7.0 Appendix

Modified Code Files (Mobile Application)

ParkingLocationAdapter.java (main modification file for parking lot feature)

```
package com.example.watechpark;

import android.annotation.SuppressLint;
import android.app.Activity;
import android.app.Notification;
import android.app.NotificationManager;
import android.app.PendingIntent;
import android.content.Context;
import android.content.Intent;
import android.content.res.ColorStateList;
import android.os.Build;
import android.os.Bundle;
import android.text.TextUtils;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.Button;
import android.widget.EditText;
import android.widget.ImageView;
import android.widget.TextView;
import android.widget.Toast;

import androidx.annotation.NonNull;
import androidx.annotation.RequiresApi;
import androidx.appcompat.app.AlertDialog;
import androidx.appcompat.app.AppCompatActivity;
import androidx.core.app.NotificationCompat;
import androidx.core.content.ContextCompat;
import androidx.fragment.app.Fragment;
import androidx.fragment.app.FragmentManager;
import androidx.fragment.app.FragmentTabHost;
import androidx.fragment.app.FragmentTransaction;
import androidx.recyclerview.widget.RecyclerView;

import com.google.android.gms.tasks.OnCompleteListener;
import com.google.android.gms.tasks.Task;
import com.google.android.material.snackbar.Snackbar;
import com.google.firebase.auth.FirebaseAuth;
import com.google.firebase.auth.FirebaseUser;
import com.google.firebase.database.DataSnapshot;
import com.google.firebase.database.DatabaseError;
import com.google.firebase.database.DatabaseReference;
import com.google.firebase.database.FirebaseDatabase;
import com.google.firebase.database.ValueEventListener;

import java.text.DateFormat;
import java.text.SimpleDateFormat;
import java.util.Date;
import java.util.List;

/*
* Project: WatechPark
* Modified code file: ParkingLocationAdpater.java for adding in the parking lot status
feature(planned for Winter 2020)
* Modified by: Vikas Sharma(Student A)
* Group Members: Elias Sabbagh, George Alexandris
* Course: CENG 355
*/


public class ParkingLocationAdapter extends
RecyclerView.Adapter<ParkingLocationAdapter.ParkingLocationViewHolder> {

private Context context;
private List<ParkingLocation> parkingLocationList;

private FirebaseAuth mAuth;
private FirebaseUser user;
FirebaseAuth.AuthStateListener mAuthListener;
```

```
private DatabaseReference mDatabase;

private DatabaseReference mDatabase1;

private DatabaseReference mDatabase2;


private FirebaseDatabase database;

private long trueCond = 1;
private long falseCond = 0;

// variable to be used later on to indicate exit status
private int status0;



public ParkingLocationAdapter(Context context, List<ParkingLocation>
parkingLocationList) {
this.context = context;
this.parkingLocationList = parkingLocationList;
}


@NonNull
@Override
public ParkingLocationAdapter.ParkingLocationViewHolder
onCreateViewHolder(@NonNull ViewGroup parent, int viewType) {
View view = LayoutInflater.from(parent.getContext()).inflate(R.layout.fragment_home,
parent, false);
return new ParkingLocationViewHolder(view);

}

@Override
public void onBindViewHolder(@NonNull final
ParkingLocationAdapter.ParkingLocationViewHolder holder, final int position) {
// get the data for the main lot and the corresponding details from the ParkingLocation
data structure
final ParkingLocation parkingLocation = parkingLocationList.get(position);
holder.lotName.setText(parkingLocation.getLotName());
holder.lotLocation.setText(parkingLocation.getLotLocation());
holder.lotDistance.setText(String.valueOf(parkingLocation.getLotDistance() +
context.getString(R.string.m)));
holder.lotCost.setText(context.getString(R.string.dol3) + parkingLocation.getCost());
holder.lotImage.setImageDrawable(context.getResources().getDrawable(parkingLocatio
n.getLotImage(), null));


// Firebase initialization
mAuth = FirebaseAuth.getInstance();
user = mAuth.getCurrentUser();


// View Details button
holder.viewDetails.setOnClickListener(new View.OnClickListener() {
@Override
public void onClick(View v) {
// read the data from each respective data structure in Firebase for further use using a
DatabaseReference object
// get a reference to the ProximityData class for the VCNL4010 Proximity sensor
mDatabase = FirebaseDatabase.getInstance().getReference().child("ProximityData");
mDatabase1 = FirebaseDatabase.getInstance().getReference().child("Cars");
mDatabase2 = FirebaseDatabase.getInstance().getReference().child("GateStatus");
// listen for active changes in the ProximityData structure
mDatabase.addValueEventListener(new ValueEventListener() {
@RequiresApi(api = Build.VERSION_CODES.N)
@Override
public void onDataChange(@NonNull DataSnapshot dataSnapshot) {
// read and store the data inside the ProximityData structure into a object of the class
ProximityData proximityData = dataSnapshot.getValue(ProximityData.class);
// get the data stored in the Cars data structure and store it into a Cars object
Cars cars = dataSnapshot.getValue(Cars.class);
// use a intent to send the name of the chosen lot from the main menu to the "View
Details" pop-up(data screen)
        Intent intent = new Intent();
intent.putExtra(context.getString(R.string.put_name),parkingLocationList.get(position).g
etLotName());
// retrieve the name of the lot and store it into a String variable
String lotName = intent.getStringExtra(context.getString(R.string.getname));

//String proximity1 = ds.getValue(ProximityData.class).getProximity();

// create a new AlertDialog box
AlertDialog.Builder dialog = new AlertDialog.Builder(context);
// inflate the view for the layout from the res folder
final View view = LayoutInflater.from(context).inflate(R.layout.activity_lot_detail, null);
TextView text = view.findViewById(R.id.textView36);
// set the textview to the lot name using the lotName variable from the previous step
text.setText(lotName);
```

```
TextView status = view.findViewById(R.id.textView37);
final EditText adminPass = view.findViewById(R.id.editText16);
// get the value entered by the user in the EditText field, store into a string variable
final String plate = adminPass.getText().toString().trim();
//adminPass.setText("admin");
Button adminBtn = view.findViewById(R.id.button3);
// open gate button
adminBtn.setOnClickListener(new View.OnClickListener() {
@Override
public void onClick(View v) {
// check if what is entered by the user equals "admin"
if (adminPass.getText().toString().equals("admin")) {

// get a new object of the AdminControl class
AdminControl status = new AdminControl();
// if it equals/matches the "admin" string value then present a success message and
open the gate
Toast.makeText(context, "Admin Access Activated: Gate is OPEN",
Toast.LENGTH_LONG).show();

// get a instance or reference of the AdminControl class and push the value of 1(open
gate) to the AdminControl structure
FirebaseDatabase.getInstance().getReference("AdminControl").child("adminStatus").set
Value(status.getAdminEntry());
ImageView adminEntry = view.findViewById(R.id.imageEntry);
// set the color of the entry arrow to blue(indicate the gate is opened through admin
access)
adminEntry.setColorFilter(context.getResources().getColor(R.color.colorBlue));
} else {
AdminControl status = new AdminControl();
// else don't allow access and present a toast for unsuccessful entry
Toast.makeText(context, "Unauthorized Admin Access! Invalid Credentials",
Toast.LENGTH_LONG).show();
// set the value as false and push a 0 to the AdminControl structure in the adminStatus
child
FirebaseDatabase.getInstance().getReference("AdminControl").child("adminStatus").set
Value(status.getAdminEntry0());
ImageView adminEntry = view.findViewById(R.id.imageEntry);
// set the color to red for the entry arrow to indicate entry is not allowed
adminEntry.setColorFilter(context.getResources().getColor(R.color.colorRed));

}
}
});

TextView prox = view.findViewById(R.id.textView42);
```

```
// get the proximity from the ProximityData structure and set the TextView
prox.setText("RT Proximity: " + String.valueOf(proximityData.getProximity()));
TextView s1 = view.findViewById(R.id.textSlot1);
String slot1 = String.valueOf(proximityData.getSlot1A());
TextView s2 = view.findViewById(R.id.textSlot2);
String slot2 = String.valueOf(proximityData.getSlot2B());
TextView s3 = view.findViewById(R.id.textSlot3);
String slot3 = String.valueOf(proximityData.getSlot3C());
TextView s4 = view.findViewById(R.id.textSlot4);
String slot4 = String.valueOf(proximityData.getSlot4D());


// recommended way by Android Studio to basically get the long value of the proximity
int value = 0;
if (android.os.Build.VERSION.SDK_INT >= android.os.Build.VERSION_CODES.N) {
value = Math.toIntExact(proximityData.getProximity());
}

// if the proximity value is less than or equal to 2500
if (value <= 2500) {
ImageView en = view.findViewById(R.id.imageEntry);
// set the color of the entry arrow to green
en.setColorFilter(context.getResources().getColor(R.color.colorGreen));
ImageView spot1 = view.findViewById(R.id.imageSpot1);
spot1.setBackgroundResource(R.color.colorGreen);
ImageView spot2 = view.findViewById(R.id.imageSpot2);
spot2.setBackgroundResource(R.color.colorRed);
ImageView spot3 = view.findViewById(R.id.imageSpot3);
spot3.setBackgroundResource(R.color.colorRed);
ImageView spot4 = view.findViewById(R.id.imageSpot4);
spot4.setBackgroundResource(R.color.colorRed);

// set the text to each corresponding status
s1.setText("Open");
s2.setText("Occupied");
s3.setText("Occupied");
s4.setText("Occupied");
// update the overall status of the lot
status.setText(context.getString(R.string.setstatus2) + " Slot 1A is available -- (1/4)");
Toast.makeText(context, "LOT HC: ENTRY IS ALLOWED! GATE IS OPEN",
Toast.LENGTH_LONG).show();

spot1.setOnClickListener(new View.OnClickListener() {
@Override
public void onClick(View v) {
```

```
Snackbar.make(view, "Do you want to reserve this slot: Slot 1A?",
Snackbar.LENGTH_LONG)
.setAction("CANCEL", new View.OnClickListener() { // leave the reservation
@Override
public void onClick(View v) {


}
}).setAction("RESERVE", new View.OnClickListener() {
@Override
public void onClick(View v) {


mDatabase = FirebaseDatabase.getInstance().getReference("ParkingLocations");

final String name = parkingLocation.getLotName();
final String location = parkingLocation.getLotLocation();
final double distance =
Double.parseDouble(String.valueOf(parkingLocation.getLotDistance()));
final double cost = Double.parseDouble(String.valueOf(parkingLocation.getCost()));
final int image = Integer.parseInt(String.valueOf(parkingLocation.getLotImage()));

ParkingLocation parkingLocation1 = new ParkingLocation(name, location, distance,
cost, image);
mDatabase.child(user.getUid()).setValue(parkingLocation1);
mDatabase.child("ParkingLocations").child(FirebaseAuth.getInstance().getCurrentUser(
).getUid())
.setValue(parkingLocation1).addOnCompleteListener(new
OnCompleteListener<Void>() {
@Override
public void onComplete(@NonNull Task<Void> task) {
if (task.isSuccessful()) {
Toast.makeText(context, R.string.success_notif, Toast.LENGTH_SHORT).show();



// send a notification to the user if the lot information is sent successfully
Notification notification = new Notification.Builder(context)


.setTicker(context.getString(R.string.not))
.setContentTitle(context.getString(R.string.reserved_aspot) + name)
.setContentText(context.getString(R.string.check_pass))
.setSmallIcon(R.drawable.logo)
.build();
notification.flags = Notification.FLAG_AUTO_CANCEL;
NotificationManager notificationManager = (NotificationManager)
context.getSystemService(Context.NOTIFICATION_SERVICE);
notificationManager.notify(0, notification);


} else {
Toast.makeText(context, R.string.data_err, Toast.LENGTH_SHORT).show();
}
}
});

AdminControl status = new AdminControl();
TextView s1 = view.findViewById(R.id.textSlot1);
// set the status of Slot 1A as occupied
s1.setText("Occupied");
// update the visual spots and arrow colors
ImageView spot1 = view.findViewById(R.id.imageSpot1);
spot1.setBackgroundResource(R.color.colorRed);
ImageView en = view.findViewById(R.id.imageEntry);
en.setColorFilter(context.getResources().getColor(R.color.colorRed));
ImageView ex2 = view.findViewById(R.id.imageExit);
ex2.setColorFilter(context.getResources().getColor(R.color.colorGreen));
TextView status2 = view.findViewById(R.id.textView37);
Toast.makeText(context,"Slot 1A booked! Please proceed to Payment after selecting
your Parking Pass...",Toast.LENGTH_LONG).show();
// update the overall status to indicate all spots are taken
status2.setText(context.getString(R.string.setstatus2) + " All slots are occupied! --
(4/4)");
// push a 0 to the AdminControl data strcuture
FirebaseDatabase.getInstance().getReference("AdminControl").child("adminStatus").set
Value(status.getAdminEntry0());
Toast.makeText(context, "LOT HC: ENTRY IS NOT ALLOWED",
Toast.LENGTH_LONG).show();


}
}).show();

}
});
// check if proximity value is greater than 5000
} else if (value > 5000) {
// update the slot color changes
ImageView en2 = view.findViewById(R.id.imageEntry);
```

```
en2.setColorFilter(context.getResources().getColor(R.color.colorRed));
ImageView ex2 = view.findViewById(R.id.imageExit);
ex2.setColorFilter(context.getResources().getColor(R.color.colorGreen));
ImageView spot1 = view.findViewById(R.id.imageSpot1);
spot1.setBackgroundResource(R.color.colorRed);
// Slot 2B,3C,4D will remain occupied(with color red as the background)
ImageView spot2 = view.findViewById(R.id.imageSpot2);
spot2.setBackgroundResource(R.color.colorRed);
ImageView spot3 = view.findViewById(R.id.imageSpot3);
spot3.setBackgroundResource(R.color.colorRed);
ImageView spot4 = view.findViewById(R.id.imageSpot4);
spot4.setBackgroundResource(R.color.colorRed);
s1.setText("Occupied");
s2.setText("Occupied");
s3.setText("Occupied");
s4.setText("Occupied");
// if the user clicks on Slot1A when all spots are occupied show a toast message
spot1.setOnClickListener(new View.OnClickListener() {
@Override
public void onClick(View v) {
Toast.makeText(context," All slots are occupied currently! Please wait a while for a
opening...",Toast.LENGTH_LONG).show();
}
});
// update the overall status
status.setText(context.getString(R.string.setstatus2) + " All slots are occupied! -- (4/4)");
Toast.makeText(context, "LOT HC: ENTRY IS NOT ALLOWED! LOT IS FULL",
Toast.LENGTH_LONG).show();

// initialize a new EntryStatus object
EntryStatus stat = new EntryStatus();
// get the status and store into a variable
status0 = stat.getEntry();

// set this value to the EntryStatus structure for the lotStatus
FirebaseDatabase.getInstance().getReference("EntryStatus").child("lotStatus")
.setValue(status0);
// check if proximity value is in between the range of open or occupied
} else if (value > 2500 && value < 5000) {
ImageView en3 = view.findViewById(R.id.imageEntry);
en3.setColorFilter(context.getResources().getColor(R.color.colorGreen));
// have the Slot 1A change background color to blue
ImageView spot1 = view.findViewById(R.id.imageSpot1);
spot1.setBackgroundResource(R.color.colorBlue);
// update the status
```

```
status.setText(context.getString(R.string.setstatus2) + " Vehicle is approaching the
parking space");
// set the TextView of Slot1A to targeted(to indicate vehicle is approaching to Slot1A)
s1.setText("Targeted");
s2.setText("Occupied");
s3.setText("Occupied");
s4.setText("Occupied");
}
// set the view of the dialog to the layout
dialog.setView(view);
// show the dialog box
AlertDialog alertDialog = dialog.create();
alertDialog.show();
}

@Override
public void onCancelled(@NonNull DatabaseError databaseError) {


}
});

// read active changes occurring at the GateStatus data structure(of the IR Break Beam
Sensor)
mDatabase2.addValueEventListener(new ValueEventListener() {
@Override
public void onDataChange(@NonNull DataSnapshot dataSnapshot) {
long gate = dataSnapshot.child("Status").getValue(Long.class);
GateStatus g = dataSnapshot.getValue(GateStatus.class);
final View view = LayoutInflater.from(context).inflate(R.layout.activity_lot_detail, null);

// check if value is equal to 1
if (gate == 1) {
final Long time = System.currentTimeMillis() / 1000;
final String newTime = time.toString();

// retrieve the status of entry and the timestamp from the GateStatus data structure
Toast.makeText(context, "HC LOT: Car has ENTERED the lot! Time: " +
convertTimestamp(g.getTimestamp()), Toast.LENGTH_LONG).show();
}
// otherwise(0) for exit
if (gate == 0) {
Toast.makeText(context, "HC LOT: Car has EXITED the lot!",
Toast.LENGTH_LONG).show();
}
```

```
}

@Override
public void onCancelled(@NonNull DatabaseError databaseError) {

}
});
}
});

// if user clicks on anywhere on the Humber College(Parking Lot) CardView in the main
meenu
holder.itemView.setOnClickListener(new View.OnClickListener() {
@Override
public void onClick(View v) {
// read the data from each respective data structure in Firebase for further use using a
DatabaseReference object
mDatabase = FirebaseDatabase.getInstance().getReference().child("ProximityData");
mDatabase1 = FirebaseDatabase.getInstance().getReference().child("Cars");
mDatabase2 = FirebaseDatabase.getInstance().getReference().child("GateStatus");
// listen for active changes in the ProximityData structure
mDatabase.addValueEventListener(new ValueEventListener() {
@RequiresApi(api = Build.VERSION_CODES.N)
@Override
public void onDataChange(@NonNull DataSnapshot dataSnapshot) {
// read and store the data inside the ProximityData structure into a object of the class
ProximityData proximityData = dataSnapshot.getValue(ProximityData.class);
// get the data stored in the Cars data structure and store it into a Cars object
Cars cars = dataSnapshot.getValue(Cars.class);
// use a intent to send the name of the chosen lot from the main menu to the "View
Details" pop-up(data screen)
Intent intent = new Intent();
intent.putExtra(context.getString(R.string.put_name),
parkingLocationList.get(position).getLotName());
// retreieve the name of the lot and store it into a String variable
String lotName = intent.getStringExtra(context.getString(R.string.getname));

//String proximity1 = ds.getValue(ProximityData.class).getProximity();

// create a new AlertDialog box
AlertDialog.Builder dialog = new AlertDialog.Builder(context);
// inflate the view for the layout from the res folder
final View view = LayoutInflater.from(context).inflate(R.layout.activity_lot_detail, null);
TextView text = view.findViewById(R.id.textView36);
// set the textview to the lot name using the lotName variable from the previous step
text.setText(lotName);
```

```
TextView status = view.findViewById(R.id.textView37);
final EditText adminPass = view.findViewById(R.id.editText16);
// get the value entered by the user in the EditText field, store into a string variable
final String plate = adminPass.getText().toString().trim();
//adminPass.setText("admin");
Button adminBtn = view.findViewById(R.id.button3);
// open gate button
adminBtn.setOnClickListener(new View.OnClickListener() {
@Override
public void onClick(View v) {
// check if what is entered by the user equals "admin"
if (adminPass.getText().toString().equals("admin")) {

// get a new object of the AdminControl class
AdminControl status = new AdminControl();
// if it equals/matches the "admin" string value then present a success messgae and
open the gate
Toast.makeText(context, "Admin Access Activated: Gate is OPEN",
Toast.LENGTH_LONG).show();

// get a instance or reference of the AdminControl class and push the value of 1(open
gate) to the AdminControl structure
FirebaseDatabase.getInstance().getReference("AdminControl").child("adminStatus").set
Value(status.getAdminEntry());
ImageView adminEntry = view.findViewById(R.id.imageEntry);
// set the color of the entry arrow to blue(indicate the gate is opened through admin
access)
adminEntry.setColorFilter(context.getResources().getColor(R.color.colorBlue));
} else {
AdminControl status = new AdminControl();
// else don't allow access and present a toast for unsuccessful entry
Toast.makeText(context, "Unauthorized Admin Access! Invalid Credentials",
Toast.LENGTH_LONG).show();
// set the value as false and push a 0 to the AdminControl structure in the adminStatus
child
FirebaseDatabase.getInstance().getReference("AdminControl").child("adminStatus").set
Value(status.getAdminEntry0());
ImageView adminEntry = view.findViewById(R.id.imageEntry);
// set the color to red for the entry  arrow to indicate entry is not allowed
adminEntry.setColorFilter(context.getResources().getColor(R.color.colorRed));

}


}
});
```

```
TextView prox = view.findViewById(R.id.textView42);
// get the proximity from the ProximityData structure and set the TextView
prox.setText("RT Proximity: " + String.valueOf(proximityData.getProximity()));
TextView s1 = view.findViewById(R.id.textSlot1);
String slot1 = String.valueOf(proximityData.getSlot1A());
TextView s2 = view.findViewById(R.id.textSlot2);
String slot2 = String.valueOf(proximityData.getSlot2B());
TextView s3 = view.findViewById(R.id.textSlot3);
String slot3 = String.valueOf(proximityData.getSlot3C());
TextView s4 = view.findViewById(R.id.textSlot4);
String slot4 = String.valueOf(proximityData.getSlot4D());


// reccomended way by Android Studio to basically get the long value of the proximity
int value = 0;
if (android.os.Build.VERSION.SDK_INT >= android.os.Build.VERSION_CODES.N) {
value = Math.toIntExact(proximityData.getProximity());
}

// if the proximity value is less than or equal to 2500
if (value <= 2500) {
ImageView en = view.findViewById(R.id.imageEntry);
// set the color of the entry arrow to green
en.setColorFilter(context.getResources().getColor(R.color.colorGreen));
ImageView spot1 = view.findViewById(R.id.imageSpot1);
spot1.setBackgroundResource(R.color.colorGreen);
ImageView spot2 = view.findViewById(R.id.imageSpot2);
spot2.setBackgroundResource(R.color.colorRed);
ImageView spot3 = view.findViewById(R.id.imageSpot3);
spot3.setBackgroundResource(R.color.colorRed);
ImageView spot4 = view.findViewById(R.id.imageSpot4);
spot4.setBackgroundResource(R.color.colorRed);

// set the text to each corresponding status
s1.setText("Open");
s2.setText("Occupied");
s3.setText("Occupied");
s4.setText("Occupied");
// update the overall status of the lot
status.setText(context.getString(R.string.setstatus2) + " Slot 1A is available -- (1/4)");
Toast.makeText(context, "LOT HC: ENTRY IS ALLOWED! GATE IS OPEN",
Toast.LENGTH_LONG).show();

spot1.setOnClickListener(new View.OnClickListener() {
@Override
```

```
public void onClick(View v) {
Snackbar.make(view, "Do you want to reserve this slot: Slot 1A?",
Snackbar.LENGTH_LONG)
.setAction("CANCEL", new View.OnClickListener() { // leave the reservation
@Override
public void onClick(View v) {


}
}).setAction("RESERVE", new View.OnClickListener() {
@Override
public void onClick(View v) {


mDatabase = FirebaseDatabase.getInstance().getReference("ParkingLocations");

final String name = parkingLocation.getLotName();
final String location = parkingLocation.getLotLocation();
final double distance =
Double.parseDouble(String.valueOf(parkingLocation.getLotDistance()));
final double cost = Double.parseDouble(String.valueOf(parkingLocation.getCost()));
final int image = Integer.parseInt(String.valueOf(parkingLocation.getLotImage()));

ParkingLocation parkingLocation1 = new ParkingLocation(name, location, distance,
cost, image);
mDatabase.child(user.getUid()).setValue(parkingLocation1);
mDatabase.child("ParkingLocations").child(FirebaseAuth.getInstance().getCurrentUser(
).getUid())
.setValue(parkingLocation1).addOnCompleteListener(new
OnCompleteListener<Void>() {
@Override
public void onComplete(@NonNull Task<Void> task) {
if (task.isSuccessful()) {
Toast.makeText(context, R.string.success_notif, Toast.LENGTH_SHORT).show();



// send a notification to the user if the lot information is sent successfully
Notification notification = new Notification.Builder(context)


.setTicker(context.getString(R.string.not))
.setContentTitle(context.getString(R.string.reserved_aspot) + name)
.setContentText(context.getString(R.string.check_pass))
.setSmallIcon(R.drawable.logo)
.build();
notification.flags = Notification.FLAG_AUTO_CANCEL;
NotificationManager notificationManager = (NotificationManager)
context.getSystemService(Context.NOTIFICATION_SERVICE);
notificationManager.notify(0, notification);


} else {
Toast.makeText(context, R.string.data_err, Toast.LENGTH_SHORT).show();
}
}
});

AdminControl status = new AdminControl();
TextView s1 = view.findViewById(R.id.textSlot1);
// set the status of Slot 1A as occupied
s1.setText("Occupied");
// update the visual spots and arrow colors
ImageView spot1 = view.findViewById(R.id.imageSpot1);
spot1.setBackgroundResource(R.color.colorRed);
ImageView en = view.findViewById(R.id.imageEntry);
en.setColorFilter(context.getResources().getColor(R.color.colorRed));
ImageView ex2 = view.findViewById(R.id.imageExit);
ex2.setColorFilter(context.getResources().getColor(R.color.colorGreen));
TextView status2 = view.findViewById(R.id.textView37);
Toast.makeText(context,"Slot 1A booked! Please proceed to Payment after selecting
your Parking Pass...",Toast.LENGTH_LONG).show();
// update the overall status to indiate all spots are taken
status2.setText(context.getString(R.string.setstatus2) + " All slots are occupied! --
(4/4)");
// push a 0 to the AdminControl data strcuture
FirebaseDatabase.getInstance().getReference("AdminControl").child("adminStatus").set
Value(status.getAdminEntry0());
Toast.makeText(context, "LOT HC: ENTRY IS NOT ALLOWED",
Toast.LENGTH_LONG).show();


}
}).show();

}
});
// check if proximity value is greater than 5000
} else if (value > 5000) {
// update the slot color changes
```

```
ImageView en2 = view.findViewById(R.id.imageEntry);
en2.setColorFilter(context.getResources().getColor(R.color.colorRed));
ImageView ex2 = view.findViewById(R.id.imageExit);
ex2.setColorFilter(context.getResources().getColor(R.color.colorGreen));
ImageView spot1 = view.findViewById(R.id.imageSpot1);
spot1.setBackgroundResource(R.color.colorRed);
// Slot 2B,3C,4D will remain occupied(with color red as the background)
ImageView spot2 = view.findViewById(R.id.imageSpot2);
spot2.setBackgroundResource(R.color.colorRed);
ImageView spot3 = view.findViewById(R.id.imageSpot3);
spot3.setBackgroundResource(R.color.colorRed);
ImageView spot4 = view.findViewById(R.id.imageSpot4);
spot4.setBackgroundResource(R.color.colorRed);
s1.setText("Occupied");
s2.setText("Occupied");
s3.setText("Occupied");
s4.setText("Occupied");
// if the user clicks on Slot1A when all spots are occupied show a toast message
spot1.setOnClickListener(new View.OnClickListener() {
@Override
public void onClick(View v) {
Toast.makeText(context," All slots are occupied currently! Please wait a while for a
opening...",Toast.LENGTH_LONG).show();
}
});
// update the overall status
status.setText(context.getString(R.string.setstatus2) + " All slots are occupied! -- (4/4)");
Toast.makeText(context, "LOT HC: ENTRY IS NOT ALLOWED! LOT IS FULL",
Toast.LENGTH_LONG).show();

// initialize a new EntryStatus object
EntryStatus stat = new EntryStatus();
// get the status and store into a variablee
status0 = stat.getEntry();

// set this value to the EntryStatus structure for the lotStatus
FirebaseDatabase.getInstance().getReference("EntryStatus").child("lotStatus")
.setValue(status0);
// check if proximity value is in between the range of open or occupied
} else if (value > 2500 && value < 5000) {
ImageView en3 = view.findViewById(R.id.imageEntry);
en3.setColorFilter(context.getResources().getColor(R.color.colorGreen));
// have the Slot 1A change backgrund color to blue
ImageView spot1 = view.findViewById(R.id.imageSpot1);
spot1.setBackgroundResource(R.color.colorBlue);
// update the status
```

```
status.setText(context.getString(R.string.setstatus2) + " Vehicle is approaching the
parking space");
// set the TextView of Slot1A to targeted(to indicate vehicle is approaching to Slot1A)
s1.setText("Targeted");
s2.setText("Occupied");
s3.setText("Occupied");
s4.setText("Occupied");


}

// set the view of the dialog to the layout

dialog.setView(view);


// show the dialog box
AlertDialog alertDialog = dialog.create();
alertDialog.show();


}


@Override
public void onCancelled(@NonNull DatabaseError databaseError) {


}
});

// read active changes occuring at the GateStatus data structure(of the IR Break Beam
Sensor)
mDatabase2.addValueEventListener(new ValueEventListener() {
@Override
public void onDataChange(@NonNull DataSnapshot dataSnapshot) {
long gate = dataSnapshot.child("Status").getValue(Long.class);
GateStatus g = dataSnapshot.getValue(GateStatus.class);
final View view = LayoutInflater.from(context).inflate(R.layout.activity_lot_detail, null);

// check if value is equal to 1
if (gate == 1) {
final Long time = System.currentTimeMillis() / 1000;
final String newTime = time.toString();

// retrieve the status of entry and the timestamp from the GateStatus data structure
```

```
Toast.makeText(context, "HC LOT: Car has ENTERED the lot! Time: " +
convertTimestamp(g.getTimestamp()), Toast.LENGTH_LONG).show();
}
// otherwise(0) for exit
if (gate == 0) {
Toast.makeText(context, "HC LOT: Car has EXITED the lot!",
Toast.LENGTH_LONG).show();
}


}

@Override
public void onCancelled(@NonNull DatabaseError databaseError) {

}
});

}
});

}

// return the size of the array list for the parking location displayed on the main menu
@Override
public int getItemCount() {
return parkingLocationList.size();
}

// convert the log format of the timestamp into a readable string form
private String convertTimestamp(String timestamp){
long yourSeconds = Long.valueOf(timestamp);
Date mDate = new Date(yourSeconds * 1000);
DateFormat df = new SimpleDateFormat(context.getString(R.string.date_for));
return df.format(mDate);

}


public class ParkingLocationViewHolder extends RecyclerView.ViewHolder {

ImageView lotImage;
TextView lotName, lotLocation, lotDistance, lotCost;
Button viewDetails, reserveButton,openGate;

TextView proximity, timestamp;
```

```
public ParkingLocationViewHolder(@NonNull View itemView) {
super(itemView);

lotImage = itemView.findViewById(R.id.image_card);
lotName = itemView.findViewById(R.id.text_locname);
lotLocation = itemView.findViewById(R.id.textView39);
lotDistance = itemView.findViewById(R.id.text_dist);
lotCost = itemView.findViewById(R.id.textView40);
viewDetails = itemView.findViewById(R.id.button11);
reserveButton = itemView.findViewById(R.id.button10);
openGate = itemView.findViewById(R.id.button3);

}
}
}
```

For reference this is the link to the modified ParkingLocationAdapter.java code for our project (as shown above):

https://github.com/VikasCENG/WatechPark/blob/master/WatechPark(APP)/app/src/main/java/com/example/watechpark/ParkingLocationAdapter.java

MainMenu.java

```
package com.example.watechpark;

import android.content.DialogInterface;
import android.content.Intent;
import android.net.Uri;
import android.os.Bundle;

import com.bumptech.glide.Glide;
import com.bumptech.glide.request.RequestOptions;
import com.example.watechpark.ui.AddACarFragment;
import com.example.watechpark.ui.Home.HomeFragment;
import com.example.watechpark.ui.Settings.PaymentFragment;
import com.google.android.gms.auth.api.signin.GoogleSignIn;
import com.google.android.gms.auth.api.signin.GoogleSignInAccount;
import com.google.android.gms.tasks.OnSuccessListener;
import com.google.android.material.bottomappbar.BottomAppBar;
import com.google.android.material.bottomnavigation.BottomNavigationView;
import com.google.android.material.floatingactionbutton.FloatingActionButton;
import com.google.android.material.snackbar.Snackbar;
import android.view.MenuItem;
import android.view.View;

import androidx.annotation.NonNull;
import androidx.appcompat.app.AlertDialog;
import androidx.appcompat.widget.SearchView;
import androidx.fragment.app.Fragment;
import androidx.fragment.app.FragmentManager;
import androidx.fragment.app.FragmentTransaction;
import androidx.navigation.NavController;
import androidx.navigation.Navigation;
import androidx.navigation.ui.AppBarConfiguration;
import androidx.navigation.ui.NavigationUI;

import com.google.android.material.navigation.NavigationView;
import com.google.firebase.auth.FirebaseAuth;
import com.google.firebase.auth.FirebaseUser;
import com.google.firebase.storage.FirebaseStorage;
import com.google.firebase.storage.StorageReference;
import com.squareup.picasso.Picasso;

import androidx.drawerlayout.widget.DrawerLayout;

import androidx.appcompat.app.AppCompatActivity;
import androidx.appcompat.widget.Toolbar;
import androidx.recyclerview.widget.LinearLayoutManager;
import androidx.recyclerview.widget.RecyclerView;

import android.view.Menu;
import android.widget.ArrayAdapter;
import android.widget.Button;
import android.widget.EditText;
import android.widget.ImageButton;
import android.widget.ImageView;
import android.widget.TextView;
import android.widget.Toast;

import java.util.ArrayList;
import java.util.List;

import de.hdodenhof.circleimageview.CircleImageView;

public class MainMenu extends AppCompatActivity {

    private AppBarConfiguration mAppBarConfiguration;
    RecyclerView recyclerView;
    ParkingLocationAdapter adapter;
    List<ParkingLocation> parkingList;


    private ImageView i1,i2,i3;
    private Button b1,b2,b3;
    private TextView t1,t2,t3,t4,t5,t6,t7,t8,t9,t10, t11, userName, userEmail;

     FirebaseAuth mAuth;
     FirebaseUser user;

    private FirebaseStorage firebaseStorage;

    protected Button addACar;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main_menu);

        parkingList = new ArrayList<>();

        recyclerView = findViewById(R.id.recyclerView2);
        recyclerView.setHasFixedSize(true);

        recyclerView.setLayoutManager(new LinearLayoutManager(this));

        parkingList.add(new ParkingLocation(getString(R.string.queens_parkway),
getString(R.string.loc0), 1.6, 6.50 , R.drawable.queens));
        parkingList.add(new ParkingLocation(getString(R.string.humber),
getString(R.string.loc1), 0.2, 8.50 , R.drawable.humber2));



        adapter = new ParkingLocationAdapter(this, parkingList);
        recyclerView.setAdapter(adapter);

       // userName = findViewById(R.id.nav_user);
       // userEmail = findViewById(R.id.nav_email);

        Toolbar toolbar = findViewById(R.id.toolbar);
        setSupportActionBar(toolbar);

        mAuth = FirebaseAuth.getInstance();
        user = mAuth.getCurrentUser();

        firebaseStorage = FirebaseStorage.getInstance();
        StorageReference storageReference = firebaseStorage.getReference();


        getSupportActionBar().setTitle(getString(R.string.home_title));


        DrawerLayout drawer = findViewById(R.id.drawer_layout);
        NavigationView navigationView = findViewById(R.id.nav_view);




        // Passing each menu ID as a set of Ids because each
        // menu should be considered as top level destinations.
        mAppBarConfiguration = new AppBarConfiguration.Builder(
                R.id.nav_home, R.id.nav_passes, R.id.nav_orderhistory, R.id.nav_addacar,
R.id.nav_manage,
                R.id.nav_payment, R.id.nav_settings, R.id.nav_help, R.id.nav_about)
                .setDrawerLayout(drawer)
                .build();
        NavController navController = Navigation.findNavController(this,
R.id.nav_host_fragment);
        NavigationUI.setupActionBarWithNavController(this, navController,
mAppBarConfiguration);
        NavigationUI.setupWithNavController(navigationView, navController);




        displayHeaderInfo();
    }


    @Override
    public boolean onOptionsItemSelected(@NonNull MenuItem item) {

        int id = item.getItemId();
        if(id == R.id.action_acc){
            Intent i = new Intent(getApplicationContext(), AccountManagement.class);
            startActivity(i);

        }else if(id == R.id.action_logout){
             logOutUser();
        }

        return super.onOptionsItemSelected(item);
    }

    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
        // Inflate the menu; this adds items to the action bar if it is present.
        getMenuInflater().inflate(R.menu.main_menu, menu);
        return true;
    }



    @Override
    public boolean onSupportNavigateUp() {
        NavController navController = Navigation.findNavController(this,
R.id.nav_host_fragment);
        return NavigationUI.navigateUp(navController, mAppBarConfiguration)
                || super.onSupportNavigateUp();
    }

    private void logOutUser(){
        AlertDialog.Builder builder = new AlertDialog.Builder(this);
        builder.setTitle(getString(R.string.confirm_out));
        builder.setMessage(getString(R.string.log_outconf));
        builder.setPositiveButton(getString(R.string.y), new
DialogInterface.OnClickListener() {
            @Override
            public void onClick(DialogInterface dialog, int which) {
                mAuth.signOut();
                startActivity(new Intent(MainMenu.this, LoginRegister.class));
                dialog.dismiss();
            }
        });

        builder.setNegativeButton(getString(R.string.n), new
DialogInterface.OnClickListener() {
            @Override
            public void onClick(DialogInterface dialog, int which) {
                dialog.cancel();
            }
        });
        AlertDialog alert = builder.create();
        alert.show();
    }

    private void displayHeaderInfo() {
            NavigationView navigationView = findViewById(R.id.nav_view);
            View headerView = navigationView.getHeaderView(0);
            final TextView userName = headerView.findViewById(R.id.nav_user);
            final TextView userEmail = headerView.findViewById(R.id.nav_email);
            final CircleImageView navImage =
headerView.findViewById(R.id.imageHeader);

            final ParkingPassInfo parkingPassInfo = new ParkingPassInfo();

            //userName.setText(user.getDisplayName());
            //userEmail.setText((user.getEmail()));

            //Picasso.get().load(user.getPhotoUrl()).fit().centerCrop().into(navImage);

//Glide.with(this).load(user.getPhotoUrl()).apply(RequestOptions.circleCropTransform()).
into(navImage);

            firebaseStorage = FirebaseStorage.getInstance();
            StorageReference storageReference = firebaseStorage.getReference();

storageReference.child(user.getUid()).child(getString(R.string.prof_2)).getDownloadUrl(
).addOnSuccessListener(new OnSuccessListener<Uri>() {
                @Override
                public void onSuccess(Uri uri) {
                    Picasso.get().load(uri).fit().centerCrop().into(navImage);

                    userEmail.setText(user.getEmail());
                    userName.setText(user.getUid());
                }
            });


//Glide.with(this).applyDefaultRequestOptions(RequestOptions.circleCropTransform()).a
sBitmap().load(user.getPhotoUrl()).into(navImage);
        }


}
```

For reference this is the link to the modified MainMenu.java code for our project (as shown above):

https://github.com/VikasCENG/WatechPark/blob/master/WatechPark(APP)/app/src/main/java/com/example/watechpark/MainMenu.java

ParkingPassesFragment.java

```
package com.example.watechpark;

import android.os.Bundle;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.TextView;

import androidx.annotation.Nullable;
import androidx.annotation.NonNull;
import androidx.fragment.app.Fragment;
import androidx.fragment.app.FragmentManager;
import androidx.lifecycle.Observer;
import androidx.lifecycle.ViewModelProviders;
import androidx.recyclerview.widget.LinearLayoutManager;
import androidx.recyclerview.widget.RecyclerView;

import com.example.watechpark.R;
import com.example.watechpark.ui.Home.HomeFragment;
import com.google.firebase.database.DataSnapshot;
import com.google.firebase.database.DatabaseError;
import com.google.firebase.database.DatabaseReference;
import com.google.firebase.database.FirebaseDatabase;
import com.google.firebase.database.ValueEventListener;

import java.io.Serializable;
import java.util.ArrayList;
import java.util.List;

public class ParkingPassesFragment extends Fragment {

    private ParkingPassesViewModel parkingPassesViewModel;

    RecyclerView recyclerView;
    ParkingPassesAdapter adapter;
    List<ParkingPassInfo> parkingPassInfoList;

    private DatabaseReference databaseReference;

    public View onCreateView(@NonNull LayoutInflater inflater,
                             ViewGroup container, Bundle savedInstanceState) {
```

```
        parkingPassesViewModel =
                ViewModelProviders.of(this).get(ParkingPassesViewModel.class);
        View root = inflater.inflate(R.layout.parking_passes_fragment, container, false);


        parkingPassInfoList = new ArrayList<>();


        recyclerView = root.findViewById(R.id.recylerView3);
        recyclerView.setHasFixedSize(true);

        recyclerView.setLayoutManager(new LinearLayoutManager(getContext()));
        databaseReference =
FirebaseDatabase.getInstance().getReference("ParkingLocation");
        databaseReference.addValueEventListener(new ValueEventListener() {
            @Override
            public void onDataChange(@NonNull DataSnapshot dataSnapshot) {
                if(dataSnapshot.exists()){
                    for(DataSnapshot ds: dataSnapshot.getChildren()){
                        ParkingPassInfo p = ds.getValue(ParkingPassInfo.class);
                        parkingPassInfoList.add(p);

                        parkingPassInfoList.add(new ParkingPassInfo(getString(R.string.hc),
getString(R.string.lot_loc1), 8.50,getString(R.string.pass2),8,getString(R.string.valid2),
getString(R.string.t2), 200));



                    }

                    adapter = new ParkingPassesAdapter(getContext(), parkingPassInfoList);
                    recyclerView.setAdapter(adapter);
                }
            }

            @Override
            public void onCancelled(@NonNull DatabaseError databaseError) {


            }
        });

        return root;
    }
}
```

For reference this is the link to the modified ParkingPassesFragment.java code for our project (as shown above):

https://github.com/VikasCENG/WatechPark/blob/master/WatechPark(APP)/app/src/main/java/com/example/watechpark/ParkingPassesFragment.java

AddACarFragment.java

```
package com.example.watechpark.ui;

import android.os.Bundle;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.view.WindowManager;
import android.widget.Button;
import android.widget.EditText;
import android.widget.TextView;
import android.widget.Toast;

import androidx.annotation.Nullable;
import androidx.annotation.NonNull;
import androidx.fragment.app.Fragment;
import androidx.lifecycle.Observer;
import androidx.lifecycle.ViewModelProviders;

import com.example.watechpark.Cars;
import com.example.watechpark.R;
import com.google.android.gms.tasks.OnCompleteListener;
import com.google.android.gms.tasks.Task;
import com.google.firebase.auth.AuthResult;
import com.google.firebase.auth.FirebaseAuth;
import com.google.firebase.auth.FirebaseUser;
import com.google.firebase.database.DataSnapshot;
import com.google.firebase.database.DatabaseError;
import com.google.firebase.database.DatabaseReference;
import com.google.firebase.database.FirebaseDatabase;
import com.google.firebase.database.ValueEventListener;

public class AddACarFragment extends Fragment {

    private AddACarViewModel addACarViewModel;

    private FirebaseAuth mAuth;
    private FirebaseUser user;
    FirebaseAuth.AuthStateListener mAuthListener;
    private DatabaseReference mDatabase;

    private TextView label;
    private EditText edit_make, edit_model, edit_number, edit_color;

    Task<AuthResult> task = FirebaseAuth.getInstance().signInAnonymously();


    public View onCreateView(@NonNull LayoutInflater inflater,
                             ViewGroup container, Bundle savedInstanceState) {
        addACarViewModel =
                ViewModelProviders.of(this).get(AddACarViewModel.class);
        View root = inflater.inflate(R.layout.add_acar_fragment, container, false);

        mAuth = FirebaseAuth.getInstance();
        user = mAuth.getCurrentUser();
        mDatabase = FirebaseDatabase.getInstance().getReference();


        edit_make = root.findViewById(R.id.edit_make);
        label = root.findViewById(R.id.textView32);
        edit_model = root.findViewById(R.id.edit_model);
        edit_number = root.findViewById(R.id.edit_number);
        edit_color = root.findViewById(R.id.edit_color);
        Button button = (Button) root.findViewById(R.id.button_add);


        button.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                addCar();
            }
        });


        return root;


    }

    private void addCar() {
        final String make = edit_make.getText().toString().trim();
        final String model = edit_model.getText().toString().trim();
        final String color = edit_color.getText().toString().trim();
```

```
        final String licensePlate = edit_number.getText().toString().trim();
        final Long time = System.currentTimeMillis() / 1000;
        final String timestamp = time.toString();

        if (make.isEmpty()) {
            edit_make.setError(getString(R.string.make));
            edit_make.requestFocus();
        } else if (model.isEmpty()) {
            edit_model.setError(getString(R.string.model));
            edit_model.requestFocus();
        } else if (color.isEmpty()) {
            edit_color.setError(getString(R.string.color));
            edit_color.requestFocus();
        } else if (licensePlate.isEmpty()) {
            edit_number.setError(getString(R.string.lic));
            edit_number.requestFocus();
        } else {

            Cars cars = new Cars(make, model, color, licensePlate, timestamp);
            mDatabase.child("Cars")
                    .push().setValue(cars).addOnCompleteListener(new
OnCompleteListener<Void>() {
                @Override
                public void onComplete(@NonNull Task<Void> task) {
                    if (task.isSuccessful()) {
                        Toast.makeText(getContext(), R.string.car_added,
Toast.LENGTH_SHORT).show();
                    } else {
                        Toast.makeText(getContext(), R.string.car_failed,
Toast.LENGTH_SHORT).show();
                    }
                }
            });


        }
    }
}
```

For reference this is the link to the modified AddACarFragment.java code for our project (as shown above):

https://github.com/VikasCENG/WatechPark/blob/master/WatechPark(APP)/app/src/main/java/com/example/watechpark/ui/AddACarFragment.java

AdminControl.java (for Servo motors)

```
package com.example.watechpark;

public class AdminControl {

    private int adminEntry = 1;
    private int adminEntry0 = 0;

    public AdminControl(int adminEntry, int adminEntry0) {
        this.adminEntry = adminEntry;
        this.adminEntry0 = adminEntry0;
    }

    public AdminControl() {
    }

    public int getAdminEntry() {
        return adminEntry;
    }

    public void setAdminEntry(int adminEntry) {
        this.adminEntry = adminEntry;
    }

    public int getAdminEntry0() {
        return adminEntry0;
    }

    public void setAdminEntry0(int adminEntry0) {
        this.adminEntry0 = adminEntry0;
    }
}
```

For reference this is the link to the added AdminControl.java class for our project (as shown above):

https://github.com/VikasCENG/WatechPark/blob/master/WatechPark(APP)/app/src/main/java/com/example/watechpark/AdminControl.java

EntryStatus.java(for IR Break Beam sensor)

```
package com.example.watechpark;

public class EntryStatus {

    private int entry = 0;

    public EntryStatus(int entry) {
        this.entry = entry;
    }

    public EntryStatus() {
    }

    public int getEntry() {
        return entry;
    }

    public void setEntry(int entry) {
        this.entry = entry;
    }
```

For reference this is the link to the added EntryStatus.java class for our project (as shown above):

https://github.com/VikasCENG/WatechPark/blob/master/WatechPark(APP)/app/src/main/java/com/example/watechpark/EntryStatus.java

GateStatus.java(for IR Break Beam Sensor)

```
package com.example.watechpark;

public class GateStatus {

    private long gStatus;
    private String timestamp;

    public GateStatus(long gStatus, String timestamp) {
        this.gStatus = gStatus;
        this.timestamp = timestamp;
    }
    public GateStatus() {
    }

    public long getgStatus() {
        return gStatus;
    }

    public void setgStatus(long gStatus) {
        this.gStatus = gStatus;
    }

    public String getTimestamp() {
        return timestamp;
    }

    public void setTimestamp(String timestamp) {
        this.timestamp = timestamp;
    }
}
}
```

For reference this is the link to the added GateStatus.java class for our project (as shown above):

https://github.com/VikasCENG/WatechPark/blob/master/WatechPark(APP)/app/src/main/java/com/example/watechpark/EntryStatus.java

ProximityData.java (for VCNL4010 Proximity Sensor)

```
package com.example.watechpark;

public class ProximityData {

    private long proximity;
    private String timestamp;
    private long slot1A;
    private long slot2B;
    private long slot3C;
    private long slot4D;

    public ProximityData() {
    }
    public ProximityData(long proximity, String timestamp, long slot1A, long slot2B, long
slot3C, long slot4D) {
        this.proximity = proximity;
        this.timestamp = timestamp;
        this.slot1A = slot1A;
        this.slot2B = slot2B;
        this.slot3C = slot3C;
        this.slot4D = slot4D;


    }

    public long getProximity() {
        return proximity;
    }

    public void setProximity(long proximity) {
        this.proximity = proximity;
    }

    public String getTimestamp() {
        return timestamp;
    }

    public void setTimestamp(String timestamp) {
        this.timestamp = timestamp;
    }

    public long getSlot1A() { return slot1A; }

    public void setSlot1A(long slot1A) {
        this.slot1A = slot1A;
    }

    public long getSlot2B() {
        return slot2B;
    }

    public void setSlot2B(long slot2B) {
        this.slot2B = slot2B;
    }

    public long getSlot3C() {
        return slot3C;
    }
    public void setSlot3C(long slot3C) {
        this.slot3C = slot3C;
    }

    public long getSlot4D() {
        return slot4D;
    }

    public void setSlot4D(long slot4D) {
        this.slot4D = slot4D;
    }
}
```

For reference this is the link to the modified ProximityData.java class for our project (as shown above):

https://github.com/VikasCENG/WatechPark/blob/master/WatechPark(APP)/app/src/main/java/com/example/watechpark/ProximityData.java

7.2 Application code

This section of the report will present the final integrated mobile application in its entirety and the end product assembled throughout the duration of the Winter 2020 semester. We will be addressing the current state of the mobile application as the project nears completion, along with the finalized features of the login/authentication activity, various screens available to the consumer which will be further represented through the data visualization activity screens implemented throughout the application. Finally, through the action control activity we will address and present the final implemented parking lot feature proposed previously through our mobile application status update. Towards the end of this section, will be the modified code files organized into the Appendix, along with the link to the complete mobile application code to be accessed through our project repository.

Demo

/1 Hardware present?

/1 Memo by student A

/1 Login activity

/1 Data visualization activity

/1 Action control activity

Report

/1 Login activity

/1 Data visualization activity

/1 Action control activity

/1 Modified Code Files in Appendix

/1 Link to Complete Code in Repository

Memo for Final Mobile Application

At this stage of the project as it nears completion, our team has fulfilled all of the planned objectives initially stated throughout our previous status updates for the mobile application. Along with some minor improvising which had to be made due to time limitations, or software limitations to meet the requirements of our application in its entirety. Therefore, we are proud to announce that we have fully completed the mobile application and have met our goals stated for our parking application.

Development of the mobile application began in the previous Fall 2019 semester and was carried into the Winter 2020 semester where we had to integrate the hardware with the mobile application including each sensor/effector, to be able to talk back and forth from the hardware to the mobile application and vice-versa. Currently, we have completed the mobile application with all proposed features from the initial status update, along with some minor tweaks and adjustments that needed to be made to meet the requirements of the project. The mobile application works alongside an online database structure through the Firebase database and on-site devices which include the VCNL4010 Proximity sensor, IR Break Beam Sensor, and the 2 servo motors running alongside the PCA9685 servo controller.

Similar to previously, the application follows a login and authentication structure and is designed with various screens in mind to support our consumer application. This includes options to add a car, manage added cars, view real-time data specific to a chosen parking lot, and provide status updates/changes through the supporting sensors/effectors and the data sent and retrieved by the online database. Other features include consumer abilities to reserve a spot in a parking lot, select a parking pass, and be guided to payment services and view order history/transactions entered by the user. As well as access settings, customize language preferences and other in-app options.

Although, the main goal of this semester and for our team in completing the mobile application was to integrate the proposed data visualization feature of the parking lot, provide real-time data through status updates/changes, monitor and track statuses of individual spots during different intervals and provide capacity management techniques. This includes, gate control through entry/exit with use of the hardware. In terms of these planned tasks, we have now finally deployed the parking lot data feature vital to our parking application, along with other key developments and additions added to enhance the final end product.

Development for the mobile application began in early January and was gradually being worked on and ended on March 20, 2020, in line with our planned schedule of events/milestones to be met from the beginning of the semester. Now, we have successfully integrated each sensor to do its job in displaying and replicating the data sent from the sensors/effectors to the online database and have retrieved the data to be displayed through the mobile application. This includes the following, data sent by the VCNL4010 Proximity sensor is sent to the online database through proximity readings and manipulated through the application code. This data is used to display the parking lot changes, provide status updates in real-time based on the occurrences of the on-site devices physically there on the parking lot model. For example, the added data implementation feature now portrays a visual representation of the lot, and four parking spots to indicate the open or occupied slots with one active spot listening to proximity changes and status updates denoted by Slot 1A.

We also made use of the IR Break Beam sensor to show the status changes on entry/exit of the gate and have that data be replicated and tracked to be notified to the consumer. In addition, with the support of the professor we also were able to successfully implement an admin feature, adding another layer of user capability and access to our application. Based on the advice received, our team decided it was best to move forward and implement an admin control feature with an option to be able to control the hardware utilized at entry from the mobile application, without having the need of having to design separate screens. This in return, also helped reduce the amount of extra work and provided us an outlet of focusing on the major requirements we needed at the end of the project.

Overall, the goal of the mobile application is to provide a sleeker, intuitive interface coupled with various types of services and features. Keeping this in mind, the final data feature was designed on a single screen for easy consumer access, with all of the main data features available at your disposal for the chosen parking lot based on consumer selection. Minor tweaks and adjustments were also made to only tackle what was needed and cut down on excess details that was not needed, as sample data was replaced with real sensor data for the purpose of our parking application.

This section is discussed by Student A (Vikas Sharma), the lead for the mobile application development and integration, along with a group effort by each team member for presenting this final mobile application milestone.

Login Activity

The final version of the mobile application did not require any major changes to the login/authentication side as most of the features stayed intact for the consumer application and its overall purpose. Similar to before, at the start of opening up the application, the user will be prompted to a screen with options to log-in, register an account, or reset your password using the Forgot Your Password screen or Verify Your Password for different forms of authentication services in-application. We constructed an authentication process, in which we believe is a simple, intuitive interface for storing/checking credentials through secured database access. The following is a portrayal of the Register/Login screen which is automatically prompted to the user at entry of the application.

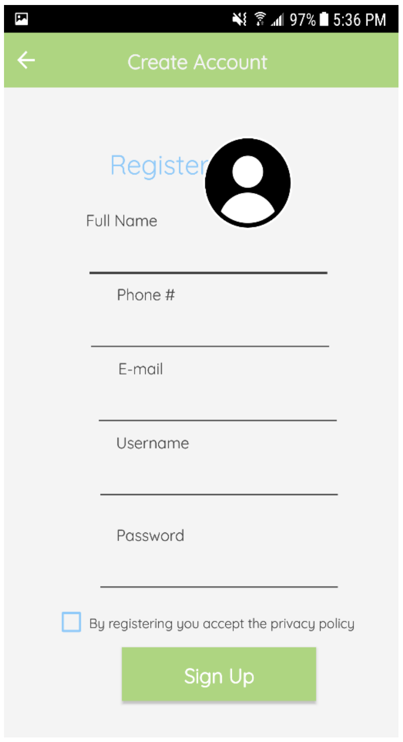


Figure 70 - Register Screen (FINAL)

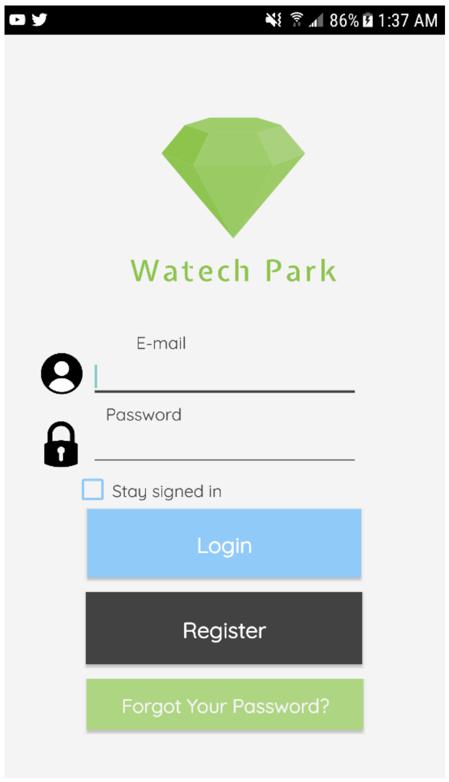
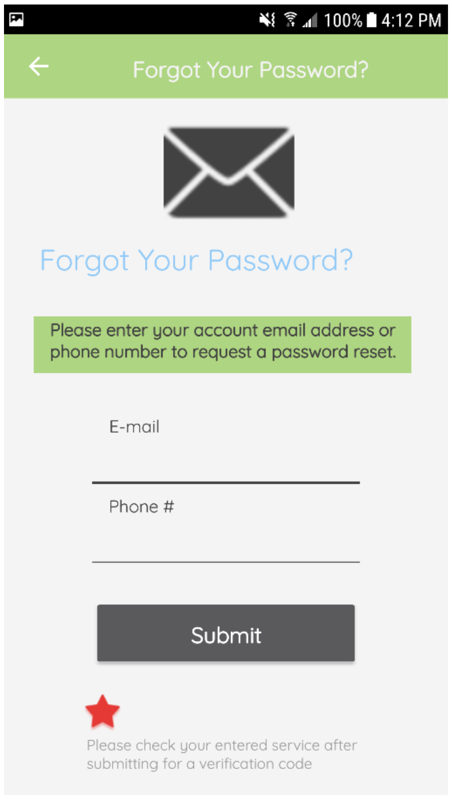


Figure 71 - Login Screen (FINAL)

The authentication part was designed with having a login/register screen separate from each other. The Register screen prompts user input in terms of personal information specific to a user. There is also the option for the user to choose a profile image from a real device through Firebase external storage permission. This account information is submitted onto the Firebase Real-Time database under a data structure, which holds the data. This information incudes, the full name, phone #, e-mail, username, image uploaded. So, once a user registers into the system the data is sent to the Firebase database, and stored to check for validation through login attempts further down the line.

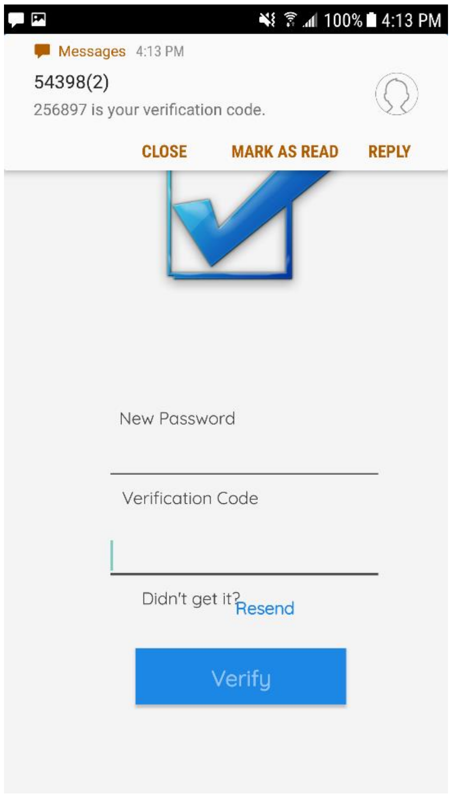
After registering, the next step is to login. To login, we used the Firebase Authentication system for e-mail and password. This is a built-in feature of Firebase, that is used to authenticate a user that exists in the database. So, in the Login screen, the user would sign-in with the “registered” email and password. Firebase checks for valid/invalid credentials based on the information stored in the Firebase database section under that UID (unique identifier). User selections are stored with a “Stay signed in” option. This means, account information is visible after the user leaves the session, or returns to resume the activity. At this time in Firebase, the logged in user would appear in the Authentication section with the corresponding email/password information.

The authentication process continues with the “Forgot Your Password” and “Verify Your Password” screen. The Forgot Your Password screen is used to allow the user to reset their account password. There is the option to use e-mail or phone # authentication. This authentication is done through the Firebase database, where once a consumer selects a service, the other unattended service is not allowed to be accessed. The user enters an email address and through valid checking Firebase will then send a verification email to the corresponding email address. Phone number authentication requires the consumer to enter in a valid phone number using registration data from Firebase. Once the phone number has been validated, and follows the required system format (+1) a verification code is sent randomly through the phone service provider.



Please enter your account email address or phone number to request a password reset.

Please check your entered service after submitting for a verification code

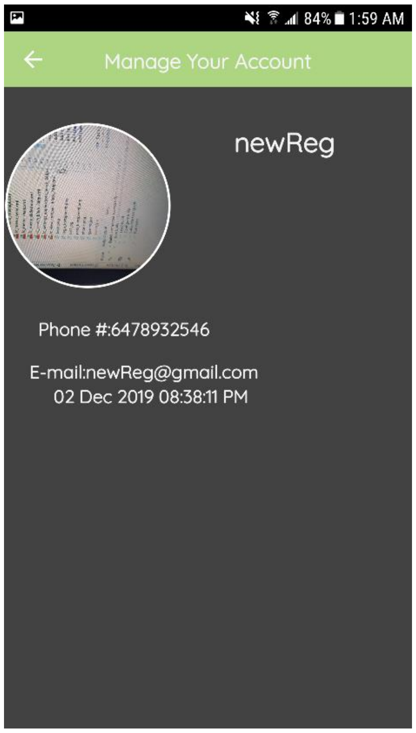


54398(2)
256897 is your verification code.

Figure 72 - Forgot Your Password Screen (FINAL

Figure 73 - Verify Your Password Screen (FINAL)

The mobile application also holds a feature, where account information is displayed specific to a user, in this case it would display different details based on the active UID. The screen followed a general layout, a little different from having to fit everything into a fragment view, we decided to use a new separate screen. This screen displays account information from the Register screen. The data is populated and displayed. Such as, the profile image stored from Firebase Storage, phone #, e-mail address, the name and a timestamp. In the final version of the application, there were no changes committed to the login or authentication side and the account screen remained as is.



Phone #:6478932546

E-mail:newReg@gmail.com
02 Dec 2019 08:38:11 PM

Figure 74 - Manage Your Account Screen (FINAL)

Data Visualization Activity

The data relevant to the parking lot available appears bundled on the main home screen and once the login authentication process is complete this is the following step the user will encounter. On this screen, is a single parking lot which was tweaked to only display a single parking lot type rather than a row of selections, which before were not doing anything special. In this case, we decided to use “Humber College” as the only parking lot visible on the screen, and eliminated the rest of the parking lots which were previously only used for sample and visual purposes from the previous semester. We believe this is the most streamlined approach taken for the users, and this allows easy and simple access of the data, and the activity all available within one main screen. This includes, proximity readings from the actual hardware, reservation capabilities, parking lot details and the actual lot itself visually represented in real-time. For this screen, we redesigned the main-menu from the proposed version, finally having a much sleeker interface to display only one parking lot location, and its information to the screen in a more convenient manner. So, as shown above this screen offers two options to the consumers. This being “View Details and “Reserve”. The View Details screen was the main modification point for the duration of this semester. There is an expandable view which pops up under an AlertDialog box and in this screen is where the parking lot is visually represented, and the sensor data is displayed along with real-time status changes of each spot.

The idea was to build from this feature and specific screen, which we were able to implement with some modifications and removals due to hardware limitations on the VCNL4010 Proximity sensor end. The reason for this was as previously mentioned in this document, we came across an issue and found out that the VCNL4010 Proximity sensor only supports a fixed I2C address of 0x13. Due to this reason, we were unable to support all four sensors with the originally four planned spots, each positioned with an individual proximity sensor, and have that data be fetched and displayed on the mobile application. Our team extensively debated whether it would be applicable to implement a I2C Multiplexer at the end of the project, but we finally came to a consensus to not follow through with that idea as it may be costly in financial terms, and lengthy in the time space we were provided with. Also, if we had decided to implement a I2C multiplexer, we may not have been able to implement other additional features such as the admin control task for the entry servo motor due to time commitments. In addition, we also would need to redesign our current final PCB unit to work with the device, which again would’ve hindered our ability to focus on the essential tasks needed to be completed.

After continuous development, now this screen portrays the data retrieved through the online database which gets sent by the real sensor/effector data through different data types and instances of the code either through raw proximity values or Boolean values which are then converted to String forms to be readable by the mobile application interface. Thus, instead of having the mobile application talk directly to the sensors, we collectively decided to have the Broadcom Development Platform being the Raspberry Pi 4 act as the sender, working in touch with the online database which is the supporting bridge structure acting as the gateway to handle and store the data sent by the hardware for each device. The mobile application in this case acts as the receiver, where it basically retrieves the sensor data and displays this data on this screen. Along with manipulating the data sent, to work with our main features on the “View Details” screen which are behind the scenes tasks performed through the application code.

The data displayed includes the following, four main spots of the parking lot in reference to the data and status of the VCNL4010 Proximity sensor, entry/exit arrows which denote the gate status through the IR Break Beam Sensor. The screen also features the addition of displaying the real-time proximity levels of the VCNL4010 Proximity sensor and is viewed in-app as “RT Proximity” followed by the proximity value fetched through the online database. To indicate the status changes on each respective parking slot, we designed this screen to show relatively three main colors being light green, red, and blue. Based on the different scenarios of the VCNL4010 Proximity sensor this data is then updated through “open” or “occupied” instances of Slot 1A with the other three spots being Slot 2B, Slot 3C, Slot 4C acting as simulated spots set with a value of 0(occupied) for the duration of the data sending/retrieving session. The data is displayed and organized with the proximity value at the bottom of the screen, fetched from the database, and cooperates with the VCNL4010 Proximity sensor to send the data. More details into how this works will be described in-depth in the upcoming Action Control Activity branch of this section, including how the application code was designed to work with the hardware in the end and vice-versa.

The other option visible is to “Reserve” a spot in the parking lot. Once the user chooses to reserve a spot, the corresponding data is sent automatically to the Parking Passes screen for that parking location and the user is notified in-app to access the payment service. For example, a user reserves a spot for the Humber College parking lot, and once the parking lot has been reserved successfully, a notification pops up in-app presenting the reserved lot and to view the parking passes for the next step. By swiping to the left of the screen, brings up the side navigation drawer that is used to hold the other fragments and features of the application. This allows the user to access every main feature of the application within a gesture, and to avoid having to go through many hurdles or displays to get to their desired destination. A representation of how the data screen and its overall design is shown on the far-right side below of Figure 77.

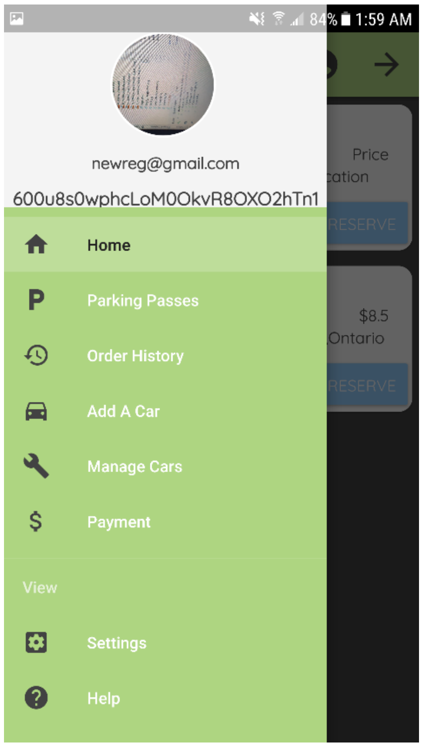


Figure 1 – Main Sidebar Screen (FINAL)

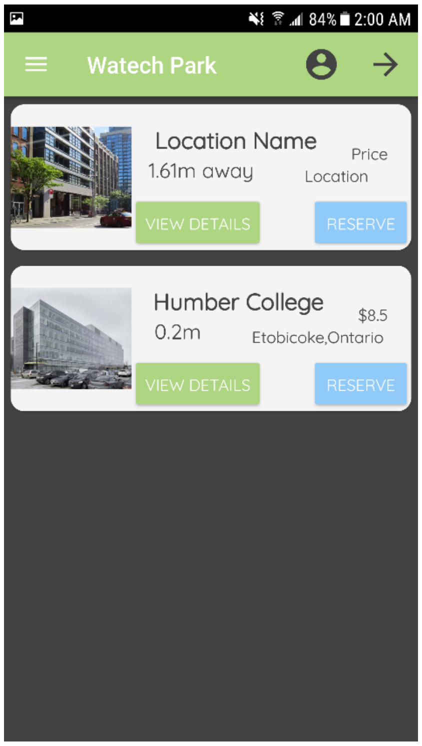


Figure 76 - Main Menu Screen (FINAL)

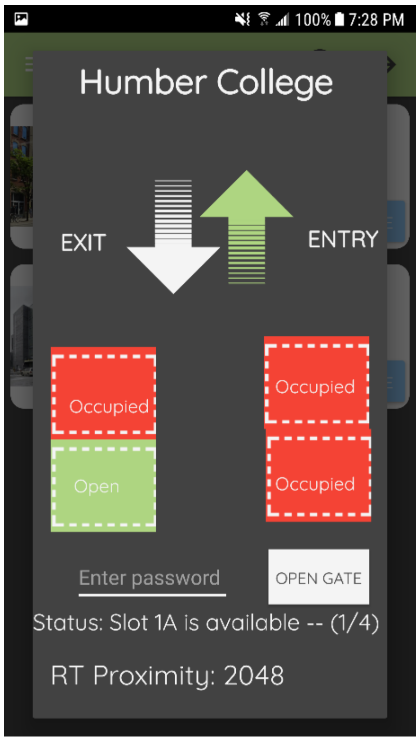


Figure 77 - Parking Lot Data Feature (FINAL)

In the “Add A Car” screen, the user enters in their vehicle details to register a car to the account. The user is prompted to enter information specific to each account, such as the Make, Model, Color, and License Plate #. We have decided to store each added car into the system and send the data according to a specific user to the database. Once a license plate image is taken on-site of a vehicle, the application will instead run a function holding a fixed license plate string and try to see if it can match it with the added license plate entered by the user through the mobile application. If the data matches, entry to the lot is allowed and the application displays the gate opening from a visual representation through the emergence of a vehicle and a color change.

The plan was to implement this feature by the timeframe of Mid-March, which we can now say we were successfully able to add this feature as part of the data screen and have the changes be visible to the app. This was accomplished by basically mirroring the hardware data to a visual form on the application. The ADD A CAR button registers the car to the Firebase database. Once the car is added into the system, and the data is sent to Firebase the consumer can access these details and the registered vehicles in the “Manage Cars” screen. To adjust to the change of having only one parking lot present for the user to choose, and view the data we also had to modify the code to only set a value in the database, and not have it push new values each time a new car is added into the system. This was done, to ensure there are no conflicts that may arise when the search for the license plate number is committed through the Raspberry Pi and to avoid multiple entries by the same user. This is because, at a time the individual is logged in as the “consumer” and this user is capable to add multiple entries based on a single account to serve the purpose of this feature. In the code, initially we pushed the new values and in the final version of the mobile application we set a new value corresponding to the logged-in user

Manage Cars is where the data is fetched from the Firebase database and the information for each “Car” is displayed following a similar format of the main menu. There are 2 options to choose from here: Edit/Delete. Edit allows the user to basically make a change and update the information to the database structure in Firebase. Once the user selects Edit, an inflated view pops up of the fragment prompting to enter in the new information. The user then would tap on the “Apply Changes” button to apply the changes automatically. The changes are visible in real - time through Firebase, once they are set. The Delete option asks the user if they are sure they want to delete the car. If the user approves, the car is deleted from the real-time database and is removed automatically after the next time you access the Manage Cars section. If the latter is chosen, the action is dropped and cancelled to continue the session.

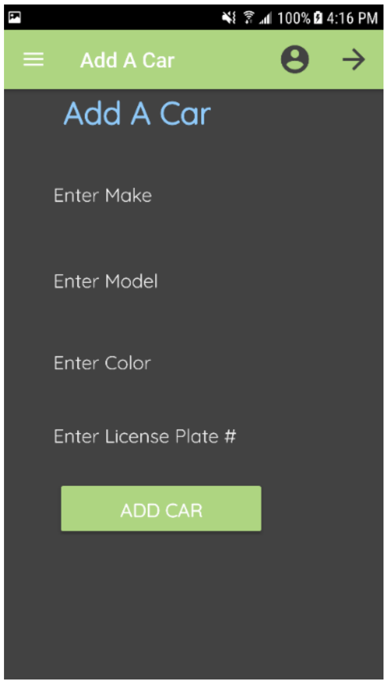


Figure 78 - Add A Car Screen (FINAL)

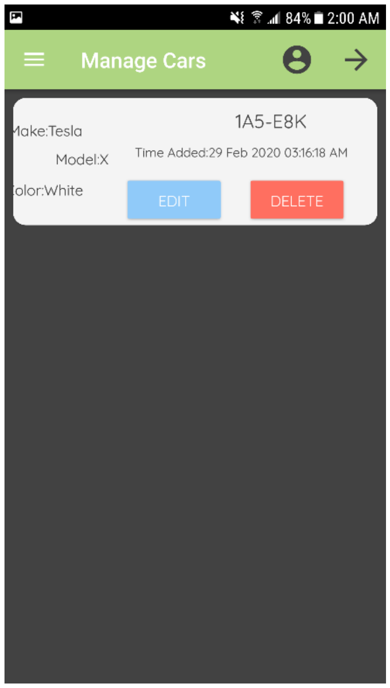
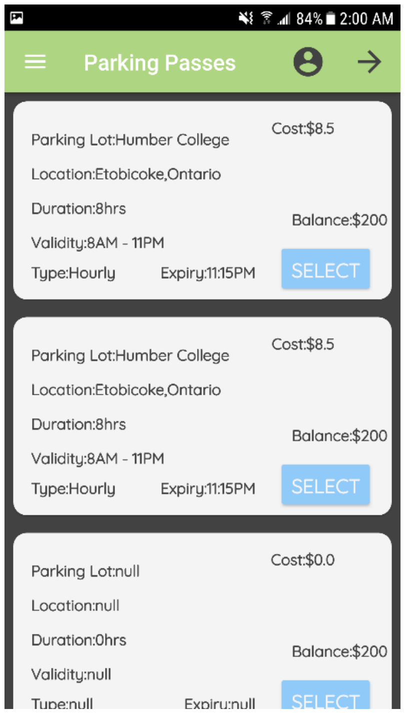


Figure 79 - Manage Your Car Screen (FINAL)

The Parking Passes screen in the application, holds all of the available parking passes for each lot. This includes, the name, location, duration (in hours), validity (time the pass is valid for), type, expiry time, cost, and the account balance before the purchase. There is a button to “SELECT” a parking lot. We also had to modify this screen, to show only one possible parking lot location of “Humber College” as this was the only parking lot available to the user when they login to the application. All other forms of sample data added into the application code was removed, and only the parking lot to hold the real data stayed intact. Once selected, the data for that lot is sent to Firebase and stored under the UID of the user. This data is also then sent to the Payment screen, which would be the next step to finalize the reservation through the transaction process.



Parking Lot:Humber College
Location:Etobicoke,Ontario
Duration:8hrs
Validity:8AM - 11PM
Type:Hourly Expiry:11:15PM
Cost:$8.5
Balance:$200

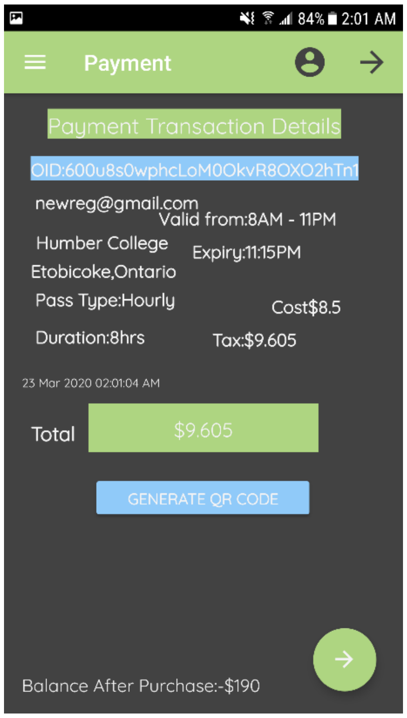
Parking Lot:Humber College
Location:Etobicoke,Ontario
Duration:8hrs
Validity:8AM - 11PM
Type:Hourly Expiry:11:15PM
Cost:$8.5
Balance:$200

Parking Lot:null
Location:null
Duration:0hrs
Validity:null
Type:null Expiry:null
Cost:$0.0
Balance:$200

Figure 80 - Parking Passes Screen (FINAL)

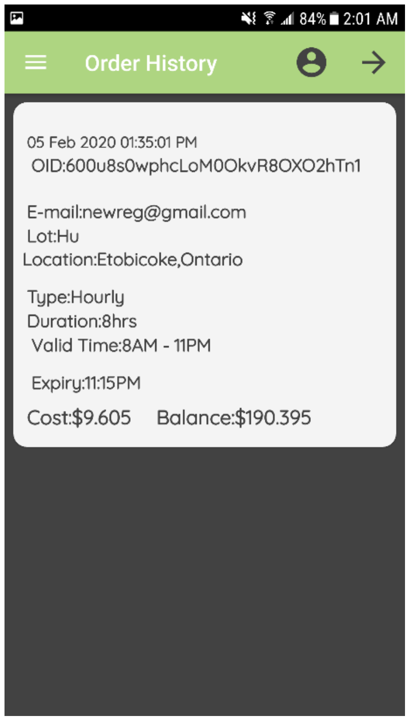
On the Payment screen, the selected parking pass is now visible with all the data related to the particular parking lot. On this screen, similar details are displayed, with the addition of an OID (order ID), and e-mail corresponding to the account that is processing the order. Also, the total is calculated for the parking pass with tax and displayed in only a readable form. The balance after the purchase is calculated on the spot and displayed according to the total accumulation and implemented through a in-app QR Code generator. The total would be calculated and based on this set value, the “Generate QR Code” button generates a random QR Code using this value. The user would then tap on the FAB (floating action button) which asks to confirm the purchase. If the order is confirmed then it has been successfully processed. A toast message appears saying “Order has been successfully placed! Please View Order History for more details”.

Order History displays the order’s that have been placed using a unique OID (order id) which is the UID used to refer to a specific account. The data is retrieved from Firebase displaying the processed information and a timestamp for when the order confirmation took place.



Payment Transaction Details
OID:600u8s0wphcLoM0OkvR8OXO2hTn1
newreg@gmail.com
Valid from:8AM - 11PM
Humber College
Expiry:11:15PM
Etobicoke,Ontario
Pass Type:Hourly
Cost$8.5
Duration:8hrs
Tax:$9.605
23 Mar 2020 02:01:04 AM
Total $9.605
Balance After Purchase:-$190

Figure 81 - Payment Screen (FINAL)



05 Feb 2020 01:35:01 PM
OID:600u8s0wphcLoM0OkvR8OXO2hTn1
E-mail:newreg@gmail.com
Lot:Hu
Location:Etobicoke,Ontario
Type:Hourly
Duration:8hrs
Valid Time:8AM - 11PM
Expiry:11:15PM
Cost:$9.605 Balance:$190.395

Figure 82 - Order History Screen (FINAL)

As the focus of the semester was to only work on the important features of displaying the parking lot and its status updates/changes in accordance to the hardware, in the end of development there were no added features/modifications to the in-app settings or options. In this case, as before the Setting screen provides a localization feature for (English/French integration). The design basically followed a simple UI, with a button to "Select Language. The user checks which one to perform and the languages change state accordingly without the need of re-entering the app. The Help screen, displays general help documentation for ways to navigate to the different screens and use the functionality. The About screen displays project and general mobile application details.

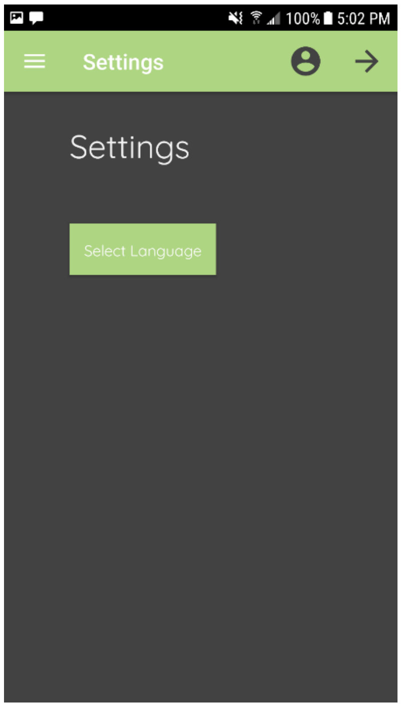
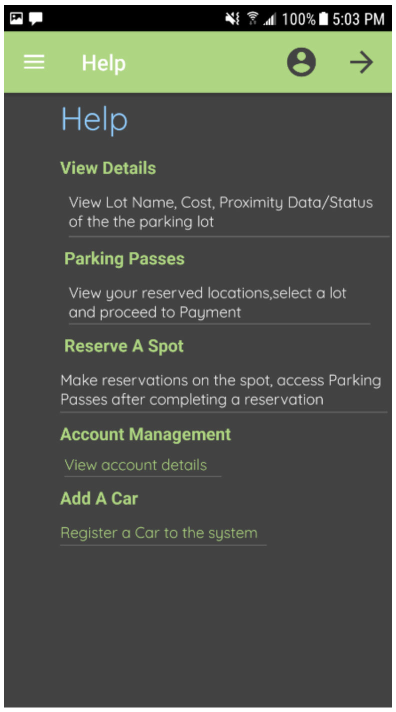


Figure 83 - Setting Screen (FINAL)



Help

View Details

View Lot Name, Cost, Proximity Data/Status of the the parking lot

Parking Passes

View your reserved locations,select a lot and proceed to Payment

Reserve A Spot

Make reservations on the spot, access Parking Passes after completing a reservation

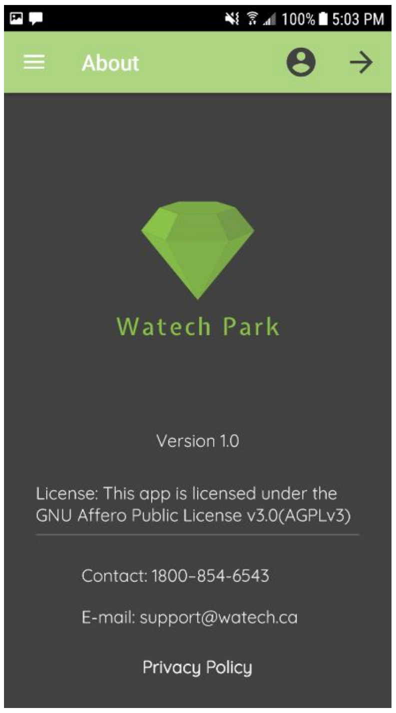
Account Management

View account details

Add A Car

Register a Car to the system

Figure 84 - Help Screen (FINAL)



Version 1.0

License: This app is licensed under the GNU Affero Public License v3.0(AGPLv3)

Contact: 1800-854-6543

E-mail: support@watech.ca

Privacy Policy

Figure 85 - About Screen (FINAL)

Action Control Activity

This section will address how the hardware communicates alongside the software, and the process behind having the mobile application control the sensors/effectors, and vice-versa. It is important to have the hardware for the project be able to talk with the application code and allow the user to track and monitor data essential to our parking application. We will be going through each individual sensor, and their main use for the mobile application for different purposes in the project. Previously, we had discussed the implementation of the three main sensors for the project into the mobile application. This includes, the VCNL4010 Proximity sensor, IR Break Beam sensor, and the 2 servo motors functioning alongside the PCA9685 servo controller. Each sensor provides a unique activity to the mobile application, and the data sent by the hardware to the online database, is further gathered and displayed in application form for consumer interest. Based on feedback received by the professor, and our team considerations we had initially planned for the two servo motors to not have anything to do with the mobile application, as the motors would be there for entry/exit purposes of the parking lot. Although through closer inspection, we ultimately decided a admin feature would be a supporting to add into our application and would further enhance the goal and not only through the consumer experience but also through an admin level priority.

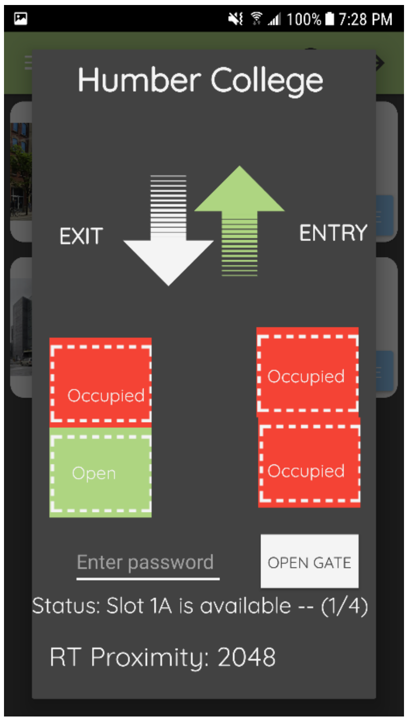
The VCNL4010 Proximity sensor is used to manage and detect a vehicles presence on a specific spot on the parking lot and update the status of the parking lot through a visual representation. At first, the VCNL4010 Proximity sensor sends the raw proximity values from its hardware, and these readings are received by the online database through the “ProximityData” structure. The data sent is in the form of a Long object type. Therefore, when the proximity value is isolated to be read, it must be in the form of a long variable type in the application code initially and then converted into a String form to be able to be manipulated and perform the tasks needed. Through the use of the online database, the data is stored each time a vehicle is detected from either far/near distances relative to the VCNL4010 Proximity sensor. In terms of the process taken in the code itself, the proximity data is checked through three different instances. These scenarios are based on the parking space status available in the lot, and are denoted by individual spots of Slot 1A, Slot 2B, Slot 3C, and Slot 4D. The procedure used to follow this for the application code is in pure resemblance to the firmware code assembled for the hardware in the watech.py python program. In the start, the VCNL4010 Proximity sensor sends various proximity values to the online database. This data is then fetched and read through a DatabaseReference object using the “ProximityData” data structure, and a separate class is created to store this data into called ProxmityData.java. This class holds two primary variables, which are the proximity, and timestamp as well as the status of each individual slot.

Through developing the code for this part of the application, we had planned to have each spot occupy a separate proximity sensor and send a status of 1 or 0 in Boolean form to indicate a true or false status for a parking space. Due to coding issues, we were met with issues in regards to converting and using the long values of 1 or 0 for each individual slot, as each time the APP would display a 0 as a default value. To solve this problem, since we are only using one active spot on the parking lot by Slot 1A, it was evident to us we only had to make use of the proximity values gathered by the VCNL4010 device, and set each other spot to the value of 1, to denote a occupied spot and show this through the APP with a red background color change for the three remaining spots. This value would stay fixed at all times and would not change as we designed it to simulate the parking lot only. For the coding portion the modifications were made to the ParkingLocationAdapter.java file.

The following steps are taken, if the proximity value read is less than the value of 2500, there are no vehicles present or near the VCNL4010 Proximity sensor. Therefore, the only action taken by the sensor is to send the proximity readings to the online database. Then, have this proximity value be read and displayed at the bottom of the screen in a String format. Since the level of proximity is less than the set value, have Slot 1A remain with a light green background color, indicating no change currently. To guide the user, toast messages are added to indicate that the “Gate is Open” and “Entry is Allowed”. If the value is greater than 5000, change the color of Slot 1A to a red representation to indicate, a vehicle is currently occupying the spot. The proximity readings sent by the sensor is updated in the database, which is then read and displayed under the “RT Proximity:” TextView resource and this value is updated each time based on the proximity value. Toast messages are also pushed to notify the user that the “Gate is Closed” and “Entry is not Allowed”. Once all spots are occupied, the overall status of the lot is full and this is then updated as a visual change visible to the user. A message then appears in real-time to change the overall status of the lot, and the number of remaining spots out of the four parking spaces is tracked and displayed.

The “EntryStatus” data structure in Firebase was created to send the status after all four spots are occupied. In this case, the application sends a status of 0 to show all spots are occupied and the lot is full. From the firmware end, this data is read and checked for a 1 or 0 value, if a 0 is to be found then there is no call committed to run the servo motors function for gate entry. Each spot excluding Slot 1A begins with a “occupied” status or 1(true), with Slot 1A being actively listening and starting the data session by a “open” status which is denoted by a 0(false). The third if statement, basically checks the condition if the value is in between the range of 2500 and 5000(open and occupied) then the status is updated with a “targeted” message on Slot 1A and a blue color change of the spot. The proximity value is also pushed by the sensor to the database as before, and processed by the mobile APP. As initially planned, following this a toast appears alerting the user “Vehicle is approaching the parking space…” before the actual change occurs.

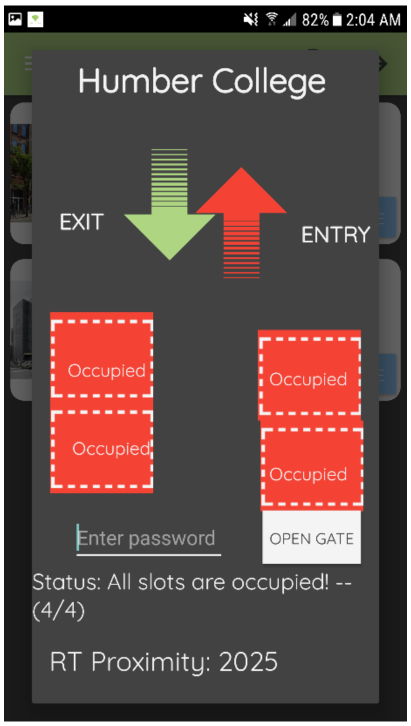
The following screenshots show the open state of Slot 1A and the entry gate, and the occupied state following a successful reservation of a spot. This feature of reserving a spot will be addressed next in this section following these screenshots.



Status: Slot 1A is available -- (1/4)

RT Proximity: 2048

Figure 86 - Data Control Screen (Open Spot)



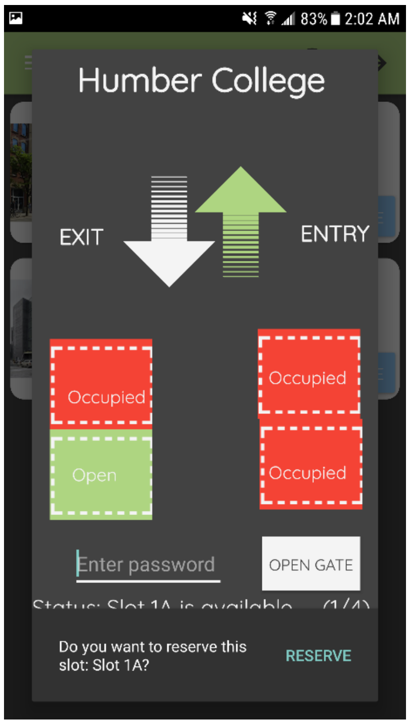
Status: All slots are occupied! -- (4/4)

RT Proximity: 2025

Figure 87 - Data Control Screen (After reserving a spot)

Another function of this screen, is to allow the consumer to be accompanied with reservation capabilities. The user can act upon one of the “open” spots at the beginning of viewing the data for the parking lot. If the gate is open and all is clear, then the application allows the user to reserve a spot by tapping on Slot 1A, as it is the only spot available and choose to reserve the spot. Once the spot has been reserved, the data is sent to the Parking Passes screen and a toast pops-up to guide the user to select the pass, and proceed to payment. The reserved spot is also visible to the user as a red color change occurs from the originally green(open) spot, and the overall status is updated immediately. A value of 0(false) is sent to the “AdminControl” data structure to show all spots are taken and entry is not further allowed. Additional details to support the process of sending and storing the data of different sensors/effectors, in the main data structures used is discussed in the Database Configuration section of this report.

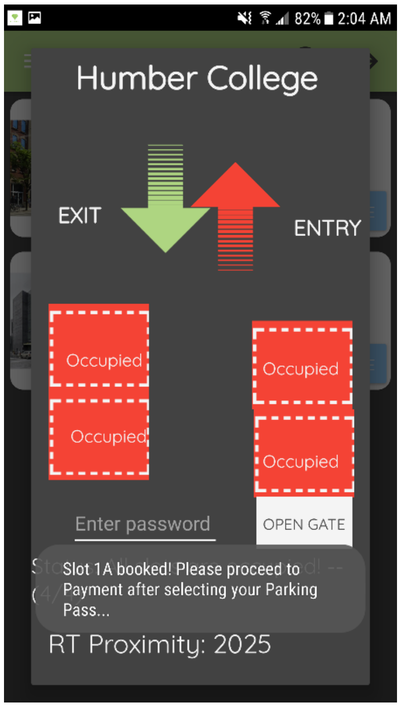
On this screen, there are also arrows for entry/exit used by the IR Break Beam sensor which are also updated, with a visual color change from green for ENTRY to red, to show entry is not allowed and the gate is closed. At this time, in-app the EXIT arrow is updated to a green color change, which means exit is the only option available next to the user once all spots are taken. It is important to mention however, for this reservation to take place successfully the proximity value must be less than 2500, or between the range of 2500 and 5000 to book the slot successfully. Any attempt made to reserve an already occupied spot which exceeds the value of 5000, then the attempt will be ignored and met with an error. The following screenshots show the different reservations capabilities and the occurrences of a user choosing Slot 1A on the parking lot.



Status: Slot 1A is available (1/4)

Do you want to reserve this slot: Slot 1A?

Figure 88 - Data Control Screen (Reserve)



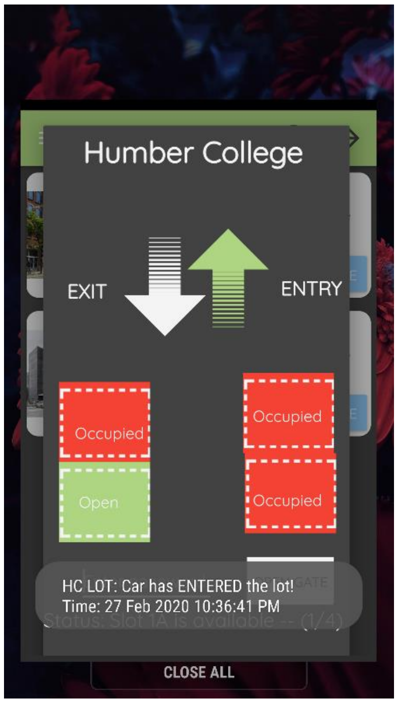
Slot 1A booked! Please proceed to Payment after selecting your Parking Pass...

RT Proximity: 2025

Figure 89 - Data Control Screen (Successful reservation)

To control gate opening and closing based on entry/exit situations, the IR Break Beam sensor is used to send the current status of the gate. On this activity, this sensor initially sends from its hardware a Boolean value of 1 or 0 to the online database. Once a vehicle is detected and the beam is broken, the license plate number is detected and assessed with the “fixed” string value. The vehicle is validated and a 1 is pushed to the online database to show entry has taken place. The mobile application uses the “GateStatus” data structure in Firebase to locate the status, and read in the value of the gate. The IR Break Beam sensor works alongside of the VCNL4010 Proximity sensor. To explain, in the code the proximity value fetched by the VCNL4010 device is used to gauge whether entry or exit is allowed by a particular vehicle. So, similar to the process of the parking spot feature, the entry/exit arrows are visible on the screen and if the proximity value is less than 2500, then the gate is open. Therefore, the entry arrow glows with a “light green” color to basically show a visual representation of the action taken and at entrance of the lot. As soon as entry is detected by a vehicle, a toast message appears which displays data fetched from the “GateStatus” data structure alerting the user “Car has ENTERED the lot!” followed with the timestamp retrieved and converted into a String format. This allows the application to also track each specific vehicle action physically occurring on-site of the parking lot.

At this time, the arrow stays green for the duration of the proximity level staying below the set amount. Once the value exceeds 5000, the entry arrows changes to a red color to indicate the gate is closed and ENTRY is not allowed. This is because, since there are four spots, with three fixed values of being “occupied”, once the last spot of Slot 1A is taken then the APP tracks the movement of the vehicles entering the lot and updates the remaining spots as mentioned above. Due to this reason, the arrow turns red to show ENTRY is not allowed until a vehicle exits the lot. During this time, the EXIT arrow switches to a green color to show the only available option to the user. The following screenshot shows the initial entry stage of the parking session, where entry is made by a vehicle and the data is sent to the mobile application.



HC LOT: Car has ENTERED the lot!
Time: 27 Feb 2020 10:36:41 PM

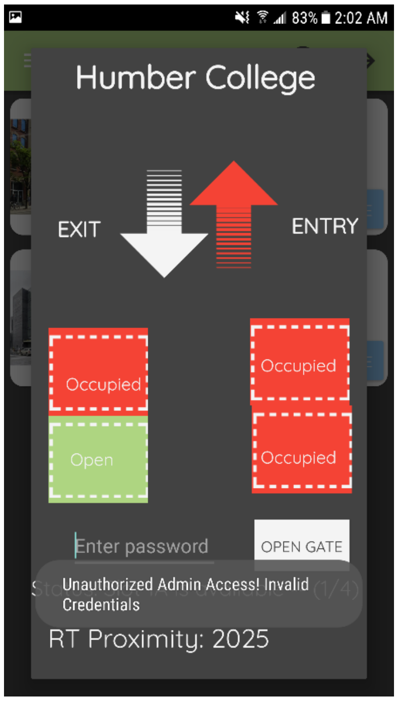
Figure 90 - Data Control Screen (Entry)

This sensor also functions with the support of the servo motors which act as the barriers for actual movement and rotation of the gate. Originally, we had planned to not have any sort of preliminary data to be sent or processed by the online database. After successfully communicating the hardware to the app and controlling the APP and its functions through the sensors/effectors, we decided to work on additional features.

The next step, is when an action is taken by the user in-app, to have a feature where the hardware is capable to be controlled by the mobile application. To do this, we introduced a new feature based on revisions for allowing two main types of users to access the APP and its functions. As shown below, we decided to integrate a "OPEN GATE" function which would allow the user to control the hardware present on the parking lot from anywhere, or any place with an accompanying Wi-Fi connection. This button was added in reference, to a mode we had discussed to implement in early January where we would have the consumer and an additional “admin” user.

This admin user would have access to different features from the consumer side. Such as, having the ability to potentially access parking lot data without being verified, or having to enter the lot. We decided due to time commitments, it would be best to drop this feature but instead workaround this functionality to have an admin feature integrated into the same screen, presenting the parking lot data. We believe this to be the most convenient, and reliable method for the consumer as it would be interesting, quick and effective. Also, working with the support of the VCNL4010 Proximity sensor and its proximity data levels, we had the button be present and added an EditText field for the user. In this field, the consumer would act as the admin, and once the password entered matches the fixed value in the code of “admin” and the button is clicked, then the servo motor function is performed in the firmware code of watech.py. To proceed with this, a value of 1 is sent each time the button is pressed after the consumer is validated as the “admin” for the duration of the session. Once validated, “admin” access is activated and a message appears mentioning the gate is open.

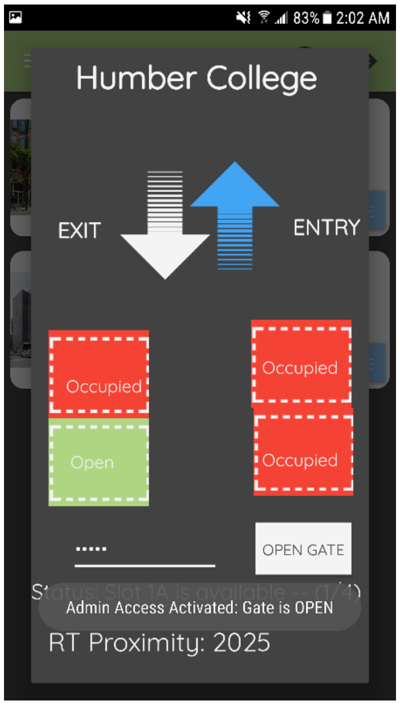
The overall point of this feature is to allow the “admin” user the ability to access the parking lot anytime, anywhere. This means the permissions for this specific user is granted and at any time of entry or exit, or an open/occupied status change where then the admin is allowed to open the gate. Through this feature, this will allow the admin to access the lot during poor weather conditions, such as in the season of Winter where snow may be blocking the entrance or parking spaces inside of the lot. This allows the admin to enter the lot in support of snow removals and other conditional activities. Other situations, can include providing daily maintenance of the lot, or viewing real-time capacity levels of each parking space and the lot overall. Now, once the user has exited from the data screen and returned to the main menu, the status value of the AdminControl structure is reset to a value of 0 along with the rest of the data. This is to ensure the gate does not always remain open at all times, and is actively listening for changes in the GateStatus and AdminControl data structures. In the occurrence the user enters in invalid credentials as the password for admin access, the system sends an error message and then sends a value of 0(false) to the AdminControl structure in the online database. This makes sure the gate does not open, without the required validation process. The following screenshots below portray how the admin access feature works, in both successful and unsuccessful trials.



Unauthorized Admin Access! Invalid Credentials

RT Proximity: 2025

Figure 91 - Data Control Screen (Invalid Access)



Status: Slot 1A is available -- (1/4)

Admin Access Activated: Gate is OPEN

RT Proximity: 2025

Figure 92 - Data Control Screen (Admin Access)

Overall, for the development of the mobile application the code built made use of only the three main sensors of the VCNL4010 Proximity sensor, IR Break Beam sensor and the two servo motors. These sensors/effectors were used to relay the parking lot data as shown above, through the data visualization feature of the parking lot and the action control features taken by the hardware or the APP itself to control the hardware through an IoT structure to accommodate the final capstone end product.

Link to Complete Application Code in Repository

The following is the link to the main code for the final version of our WatechPark android application. Through the “app” folder are the java classes and main code assembled for the project. This can be found under the “WatechPark(APP) folder designated on the main “master” branch of our project repository.

https://github.com/VikasCENG/WatechPark/tree/master/WatechPark(APP)/app/src/main/java/com/example/watechpark