Lawn Wizard

Project Overview

This app creates a crowdsourcing environment for local yard care work.

Team Organization

Project Manager: Kelsye Anderson (may change over the course of the project)

Designers and Developers: Camila Vulcano, Thomas Lau, James Seelos

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 Rob Johnson.

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 – None
- Verification System
    - Likelihood – Low
    - Severity – Med
    - Consequences – Lack of security concerning exchange of lawn care services
    - Work-Around – Verify by name
- 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 google play store

Configuration management

See the README.md in the Git repository

Requirements

Introduction and Context

Finding time for lawn care can be a hassle and looking for lawn care workers can be even more of a struggle. To help fix this problem, this project aims to provide an application for homeowners to make finding lawn care workers stress-free and easy.

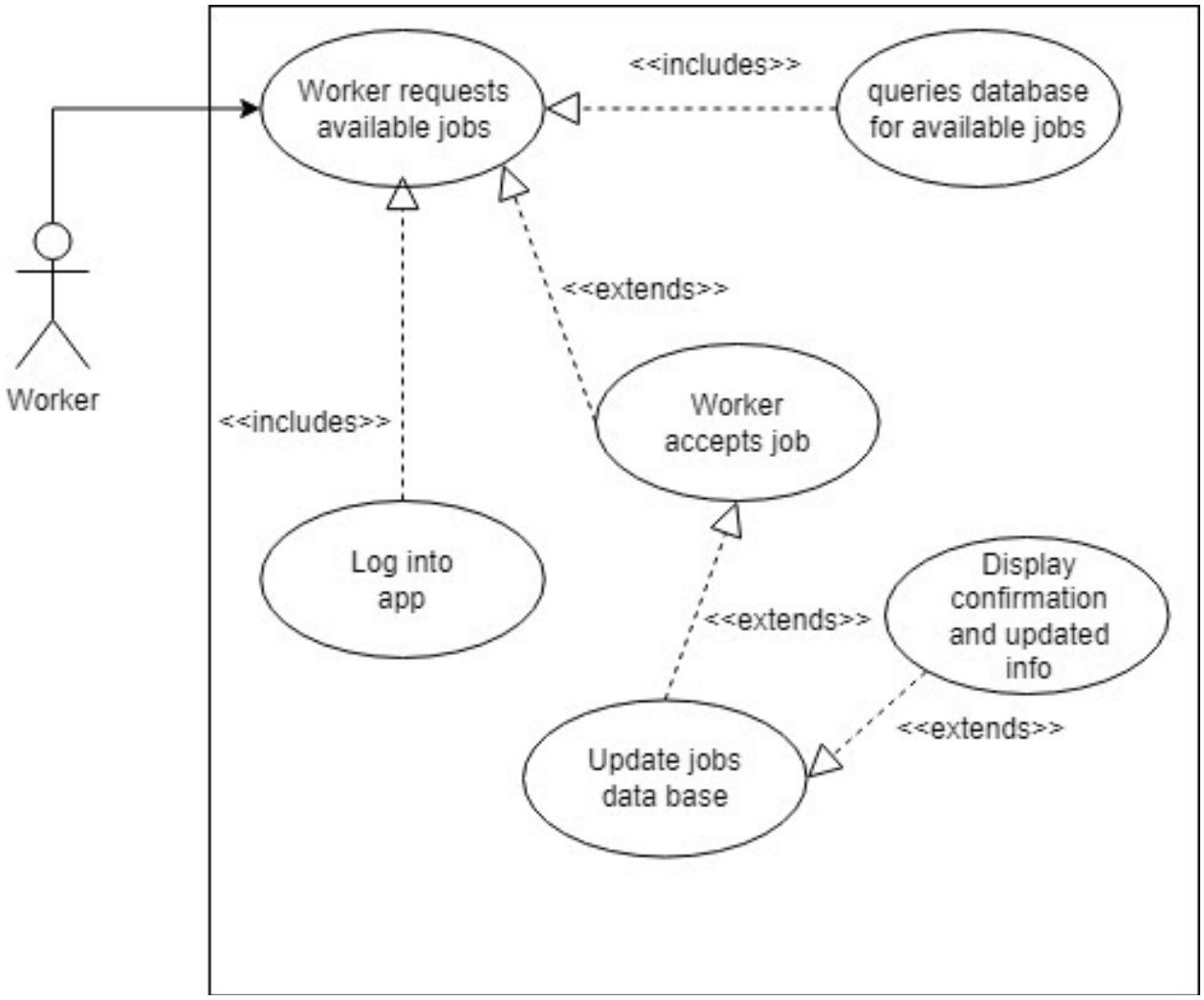
In this app, users will be able to post a job to the app where local workers will be able to then view the job. The posted job will contain the job description, pictures of the lawn, location of the home, the payout, estimated time of labor, when the job needs to be done, and any other additional notes the worker will need to know. The posted job will be sent out to the top 10 workers which will be determined based on location, rating, etc. The first worker to accept the post will then be assigned to that job. If now worker accepts the job, it will be sent to the next 10 workers.

The worker will then go do the job based on the information from the job post. Once completed, the payment will be processed, and the homeowner will be able to leave a rating for the worker. If the job was not completed, the homeowner will be able to submit a complaint stating how the job was not completed, and if approved, the homeowner will be refunded.

By giving workers and homeowners a platform like this, finding lawn care jobs should never be a problem again.

Users and their Goals

Figure 1 - Worker accepts a job



Participating actor: Worker

Entry Conditions:

- Worker gets sent a job

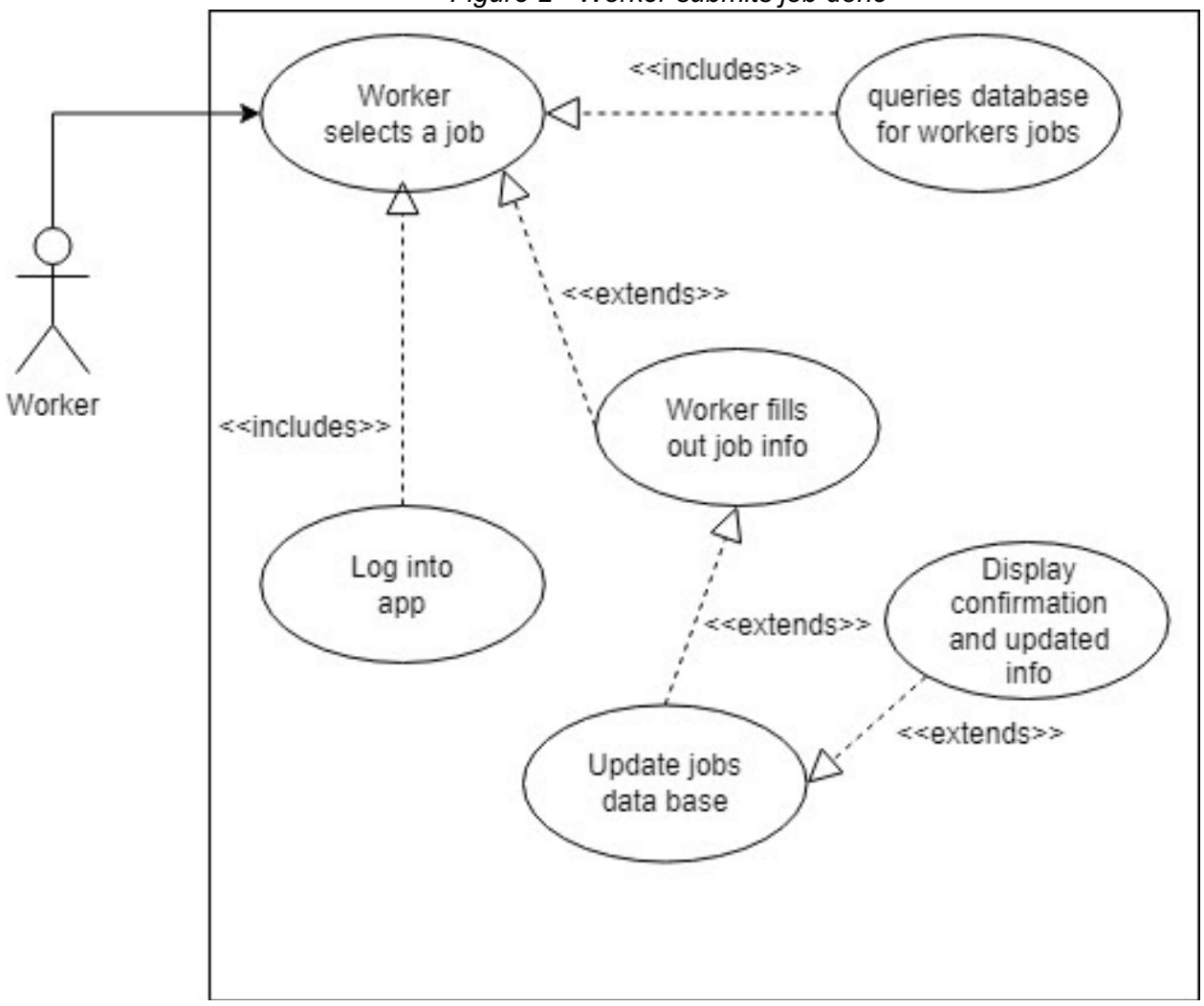
Exit conditions:

- Worker accepts the job
- Worker does not accept job

Event Flow:

1. Worker logs on to system.
2. System displays available jobs.
3. Worker accepts job.
4. Customer is notified that worker has been found.

Figure 2 - Worker submits job done



Participating actor: Worker

Entry Conditions:

- Worker submits that job is done

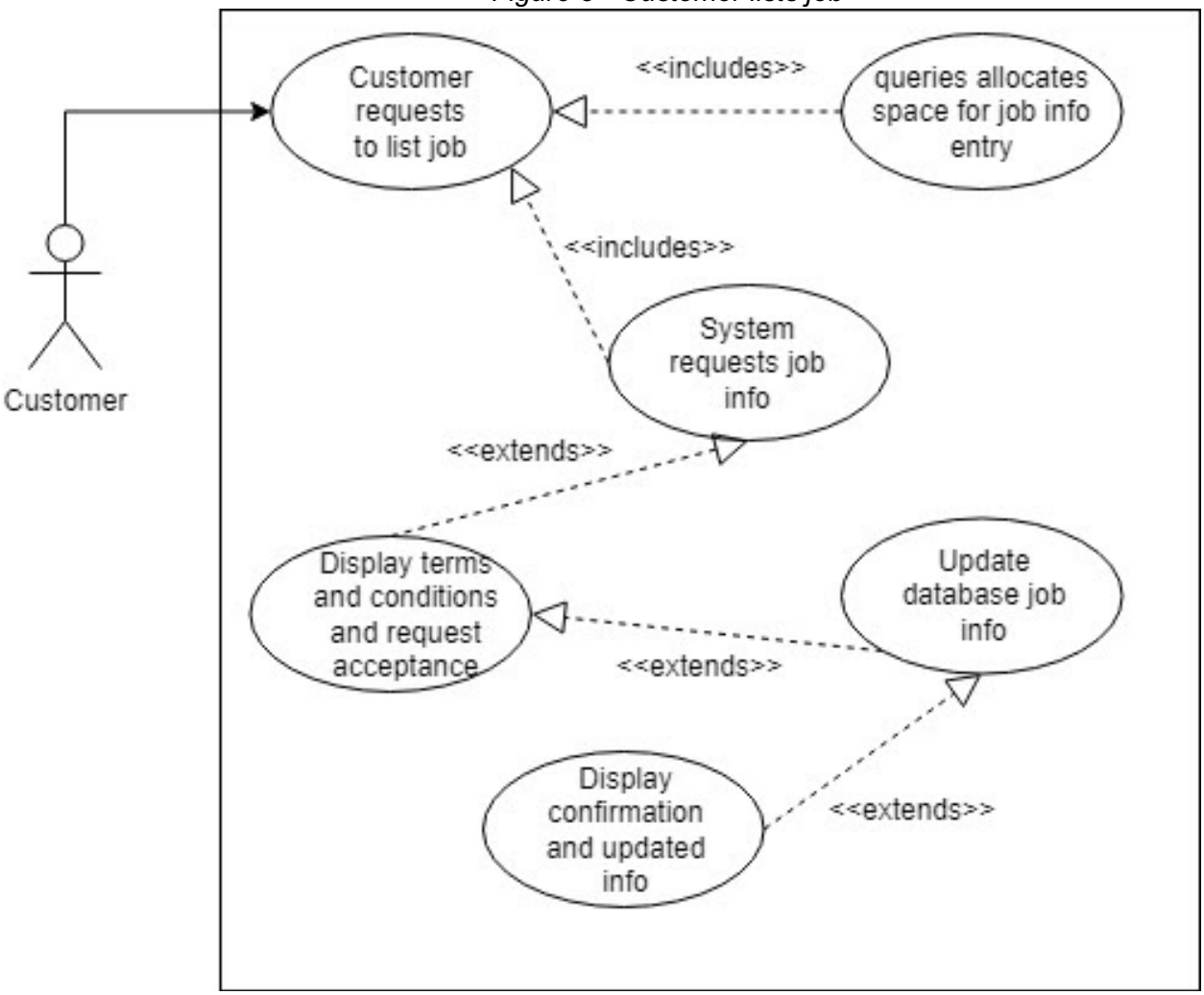
Exit conditions:

- Worker gets paid

Event Flow:

1. Worker signs into app
2. Worker selects job that has been done and submits that the work is done
3. System transfers money from Customers account to workers account

Figure 3 - Customer lists job



Participating actor: Customer

Entry Conditions:

- Customer has a job that they want done

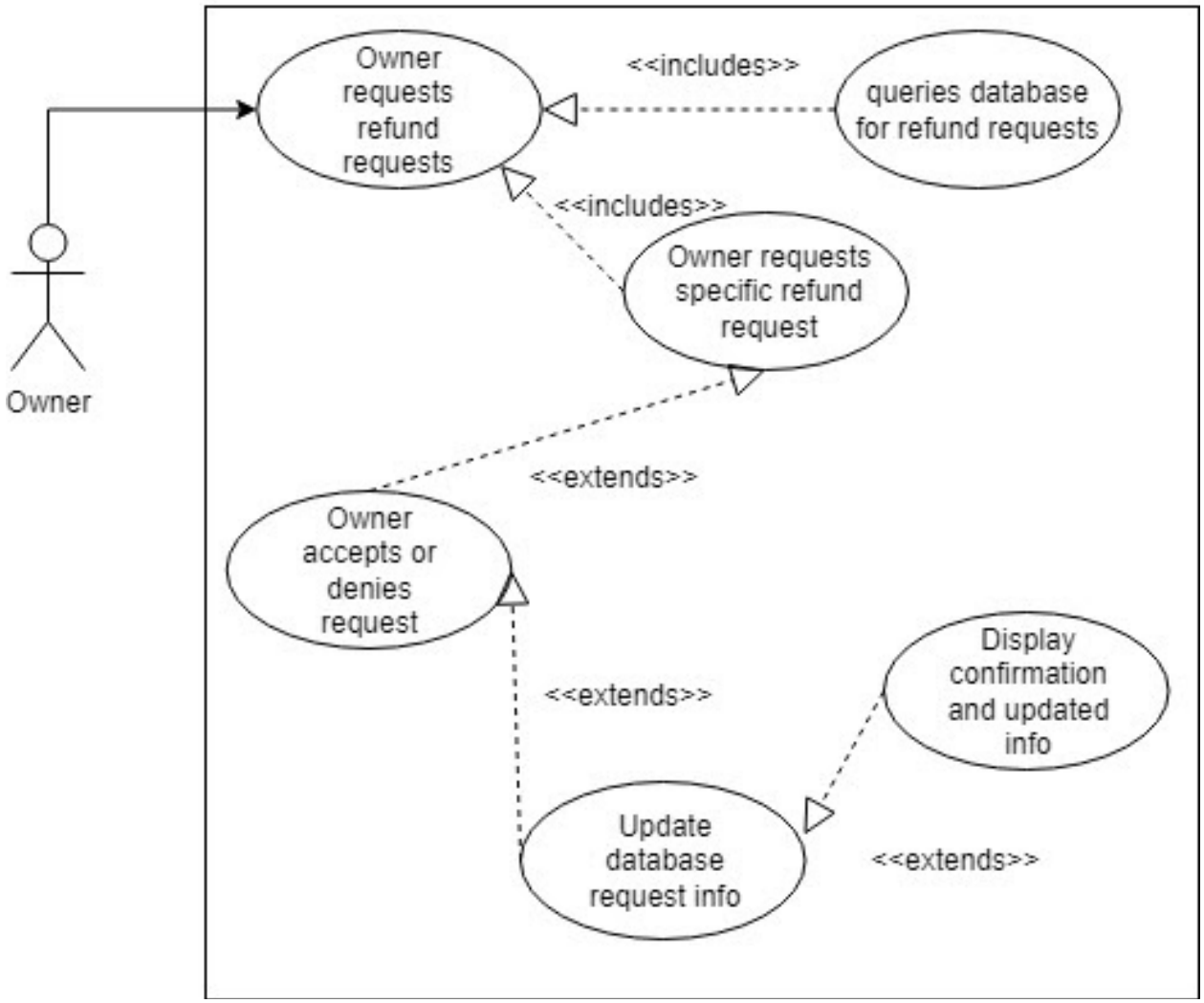
Exit conditions:

- Customer lists job
- Customer cancels job

Event Flow:

1. Customer requests to list job
2. System requests job information
3. Customer inputs information and accepts terms
4. If the Customer dosent accept terms then the job is canceled
5. System updates database
6. System confirms to Customer that the job is listed

Figure 4 - Owner issues refunds



Participating actor: Owner

Entry Conditions:

- Owner is logged in to system

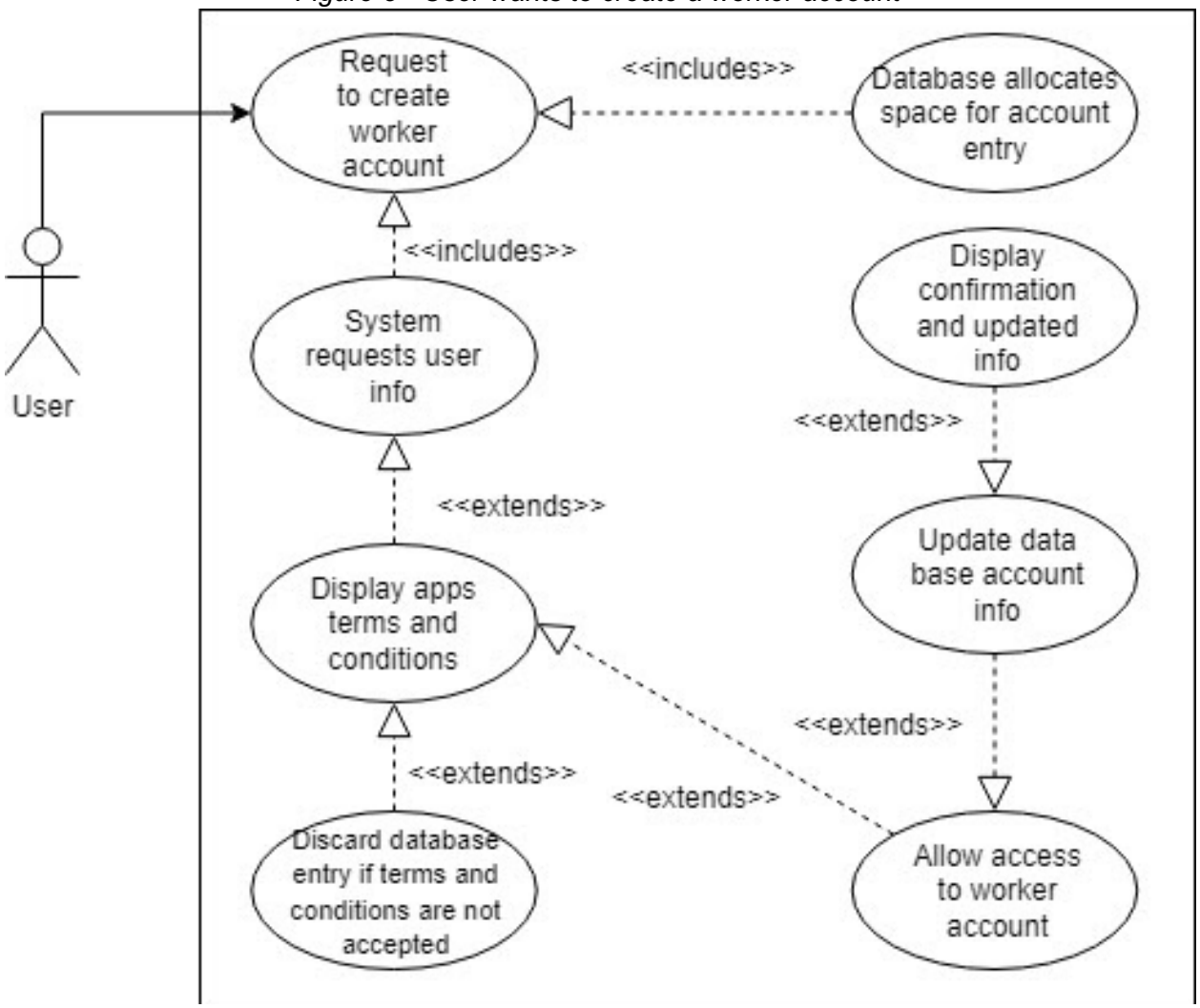
Exit conditions:

- Refund is issued
- Refund is denied

Event Flow:

1. Owner requests to list refunds
2. System displays refund requests
3. Owner requests specific refund
4. System displayed refund information
5. Owner issues refund
    a. If the owner denies the refund the refund request is taken out of the system and customer is notified
6. System takes the amount refunded from the worker and gives it to the customer
7. The worker and customer are both notified

Figure 5 - User wants to create a worker account



Participating actor: User

Entry Conditions:

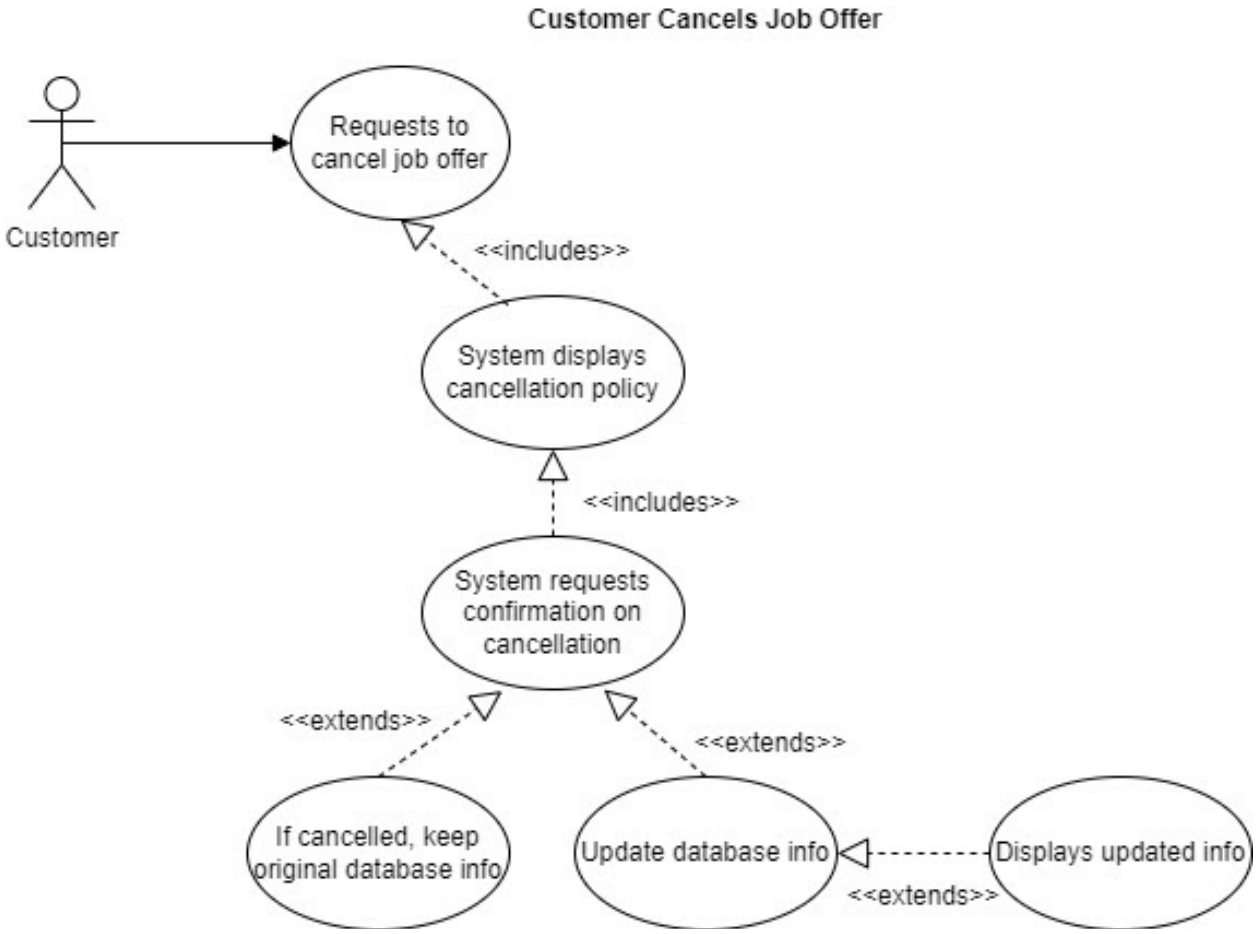
- User wants to create a worker account

Exit conditions:

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

Event Flow:

1. User requests to create worker 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



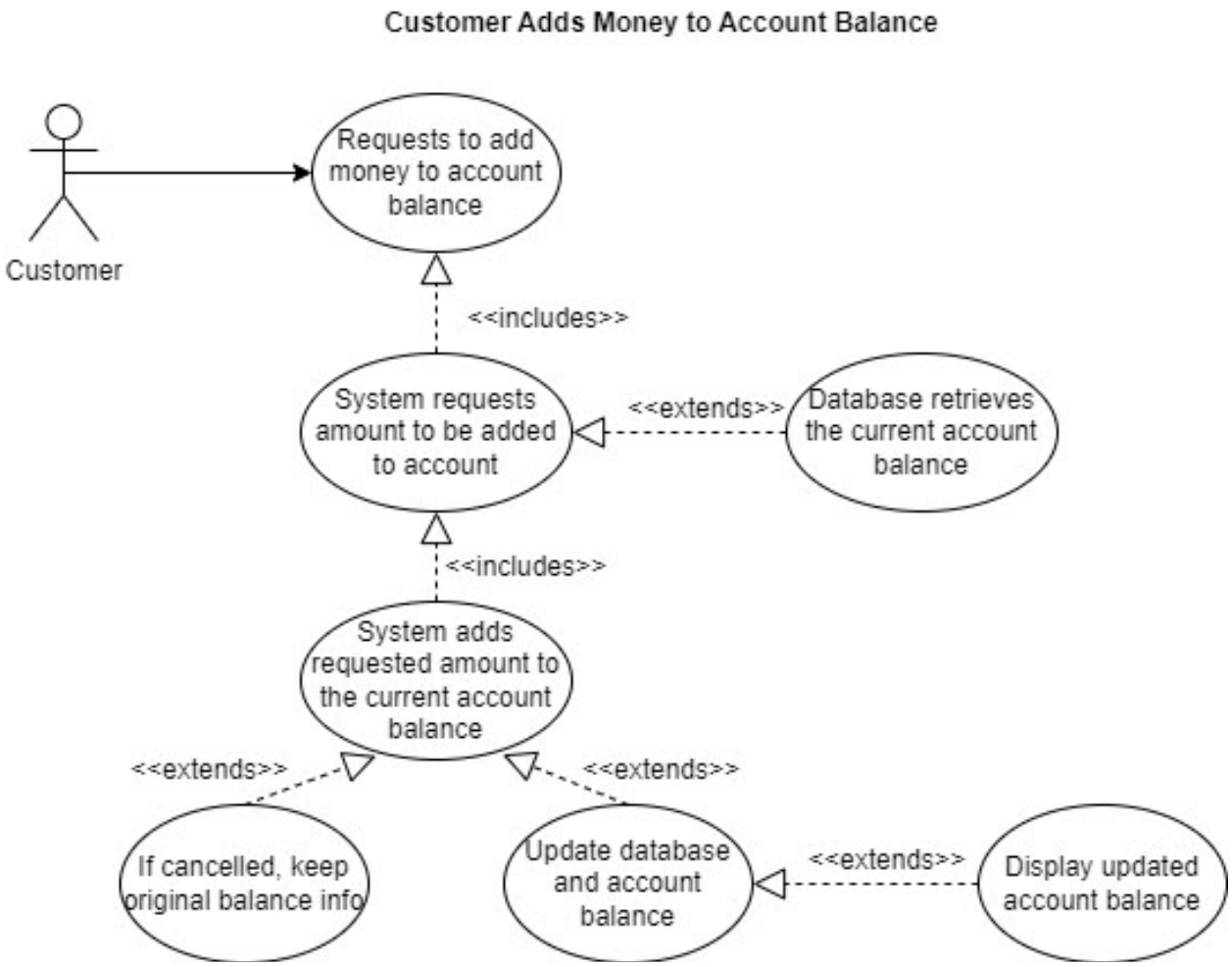
Participating Actor: Customer

Entry Conditions:

- Customer is logged in
- Customer has a job offer available

Exit Conditions:

1) Job offer is cancelled
2) Action is cancelled
3) Event Flow:
4) Customer requests to cancel available job offer
5) System displays the cancellation policy
6) System requests confirmation
    a) If confirmed, job offer is removed
    b) If cancelled, no changes are made



Participating Actor: Customer

Entry Conditions:

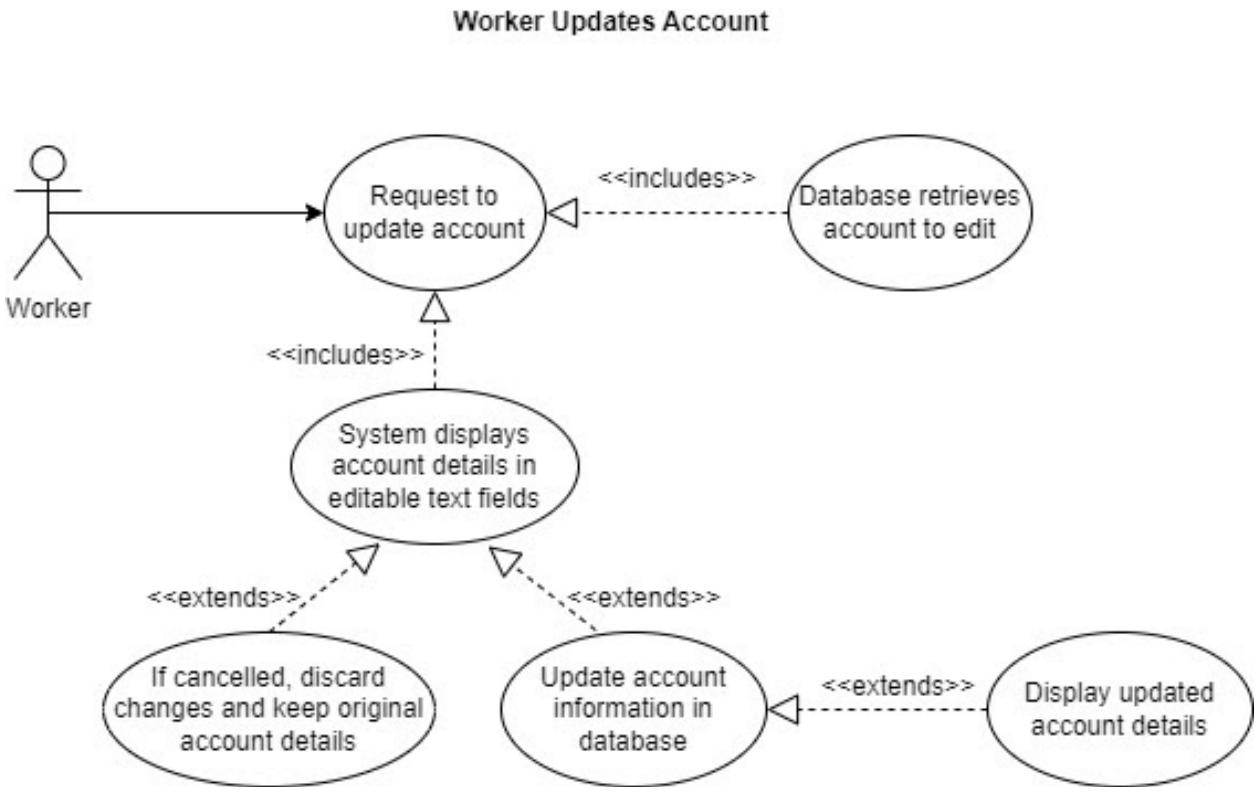
- Customer is logged in

Exit Conditions:

- Money is added to account balance
- Action is cancelled

Event Flow:

1. Customer requests to add money to account balance
2. Customer inputs the amount they wish to add
3. System saves new account balance
    a. If cancelled, no changes are made to balance



Participating Actor: Worker

Entry Conditions:

- Worker is logged in

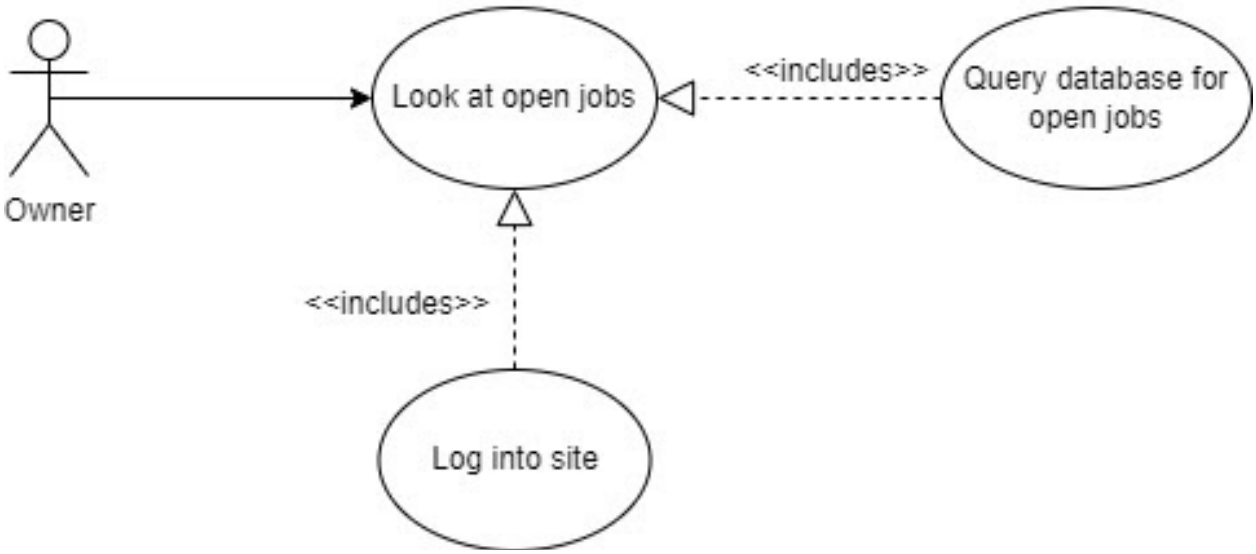
Exit Conditions:

- Account is updated
- Action is cancelled

Event Flow:

1. Worker requests to update their account
2. System displays account details in editable text fields
3. Worker edits the fields
4. System saves account details
    a. If cancelled, no changes are made to the account details

Owner Looks at Open Jobs



Participating Actor: Owner

Entry Conditions:

- Owner is logged in

Exit Conditions:

- Owner views open jobs

Event Flow:

1. Owner logs into system
2. Owner requests to look at open jobs
3. System displays open jobs

environment for local yard care work.

Functional Requirements

1. User Authentication and Access
    1.1. The user must login to the app using an email/username and a password
        1.1.1. If it's the user's first login, they must create an account by using an email/username and password
            1.1.1.1. The app will be able to make sure the account information is secure
        1.1.2. If it's not the user's first login, they will then be given access to the app. If anything is incorrect in the login process, the user will try again
    1.2. Users can have 3 different types of accounts: Owner, Employer, and Worker
        1.2.1. An owner has admin rights on the app, this will not be accessible to the public
        1.2.2. An employer is the homeowner and the one needing the lawn care
            1.2.2.1. On creation, they will be able to select information about their profile such as home location, pictures of their property, etc. (See 4.1)
        1.2.3. A worker is the person who would go do the lawncare for the employer
2. User Profile
    2.1. The user profile will display needed information about the user where they can view and edit the information
    2.2. Login credentials will be viewable to the user where they will be able to edit them if needed
    2.3. Their location will be viewable here to the public, and it will also be editable
    2.4. A simple history page showing their past jobs/hires will be here
    2.5. Their balance will be shown on the user profile page where they can deposit/load money from/into their account
    2.6. A past transactions page can be shown also
3. Owner/admin Features

3.1. Only someone given admin rights will have the option to sign up as an owner
3.2. Owners will have the power to interfere with employer/worker interactions if it is needed
3.2.1. They will be able to review disputes sent to the app
3.2.2. If appropriate, they can reverse transactions that have taken place
3.3. Owners will have the power to ban/suspend accounts if rules are broken

4. Employer features
4.1. Creation of an employer account
4.1.1. Employer must give the location they live at so workers know where to go
4.1.2. Employers can show pictures of their property
4.1.3. Employers will be able to give any extra needed notes
4.2. Creation of a job
4.2.1. When creating a job, the location given on account creation will be the default, but this can be changed
4.2.2. The user will give a description of the job that is needed
4.2.3. A picture of the job is not required, but will be recommended
4.2.4. A payout will be selected here for when the job is finished
4.2.5. Other job specific notes will be given which could include equipment that is provided, tips, etc.
4.2.6. The job will be sent out to the top workers in the area
4.3. After the job
4.3.1. After the job, the transaction will process automatically
4.3.2. If the user is not satisfied with the job, they will be able to dispute the transaction
4.3.2.1. They will show after pictures of the job
4.3.2.2. They will give a description of the issues that occurred
4.3.2.3. An Owner will review the dispute and decide (See 3)
4.3.3. The Employer will then be able to leave a rating for the worker, and the job is terminated

5. Worker features
5.1. Creation of a worker account
5.1.1. The worker will provide their location
5.1.2. The worker will be able to give a basic bio about themselves
5.1.2.1. This includes age, able to drive, equipment they can use, skill, etc.
5.1.3. Their rating will be shown as well as their past jobs
5.1.4. They will also see their balance
5.2. Viewing a job
5.2.1. When a job is created by an employer, it is sent to the top 10 workers
5.2.2. The worker will be notified of the job in the area, and the first worker to accept the job will get it
5.2.3. Before accepting, the worker will be able to view the job information
5.2.4. The worker will have the option to decline the job

5.2.5. If accepted, the worker will be assigned that job, and the worker will be given information about the employer to contact them

5.3. After the job

5.3.1. Once the job is done, the transaction will go through

5.3.2. The worker will be able to leave final comments, and then give a rating for the employer

5.3.3. If the employer is unhappy, they can dispute the transaction (See 4.3)

5.3.4. If everything goes through, the job is then terminated

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 location of the customer/worker represented in the system

   1.3. The system must keep track of jobs that are currently open and who is close geographically

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 downloaded onto an android device

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 Systems interface could be converted to a map where the locations of workers nearby is shown
2. The App could have a chat box that appears for the worker who accepts the job to be able to communicate with the customer through the app
3. The App could show a progress bar to the customer so they can see how close they are to having their job completed

Glossary

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

Customer - a user that uses the system to receive yard work (pays money)

Worker - a user that uses the system to find yard work (receives money)

Owner - a user that receives and manages the transferring of money

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

User - refers to any of the three types of users of the system (Customers, Worker, Owner)