3.3 Integration

Mar31/Apr 1 Database connections fully functional and following posted:

your repository/Documentation/GroupNameDatabase.pdf

/1 Participated: Online session

/4 Online demo

Database description (fits into poster template's results column along with mobile application description):

/1 Addresses connection to enterprise wireless (v.s. home/open Wi-Fi/hotspot)

/1 Database configuration mentioned

/1 Security considered

May replace printing/enclosure block in poster template:

/2 Unit and production testing considerations

3.3.1 Enterprise Wireless Connectivity

Connecting our SMART Parking lot prototype wirelessly to the database is done through the Raspberry Pi 4 Model B. The connection between the Raspberry Pi and the WiFi is done through a configuration file on the Pi. The configuration file is found in the /etc/ directory. This directory holds all the system-wide configuration files and holds the /etc/wpa_supplicant/wpa_supplicant.conf. WPA is an acronym for WiFi Protected Access this is essentially a security certification program to secure wireless host networks. This file holds all the information about networks configured on your device. We configured the Raspberry Pi for our prototype by connecting it to Humber's network named "eduroam". To configure the device to connect to eduroam we had to edit the wpa_supplicant.conf and add a network line like this:

```
network={
        ssid="eduroam"
        key_mgmt=WPA-EAP
        auth_alg=OPEN
        eap=PEAP
        identity="n########@humber.ca"          #humber account number
        password="HumberAccPassword"            #humber account password
        phase2="auth=MSCHAPV2"
        priority=999
        proactive_key_caching=1
}
```

We then restarted the RPi and were able to connect remotely to the device from VNC Viewer. The Raspberry Pi is now allowed to be remotely accessed from home using an open Wi-fi spot, and work on the Raspberry Pi is done wirelessly instead of connecting all the peripherals physically.

3.3.2 Database Configuration

The online database is configured based on essential criteria needed to access the mobile application, hardware and vice-versa. The main source of delegating data is done through the Google Firebase database. SQL scripts are used to design these tables through the Firebase structure and data is populated based on a SQL format.

The parking lot prototype establishes its connection through setting up and initializing the firebase/pyrebase credentials, to gain access to the Firebase API. This includes, adding the API Key to send/retrieve data. Pyrebase is essentially a wrapper class used with python programming, to allow access to Firebase and manipulate the data.

Following this, the dependencies were installed using the “sudo pip install pyrebase” command for each sensor to make the connection to the database.

There are five main data structures used in the project, along with four sub-siding tables used for the purpose of the mobile application, and other intended functionality of our parking application.

The ‘TestUsers’ data structure stores registration details specific to the user. The UID (user ID) acts as the primary key identifying each existing user and its registered account information.

Figure 93 - TestUsers table

The ‘ProximityData’ structure stores raw proximity values sent from the VCNL4010 hardware device, and is retrieved by the mobile application to display the real-time proximity levels of the lot.

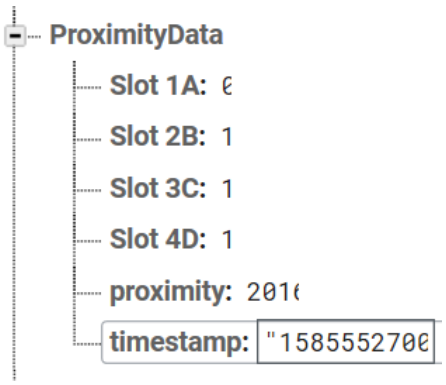

Figure 94 - ProximityData table

The ‘ParkingLocations’ table stores parking lot data under the specific UID of the current logged in user.

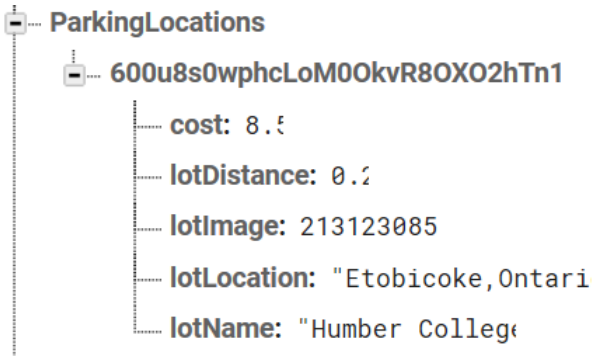

Figure 95 - ParkingLocations table

The ‘ParkingLocation’ table stores the reserved lot information. Data is retrieved through the use of foreign keys from the ‘ParkingLocations’ table. This includes the cost, location, duration, validity hours, type of pass, and the expiry time.

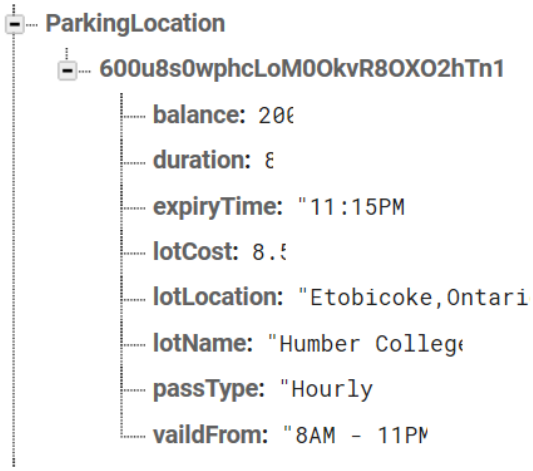

Figure 96 - ParkingLocation table

The ‘Orders’ table stores the payment processing details of the user after a parking pass purchase has been made. An OID is used in reference as a foreign key to each existing UID located in the ‘TestUsers’ table, working in accordance to identify each order based on a logged in/existing user.

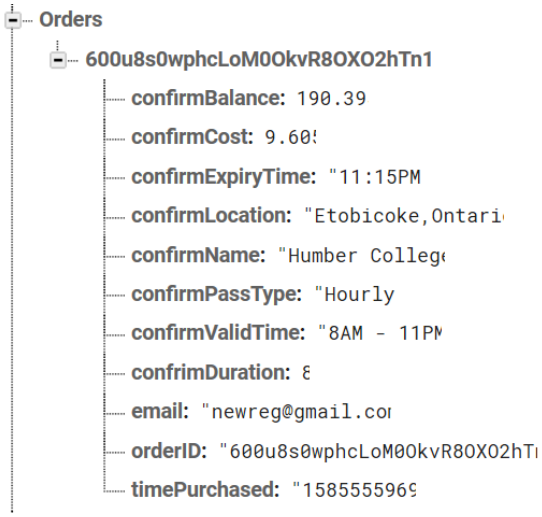

Figure 97 - Orders table

The ‘GateStatus’ table stores IR Break Beam entry/exit status and a timestamp to indicate the exact time an action is performed.

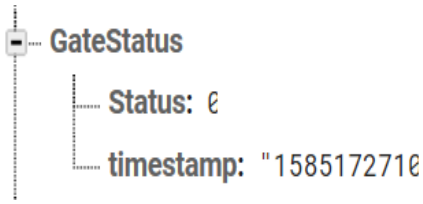

Figure 98 - GateStatus table

The ‘EntryStatus’ table stores a value of 0 each time the lot is full to notify the database of invalid entry.

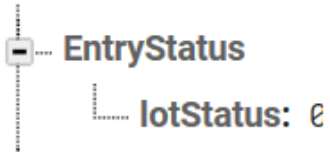

Figure 99 - EntryStatus table

The ‘Cars’ table stores the make, model, color, license plate number of the vehicle.

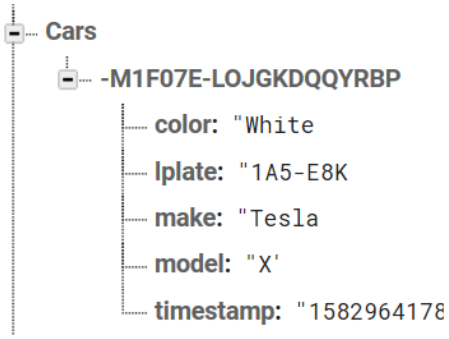

Figure 100 - Cars table

The ‘AdminControl’ table stores the status of the gate and sends a value of 1 or 0 to indicate an opening or closing.

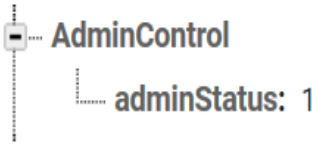

Figure 101 - AdminControl table

3.3.3 Security

Security is a vital feature in our parking application. The main goal of this feature being to prevent unwanted attacks, or potential harm to personal or sensitive data where data can be manipulated or abused. In our database setup, we have our read/write permissions set to limited access to outside users who may be trying to access the WatechPark database.

The read permission is set to “true” which allows any user to have access to the database, but for viewing purposes only. The write permission is set to “auth! = null” which means only authenticated users have access to modify the database.

Sensitive data such as passwords or payment information essential to a user are not stored in Firebase, through any form of encrypted credentials. This eliminates the possibility of a breach of data from any anonymous source that may serve as a threat, and ensures data integrity.

3.3.4 Unit Testing/Production Testing

Tests completed on individual parts will be described here. This includes the mobile application and Raspberry Pi parking lot prototype testing. Along with, the necessary

steps taken after deployment of the project in checking for any potential bugs, crashes or flaws in production.

WatechPark Android App

Test #1: Register

- Tested if entered registration data can be sent to Firebase and pushed to ‘TestUsers’ table

Test #2: Forgot Your Password

- Tested email and phone verification through Firebase provided functions
- Tested if email gets sent for new password request, notification is received through phone verification
- Tested for bugs, crashes after using email verification method, observed each method for any glitches, or possible crashes through the “debug” and logcat functions in Android Studio.

Test #3: Verify your Password

- Tested verification code process, if received notification with code to retrieve a new password
- Test if new password entered is updated for the user, and passwords match at Login.

Test #4: Login

- Test if email exists in Firebase data structure ‘TestUsers’, check for valid credentials

- If email exists in ‘Authentication’ section of Firebase and password credentials match go to Home screen, else display an error message

Test #5: Home Screen

- Tested if ‘Humber College’ parking lot is available in CardView structure along with a ‘View Details’ and ‘Reserve’ button, re-designed the menu to display a single lot choice, allowing a much faster, sleeker UI(user interface).
- Test when View Details button is pressed, information is displayed about the parking lot,
- Test when Reserve is pressed then it will allow you to get a parking pass, check Firebase if values have been added into ‘ParkingLocations’ table using the UID
- Check if user received notification on phone when Parking Pass is available.

Test #6: Parking Passes

- Check if data is sent and displayed from ‘ParkingLocation’ table and can access the reserved lot
- Test if ‘SELECT’ button sends the data to ‘ParkingLocation’ table and can be accessed through the Payment Screen

Test #7: Order History

- After parking pass is bought check if it appears on the Order History Screen using the ‘Orders’ table

Test #8: Payment

- Test if data is displayed based on the selected parking pass from Parking Passes screen using the ‘ParkingLocation’ table

- Check ‘Orders’ table on Firebase to make sure order went through once payment is processed

Test #9: Add a Car

- Test if users can add a car by entering make, model, color, and license plate number.
- Check if the data is sent to the Firebase data structure “Cars”

Test #10: Manage Cars

- Check if added cars from the ‘Cars’ table appear here.
- Test if user can edit or delete cars by pressing the coordinating buttons

Test #11: View Details Button (Parking Lot Data Screen)

- Test if ‘GateStatus’ value on Firebase is changed for entry and exit and colors change on app accordingly, along with a timestamp
- Test proximity value reading from Firebase using the ‘ProximtityData’ table, and that values is updated in real-time
- Test admin control sending values for gate control through ‘AdminControl’ table

SMART Parking Lot Prototype

Test #1: IR Break Beam

- Test if the sensor can detect if an object is blocking IR beam, and if receiver can detect the beam being broken
- Test if a HIGH value is received when the beam is detected, noEntry is sent to close gate, and LOW value when beam is not detected, entry is set to open gate using ‘GateStatus’

Test #2: VCNL Proximity

- Test if proximity values are sent based on if a vehicle approaches the sensor. Report the distance to Firebase under ‘ProximityData’ table, database should be updated according to sensor value
- Check if Slot 1A is available if the sensor reads a value less than or equal to 2500. If the value is greater than 5000 the space is taken. If value is between 2500 and 5000, car is approaching the space.

Test #3: Servo-Motor

- Once the IR beam receiver detects interference with IR beam it will raise the servo motor arm and allow entry/exit
- Check if ‘GateStatus’ on Firebase has changed value for when gate is allowing “entry” (1) for true condition or “noEntry” (0) for false condition. Gate will lower arm after 2 seconds and status will change to noEntry.