



Odd Jobs - Social Network for Side Hustles

- Logan Ballard (Front-end Wizard)
- Aaron Slade (Database Aficionado)
- Matt Scribner (Authentication Guru)
- Alex Hardy (JavaScript Mastermind)



Goals / Purpose

- Allow customers and workers to connect and post jobs that they need done or accept jobs to complete and earn money.
- Provide a straightforward user experience for both customers and workers (don't make them think)



Major Design Decisions

- Responsive design for both desktop and mobile users using the Bootstrap v5.0 CSS framework and JavaScript plugins
- Easy account creation with ability to add/remove funds
- Different account types (Customer, Worker, Owner, Admin)
- Simple database to store Jobs and User info
- Rating system for Jobs



Scrum Practices

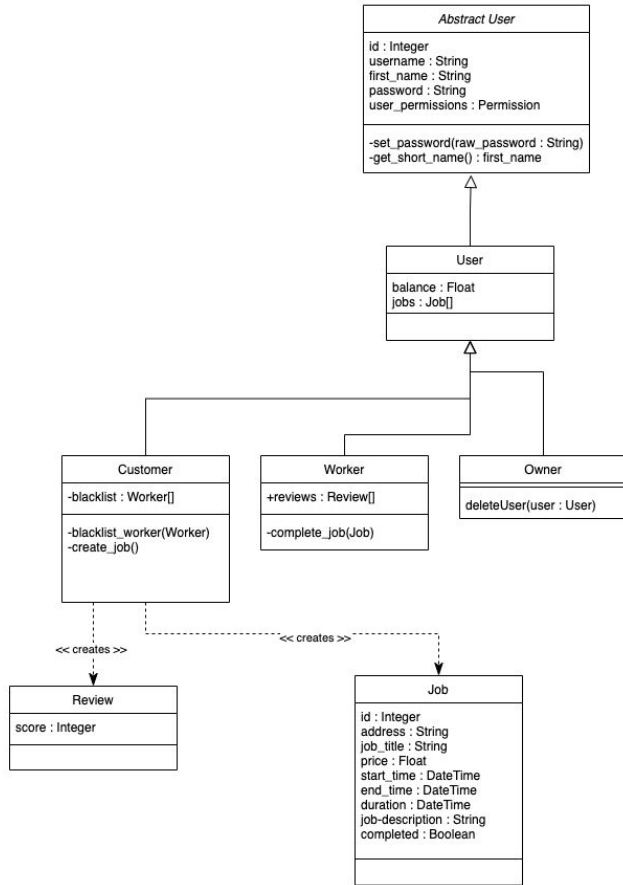
- Sprint planning meetings
 - Compile a list of issues/features to implement
- Standup meetings
 - Review issues completed since last meeting
 - Pick new issues to implement over a few days



Database and Model Classes

- Requirement location in documentation:
<https://github.com/504knight/7-little-ducklings/blob/master/docs/Requirements%20Definition.md>
- Initial Requirements Gathered:
 - i. Database will be used to store information.
 - 1.1. The database will store account info including:
 - 1.1.1. Account type
 - 1.1.2. Username
 - 1.1.3. Password
 - 1.1.4. Balance
 - 1.1.5. First Name
 - 1.2. The database will store job info including:
 - 1.2.1. Job location
 - 1.2.2. Job type
 - 1.2.3. Job payment offer
 - 1.2.4. Job time window

Database and Model Classes



<-Model Classes UML Diagrams

Translation into Prototype:

- Relatively few modifications
- Created Model Prototype Class:
<https://github.com/504knight/7-little-ducklings/blob/master/prototypes/database/test/prototype/dbtest/models.py>
- Initial Prototype Database Schema:
<https://github.com/504knight/7-little-ducklings/blob/master/prototypes/database/database-schema.pdf>
- Prototype was a quick working group of models with unoptimized queries and schema.



Database and Model Classes

Redesign and Modifications During Implementation:

- Review Model was pointless
- Any raw SQL queries were optimized, and non-parameterized queries for initial testing/implementation were replaced with parameterized versions for security.
- Some features required the addition of more methods in the Model classes.
- Example optimization and parameterization of query:

Old:

```
return Job.objects.raw(f"SELECT * FROM OddJobs_jobs WHERE worker_id={self.id} OR customer_id={self.id}")
```

New:

```
type_condition = 'customer_id' if self.type == UserType.CUSTOMER or self.type == UserType.ADMIN else 'worker_id'  
return Job.objects.raw(f"SELECT * FROM jobs WHERE {type_condition} = %s", [self.id])
```





The new optimized query uses one less comparison per record in the WHERE clause.

Database and Model Classes

Updated Schema:

<https://github.com/504knight/7-little-ducklings/blob/master/docs/notes/database-schema.pdf>

Main Related Tasks From Product Sheet:

	Title	Assignees	Story Points	Target Date
7	✓ Class Diagram	 504knight	7	Feb 15, 2022
8	✓ Database Prototype	 jsladea	5	Feb 15, 2022
16	✓ Database	 jsladea	5	Mar 24, 2022
36	🔗 Populate DB Script,	 504knight	2	Mar 24, 2022

Some minor modifications to the model methods occurred outside of these tasks as we worked on implementing various functionalities.

Testing was accomplished by verifying that the model methods used in the views, resulted in the correct output in the final application.



Authentication and Account Creation

Initial Requirements:

<https://github.com/504knight/7-little-ducklings/blob/master/docs/Requirements%20Definition.md>

- i. User Account Creation and Access
 - 1.1. User will be required to log in with an account to access system services.
 - 1.1.1. User will be able to create an account of a specific type with the following information:
 - 1.1.1.1. The account type
 - 1.1.1.2. A username
 - 1.1.1.3. A password
 - 1.1.1.4. A name to be associated with the account (First name is recommended.)
 - 1.1.2. User will be able to login to a previously created account with a username and password.



Authentication and Account Creation

The Django framework has a built-in authentication service that can be implemented to custom User models.

We had to create our own User model that extended the AbstractUser class from Django. Then this model could be used as persistent user data that auto-hashes passwords. Validation goes through Django back-end so that nothing can be leaked or manipulated on the client side.

We decided to use this service for the drastically reduced time it would take to make an authentication system, for it's customizability, and it's security.



Authentication and Account Creation

Milestone 2:

- Authentication Prototype (Matt)

Milestone 3:

- Authentication (actual) (Matt)
- Django Core (Matt)
- Registration Page Template (Matt)
- Database (Aaron)

Milestone 4:

- Registration Form Error Handling (Matt)

Unit tests:

- Automated tests to ensure an account can be created and verified.
- `python manage.py test`

Account Types and Their Specific Processes

Initial Requirements:

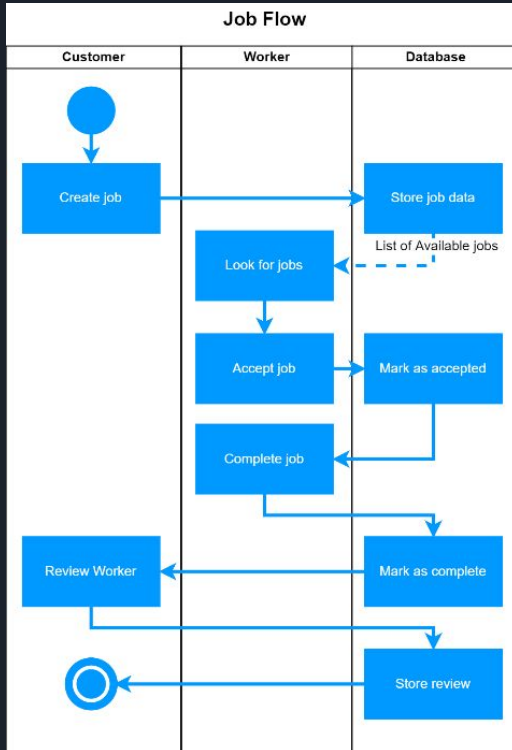
<https://github.com/504knight/7-little-duckings/blob/master/docs/Requirements%20Definition.md>

Summary:

- There MUST be the 3 different account types: Customer, Worker, and Owner
- They each MUST have access to account type-specific features

- ii. User Account Types
 - 2.1. There will be a customer account type with full access to customer-specific features.
 - 2.1.1. User with customer account (hereafter referred to as "Customer") will be able create a job.
 - 2.1.1.1. Customer will have to select a job type from a list of job types.
 - 2.1.1.1.1. Customer will be able to fill in a custom job type upon choosing the "other" option.
 - 2.1.1.2. Customer will have to input the location at which the job will take place.
 - 2.1.1.3. Customer will have to input their preferred time of day and day of the week for the work to take place.
 - 2.1.1.4. Customer will have to input how much they will pay for the completion of the job.
 - 2.1.2. After a Worker accepts an offered job, Customer will have to confirm and approve job acceptance.
 - 2.1.2.1. Customer should receive a reminder to tip their worker after completion of the job if they are satisfied.
 - 2.1.3. Customer will be able see jobs they have listed.
 - 2.1.3.1. Customer should have the ability to delete an unconfirmed job.
 - 2.1.3.2. Customer could be able to edit an unaccepted job.
 - 2.1.4. Customer should have the ability to review a Worker after the completion of a job.
 - 2.1.5. Customer should be able to blacklist a worker so that Worker cannot see their jobs anymore.
 - 2.2. There will be a worker account type with full access to worker-specific features.
 - 2.2.1. User with worker account ("Worker") will be able to view a list of available jobs.
 - 2.2.1.1. Worker should be able to filter jobs by distance based on an inputted zip code.
 - 2.2.1.2. Worker should have the ability to filter jobs by job type.
 - 2.2.2. Worker will be able to accept a listed job.
 - 2.2.2.1. Worker will be able to see the exact job location only after job acceptance is confirmed by Customer to ensure privacy.
 - 2.2.3. Worker should be able to see their rating.
 - 2.2.4. Worker should have the ability to blacklist a Customer so they cannot see that Customer's jobs.
 - 2.3. There will be an owner account type with full access to owner-specific features.
 - 2.3.1. Owner will be able to see a list of all requested, accepted, and confirmed jobs.
 - 2.3.1.1. Owner could be able to see all completed jobs.
 - 2.3.2. Owner will be able to see a list of all users.
 - 2.3.3. A portion of every transaction will be deposited into the Owner's account balance.
 - 2.3.3.1. The owner could be able to change the size of the portion.

Account Types and Their Specific Processes



<- Job Flow Activity Diagram

Design Choices:

- User Centered
- Both Customers and Workers have control
- Simple Processes



Account Types and Their Specific Processes

Milestone 1:

- Use Case Diagrams (Aaron)
- Requirements Definitions (Alex/Everyone)

Milestone 2:

- Wireframe (Alex)
- Class Diagram (Logan)
- Activity Diagrams (Alex)

Milestone 3:

- Rating System (Aaron)
- Account Specific Navigation (Logan)
- Job Finding/Acceptance Page (Alex)
- Current Jobs(Worker) Page (Alex)
- Job Creation Page (Logan)
- Current Jobs(Customer) Page (Alex)
- Account Specific Personalized Messages (Logan)

Milestone 4:

- Job Cancellation (Alex)
- Error Handling (Everyone)
- Unit Testing (Matt)



Account Types and Their Processes

Testing:

- Visual testing of specific functionality
- Visual testing of overall processes

Dependencies:

- Database
- Authentication



Resources

- Django Docs: <https://docs.djangoproject.com/en/4.0/>
- Django Tutorial: <https://docs.djangoproject.com/en/4.0/intro/tutorial01/>
- User Authentication in Django:
<https://docs.djangoproject.com/en/4.0/topics/auth/>
- Bootstrap v5.1 Docs:
<https://getbootstrap.com/docs/5.1/getting-started/introduction/>
- Bootstrap 5 Crash Course Tutorial #19 - Customizing Bootstrap:
https://www.youtube.com/watch?v=nCX3QVI_PiI