Gabriel Cabrera, Nicholas Greer, Michael Mogannam

parkingFriendly
SOFTWARE DESIGN SPECIFICATION

1.0 Introduction

parkingFriendly is an application which will make meter parking much more friendly. By eliminating the need for change to feed your meter this application will allow for easier payment, less parking violations and less stress, overall. The application will utilize a database and paypal access to allow users to purchase credit for paying for meters. The database will store information pertaining to all aspects of the crediting process(creating an account, purchasing credit, using credit to pay for a meter, and checking out of meters afterwards as an initiative to play the game of chance involved with the meter access)

1.1 Goals and objectives

To achieve a working application that properly utilizes Paypal, Leaflet and Javascript to allow individuals to find and pay for metered parking in an easier and less stressful environment.

1.2 Statement of scope

Requirements	Priority
Getting HTML and database up and running properly in AWS	Essential
Get paypal to work with interface so users can purchase credit	Desirable
QR Code access to make paying for meter easier	Future
Offer mobile application	Future
Leaflet working and displaying meters	Essential
Game aspect of application	Desirable

1.3 Software context

Parking at meters can be problematic in 2018 because currency has become predominantly digital and most meters only accept coins. The parkingFriendly app allows vehicle operators to pay for meters that they would otherwise skip entirely and parks elsewhere due to lack of change on their person. This software encourages the use of meters and allows digital payment without the need to trash old hardware designs.

1.4 Major constraints

Possible issues using Paypal to test credit purchasing or giving credit to accounts.

2.0 Data design

Database where all data will be stored and accessed.

2.1 Internal software data structure

The web app will utilize queries made to the database and display what is needed (credits on account, list of meters available) and stores info that is created/updated (creating an account, crediting a meter, checking out of meter).

2.2 Global data structure

A self-contained database is being used as our primary data structure where all data will be stored.

2.3 Temporary data structure

Currently no other temporary data is utilized, it is all self-contained in the database.

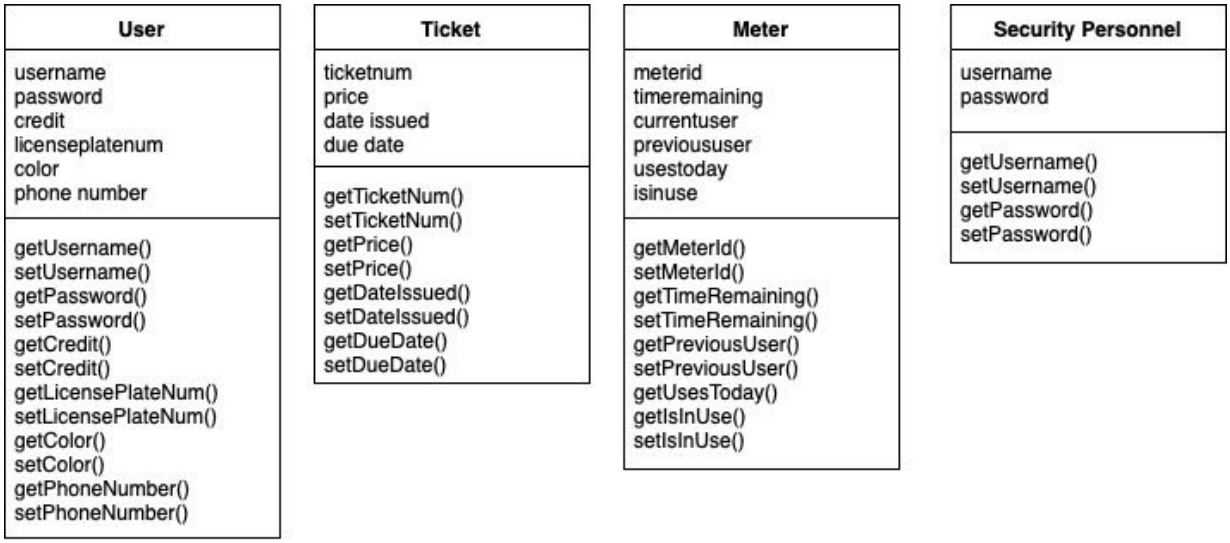
2.4 Database description

The underlying database is going to store the users information for accessing the web app (logging in, crediting their account, and paying for a parking meter). It also stores the credit amount that each user has in their “wallet”, it holds all ticketing information for any violations that may occur. The database also stores information pertaining to: Meters, individual users, security personnel, tickets, and account information.

3.0 System Structure

The system comprises of a login page where you can log into the web application, or create an account, following the login page, you have a main page where you are given options to either, add credit to your account via paypal, apply credit to a meter for parking, pay a parking violation ticket, or check out of said parking meter.

3.0.1 Architecture diagram



4.0 User interface design

The web application starts out with a login page where you can log in or create an account, following this page you are sent to an options page where it includes a map of the parking meters in the area with descriptions of each meter. It gives you options to purchase credit, credit a meter, pay a parking violation ticket, or update your account. Each respective page has attributes according to each action. The purchasing credit page will send you to a paypal link to purchase meter credit. The pay meter page gives you options on how to pay for a meter and check out of that meter. The pay parking ticket page will give you the option to pay off a violation ticket. The update account page will allow you to add a vehicle to your account, or change information if needed.

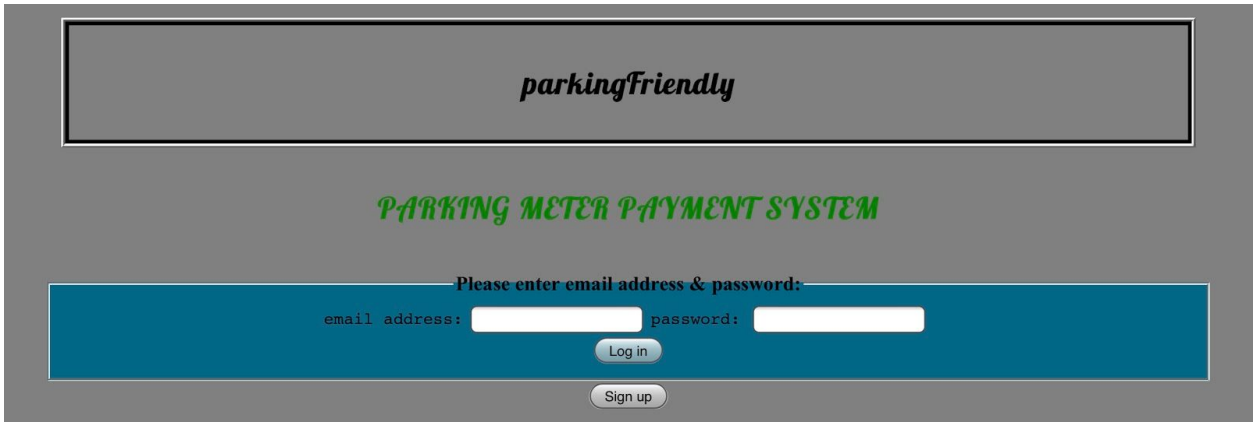
4.1 Description of the user interface

A detailed description of user interface including screen images or prototype is presented.

a) login/homepage
    i) contains logo, link to refresh page, input for email (username), input for password, login button (for submitting), and 'sign up' button outside of fieldset to allow brand new users to register an account
b) sign up page
    i) contains two input boxes for email address (username) {TWICE for confirmation}, and two input boxes for a secure password and confirmation, as well as a submit button; When sign up is submitted user receives a confirmation email with link to authenticate their email
c) enter meter & amount menu
    i) contains drop down box with list of all meters, map with live GPS to link user directly to location and meter they are closest to, an input box for the amount they would like to pay with an automated timer that fills up accordingly, and a submit button to pay for the meter. Each time a meter is paid for by pressing submit a popup with all the information about the purchase as for confirmation before payment is official.
d) meter_id error page
    i) if user enters an amount but not a meter_id then the meter error page displays and requires the user to start the process of payment over. Refreshing the page is the penalty to ensure the user is paying attention to the payment system so as reduce user error or mistake.

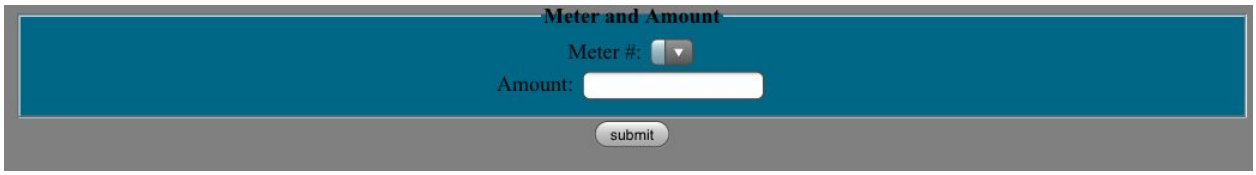
4.1.1 Screen images

1. login/home page



2) sign up page

3) enter meter & amount menu



4) meter_id error page

4.1.2 Objects and actions

Text Boxes, Buttons, and Drop-Down Menus will be utilized to allow user input. The leaflet map will be interactive and will allow users to click on a meter that will auto-fill the text boxes required for crediting to an account.

4.2 Interface design rules

The Eight Golden Rules of Interface Design
-Strive for consistency
-Seek universal usability
-Offer informative feedback
-Design dialogs to yield closure
-Prevent errors
-Permit easy reversal of actions

-Keep users in control
-Reduce short-term memory load

4.3 Components available

Leaflet map overlay to display meter locations and availability will be implemented in the web application.

5.0 Restrictions, limitations, and constraints

Parking meters are very close to each other and online maps can only pinpoint a person's location with a certain degree of accuracy. Although the map will be available it is crucial that the user ensures that the parking meter they are paying for is the actually parking meter they are parked at. We recommend that the user checks the physical meter for its identifier to ensure that it is the meter represented based on their location shown on the map.

6.0 Testing Issues

Testing would consist of, making sure there is no bottlenecking when a mass of users use the application at one given moment of time, making sure that the application correctly and progressively displays when meters are available or occupied, making sure that accounts being created are stored properly in the database and that the database queries the correct information when required to.

6.1 Classes of tests

Testing, for the post part, will be done via black-box testing, as the testers will not be able to look at back end code, or any code in general. This is done so that the end user will not have to worry about any specific inputs to enter, as it will be streamlined to be easy to use.

6.2 Expected software response

The expected results include, properly displaying available and occupied meters, correct queries where ever applied, and proper insertion, deletion, and updates of the database.

6.4 Identification of critical components

Critical components include the database, leaflet map, and paypal utilization for purchasing credit.

Appendix A: minimum requirements

1. User stories

---
parkingFriendly User

As an

user;

I want to
pay for parking;

So i can
park my car without getting a ticket;

As an
User;

I want to
Setup an account;

So i can
Pay for parking;

As an
User;

I want to
Change account information;

So I can
Keep information secure/up-to-date;

As an
User;

I want to
Pay my ticket;

So I can
Keep my account from being disabled;

Security Personnel

As a
Security personnel;

I want to
Monitor meters;

So I can
Do my job properly ;

As a
Security personnel;

I want to
Access parking records;

So I can
Issue a ticket;

---
parkingFriendly admin

As an
admin;

I want to
Access parking payment records;

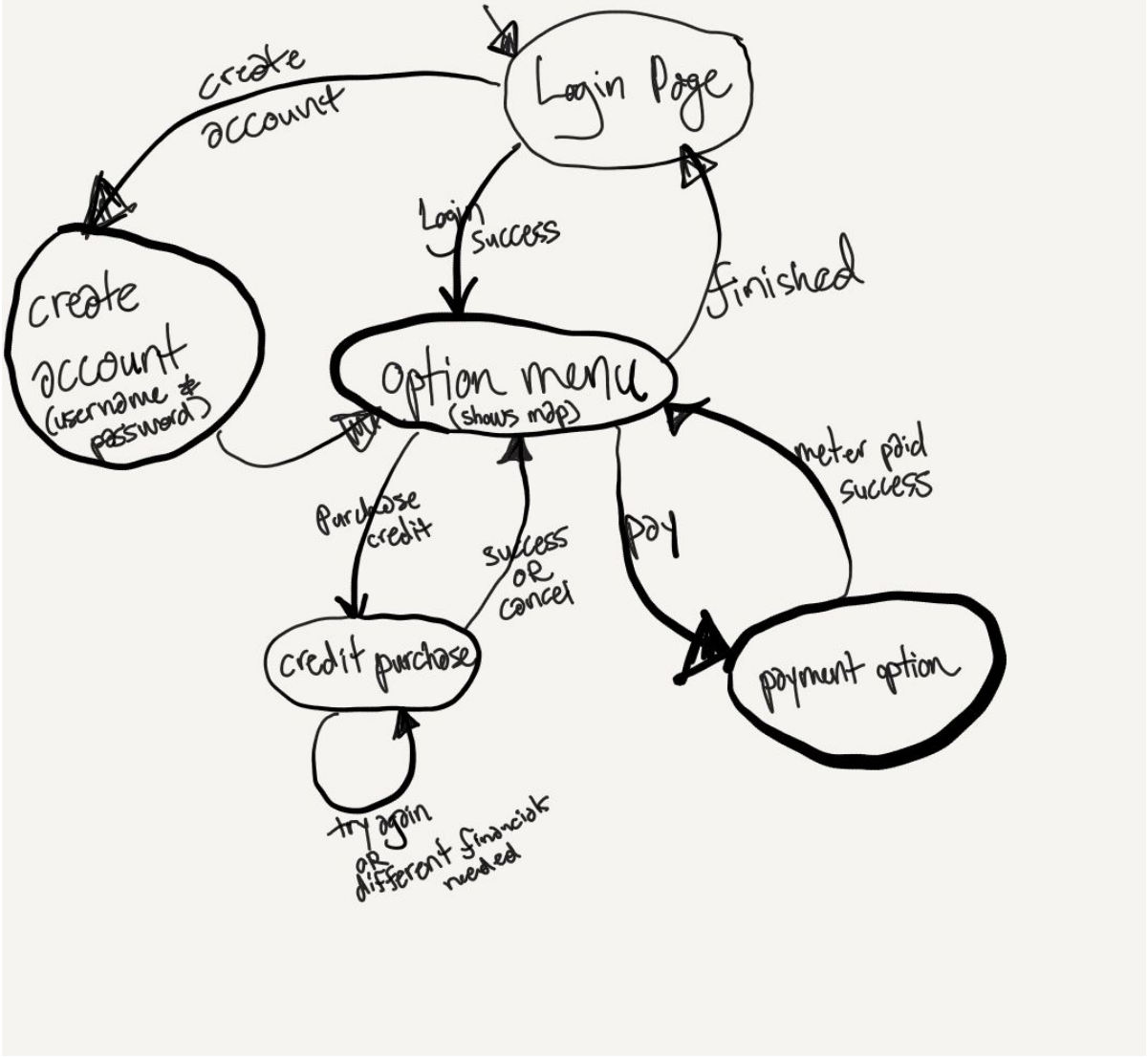
So I can
Report details for an accident;

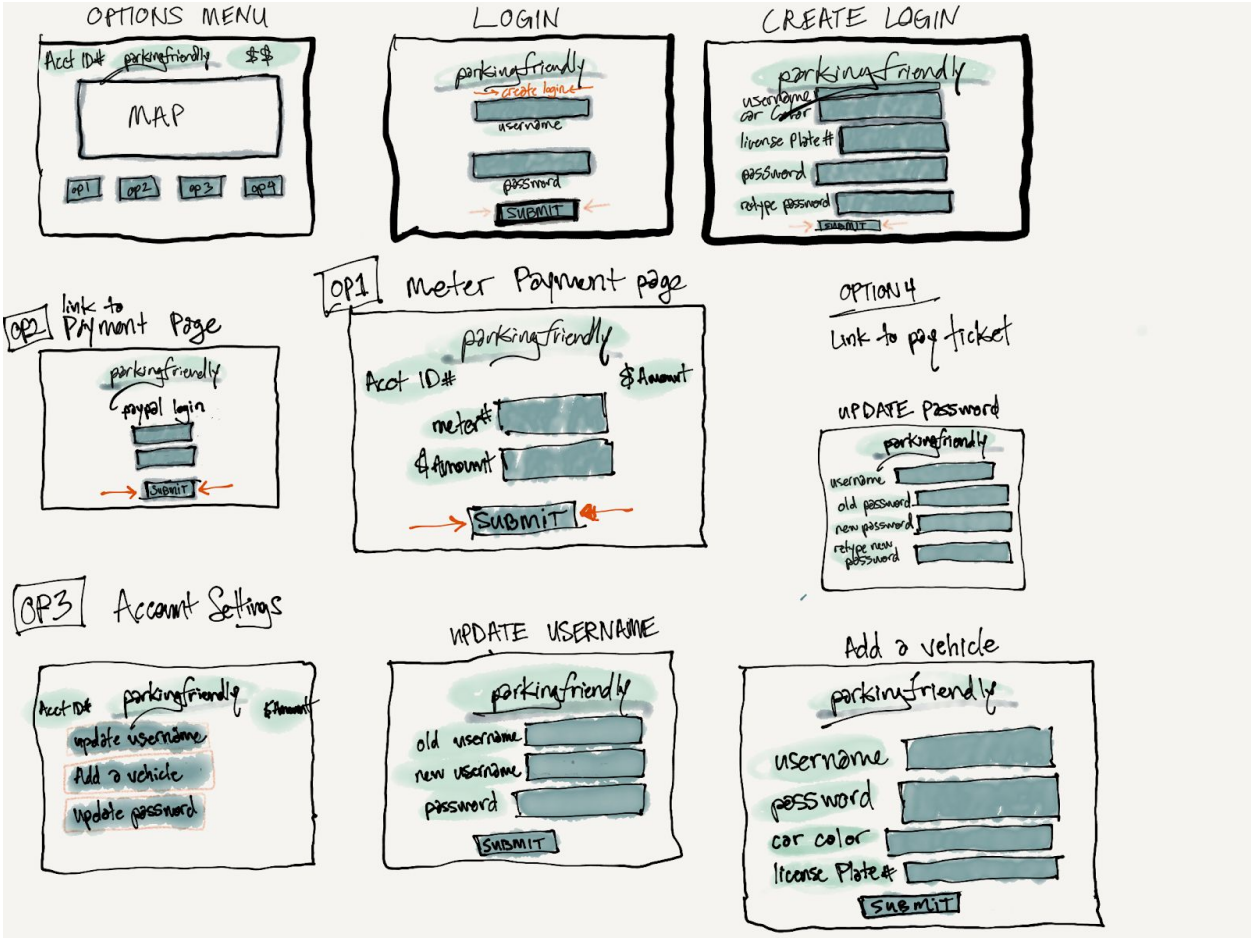
As an
admin;

I want to
Update parking meter locations;

So I can
Keep application up-to-date;

2. User Interface Design


Login Page
create account
create account (username & password)
Login success
option menu (shows map)
finished
Purchase credit
success or cancel
credit purchase
try again or different financial needed
pay
payment option
meter paid success



3. Data Design

Relational Structure form

Person(PERSONID, firstname, lastname, phonenumber, issecurity, isuser)

Security_Personnel(PERSONID)
foreign key (personid) references Person

Users(credit, licenseplatenum, color PERSONID)
foreign key (personid) references Person

Account(ACCOUNTID, email, password, personid, isregular, isadmin)
foreign key (personid) references Person

Admin_Acct(ACCOUNTID)
foreign key (accountid) references Account

Regular_Acct(ACCOUNTID)
foreign key (accountid) references Account

Security_Issues(PERSONID, TICKETID)
foreign key (personid) references Person
foreign key (ticketid) references Ticket

Security_Monitors(PERSONID, METERID)
foreign key (personid) references Person
foreign key (meterid) references Parking_Meter

Security_Gets(PERSONID, SCHEDULEID)
foreign key (personid) references Person
foreign key (scheduleid) references Schedule

User_Pays(PERSONID, TICKETID)
foreign key (personid) references Person
foreign key (ticketid) references Ticket

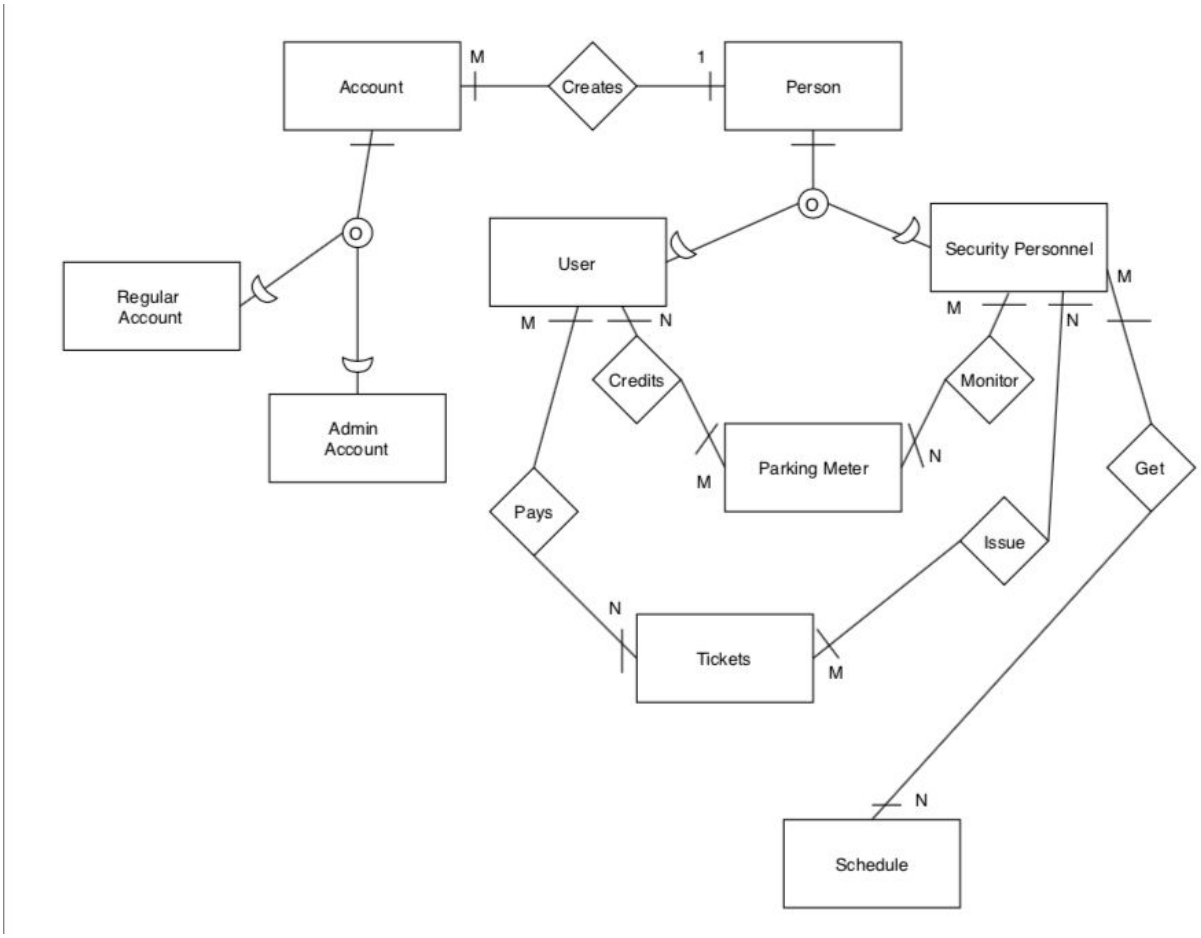
User_Credits(PERSONID, METERID)
foreign key (personid) references Person
foreign key (meterid) references Meter

Parking_Meter(METERID, timeremaining, currentuser, previoususer, usestoday, isinuse)

Ticket(TICKETID, price, due_date, date_issued, issuedby, issuedto, ispaid)

Schedule(SCHEDULEID, daytowork)

ER Diagram



4. Coding Standards
   - Languages going to be utilized
     - Javascript
     - SQL(postgreSQL/OracleSQL/mySQL)
   - Camel Case Notation
     - Functions with more than one word connected by underscore
   - Coding standards based off of W3schools.com standards for javascript
     - With the exception of changing from space indentation to tab indentation for ease of reading code
     - For compound statements, opening brackets will be on the next following line
     - Avoid lines longer than 80 characters
     - Constants and global variable names will be in upper-case