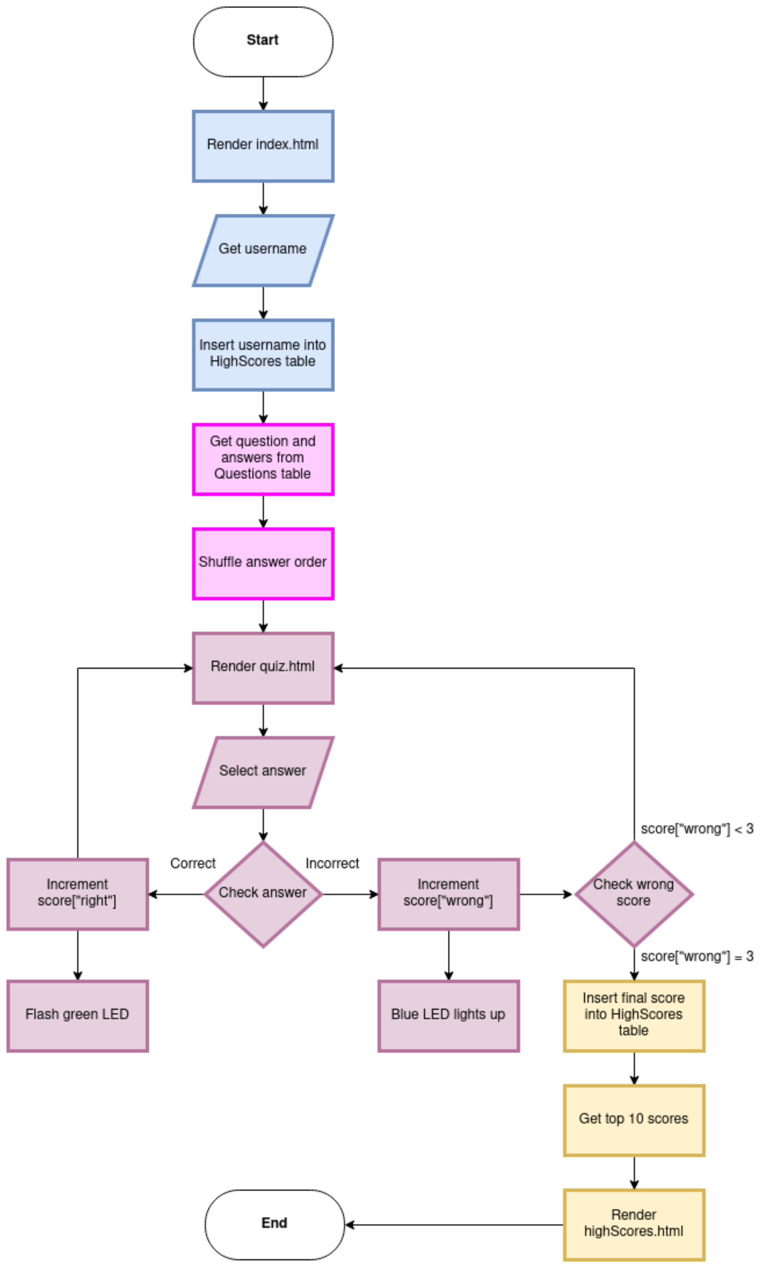
PyPi (2010). *psycopg2 2.8.6* [online] Available at: https://pypi.org/project/psycopg2/ (Accessed 12th December)

Raspberry Pi (2020). *Kernel building* [online]. Available at: https://www.raspberrypi.org/documentation/linux/kernel/building.md (Accessed 1st January 2021)

SuperMairio (2020). *Pi Quiz* [online] GitHub. Available at: https://github.com/SuperMairio/Pi-Quiz (Accessed: 9th December 2020)

5 Appendices

5.1 Flow Diagram for Project


Start
Render index.html
Get username
Insert username into HighScores table
Get question and answers from Questions table
Shuffle answer order
Render quiz.html
Select answer
Correct
Check answer
Incorrect
Increment score["right"]
Increment score["wrong"]
Check wrong score
score["wrong"] < 3
score["wrong"] = 3
Flash green LED
Blue LED lights up
Insert final score into HighScores table
Get top 10 scores
Render highScores.html
End