EMBEDDED INTERFACE DESIGN

Super Project 2018

AUTOMATED BIKE DOCKS

Harsimran Singh Bindra | Yasir Aslam Shah | Sharanjeet Singh Mago

Introduction

The main idea of this project is to implement an Automated Bike Docking station. In this project two Raspberry Pi3s are used to represent two bike stations containing multiple bike slots. A bike is locked or unlocked in any bike station using RFID / Login based operations. Both the dock modules are connected to a centralized AWS database. The project also contains a webpage through which a user can book or checkout any slot. The website also fetches the data from the AWS database and contains status about each bike station i.e. the number of bikes docked and the number of slots available for bike parking. The webpage is also updated if any bike at a station is locked, unlocked or if any slot at a bike station is booked by a user. Each user has a RFID tag which is associated with a unique username and contact number. An AWS simple Notification Service is implemented in this project to notify users books any slot for bike parking based on username or the RFID tag used or checkout a slot. MQTT protocol is used to connect the bike stations to the AWS database whereas HTTPS communication protocol is used to connect the webpage to the AWS database. The communication between the webpage and the database is bidirectional i.e. a user can book an available bike slot from the webpage. A QT application displaying the bike slots available at a docking station is available at each docking station by connecting a display to the raspberry pi 3 module that interfaces with a user for slot booking and checking out.

Project Requirements:

- **Python and Node.js Elements:**

  Various modules of project are written in Python. Python language is used for RFID module interfacing, QtPy application, MQTT & Lambda function interfacing and database creation & integration.Node.js is used to write the lambda function for database integration with the website.

- **QT and HTML Uis:**

  PyQt application is used to create UIs to interact with users at Dock locations. The program for QtPy5 is written in Python and allows a user to sign in into the system and book a slot or check one out. HTML Uis are used for webpage creation, formatting and database operations.

- **Communication protocols:**

  In this project, various communication protocols are implemented as MQTT, SPI, websockets, interfacing various hardware and software modules.

- **AWS IoT Framework:**

  1. AWS IoT thing an Lambda function
  2. S3 service
  3. API gateway
  4. Simple notification service
  5. MQTT service
  6. Python SDK
  7. DynamoDB

- **Sensors:**

  RFID tags and sensors

- **Two RPi3s:**

  Two RPI3s are used as two Dock modules with PyQt loaded for user interaction while booking and checking any slots.

- **Message Queuing**:

  SNS simple Notification Services is used to notify a user using SMS service. SNS itself has its own messaging queuing service.

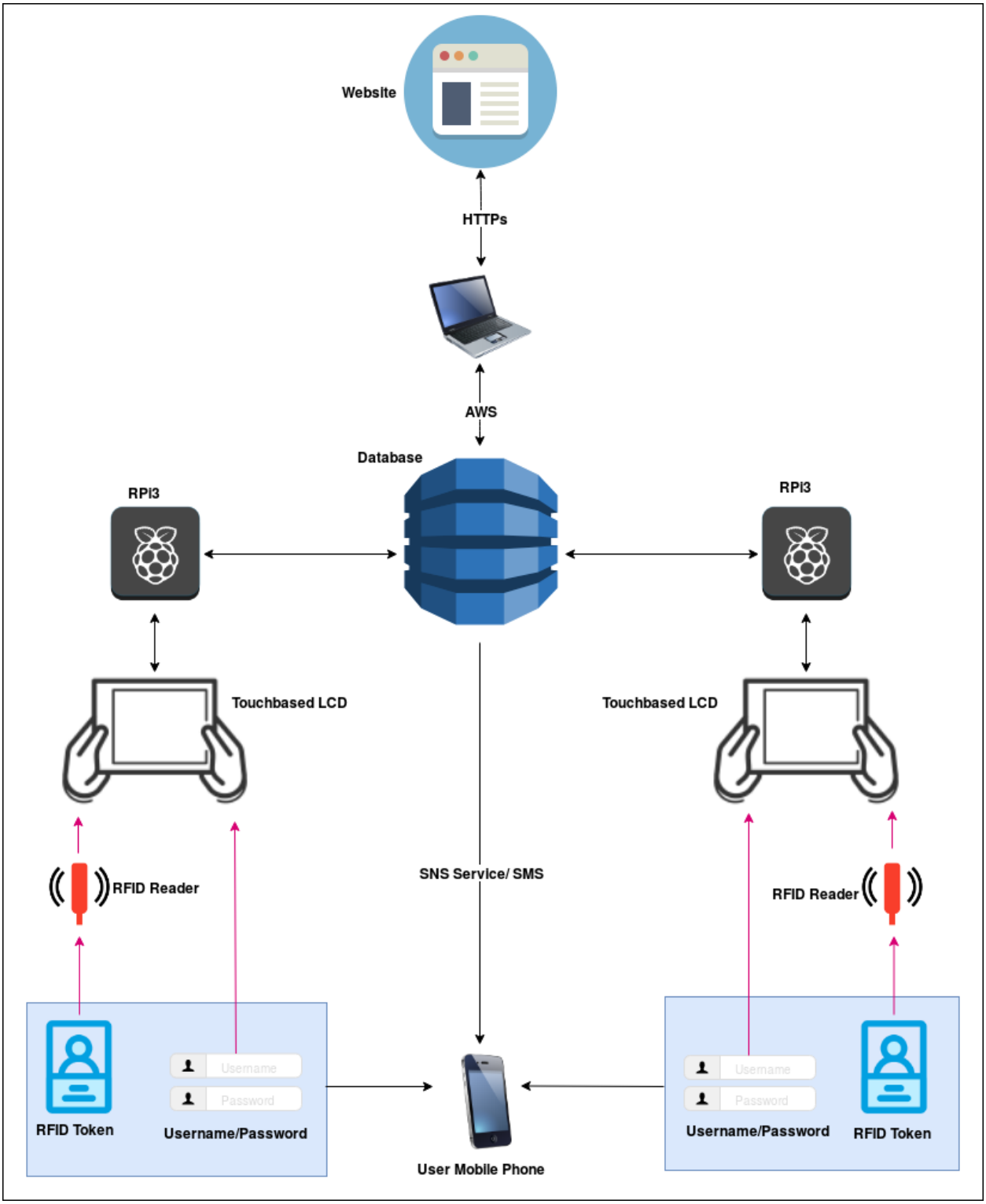
Hardware Requirements:

The system contains the following hardware modules

- Two RPi3s

- RFID modules and tags
- Touch based LCD panels

Functional Design Overview



Daigram1: Project Flow Diagram

- The system contains a touch-based module at one of the Dock.
- There are two Docks with ten slots each.
- A user inputs its username/password to login into the system.
- A user is also provided with an option to login with a RFID tag
- A user can only book one slot at a time, so a user can login to either book a fresh slot or checkout from an existing slot.
- A user can also book or checkout a slot remotely using website also
- each dock module and the website work cohesively with a centralized database that contains tables about user details and slot availability.

Procedure Steps

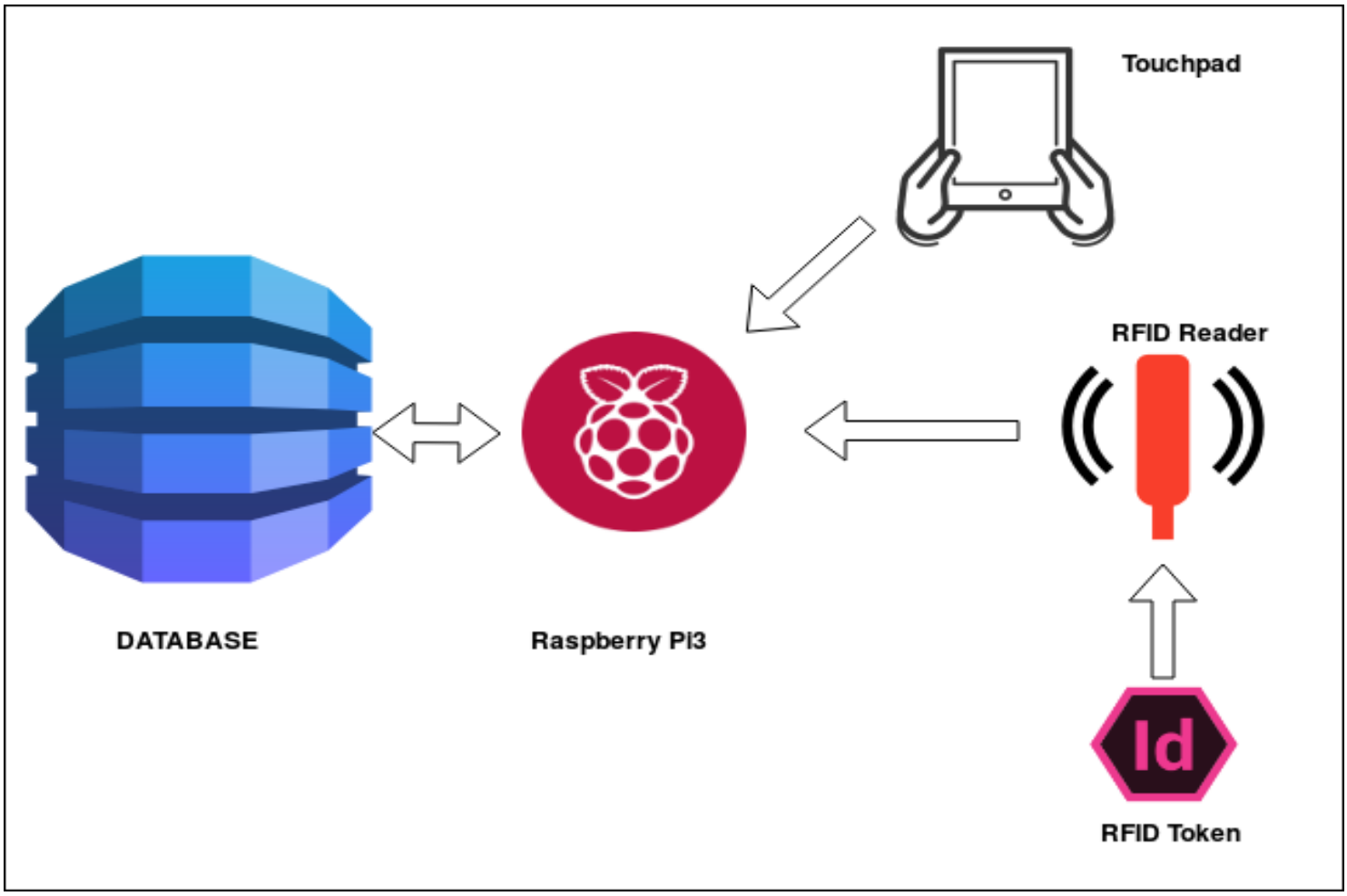
- Authentication of User. This is done by two methods
    - RFID Card Scan
    - Manual Login
- If logged online, user selects the dock to unlock from available options or checkout any existing booking.
- Once logged in using dock modules, the system checks if the user has an existing booking in system and hence directs the user to checkout window, otherwise the booking window is loaded.
- Database updated with the information of Person and dock number whenever user books or checkout any slot.

System Outputs

- Locking and Unlocking of cycle docks.
- Notification on mobile phone.
- Information on website/dock module
    - Slots availability

Hardware Flow Diagram:

- RFID tokens to login into the system.
- Touch based LCD modules to interact with a user
- RPI3 interacting with the database



Daigram2: Hardware System

Project Modules:

This project contains the following three parts:

1.PyQT

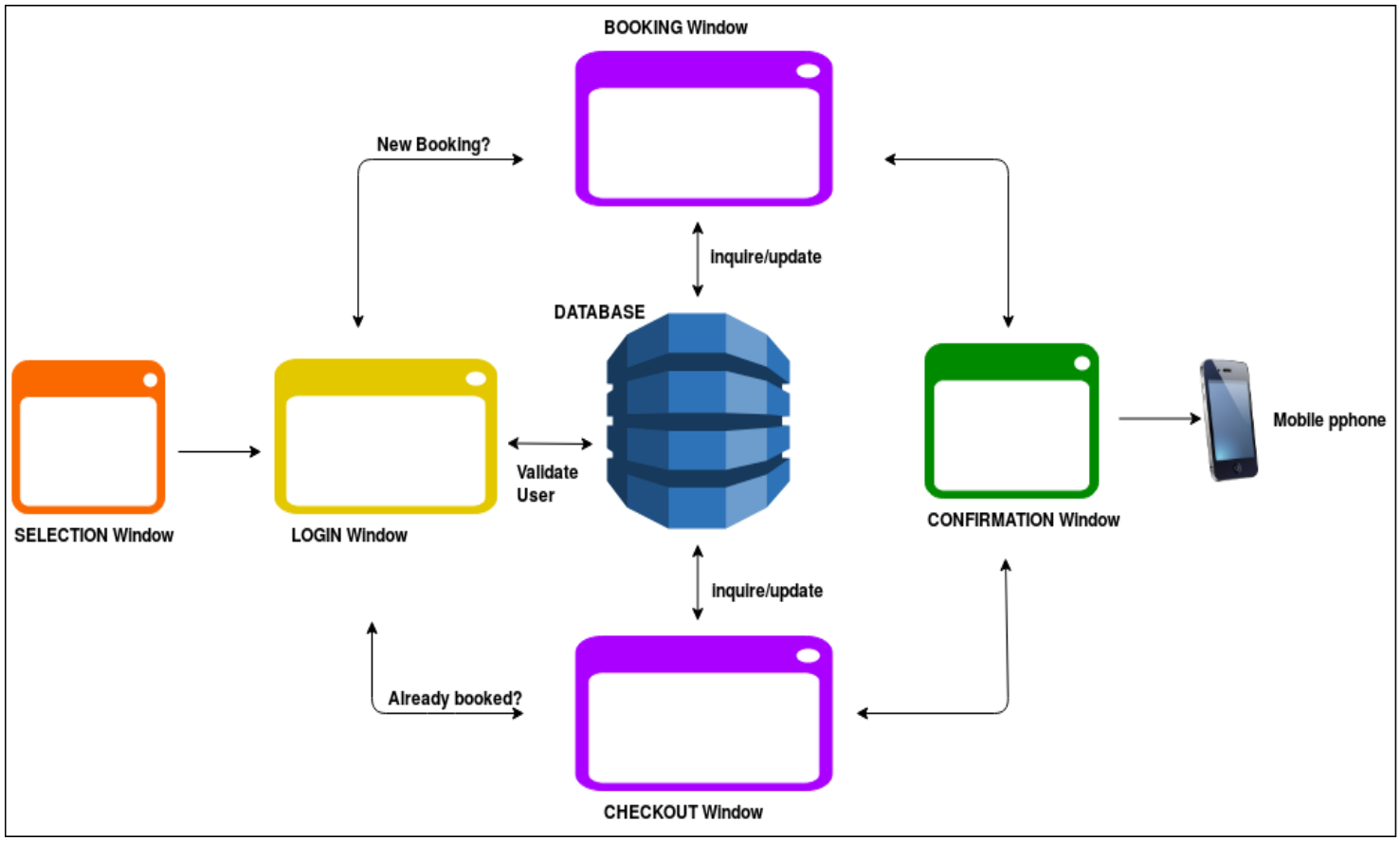
PyQt is a python binding for QT cross-platform. In this project we are using PyQt based multiple interfaces to interact with the user to book a slot or checkout any booked slot at any dock.

In this project, there are two docks present with ten slots available at each dock. A touch-based module is present on each dock that allows a user to book or checkout slots.

PyQt based interface is loaded onto these touch-based modules. The interface allows a valid user to book a slot or checkout an existing booking. The interface contains a total of five windows. The home window is a **SELECTION window** that allows a user to select login options as RFID login or Username/password. A user can login using RFID token and the next window will be either Booking window or Checkout window based on a user booking a slot or checking out a slot respectively.

If a user selects to input login credentials then a **LOGIN window** asking a user to input a username and a password pops up, thereby allowing only registered users to access the service. The system is loaded with a set of users that can access the bike docking system based on a valid username/Password or RFID token. Any new user registering for access needs to setup himself up with the service provider. If the input credentials are correct, next window pops up.

Daigram3: QtPy Design Flow



The next window is different for different users. If the user has already booked a slot previously, then the next window called as **CHECKOUT window** allows the user to check out the bookings, however if the user has zero bookings in the system, then the next window called as **BOOKING window** allows a user to check if any slot is available and therefore book that particular slot.

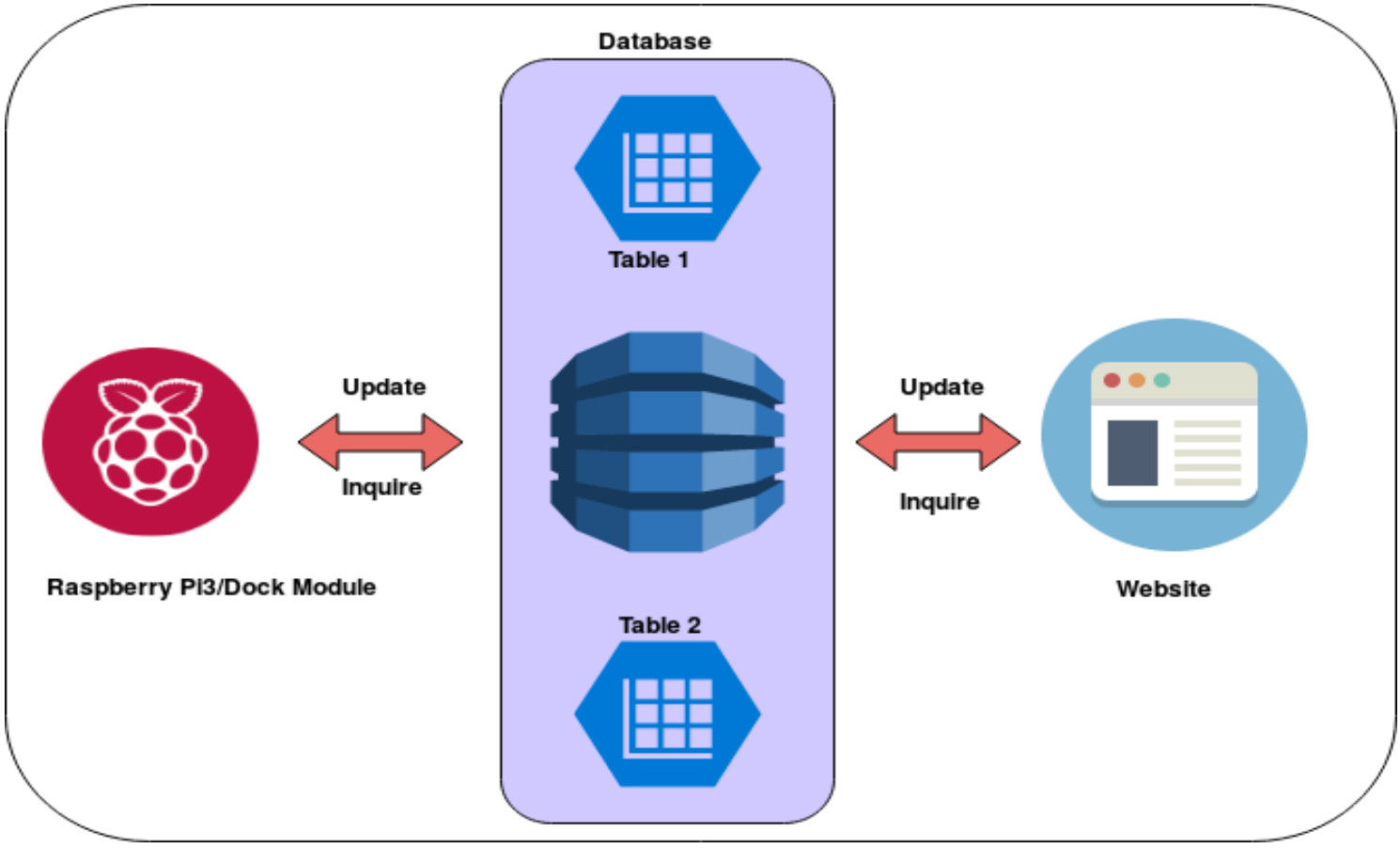
In the booking window, the interface allows a user to select the slot as well as the dock and check if the particular slot is available or not. Once the selected slot is open for booking, a user can hence book the slot.

The final window is **CONFIRMATION window** that provides confirmation regarding any booking or checkout. A user is notified using an SMS to his registered contact number about his booking or checkout operation.

All the QtPy interfaces are python based and the data about a user booking a slot or checking out any slot is maintained in a database using dynamodB.

2.**Database**

In this project, each time a user books a slot from the system or checks one out, related tables are updated. There are two tables Table 1 and Table 2; Table 1 containing the details about each registered users and their corresponding booking details and table 2 contains details about what all slots are available and what slots are occupied.



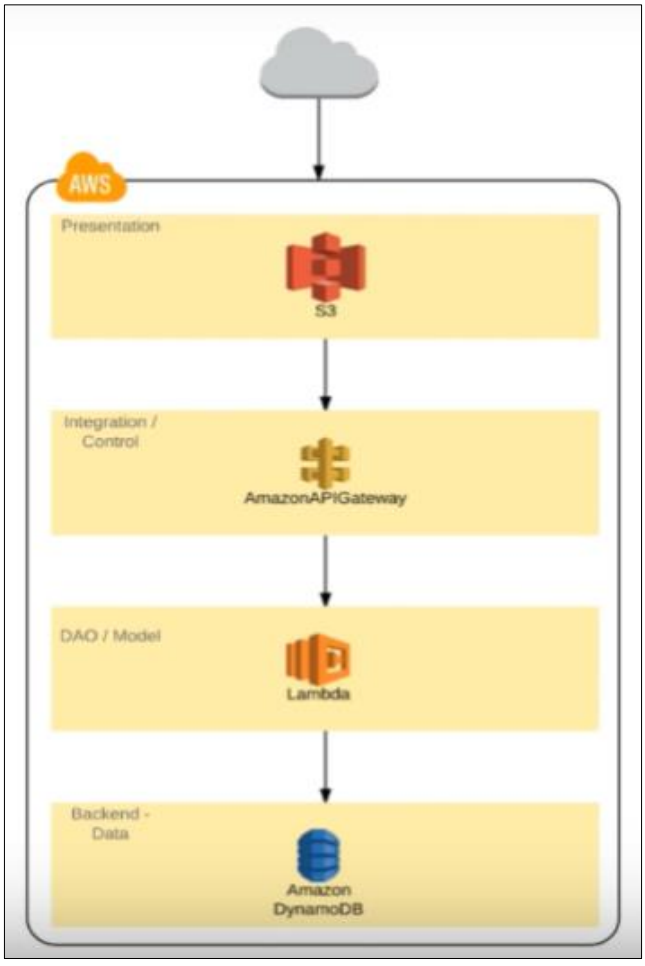
Daigram4: Database Operation

The database is maintained using DynamoDB. DynamoDB is a fully managed proprietary No SQL database service that supports key-value and documents data structures and is offered by Amazon.com as part of the Amazon Web Services portfolio. Each PyQt interface interacts with the database by sending a query every time a user inputs its user credentials or checks if a selected slot is available and the system in return checks if the user is valid or if the slot is available. Whenever a user books a slot or checks out any existing slot, the system updates the database tables using dynamoDB. The system requests data from the database as data query and updates the tables after every operation

While interacting with the database, lambda function is also implemented to inquire and upgrade the database every time a user interacts with the system.

3.**Website**

Daigram6: Website and AWS[ref]



The third part of the project is a website-based system that allows a user to book a slot or checkout any slot remotely using a website. The website interacts with the DynamoDB using Lambda Function, S3 and API Gateway.

Amazon API Gateway is a fully managed service that makes it easy for developers to create, publish, maintain, monitor and secure APIs at any scale. With a few clicks in the AWS Management Console, a user can create API that acts as a "front door" for applications to access data or functionality from any back-end services, such as workloads running on code running on AWS Lambda or any web application.

AWS Lambda is a compute service that lets a user run his code on a high-availability compute infrastructure and performs all the administration of the compute resources, including server and operating system maintenance, capacity provisioning and automatic scaling, code monitoring and logging. In this project the code supplied to the lambda function is written entirely in Node.js thus fulfilling the core requirement of the project.

Amazon S3 or Amazon Simple Storage Service is a simple storage service offered by Amazon Web Service (AWS) that provides object storage through a web service interface. Amazon S3 uses the same scalable storage infrastructure that Amazon.com uses to run its global e-commerce network.

Lessons learned

Website

Setting up the flow for website backend was a herculean task in itself as using the AWS services for serverless website hosting had few features which were recently incorporated in the AWS framework. To host a website, we need 4 primary elements namely 1. Presentation layer (S3 bucket) 2. Integration/control layer (Amazon API gateway) 3. DAO/Model (Lambda function) 4. Backend data storage (DynamoDB).

Data parsing received from dynamo DB and using it in Java script for lambda function was tricky as data saved in the Dynamo DB is saved in a particular format.

AWS Service integration issues

There are many issues associated while using the AWS services. Initially when we started using the AWS services using our starter accounts, we faced a lot of setbacks as we many services are restricted to use with a starter account. For example, we couldn't update the Dynamo DB without creating an IAM user which is not permitted in a starter account. Similarly, many other services were inaccessible with the starter account. We figured out this issue after a few days and then created a normal account and linked it with the AWS educate account to get a 100-dollar credit.

Next issue which we faced while working with the AWS services was to pass data from AWS SQS service onto the SNS service. After researching and reading about both the services we figured that SNS service uses its own queue and it cannot consume data from the SQS service. Hence we substituted SQS message queue with the SNS service after consulting with the professor.

QtPy application issue

while designing QtPy application, the project demanded multiple windows that can interact as per user input. In our project, there are five windows and each window responds to a particular user input, hence switching between different multiple windows dynamically itself had a great deal of learning and issues. Switching to a proper window after a user has input his/her request, allows a system to check for proper flags and hence schedule the display of appropriate window.

The button click function was called multiple times on clicking the button once. Then we discovered the auto-connect feature provided by the QT application. Next up was the issue of repainting the text label to update it as it doesn't update unless the QT window is loaded again.

References:

- https://en.wikipedia.org/wiki/Amazon_DynamoDB
- https://aws.amazon.com/dynamodb/
- https://aws.amazon.com/s3/
- https://aws.amazon.com/api-gateway/
- https://boto3.amazonaws.com/v1/documentation/api/latest/guide/dynamodb.html
- https://aws.amazon.com/sns/
- https://stackoverflow.com/questions/35422490/pyqt5-add-image-in-background-of-mainwindow-layout
- https://docs.aws.amazon.com/lambda/latest/dg/python-programming-model-handler-types.html
- https://pimylifeup.com/raspberry-pi-rfid-rc522/
- https://www.youtube.com/watch?v=Byhg9BBsbJw
- https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/CurrentAPI.html