Design Document for Team 7

Recyclr

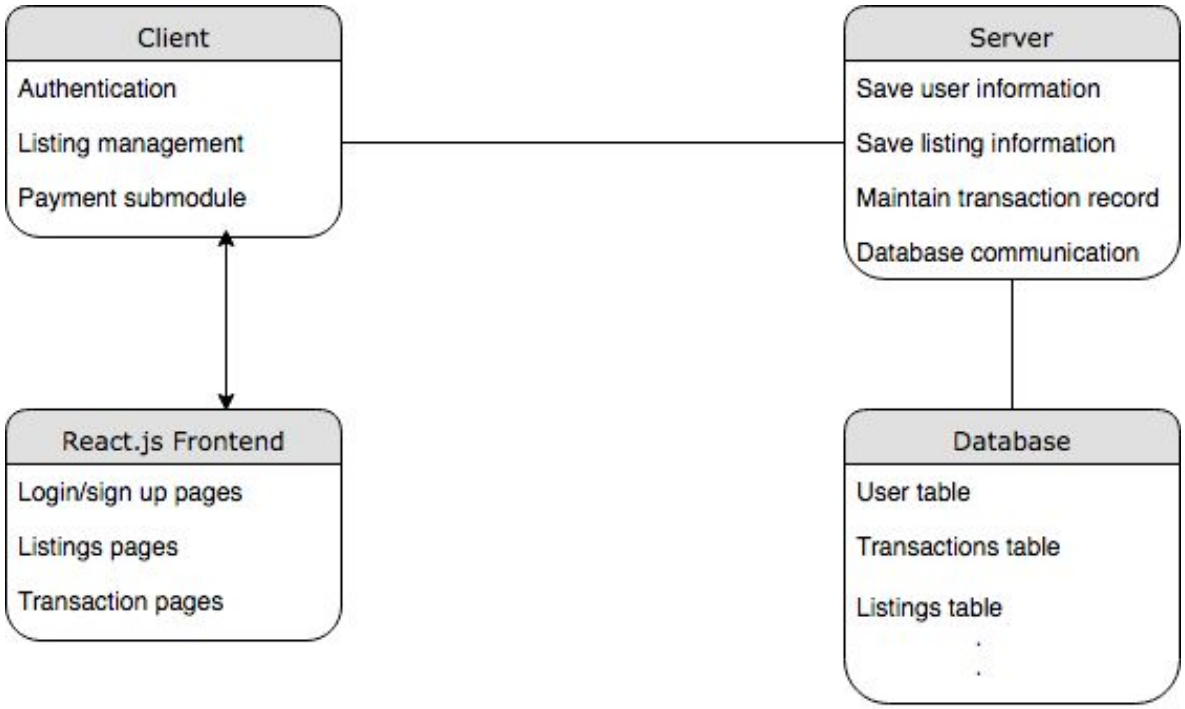
Cory Laker (claker@purdue.edu), Geoffrey Myers myers259@purdue.edu),
Pranav Vasudha (pvasudha@purdue.edu), Ryan Walden (waldenr@purdue.edu),
Vedant Nevetia (vnevatia@purdue.edu), Zachary Rich (richz@purdue.edu)

Purpose

Recyclr is a web application that has both a recycling user side for people who want to recycle and an organization side for companies that are willing to pick up recycling. Recycling is becoming increasingly more important in today's world. There are a lot of 'dead spots' where recycling isn't an option for many people. Recyclr aims to create a community of recyclers that are excited to be able to improve our recycling system as a whole. As previously explained, people who have a lot of products to recycle are encouraged to hold on to it until they have a substantial amount of waste, which they can then put up as a listing on Recyclr, and interested recycling companies can coordinate a time to come pick it up.

Design Outline

We are adopting a client-server model, utilizing Golang as the language of choice for the backend, and React.js as the framework of choice for designing our frontend. Our system will theoretically consist of 3 main components: the backend, frontend and the database system.



The backend will expose a REST API that can be used by the frontend to gather data as needed. If time permits, we will also utilize GraphQL for API calls. The backend system will be the intermediary in communicating with the database and sending the results back to the frontend.

The frontend will include many components, such as:

- Authentication
    - This component will handle the user's account information, as well as handling sign in/sign out/ sign up authentication.

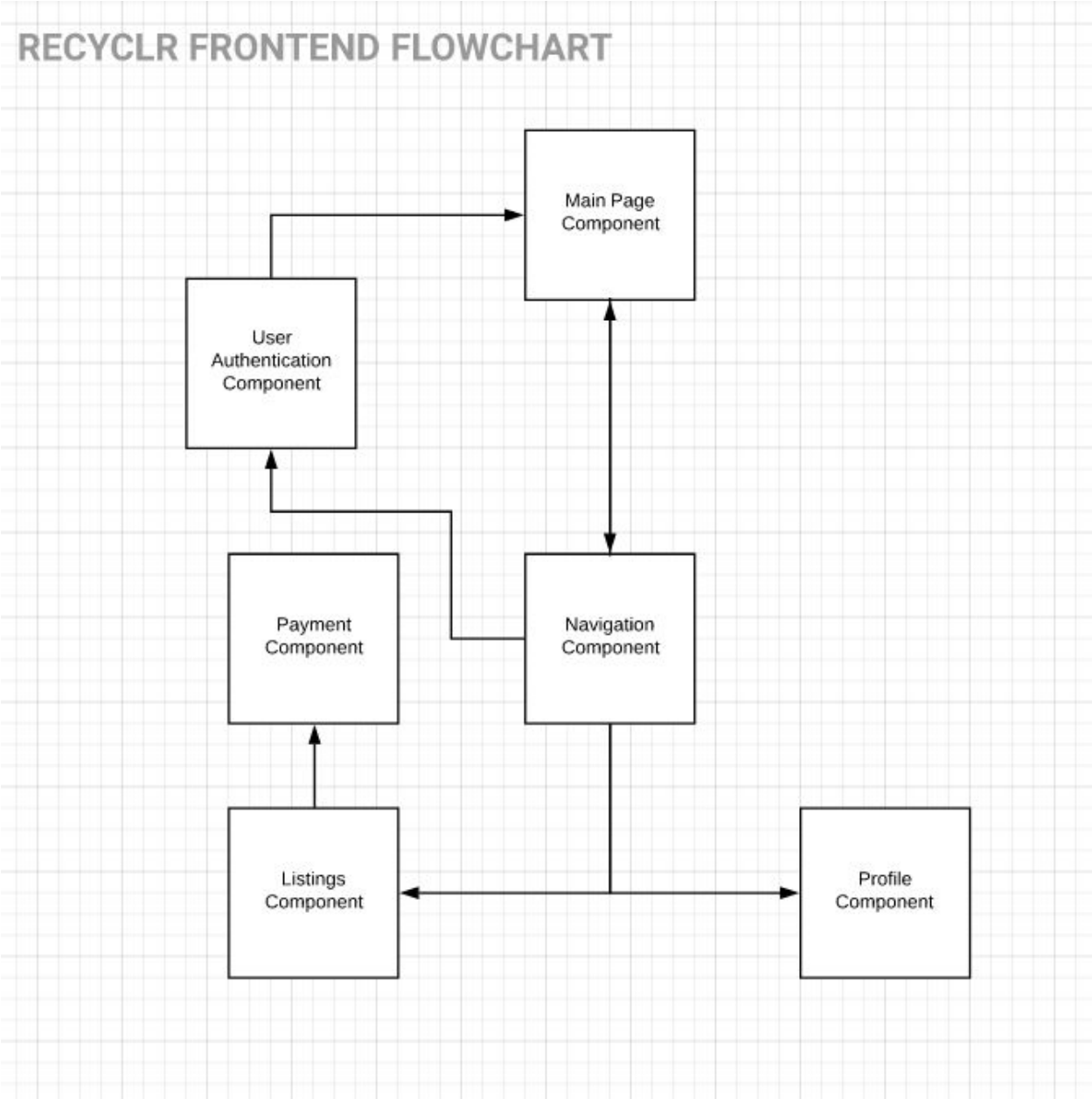
- Main/home page
    - This component will consist of the main view of the site. It will be what the user sees when starting the app, as well as be the main component to offer information to the user.

- Profile
    - This component will show the user their personal information, as well as allow the user the ability to modify some of the information.

- Navigation
    - Will be the main way the user navigates throughout the app in a manner that is convenient and easy to use.

- Listings
    - This will be the component that provides the primary functionality of the app. It will allow for the creation of listings, a viewing of available listings, and an interface for the fulfillment of listings.

- Payment
    - This component will allow users to pay companies to come pick up their recyclables via a third party API.



The navigation component will always be visible to the user. It will allow the user access to the profile, listings, main page, and authentication. The authentication module will authenticate the user and allow the user access to the profile component. The listing component will, depending on the users' actions, will allow the user to access the payment module to complete payments.

A user can either be a recycler or an organization, the flow chart will remain the same for either type of user. The content they see or can access on each component may vary however.

Design Issues

Functional Issues

- **Process management** - User must have sufficient confirmation that the recycling process facilitated by the app is correctly flowing.
  - Some solutions to this include:
    - Only sending notifications only when major events happen, such as pickup or listing.
    - Sending notifications at every step of the process.
  - This can be solved by having notifications sent every time a client successfully sends or receives a request, this way if a user does not receive an expected notification, they will know an error has occurred and can take appropriate action.

- **User Type Definitions** - Difference between a recycler and an organization not being clearly defined on the start screen leading to confusion for users newly signing up.
  - Solutions to this problem include:
    - Having text on the signup screen explaining the differences in the account types.
    - Having both user types use the same authentication flow to use the application.
    - Splitting the authentication to have separate authentication processes for both account types.
  - As a solution, having separate buttons and OAuth flows for both types of users would be helpful.

- **Scheduling** - Making scheduling times easy for all users, recyclers and companies, to use and understand is essential for the app to be user friendly.
  - Solutions to this problem include:
    - Having a notification system and a clearly defined calendar.
    - Have users be able to view their scheduled pickups in a list.
    - Have users view a transaction history to view when their pickup is.

        - By having a notification system and a clearly defined calendar for users to be able access after scheduling is imperative.

    - **Rating system** - We need to actively have a system to maintain users who uphold their end of the recycling and show up on time to keep the app running and be able to ban those who abuse the system.
        - Solutions to this problem include:
            - Have users be able to block users who abuse the system
            - Have a rating system for both users and companies
            - Have users send messages to the admin about users that do not show up on time or are abusive/malicious.
        - By designing a rating system for both recyclers and organization we can ensure that users are upholding their end of their responsibilities as users so that we can keep our community strong.

- # Non-Functional Issues
    - **Backend scalability** - Being able to scale the backend to handle many users at once and many concurrent operations is an issue that is faced by almost any modern application.
        - Solutions to this product include:
            - Having the API create a new thread for each user that is using the application.
            - Use load management for the API.
            - Only allow a certain number of users to make requests at a certain time and make any other users wait.
        - As a solution, we will create an API that can easily handle many incoming requests and use proper load management on the server so that it will not crash, along with performing stress tests before deployment.

    - **Secure payment options** - Being able to ensure your payments are secure online is extremely important.
        - Solution to this problem include:
            - Using an existing API.
            - Hashing the transaction information and processing it server-side.
            - Relying on HTTPS to keep payment information secure.
        - We will use a third party payment system that is already well established to ensure that our users will have adequate security for any payments made on our site.

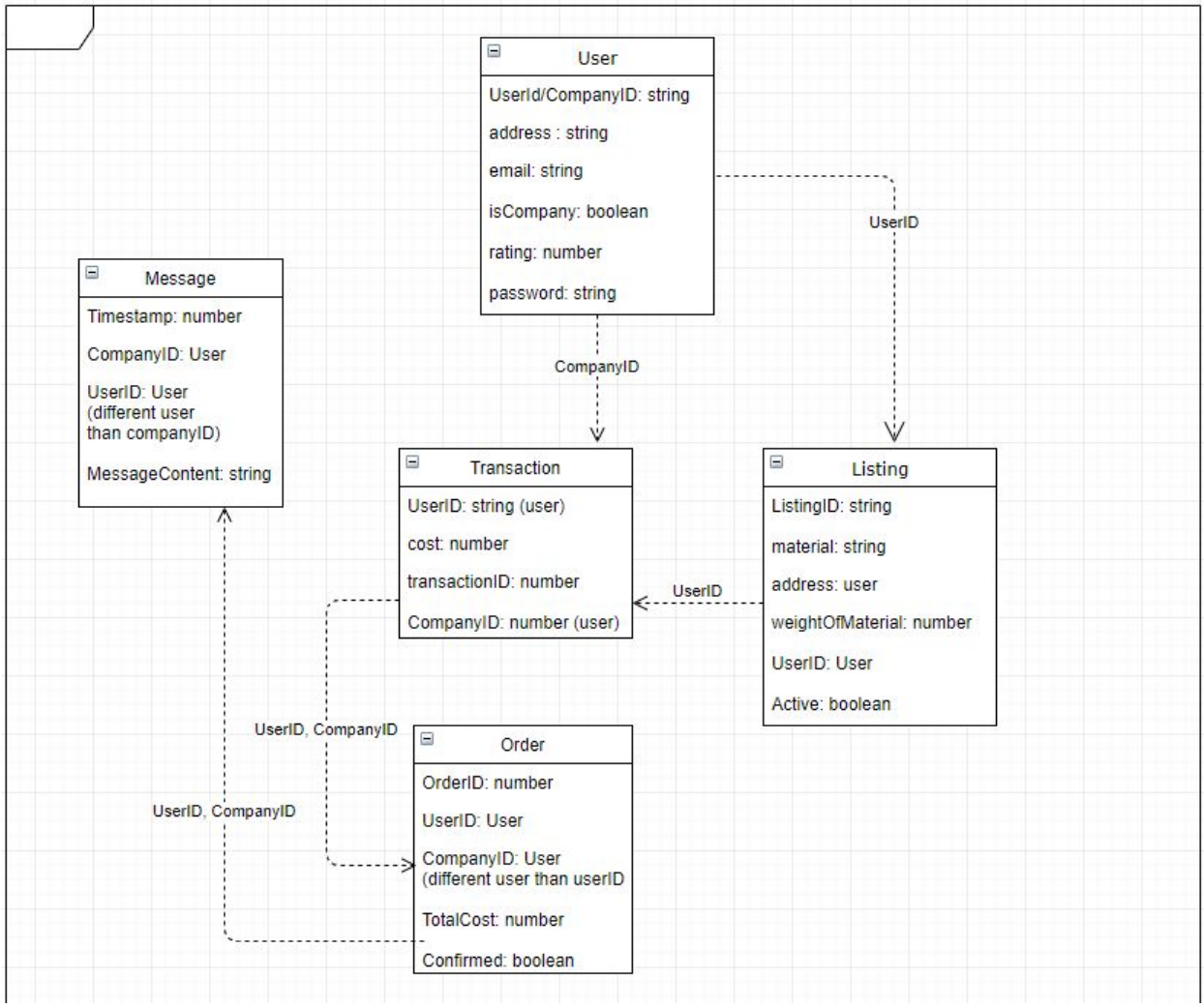
    - **Account security** - Keeping user accounts secure is necessary for any application.
        - Solutions to this problem include:
            - Store the full text in the database, but take extra care that the database is only accessible when credentials are correct.
            - Salt and hash all of the sensitive data and store that in the database.
        - We will make sure that user's passwords are both salted and hashed, along with any other sensitive information that will be stored in our database.

- **Cross site scripting** - Being able to run malicious scripts makes a web application insecure and open to stealing user information and other malicious intentions.
    - Solutions to this problem include:
        - Implement a great number of tests that specifically target cross site scripting.
        - Parse information in the backend, and if there is any malformed input, throw out the request.
        - Ensure user input is sanitized before it reaches the API.
    - We will run tests to ensure that cross site scripting does not work on our application, along with sanitizing user input.

- **Database Scalability** - Having a database that does not slow down or become too cluttered is essential for quick data retrieval and efficient programming.
    - Solutions to this problem include:
        - Picking between different database solutions.
        - Splitting data into separate tables as required to optimize query times.
    - We will use PostgreSQL with an ORM to ensure that data models are simple and concise, along with being easy to use and create, read, update, and delete data.

Design Details

Class Diagram



Class Structure

- **User** - All users will be represented by this data structure. Contains all information needed to login. It also contains the profile info that will be created by the users.
    - UserID

    - Email
    - Address
    - Password
    - Name
    - Rating

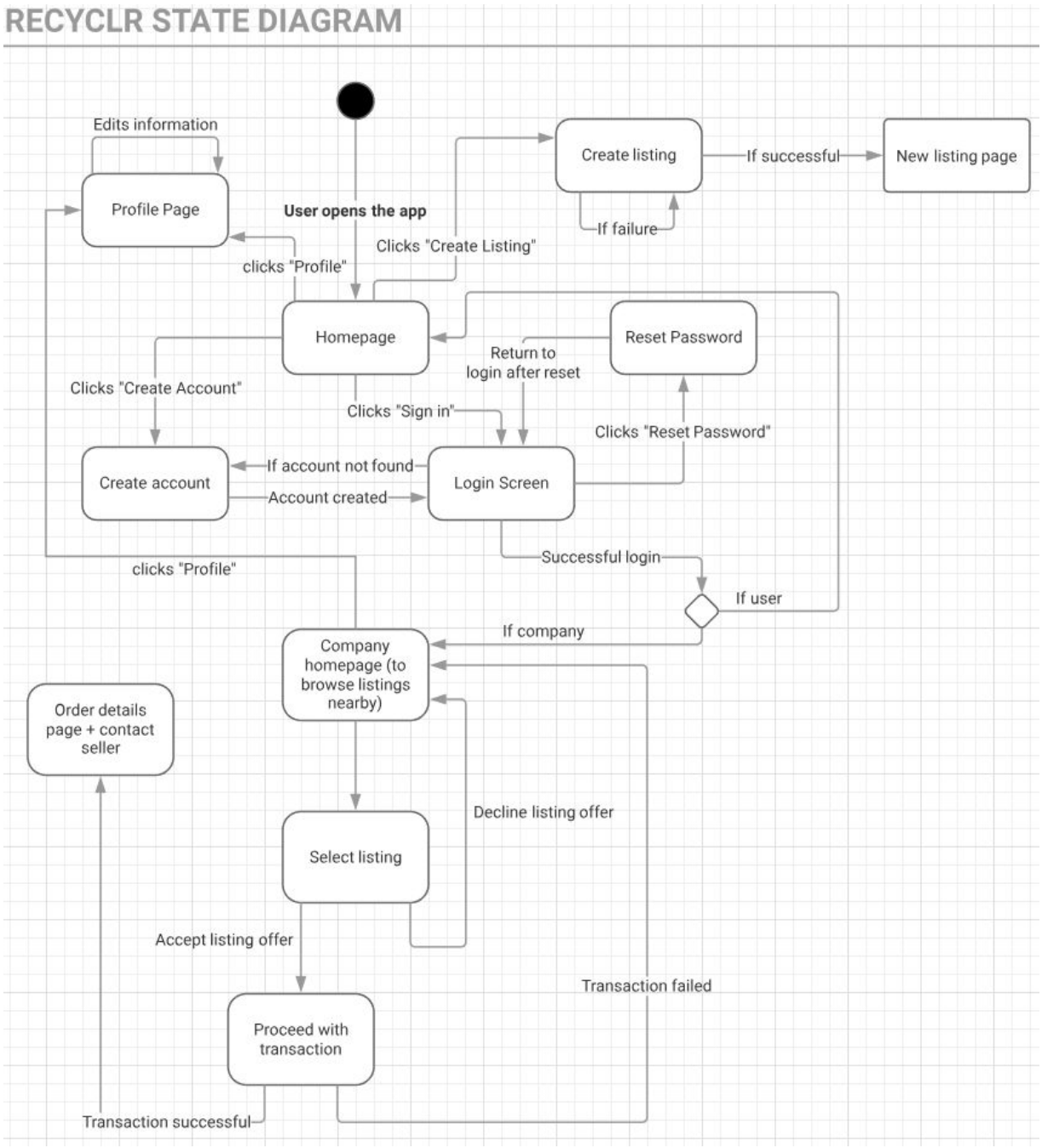
- **Company** - Stores basic information about companies in which the upgrade Recyclr is active.
    - Location
    - Name
    - Rating
    - Password
    - Email
    - CompanyID

- **Listing** - Listings of recycling items are going to be using this information
    - ListingID
    - Weight
    - Price
    - UserID
    - Active/Inactive

- **Transaction** - Every transaction that takes place between a user and a company
    - Cost
    - UserID
    - CompanyID
    - TransactionID

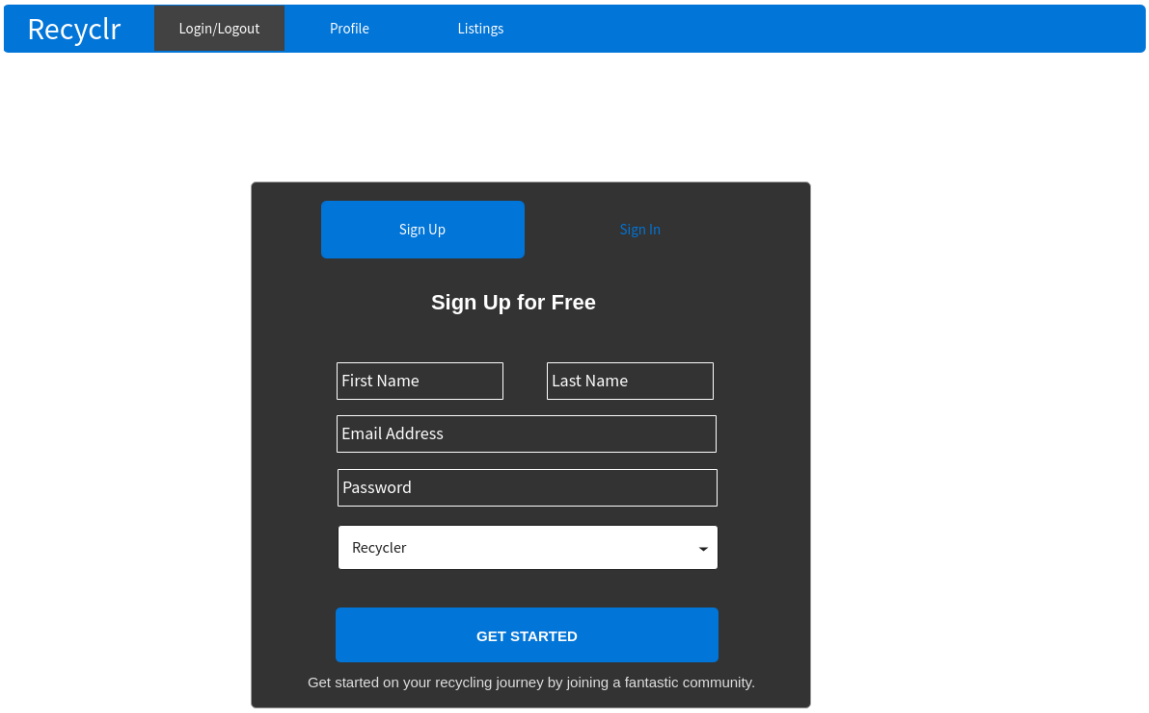
- **Message** - Contains information pertaining to messages exchanged between companies and users post-order confirmation
    - Timestamp
    - CompanyID
    - UserID
    - Message Content

- **Order** - Contains information about a particular order
    - OrderID
    - UserID
    - CompanyID
    - TotalCost
    - Confirmed/Unconfirmed



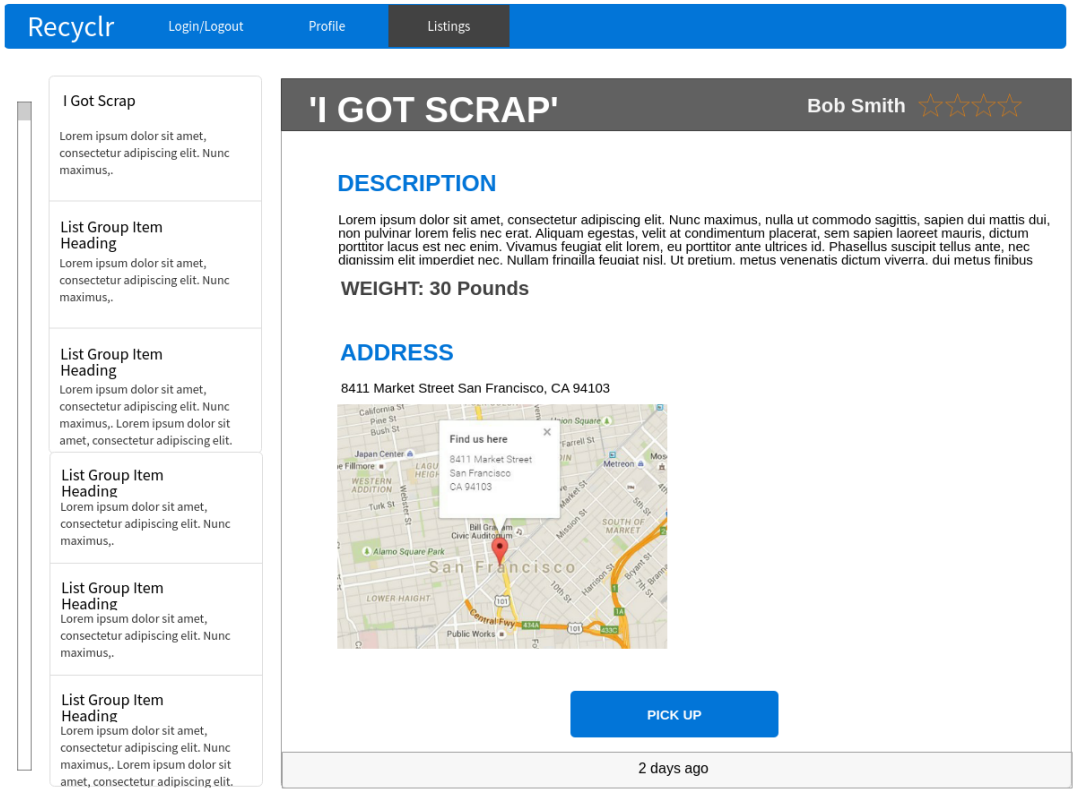
UI Mockups

Sign In Component



The sign in and sign up functionality is encapsulated in this component. The sign in flow accounts for a user being able to sign up as a Recycler or a Organization through the dropdown field below the password. The sign up is depicted in the picture above, and the sign in is nearly identical without the First Name, Last Name, and Dropdown fields.

Listings Component



This diagram is a potential mockup of the listings component from an Organization's perspective. The scroll bar on the left most side controls the list view which has listings from users who are looking to get their recycling material picked up. Clicking on one of the listings gives the Organization a more detailed view of the listing with a detailed description, weight, the pickup address and an option to opt to pick the recycling material up. If the user clicks the pick up button, then the Organization is directed to a scheduling page where they can coordinate times with the Recycler.