Readme.md Example

Big Blue Parking Genie

This app creates a marketplace for parking space/lot owners to list parking spaces for local events.

Workspace layout

The Big Blue Parking Genie web app will be stored in this repository.

The documentation and resources for this project will be kept in the "docs" folder. This includes use case diagrams, the project plan, database diagrams, and more as the project progresses.

The project will kept in the folder "app".

Version-control procedures

Collaborators should have a forked repository of the app in Alex's account of the project "group-7", in their Github. Each collaborator should clone the forked repository. Before each meeting, collaborators should submit a pull request so we can monitor progress and discuss issues.

Tool stack description and setup procedure

Django – convenient framework for this app since it comes with a simple SQLLite database. Since this app will use the database features while staying relatively small, Django was the obvious choice. Python – Django uses Python, and we are all very familiar with it.

Build instructions

Clone the project in gitbash. `bash $ git clone https://github.com/alexanderthurston/group-7`

Create and start a virtual environment `bash $ virtualenv --no-site-packages`

Install the project dependencies `bash $ pip install -r requirements.txt`

Create a file named "secret.sh" `bash touch secrets.sh`

Obtain a secret from MiniWebTool key and add to secret.sh `bash export SECRET_KEY='<secret_key>'`

Add secret.sh to .gitignore file Create a postgres db and add the credentials to settings.py

```
DATABASES = {
    'default':  {
        'ENGINE': 'django.db.backends.postgresql_psycopg2',
        'NAME': 'db_name',
        'USER': 'name',
        'PASSWORD': '',
        'HOST': 'localhost',
        'PORT': '',
        }
    }
```

Migrate in bash `bash $ python manage.py migrate`

Create an admin account `bash $ python manage.py createsuperuser`

Then complete migrations `bash $ python manage.py makemigrations group-7`

Then migrate again `bash $ python manage.py migrate`

and finally `bash $ python manage.py runserver`

Type localhost:8000 in a browser to see the app running.

Unit testing instructions

Unit tests will cover all use cases laid out in the use case diagrams. They can be found in the `unittests.py` file. The unit test class will prompt the user to select which use cases should be executed. The following use cases will be offered.
`bash Select which tests to perform. (ex. -> 1 3 8) 1. Customer order 2. Attendant validates customer at parking lot 3. Owner lists park`
At the end of the test run, the results will be given.
`bash 1. Customer order was completed successfully 3. Owner lists parking lot has failed 8. Customer adds money to current balance was c`

System testing instructions

Start by running an instance of the web app by first entering the correct repository and then by entering the following `bash $ python manage.py runserver` Now that the app is running, open an internet browser and enter the address `localhost:8000`. Login to the web app using the following credentials. Username: SystemTest, Password: systest

These credentials allow access to perform all actions as a customer, supervisor, owner, and attendant in a test environment.

Other development notes, as needed

Big Blue Parking Genie

This app creates a marketplace for parking space/lot owners to list parking spaces for local events.

Workspace layout

The Big Blue Parking Genie web app will be stored in this repository.

The documentation and resources for this project will be kept in the "docs" folder. This includes use case diagrams, the project plan, database diagrams, and more as the project progresses.

The project will kept in the folder "app".

Version-control procedures

Collaborators should have a forked repository of the app in Alex’s account of the project "group-7", in their Github. Each collaborator should clone the forked repository. Before each meeting, collaborators should submit a pull request so we can monitor progress and discuss issues.

Tool stack description and setup procedure

Django – convenient framework for this app since it comes with a simple SQLLite database. Since this app will use the database features while staying relatively small, Django was the obvious choice. Python – Django uses Python, and we are all very familiar with it.

Build instructions

Clone the project in gitbash. `bash $ git clone https://github.com/alexanderthurston/group-7`

Create and start a virtual environment `bash $ virtualenv --no-site-packages`

Install the project dependencies `bash $ pip install -r requirements.txt`

Create a file named "secret.sh" `bash touch secrets.sh`

Obtain a secret from MiniWebTool key and add to secret.sh `bash export SECRET_KEY='<secret_key>'`

Add secret.sh to .gitignore file Create a postgres db and add the credentials to settings.py

```
DATABASES = {
    'default':  {
        'ENGINE': 'django.db.backends.postgresql_psycopg2',
        'NAME': 'db_name',
        'USER': 'name',
        'PASSWORD': '',
        'HOST': 'localhost',
        'PORT': '',
        }
    }
```

Migrate in bash `bash $ python manage.py migrate`

Create an admin account `bash $ python manage.py createsuperuser`

Then complete migrations `bash $ python manage.py makemigrations group-7`

Then migrate again `bash $ python manage.py migrate`

and finally `bash $ python manage.py runserver`

Type localhost:8000 in a browser to see the app running.

Project Plan Example

Big Blue Parking Genie

Project Overview

This project aims to build a system for finding multiple modes of parking for USU events.

The system will allow customers to search through parking options in parking lots, driveways, lawns, and other privately owned parking options. The system will also accept listings from the owners of private lots for a 25% fee on their profits and provide a simple purchase verification system to be utilized by the owner.

Team Organization

Project Manager: Alex Thurston (may change over the course of the project)

Designers and Developers: Nathan Merrill, Jay Peterson, Alex Thurston

Software Development Process

The development will be broken up into five phases. Each phase will be a little like a Sprint in an Agile method and a little like an iteration in a Spiral process. Specifically, each phase will be like a Sprint, in that work to be done will be organized into small tasks, placed into a "backlog", and prioritized. Then, using on time-box scheduling, the team will decide which tasks the phase (Sprint) will address. The team will use a Scrum Board to keep track of tasks in the backlog, those that will be part of the current Sprint, those in progress, and those that are done.

Each phase will also be a little like an iteration in a Spiral process, in that each phase will include some risk analysis and that any development activity (requirements capture, analysis, design, implementation, etc.) can be done during any phase. Early phases will focus on understanding (requirements capture and analysis) and subsequent phases will focus on design and implementation. Each phase will include a retrospective.

Phase	Iteration
1.	Phase 1 - Requirements Capture
2.	Phase 2 - Analysis, Architectural, UI, and DB Design
3	Phase 3 - Implementation, and Unit Testing
4	Phase 4 - More Implementation and Testing

We will use Unified Modeling Language (UML) to document user goals, structural concepts, component interactions, and behaviors.

Communication policies, procedures, and tools

Discord – Main channel for communication. Used for group calls, file sharing, and other collaborative activities.

Google Drive – Storage for files needing collaborative effort and review for Milestone 1.

GitHub – Formal repository used for submissions, version control, data tracking, and communication with Professor Dan Watson and Bradley Payne.

Risk Analysis

- Database Structure
  - Likelihood – Low
  - Severity – Very High
  - Consequences – Ineffective data tracking leading to confusion concerning balances, transactions, account information, etc.
  - Work-Around – None. System loses value and functionality without proper database implementation.
- Login
  - Likelihood – Low
  - Severity – Med-High
  - Consequences – Dissatisfactory customer experience concerning preferences, transaction information, cancellations, and current balance.
  - Work-Around –
- Verification System
  - Likelihood – Low
  - Severity – Med
  - Consequences – Lack of security concerning exchange of parking services
  - Work-Around – Verify by name and vehicle type
- UI
  - Likelihood – Low
  - Severity – Very High
  - Consequences – Inability to interact with users in a clear and efficient way.
  - Work-Around – None. System loses value and functionality if users are not able to interact with it.

- Hosting
    - Likelihood – Low
    - Severity – Med
    - Consequences – Inability for system to host or serve information essential for system functionality
    - Work-Around – Host system through a hosting service

Configuration Management

See the README.md in the Git repository.

Requirements Definition Example

Requirements Definition
Group 7

Big Blue Parking Genie

1. Introduction and Context

During large USU events, demand for parking is high and USU-owned parking spaces are limited. To help fix this problem, this project aims to build a system to allow users to list and rent parking spaces for USU events.

The system will allow customers to search through parking options in parking lots, driveways, lawns, and other privately owned parking options. Customers will be able to reserve and pay for parking spots through the system, and be given a way to validate their parking purchase to a parking lot attendant when they arrive at the lot.

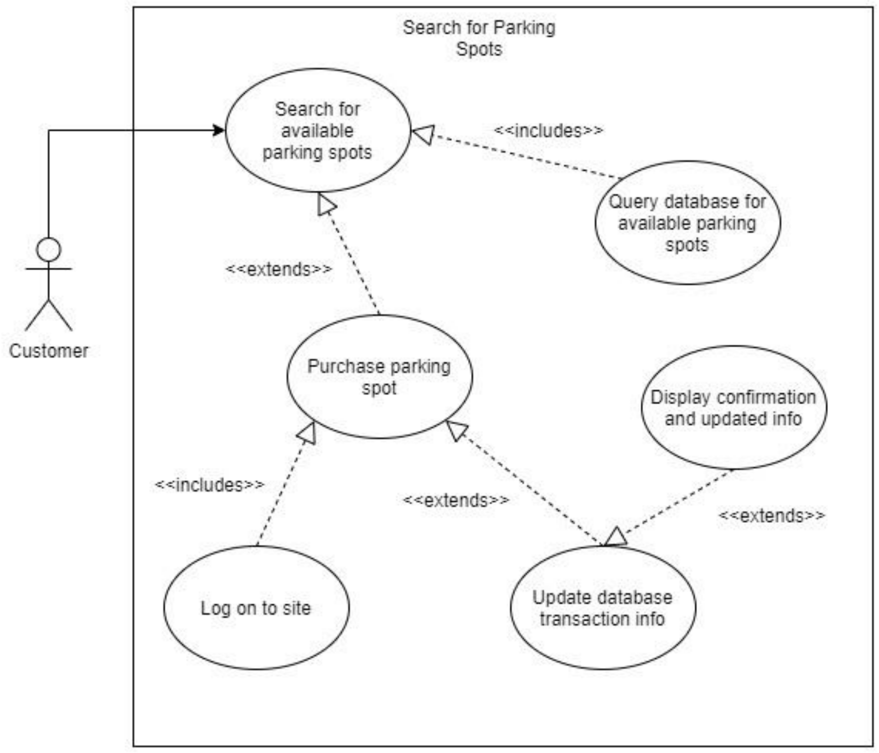
The system will allow parking lot owners to list their parking spaces, and will handle payment collection. It will also give parking lot owners a way to check customers' parking validations, as well as authorize other parking lot attendants to be able to check parking validations.

By giving private parking owners a platform to rent out their parking spaces, the system will alleviate the scarcity of parking usually experienced at large USU events.

2. Users and their Goals

The following UML use case diagrams will describe the system's actors and the actors' goals.

Figure 1 - Customer searches for parking spot


Search for Parking Spots
Search for available parking spots
<<includes>>
Query database for available parking spots
<<extends>>
Customer
Purchase parking spot
Display confirmation and updated info
<<includes>>
<<extends>>
<<extends>>
Log on to site
Update database transaction info

Participating actor: Customer

Entry Conditions:

- Customer wants to rent a parking spot

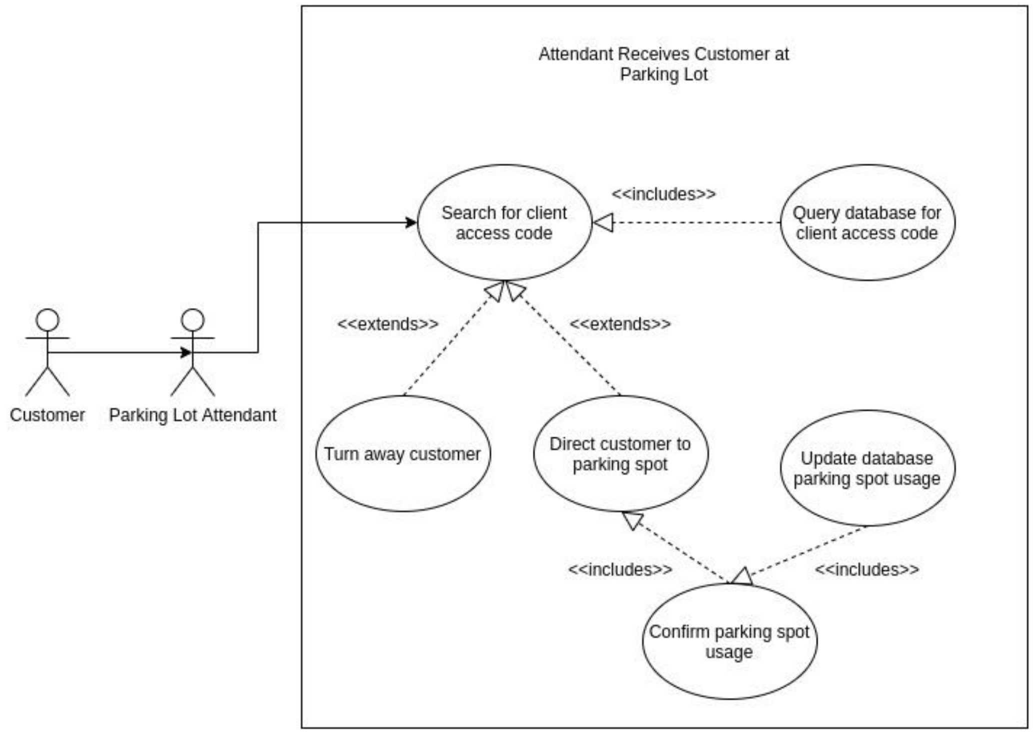
Exit Conditions:

- Customer purchases parking spot
- Customer stops searching for parking spot

Event Flow:

1. Customer logs on to system.
2. Customer requests available parking spots.
3. System displays available parking spots
4. Customer selects a parking spot.
5. Customer pays for parking spot.
6. System displays confirmation of purchase.

Figure 2 - Attendant receives Customer at Parking Lot


Attendant Receives Customer at Parking Lot
Search for client access code
<<includes>>
Query database for client access code
<<extends>>
<<extends>>
Customer
Parking Lot Attendant
Turn away customer
Direct customer to parking spot
Update database parking spot usage
<<includes>>
<<includes>>
Confirm parking spot usage

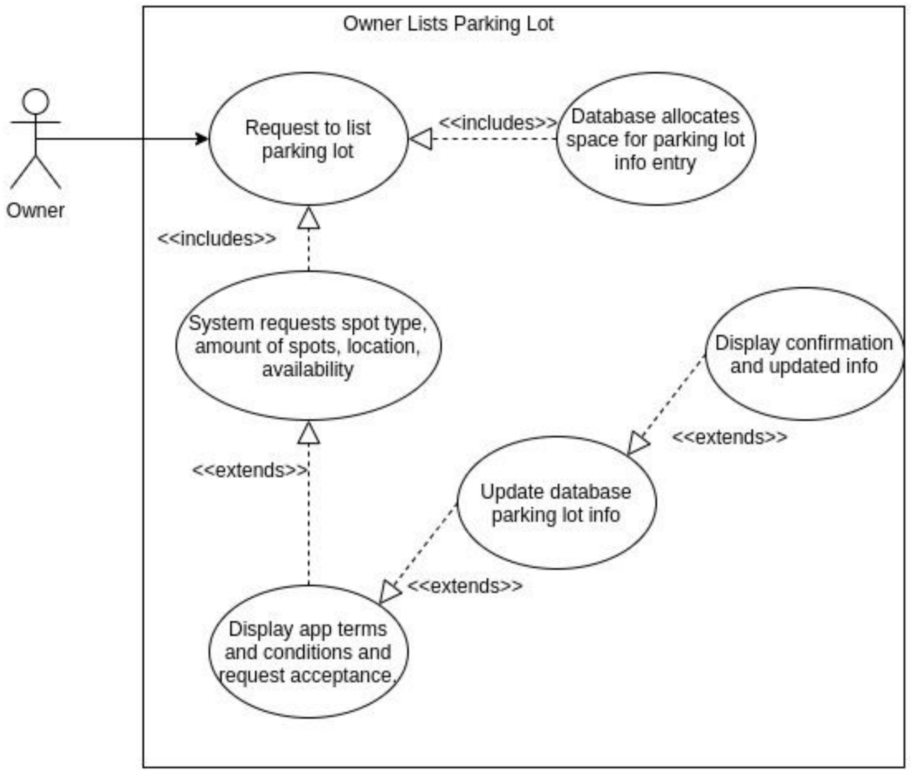
Participating Actors: Customer, Lot Attendant

Entry Condition:
- Customer arrives at parking lot

Exit Conditions:
- Customer is turned away
- Customer is directed to parking spot

Event Flow:
1. Customer arrives at parking lot.
2. Customer shows confirmation code to Lot Attendant.
    a. If Customer does not have confirmation code, Lot Attendant turns Customer away.
3. Lot Attendant verifies confirmation code using System.
    a. If confirmation code is correct, Lot Attendant directs Customer to parking spot.
    b. If confirmation code is incorrect, Lot Attendant turns Customer away.

Figure 3 - Owner lists Parking Lot


Owner Lists Parking Lot
Owner
Request to list parking lot
<<includes>>
Database allocates space for parking lot info entry
<<includes>>
System requests spot type, amount of spots, location, availability
Display confirmation and updated info
<<extends>>
<<extends>>
Update database parking lot info
<<extends>>
Display app terms and conditions and request acceptance.

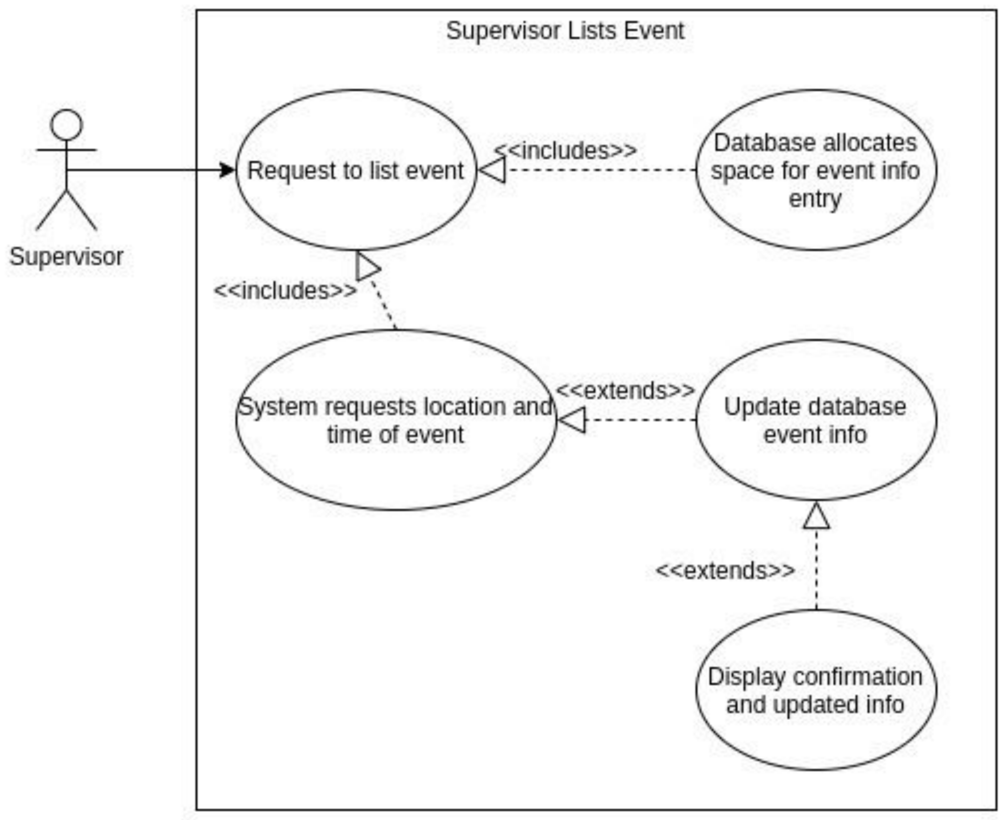
Participating Actor: Owner

Entry Conditions:
- Owner is logged in

Exit Condition:
- Parking lot is listed in the system

Event Flow:
1. Owner requests to list parking lot
2. System requests parking lot information
3. Owner inputs information and accepts terms
4. System updates database
5. System displays confirmation to Owner that the parking lot is listed

Figure 4 - Supervisor lists Event


Supervisor Lists Event
Supervisor
Request to list event
<<includes>>
Database allocates space for event info entry
<<includes>>
System requests location and time of event
<<extends>>
Update database event info
<<extends>>
Display confirmation and updated info

Participating Actor: Supervisor

Entry Condition:

- Supervisor is logged in to system

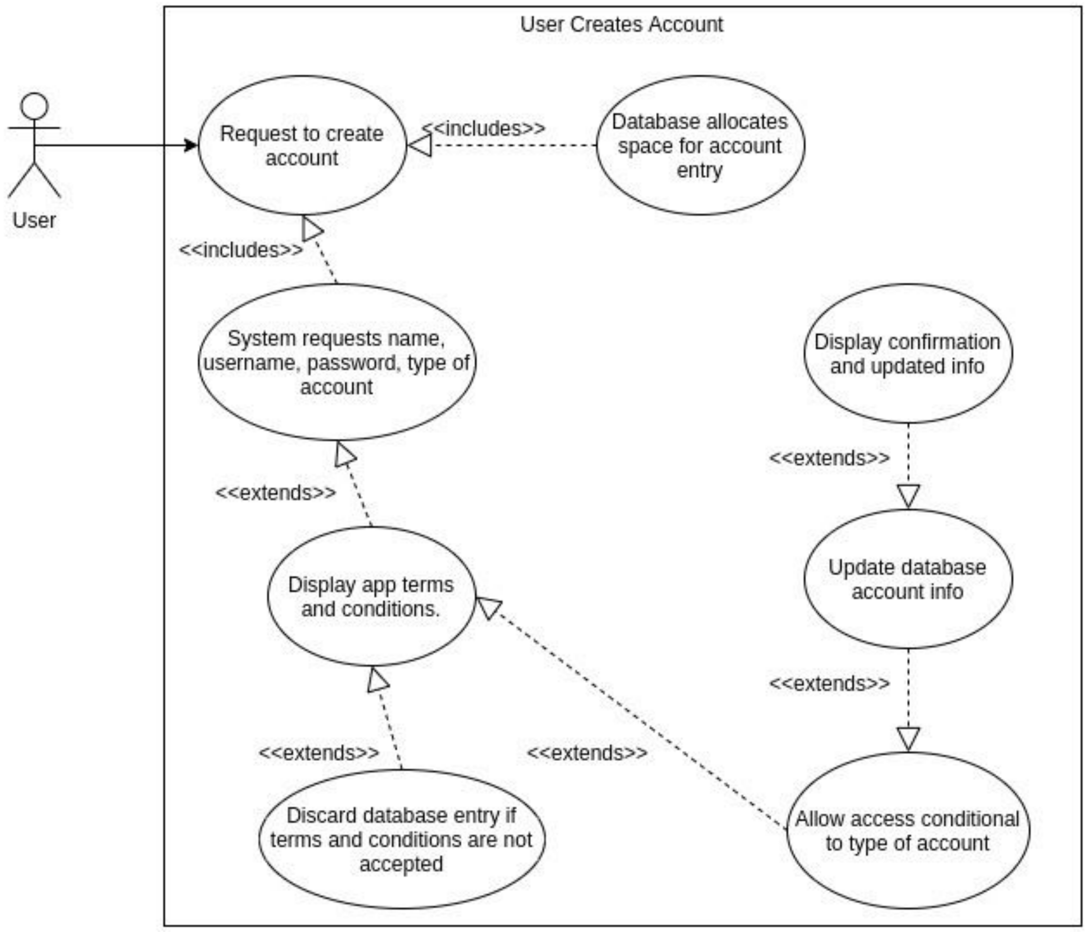
Exit Condition:

- Event is listed in system

Event Flow:

1. Supervisor requests to list event
2. System requests event information
3. Supervisor inputs information
4. System updates database
5. System displays confirmation to Supervisor that the event is listed

Figure 5 - User creates account


User Creates Account
User
Request to create account
<<includes>>
Database allocates space for account entry
<<includes>>
System requests name, username, password, type of account
Display confirmation and updated info
<<extends>>
<<extends>>
Update database account info
Display app terms and conditions.
<<extends>>
<<extends>>
<<extends>>
Discard database entry if terms and conditions are not accepted
Allow access conditional to type of account

Participating Actor: User

Entry Condition:

- User wants to create an account

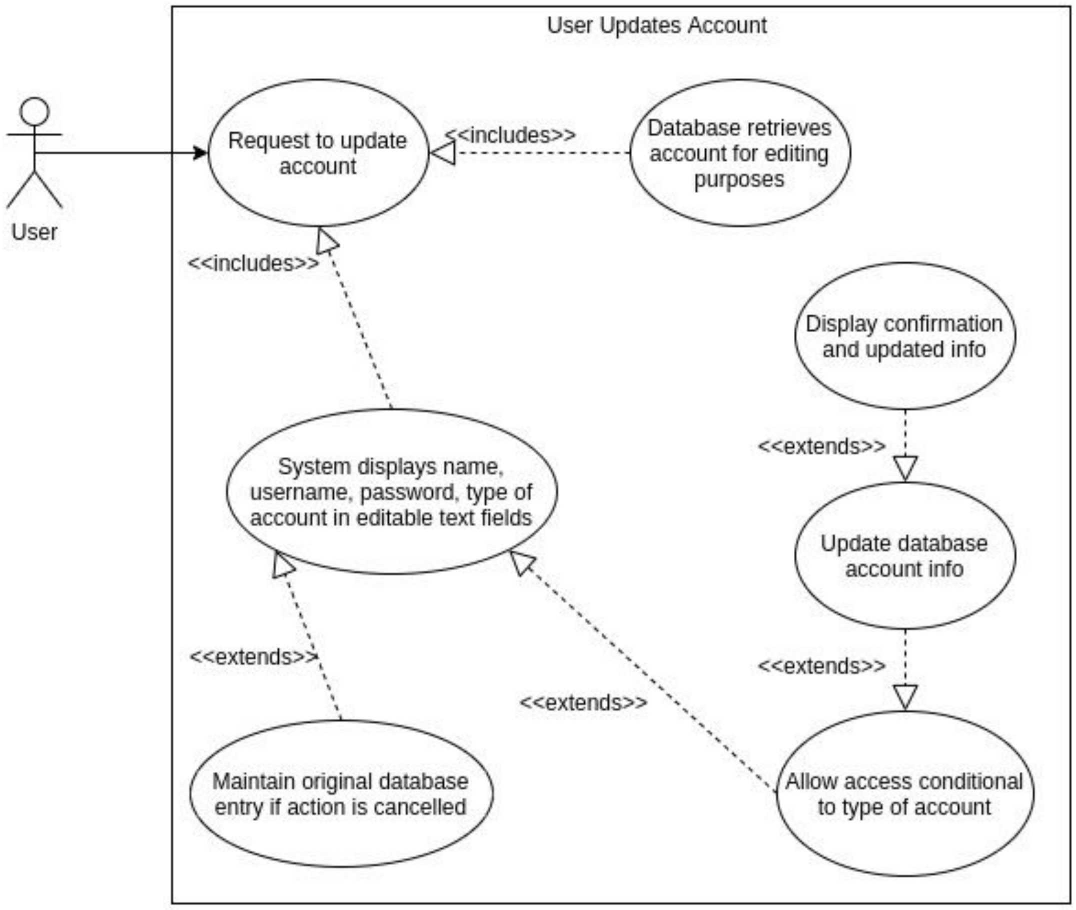
Exit Conditions:

- Account is created
- User rejects terms and account is not created

Event Flow:

1. User requests to create account
2. System requests User's information
3. System displays terms and conditions
    a. If User accepts, account is created
    b. If User declines, account is not created

Figure 6 - User updates account


User Updates Account
User
Request to update account
<<includes>>
Database retrieves account for editing purposes
<<includes>>
Display confirmation and updated info
<<extends>>
System displays name, username, password, type of account in editable text fields
Update database account info
<<extends>>
<<extends>>
<<extends>>
Maintain original database entry if action is cancelled
Allow access conditional to type of account

Participating Actor: User

Entry Condition:

- User is logged in

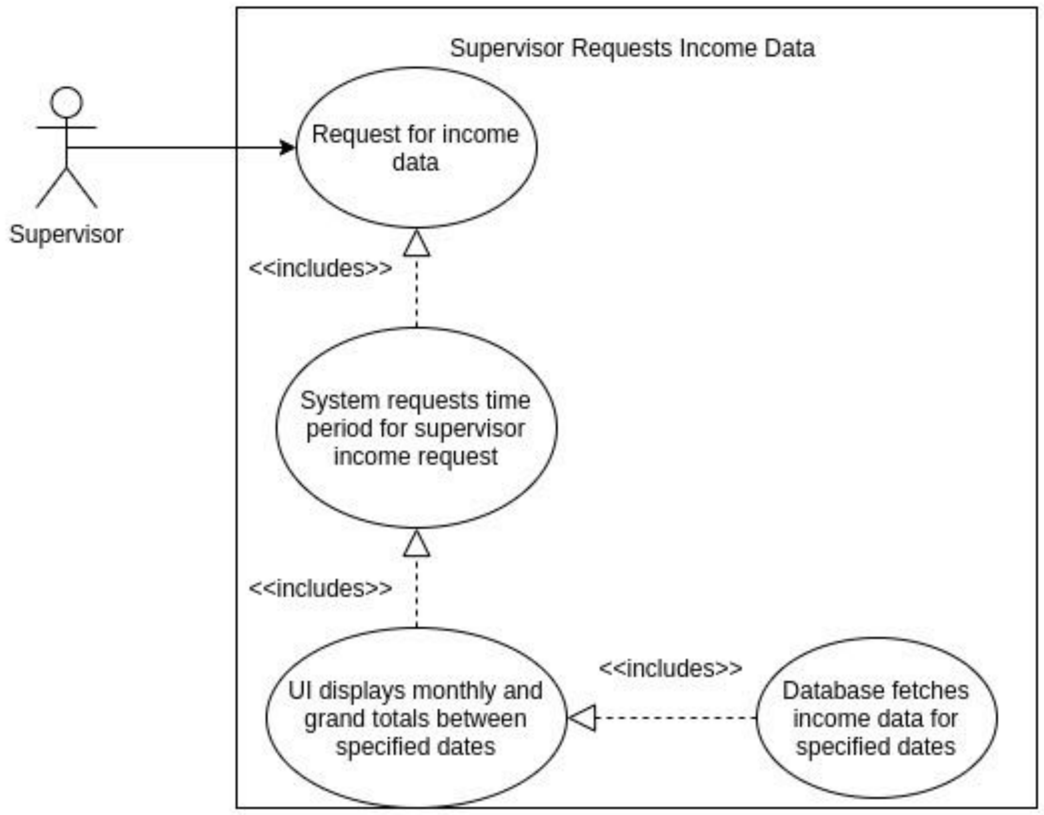
Exit Conditions:

- User cancels action
- User account is updated

Event Flow:

1. User requests to update account
2. System displays editable account features
3. User can edit fields
    a. If user chooses to save, new account features are saved
    b. If user chooses to cancel, account features are not saved

Figure 7 - Supervisor requests income data


Supervisor Requests Income Data
Supervisor
Request for income data
<<includes>>
System requests time period for supervisor income request
<<includes>>
UI displays monthly and grand totals between specified dates
<<includes>>
Database fetches income data for specified dates

Participating Actor: Supervisor

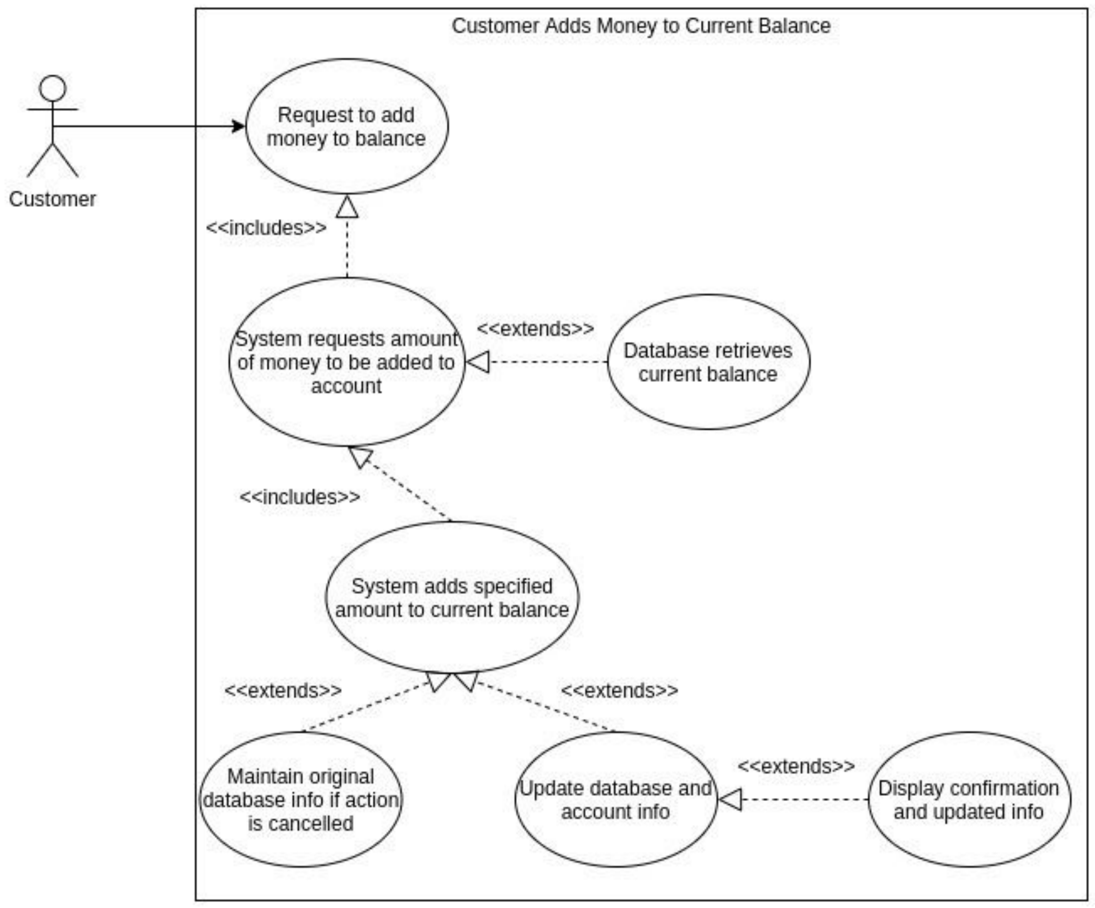
Entry Condition: Supervisor is logged in

Exit Condition: Supervisor navigates away from page

Event Flow:

1. Supervisor requests to see income data
2. Supervisor specifies time and date of desired data
3. System displays data

Figure 8 - Customer adds money to current balance


Customer Adds Money to Current Balance
Customer
Request to add money to balance
<<includes>>
System requests amount of money to be added to account
<<extends>>
Database retrieves current balance
<<includes>>
System adds specified amount to current balance
<<extends>>
<<extends>>
Maintain original database info if action is cancelled
Update database and account info
<<extends>>
Display confirmation and updated info

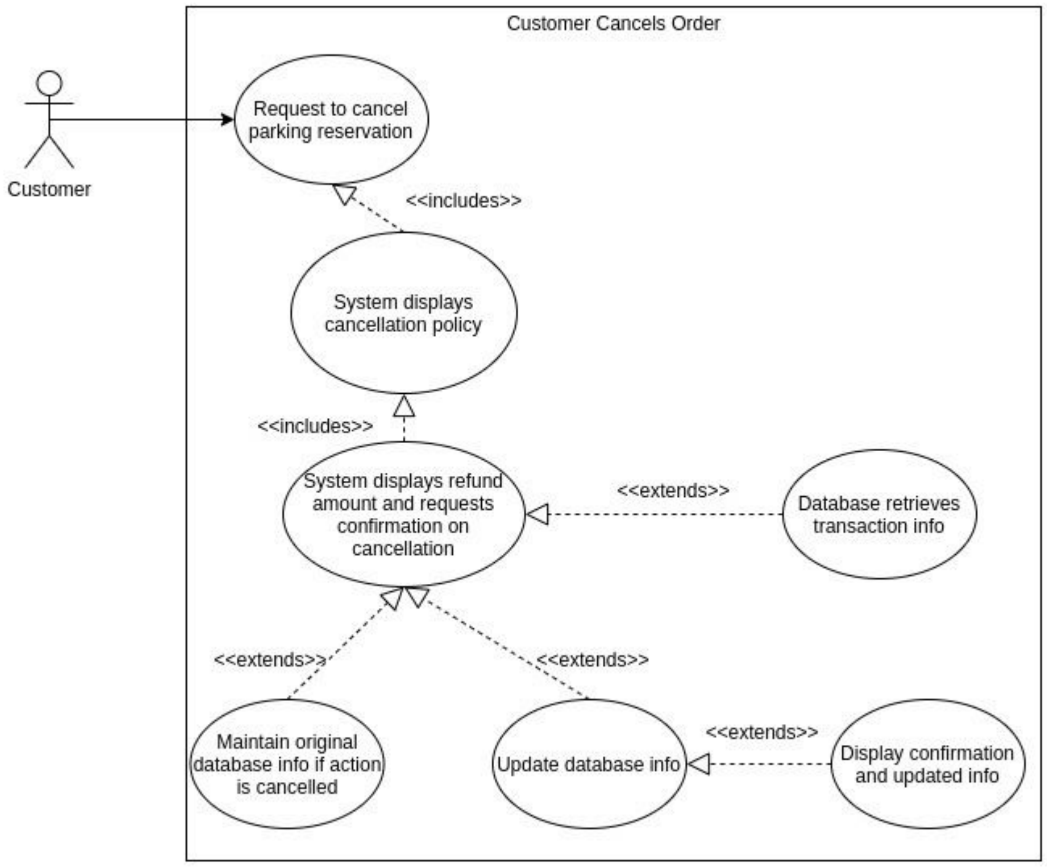
Participating Actor: Customer

Entry Condition:
- Customer is logged in

Exit Conditions:
- Money is added to the balance
- Action is cancelled

Event Flow:
1. Customer requests to add money to their balance
2. Customer inputs amount of money to enter
3. System saves new balance amount
4. System displays confirmation to Customer

Figure 9 - Customer cancels order


Customer Cancels Order
Customer
Request to cancel parking reservation
<<includes>>
System displays cancellation policy
<<includes>>
System displays refund amount and requests confirmation on cancellation
<<extends>>
Database retrieves transaction info
<<extends>>
<<extends>>
Maintain original database info if action is cancelled
Update database info
<<extends>>
Display confirmation and updated info

Participating Actor: Customer

Entry Conditions:

- Customer is logged in
- Customer has an order in the system

Exit Conditions:

- Order is cancelled
- Action is cancelled

Event Flow:

1. Customer requests to cancel reservation
2. System displays cancellation policy
3. System displays refund amount
4. System requests confirmation
    a. If action is cancelled, no change is made to Customer's reservation or account balance
    b. If action is confirmed, Customer's account is credited and reservation is removed

3. Functional Requirements

1. User Authentication and Access
   - 1.1. The system must require all users to authenticate themselves before giving them access to the system.
     - 1.1.1. On first login, the system must allow the user to create an account with a username and password.
       - 1.1.1.1. The system must get the user's email address during account creation
     - 1.1.2. On subsequent login, the system must allow users to enter their username and password. If entered correctly, the user must be given access to the system. If entered incorrectly, the system must allow the user to try again.
   - 1.2. Users can have a combination of the following access rights: Customer, Lot Attendant, Lot Owner, and Supervisor.
     - 1.2.1. Users with Customer rights must have access to all Customer features. See FR #3
     - 1.2.2. Users with Lot Owner rights must have access to all Lot Owner features. See FR #4
     - 1.2.3. Users with Lot Attendant rights must have access to all Lot Attendant features. See FR #5
     - 1.2.4. Users with Supervisor rights must have access to all Supervisor features. See FR #6

2. User Profile Features
   - 2.1. The system will allow any authenticated user (one who is logged in) to modify their own password.
   - 2.2. The system will allow any authenticated user (one who is logged in) to modify their own email address.
   - 2.3. The system should not allow any user that does not have Supervisor rights to view or modify any other user profile
   - 2.4. The system should allow users to view the balance of their account
   - 2.5. The system should allow users to preload money into their account
     - 2.5.1. No real money is required, users can preload money by typing a number in
   - 2.6. The system should allow users to withdraw money from their account

3. Customer Features
   - 3.1. All users will be given Customer rights and will have access to all Customer features.
   - 3.2. The system will allow users with Customer rights (hereafter referred to as "Customers") to browse, reserve, and pay for parking spaces.
     - 3.2.1. The system will allow Customers to select an event from a list. When the Customer selects an event, the system will display a list of available parking spaces.
       - 3.2.1.1. The system will display the address, distance from event location, type, and price of the available parking spaces to the Customer.
     - 3.2.2. Customers will be able to select a parking space from the list to rent.

- 3.2.3. If a Customer decides to rent a parking space, the system will require the Customer to pay for the parking space immediately.
  - 3.2.3.1. Customers pay for parking from their preloaded account balance
- 3.3. After a Customer pays for a parking space, the system must give the Customer a way to show the parking lot attendant that they have rented a parking space
  - 3.3.1. Confirmation codes can be a QR code or an access code, e.g. “ABC123”
- 3.4. The Customer should be able to view their parking spot reservations from their account.
  - 3.4.1. The Customer should be able to access their confirmation codes for their parking reservations from their account.
- 3.5. The Customer should have the ability to cancel their parking spot reservations
  - 3.5.1. Refunds will only be given if cancellation occurs at least 24 hours before the event time

4. Lot Owner Features
   - 4.1. All users will be given Lot Owner rights and have access to all Lot Owner features.
   - 4.2. The system will allow the users with Lot Owner rights (hereafter referred to as “Lot Owners”) to list parking spots to be rented for events.
     - 4.2.1. When listing a parking lot, the system will prompt the Lot Owner for the address of the lot, the event name, the distance from the lot to the event location, the type and number of parking spots, and their prices.
     - 4.2.2. The “type” refers to what the spot is best suited for, e.g. motorcycles, compact cars, normal cars, oversize, tailgating
   - 4.3. Lot Owners can see how many of their parking spots have been rented
   - 4.4. Lot Owners are also given Lot Attendant rights for their parking lots
   - 4.5. Lot Owners should have the ability to give other user accounts Lot Attendant rights for their parking lots.
   - 4.6. When Customers rent a parking place owned by a Lot Owner, 75% of the listed price will be deposited in the Lot Owner’s account
     - 4.6.1. This percentage can be changed by the Supervisor

5. Lot Attendant Features
   - 5.1. Only users with Lot Attendant rights (hereafter referred to as “Lot Attendants”) should have access to Lot Attendant features.
   - 5.2. To gain Lot Attendant rights for a parking lot, users must either be the Lot Owner for the lot or be given Lot Attendant rights by the parking lot’s owner.
     - 5.2.1. The system must provide a way for Lot Owners to authorize other users as Lot Attendants for their parking lots.
   - 5.3. Lot Attendants must be able to check Customers in using the Customers’ QR code or access code
     - 5.3.1. The system must give Lot Attendants a way to scan or enter the code to validate that the Customer has rented a parking spot
       - 5.3.1.1. If a QR code is used, Lot Attendants can use a separate QR code scanning app on their phone, but the QR code must link back to the system for confirmation

5.3.2. After validation, the system must display the ID number of the parking spot the Customer has rented so the Lot Attendant can direct the Customer to their parking spot.

6. Supervisor Features
   6.1. Only the user with Supervisor rights (hereafter referred to as the "Supervisor") should have access to Supervisor features.
   6.2. The Supervisor must have the ability to list events.
      6.2.1. When listing a new event, the system will prompt the Supervisor for the name, date, and address of the event
   6.3. The system will deposit a percentage of all transactions made into the Supervisor's account
      6.3.1. This percentage will start at 25%
      6.3.2. The Supervisor should be able to change this percentage
   6.4. Supervisors should have the ability to view high-level usage statistics for the system.

4. Non-Functional Requirements

1. The system must use a database
   - 1.1. The system's database must store user account information, including the following fields: Username, Password, Email Address, Account Balance
   - 1.2. The system must store information about the parking lots represented in the system
     - 1.2.1. The system must keep track of individual parking spaces within a lot by representing them with integers from 1 to n

2. The team will use the Git version control system, with GitHub as a remote repository.

3. The system must be deployable
   - 3.1. Can be either locally hosted or hosted by a cloud service

4. The system's interface must be mobile device friendly.

5. Future Features

This section contains a list of features that are beyond the scope of the project, but could be implemented in future versions.

1. The system's list interface could be converted to a map interface, where parking lots are listed as graphical icons
2. The system could calculate distance from the event automatically, without requiring input from Lot Owners
3. The system could provide users with a map of their chosen parking lot to make it easier to find their spot
4. The system could email users a confirmation of their parking spot rental, which would include their access code

6. Glossary

This section contains a list of important terms and their definitions.

Customer - a user that uses the system to rent parking spaces that others have listed

Lot Attendant - a user, either a Lot Owner or authorized by a Lot Owner, that checks Customers' validations when Customers arrive at parking lots

Lot Owner - a user that lists parking spaces to be rented out

Supervisor - the user that lists events and collects part of all profits

System - refers to the application that the project aims to build

User - refers to any of the four types of users of the system (Customers, Lot Owners, Lot Attendants, Supervisor)