Marketplace Project - Online Garage Sale Web Application

by Zumin Yu

Website: https://online-garage-sale-app.herokuapp.com/

GitHub Repo: https://github.com/Yumi2121/CA-T2A2-Marketplace-Project

Table of Contents

Marketplace Project - Online Garage Sale Web Application
by Zumin Yu
Table of Contents
Application Purpose
Target audience
User stories
Functionality and Features
Screenshots
Sitemap
Wireframes
Application components and their associations
ERD
Database
Tech stack
Third party services
Project plan and Task Management

Application Purpose

Garage sale, always been a very popular event in Australia with a long history. However, it is always comes with few problems:

- Hard to know where garage sales events near me
- Hard to know the garage sale time
- Hard to know if the garage sale got the items I want
- Not practical if the location is too far from me
- For a garage sale host, the unpredictable weather is a risk

The above problem needs to be solved because garage sale not only brings us a sustainable environment, but also brings the communities together and fostered neighborhood relationships. On top of that, selling an unwanted item and getting back extra money are always a fun deal.

This online garage sale application was inspired by the traditional garage sale, and aiming to help on the problem was mentioned above.

Target audience

The target audience for this application are those who want to sell their unwanted items, and those who want to buy second-hand items with reasonable prices, especially for those people who support sustainable lifestyles.

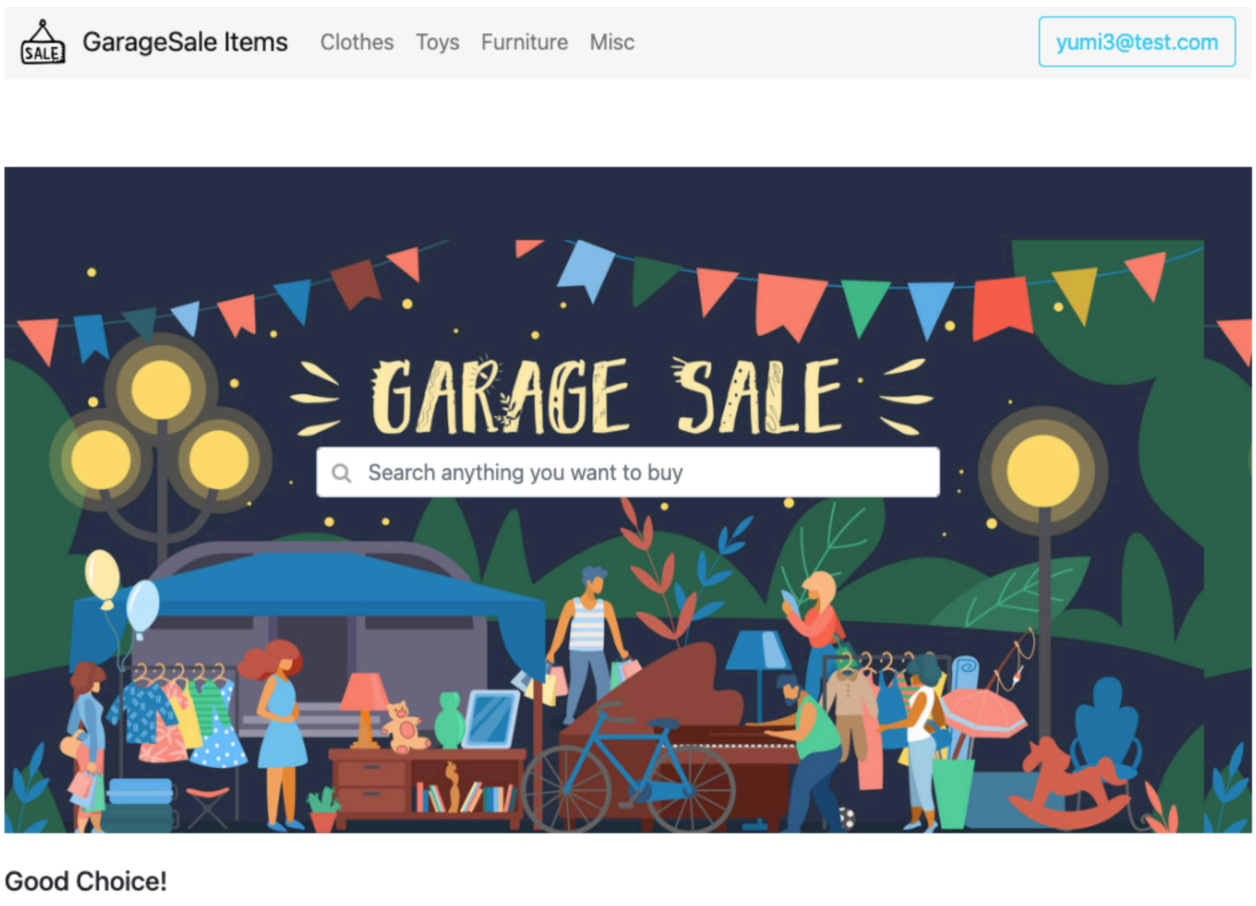
User stories

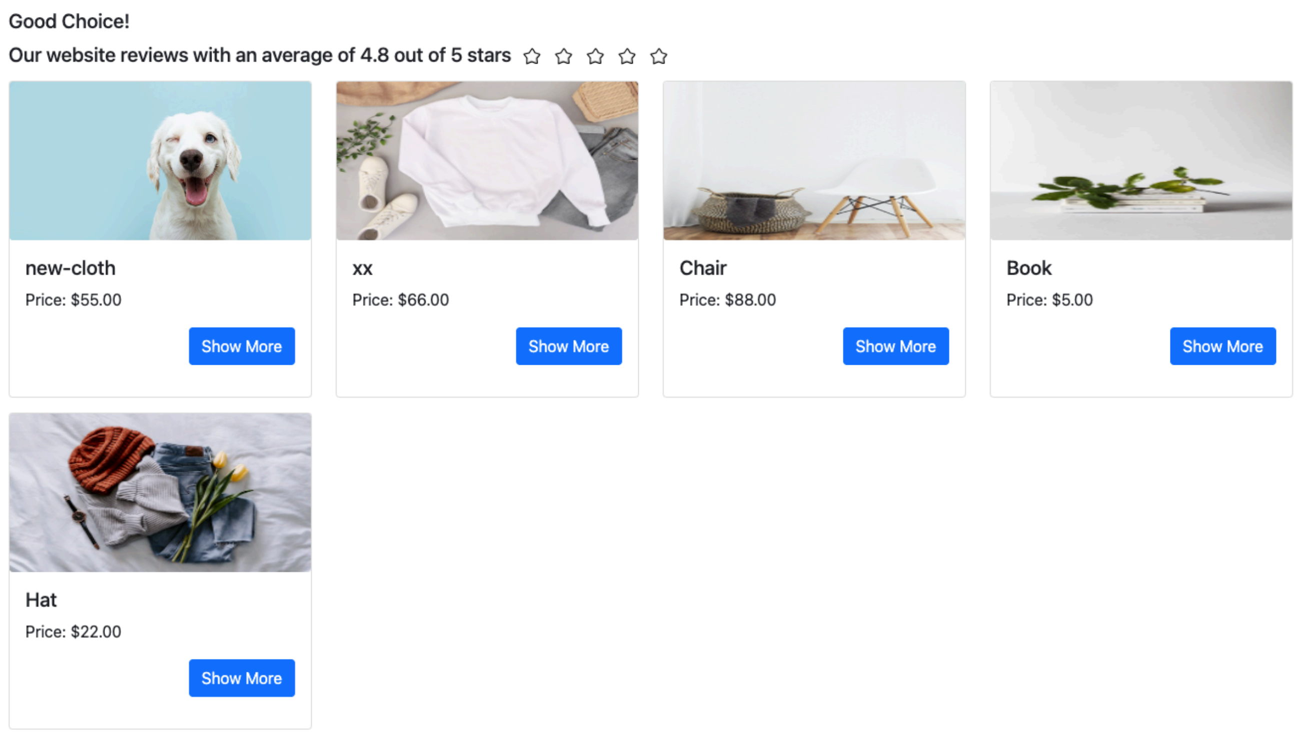
- As a new app user, I want to be able to sign up, so that I can buy and sell items.
- As an existing app user, I want to be able to sign in, so that I can buy and sell items.
- As a signed in user, I want to be able to sign out, so I feel safe when I close the website.
- As a buyer Amy, I want to be able to find them by category, so I can easily find the item I want to buy.
- As a buyer, Bobbi, I want to be able to search for the item I want to buy, so that I don't need to look through all the items which I think is wasting my time.
- As a seller of Cici, I want to be able to sell my items in this app, so that I can get extra money.
- As a buyer of Didi, I want to be able to see more details of the items I want to buy, for example a bigger image of the item and the location of the seller, so that I can consider all the situations to help me make a decision of purchase.
- As a buyer Fang, I want to be able to make payments online by using a credit card, so that I don't need to use cash.
- As a user Lucy, I want to have a separate page of my profile, so that I can view my profile details and update them if necessary.
- As a user of Zizi, I am a seller and a buyer. I want to be able to view my selling items, so I have better control of it.
- As a user of Zizi, I am a seller and a buyer. I want to be able to view my bought items, so I have better control of it.
- As a user of Zizi, I am a seller and a buyer. I want to be able to view my sold items, so I have better control of it.
- As a buyer Elle, I want to be able to view the photo of the item I am browsing.
- As a seller Qiqi, I want to be able to upload photo of item I am selllng.
- As a seller Mary.A, I want to be informed by email that my item has been sold, so that I have better track of my selling.
- As a buyer Mary.B, I want to receive an email to confirm my purchase, so that I can save as a record.

Functionality and Features

- Sign in, sign up & sign out
- Filter sale items by catagory
- Search items by keyword
- Selling item (become a seller)
- View item details
- Puchase item online by using credit card
- View sellers location
- My profile page, used to view, update any user infomation
- View the item i am selling (as a seller)
- View all bought item (as a buyer)
- View all sold items (as a seller)
- Email notification to buyer and seller after item has been purchased

Screenshots





Good Choice!

Our website reviews with an average of 4.8 out of 5 stars ☆ ☆ ☆ ☆ ☆

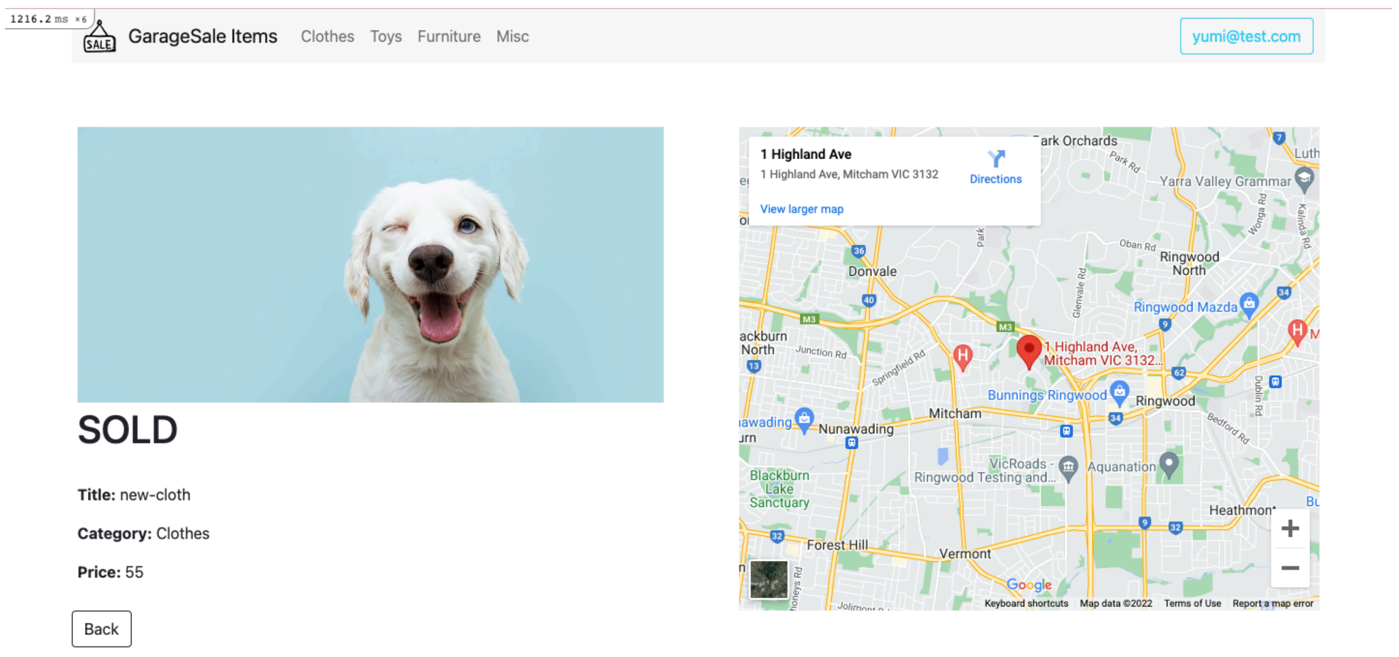
new-cloth
Price: $55.00
Show More

xx
Price: $66.00
Show More

Chair
Price: $88.00
Show More

Book
Price: $5.00
Show More

Hat
Price: $22.00
Show More



1216.2 ms ×6

GarageSale Items Clothes Toys Furniture Misc yumi@test.com

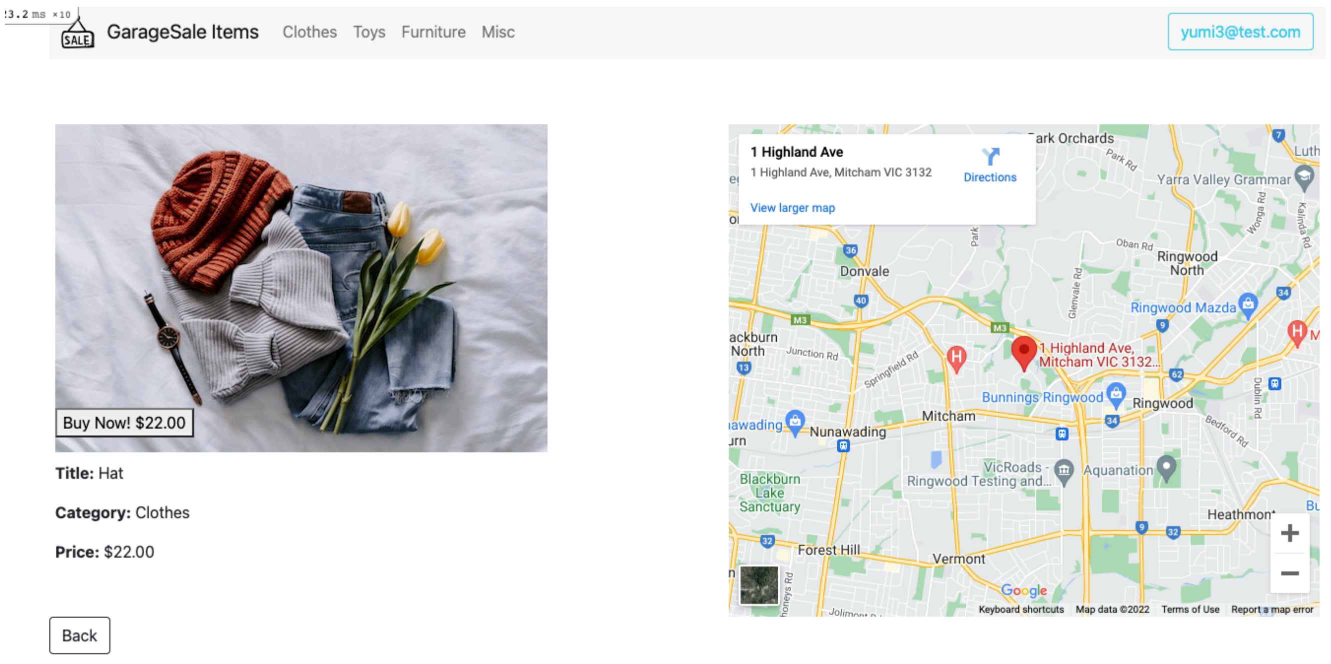
SOLD

Title: new-cloth

Category: Clothes

Price: 55

Back

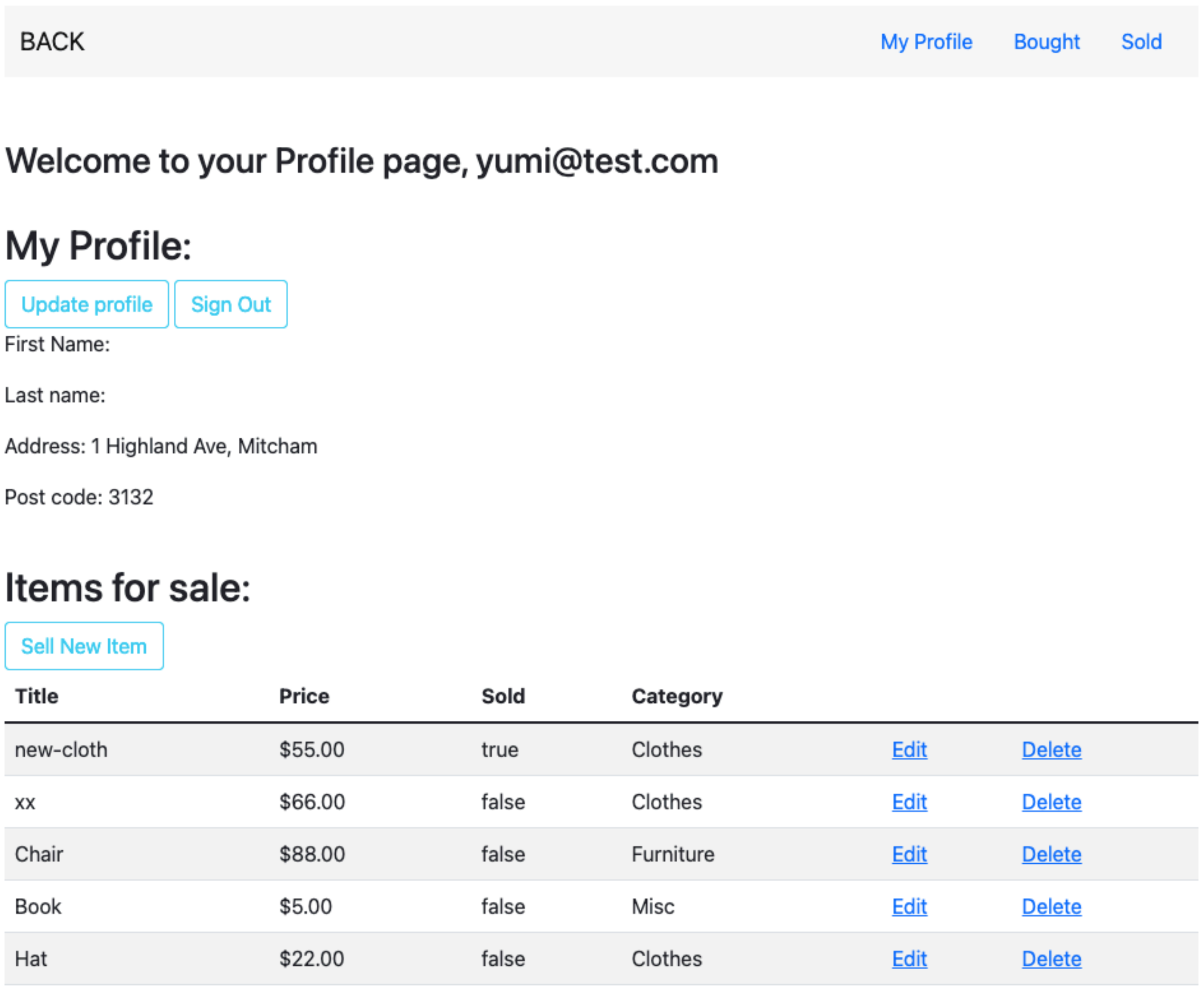


Title: Hat

Category: Clothes

Price: $22.00

Back



Welcome to your Profile page, yumi@test.com

My Profile:

First Name:

Last name:

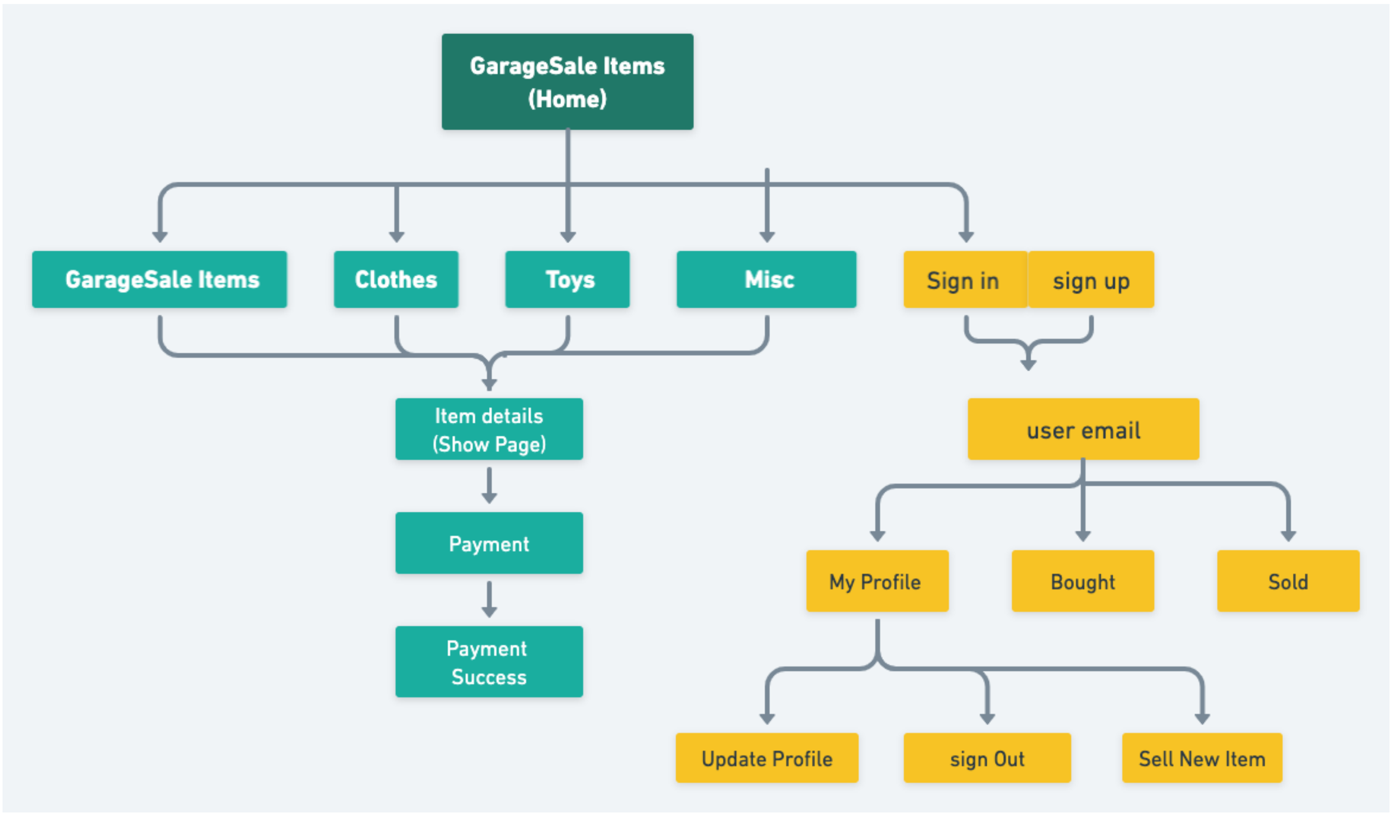
Address: 1 Highland Ave, Mitcham

Post code: 3132

Items for sale:

Title	Price	Sold	Category		
new-cloth	$55.00	true	Clothes	Edit	Delete
xx	$66.00	false	Clothes	Edit	Delete
Chair	$88.00	false	Furniture	Edit	Delete
Book	$5.00	false	Misc	Edit	Delete
Hat	$22.00	false	Clothes	Edit	Delete

Sitemap



Wireframes

Please find the wireframe from docs/Garage sale Wireframe@2x.png

Application components and their associations

As a typical Rails application, this online garage sale app uses Model View Controller (MVC) architecture pattern.

- **Model**

  Models represent the data in the application. When a database is being used, each model will represent each table in the database. It also performs data validation to make sure the integrity of the data before they are saved into the database.

  The following models are used in garage sale applications:

  - `Item`, which inherits from ApplicationRecord. `Item` model has_many `orders` and belongs_to `User`.
  - `Order`, which inherits from ApplicationRecord. `Order` model belongs_to `item` and `user` (buyer and seller).
  - `User`, which inherits from ApplicationRecord. `User` model has_many `items`, also has_many `sold_orders` and `bought_orders`.
  - ApplicationRecord inherit from `ActiveRecord::Base`, which defines some helpful method that will apply to all models belonging to ApplicationRecord. in this app, define

    `self.abstract_class= true` .

- **Controller**

  It acted as an interface between the view and the model. It also consists of actions, which map to specific routes. The following controller are used in garage sale applications:

  - `Items` Controller, which inherits from ApplicationController. It defines methods such as index, show, new, edit, create, update and destroy. Each of the method has its corresponding view page and mapped to specific routes in `routes.rb`.
  - `Orders` Controller, which inherits from ApplicationController. It defines `bought` and `sold` action.
  - `Payment` Controller, which inherits from ApplicationController, method `success` was defined in this controller and linked to the success page after the payment.
  - `User` Controller, which inherits from ApplicationController. `Show` action was defined in this controller to link to my profile page.
  - Only ApplicationController inherits from `ActionController::Base`, all other controllers inherit from ApplicationController. This offers special configure requests, for example request forgery protection or give permit to specific parameters in this case.

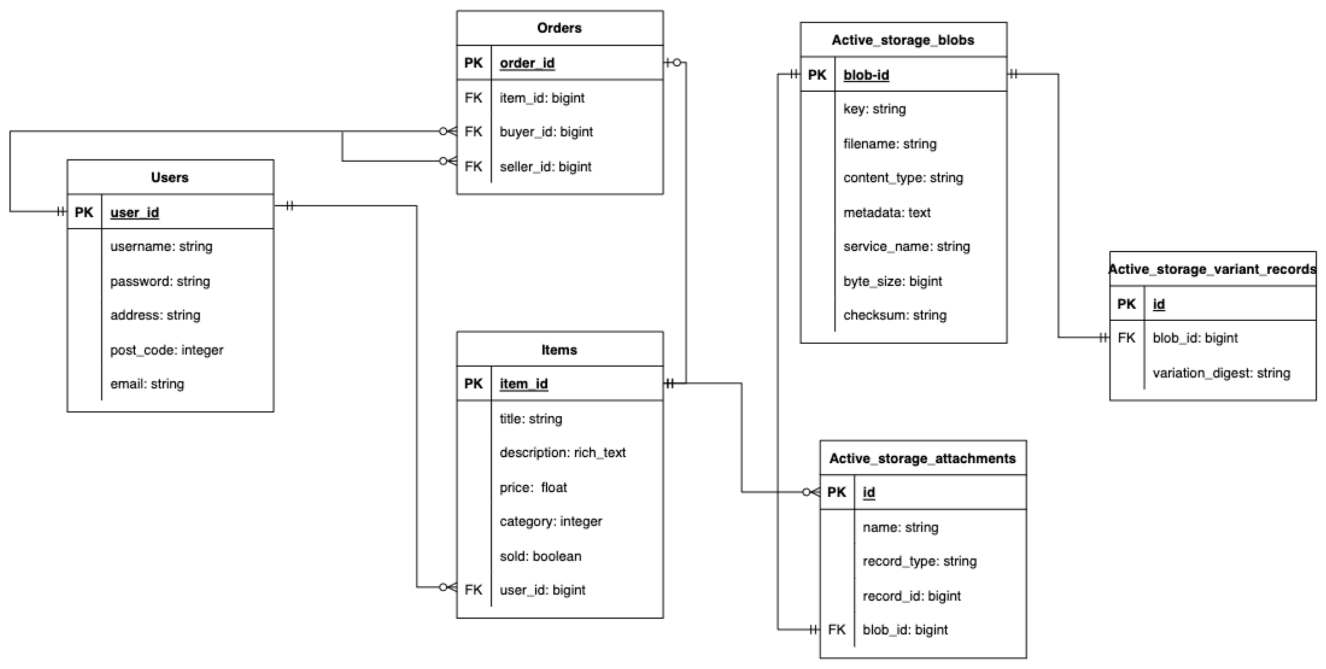
- **View**

  It represents the user interface of an application through the HTML file that is embedded in Ruby code in the form of `html.erb` file. View serves two purposes, first it provides controls which allow users to provide the inputs to the application. Second, to display the output of an operation.

  In this app, each controller method has their own corresponding view such as the `user` related view in the 'devise' folder,and it includes sign in, sign up, password change view etc. `items` related view are in 'items' folder named index, new, show, update etc. `Orders` folder includes 'bought' and 'sold' list page. And the success page in the 'payment' folder.

ERD

You can also find ERD in docs/Garage-sale-app.drawio.png



Database

A relational database model represents the database as a collection of tables, all the relations are saved as table format. And foreign keys are structured into a relational database to link together two or more tables.

This application implimented relational database model and includes 3 tables:

- **Users table**

  User has `user_id` as primary key, and it `has_many` orders. Each user can have 0 to many orders, this includes buying and selling order. User also `has_many` Items, each user can create 0 to many items for sale.

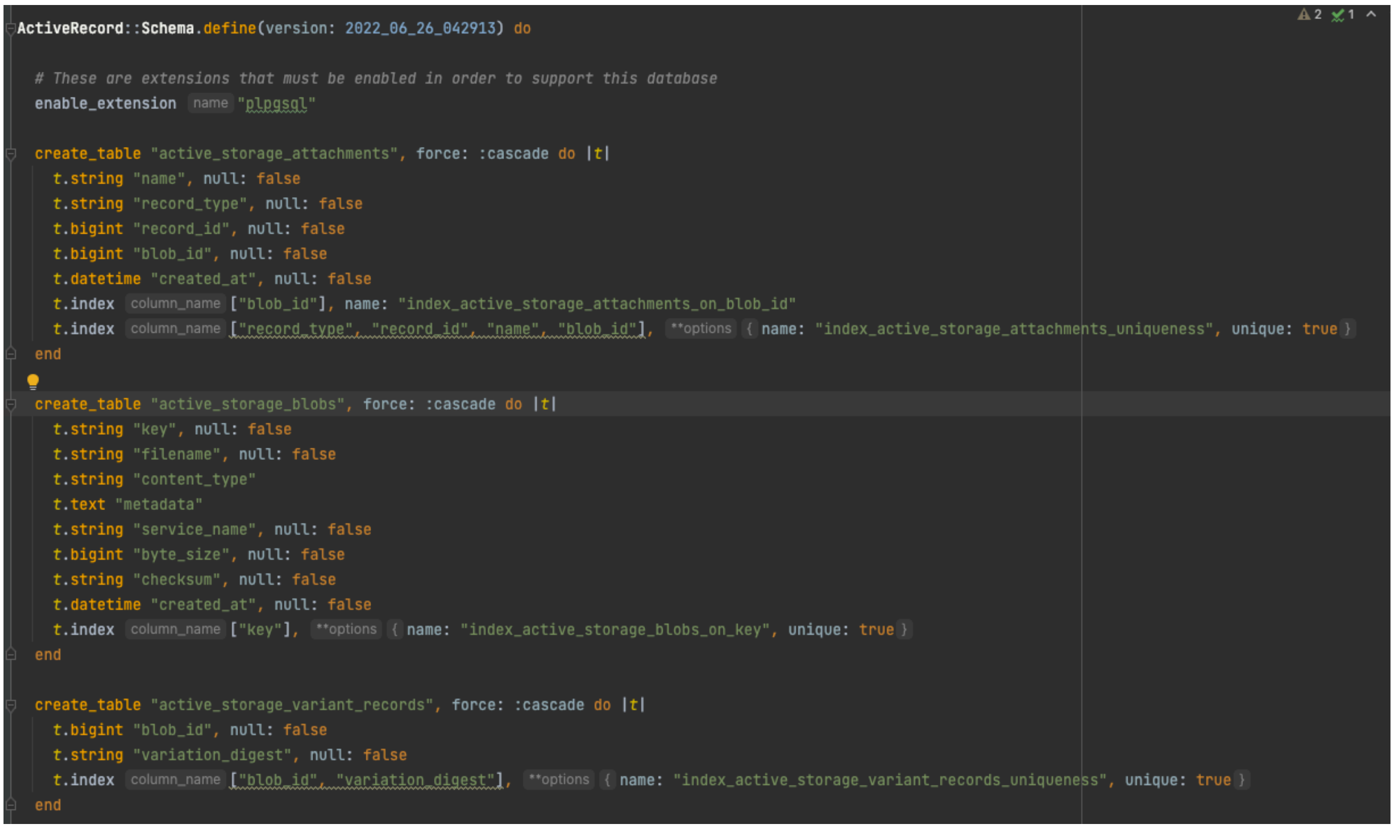
- **Items table**

  Item `belongs_to` Users, it has `item_id` as primary key and a `user_id (seller_id`) as foreign key. Each item has one and only one user, it means an new item will and have to associate with one user. Item also can have 0 or one orders only, it means one item can have no orders, or only one orders associate with it.

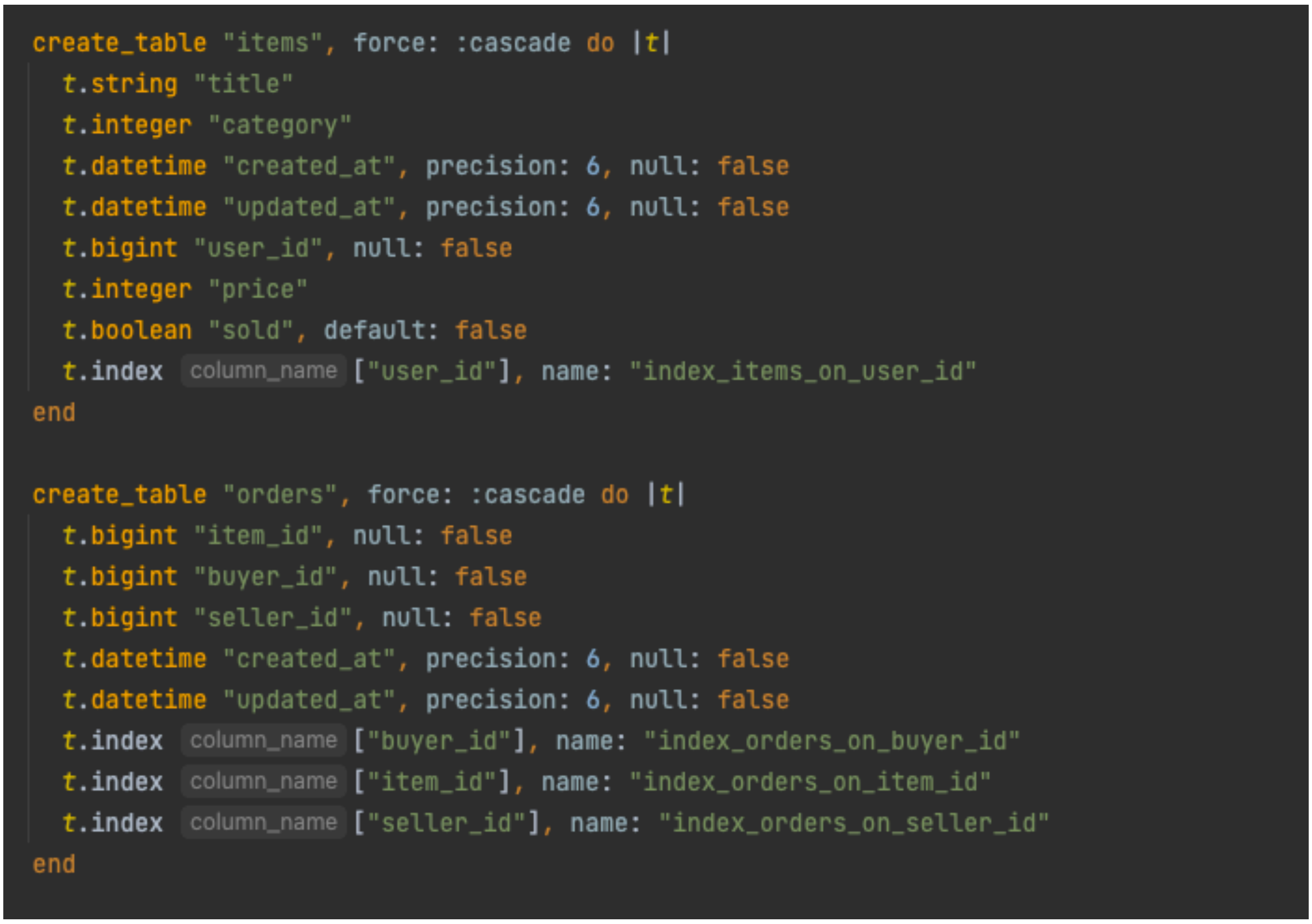
- **Orders table**

  Orders `belongs_to` Users (seller and buyer) and `belongs_to` Items. It has `order_id` as primary key and three foreign keys: `item_id`, `buyer_id` and `seller_id`. Orders have one-to-one relationship with both Users table and Items table. Each order will link to to one and only one item, same as user.

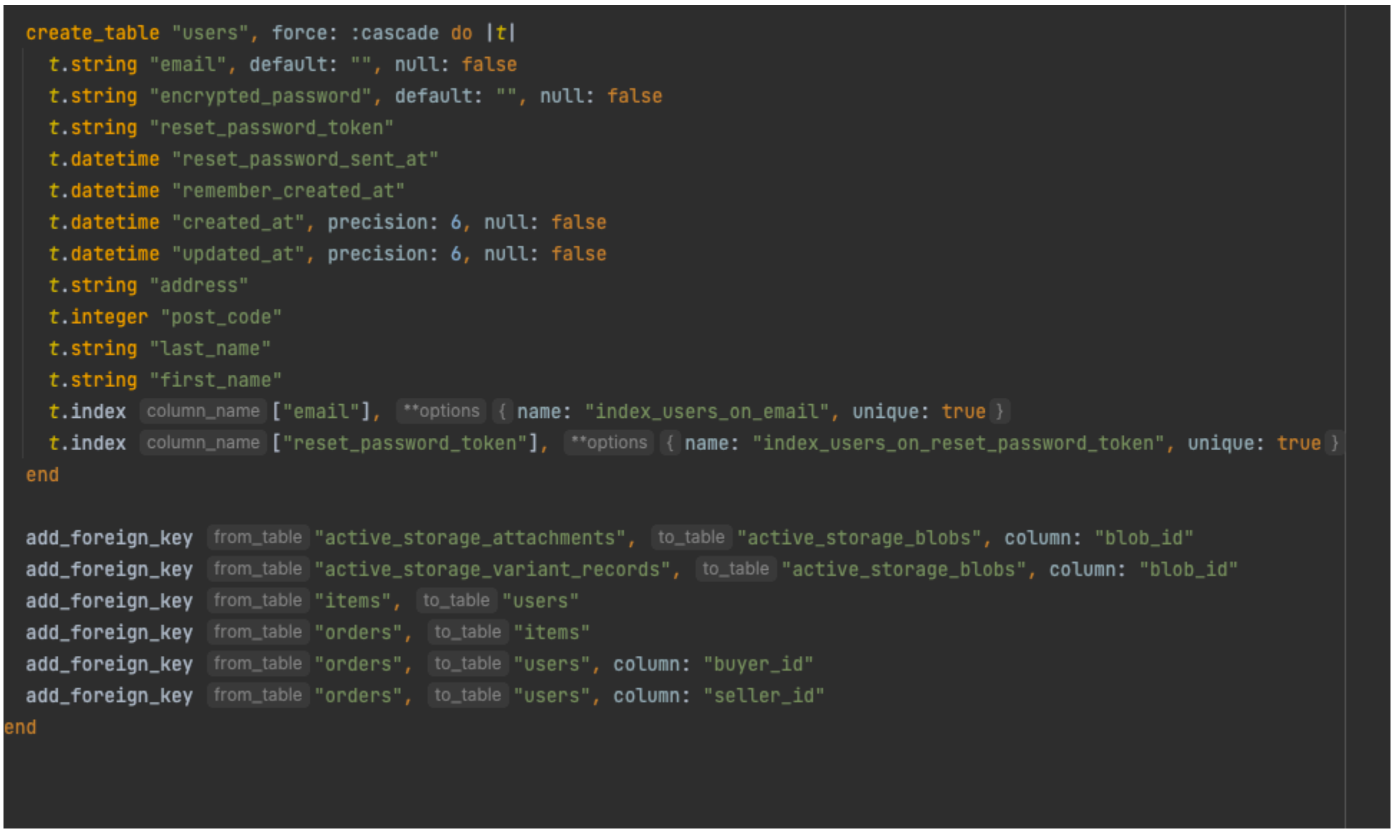
Database schema design:



```
ActiveRecord::Schema.define(version: 2022_06_26_042913) do

  # These are extensions that must be enabled in order to support this database
  enable_extension name "plpgsql"

  create_table "active_storage_attachments", force: :cascade do |t|
    t.string "name", null: false
    t.string "record_type", null: false
    t.bigint "record_id", null: false
    t.bigint "blob_id", null: false
    t.datetime "created_at", null: false
    t.index column_name ["blob_id"], name: "index_active_storage_attachments_on_blob_id"
    t.index column_name ["record_type", "record_id", "name", "blob_id"], **options { name: "index_active_storage_attachments_uniqueness", unique: true }
  end

  create_table "active_storage_blobs", force: :cascade do |t|
    t.string "key", null: false
    t.string "filename", null: false
    t.string "content_type"
    t.text "metadata"
    t.string "service_name", null: false
    t.bigint "byte_size", null: false
    t.string "checksum", null: false
    t.datetime "created_at", null: false
    t.index column_name ["key"], **options { name: "index_active_storage_blobs_on_key", unique: true }
  end

  create_table "active_storage_variant_records", force: :cascade do |t|
    t.bigint "blob_id", null: false
    t.string "variation_digest", null: false
    t.index column_name ["blob_id", "variation_digest"], **options { name: "index_active_storage_variant_records_uniqueness", unique: true }
  end
```



```
create_table "items", force: :cascade do |t|
  t.string "title"
  t.integer "category"
  t.datetime "created_at", precision: 6, null: false
  t.datetime "updated_at", precision: 6, null: false
  t.bigint "user_id", null: false
  t.integer "price"
  t.boolean "sold", default: false
  t.index column_name ["user_id"], name: "index_items_on_user_id"
end

create_table "orders", force: :cascade do |t|
  t.bigint "item_id", null: false
  t.bigint "buyer_id", null: false
  t.bigint "seller_id", null: false
  t.datetime "created_at", precision: 6, null: false
  t.datetime "updated_at", precision: 6, null: false
  t.index column_name ["buyer_id"], name: "index_orders_on_buyer_id"
  t.index column_name ["item_id"], name: "index_orders_on_item_id"
  t.index column_name ["seller_id"], name: "index_orders_on_seller_id"
end
```



```
  create_table "users", force: :cascade do |t|
    t.string "email", default: "", null: false
    t.string "encrypted_password", default: "", null: false
    t.string "reset_password_token"
    t.datetime "reset_password_sent_at"
    t.datetime "remember_created_at"
    t.datetime "created_at", precision: 6, null: false
    t.datetime "updated_at", precision: 6, null: false
    t.string "address"
    t.integer "post_code"
    t.string "last_name"
    t.string "first_name"
    t.index column_name ["email"], **options { name: "index_users_on_email", unique: true }
    t.index column_name ["reset_password_token"], **options { name: "index_users_on_reset_password_token", unique: true }
  end

  add_foreign_key from_table "active_storage_attachments", to_table "active_storage_blobs", column: "blob_id"
  add_foreign_key from_table "active_storage_variant_records", to_table "active_storage_blobs", column: "blob_id"
  add_foreign_key from_table "items", to_table "users"
  add_foreign_key from_table "orders", to_table "items"
  add_foreign_key from_table "orders", to_table "users", column: "buyer_id"
  add_foreign_key from_table "orders", to_table "users", column: "seller_id"
end
```

Tech stack

- **Ruby on Rails**
  - ruby '2.7.5'
  - rails '~> 6.1.5'
- **PostgreSQL**
- **HTML**
- **SCSS**
- **Heroku**
- **GitHub**
- **Ruby Gems**
  - Bootstrap
  - devise
  - simple_form, "~> 5.1"
  - stripe, "~> 6.4"
  - mailgun-ruby, "~> 1.2"

Third party services

- **AWS S3**

Amazon Simple Storage Service (Amazon S3) provides cloud storage for web applications, for example websites and images. In this app, AWS S3 is used for the image upload, retrieving and deleting.

- **Heroku**

  Heroku is a cloud platform as a service (PaaS), it allows developers to build, deploy and manage web applications in the cloud. And It supports most of the popular languages such as Ruby and Node.js. This app uses the free model of Heroku to manage the staging app until deployment of the final production version .

- **Google map**

  Google map is a web mapping platform, it offered satellite imagery, maps, route planning etc. all sort of functions that related to map. This app uses Google map to display the seller's location, so the buyer will easily know the distance between sellers to help to make a purchase decision.

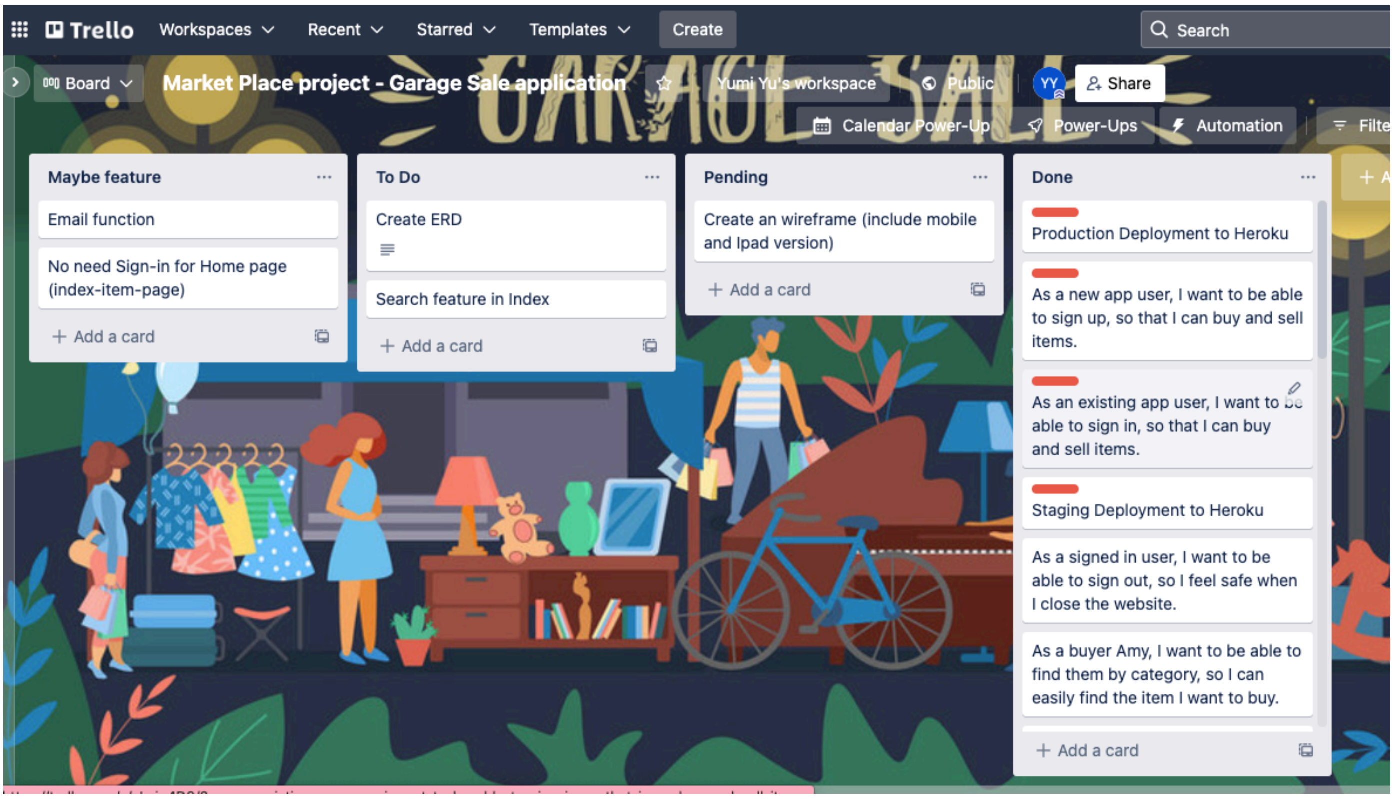
Project plan and Task Management

This application adopted an agile project management methodology. Starting from the planning stage, I listed all the user stories and used them as guides to design the features of this app. According to the main features, I have also drafted the ERD and wireframe, which is really helpful as ERD displays the relationship of models and Wireframes provide a solid view that puts all the ideas and features together.

Trello is the project management tool that I'm using and found it's very powerful and easy to use. Each user story becomes one of my cards that needs to be completed in Trello. I have 4 main list sections in Trello: To do, pending, done and maybe feature. 'Maybe Feature' is created for more optional features if I have enough time in the end. Cards will be allocated to the corresponding list during the process and each card got the priority level, so I can prioritize the important tasks and complete the high priority one first. The task I am working on is marked as pending and when completed it is marked as done. (please see Trello screenshot below)

GitHub is also a very important tool I used throughout this project. I have created a repo in GitHub and use a dev branch to update any feature during the development stage. It is really useful as you can see any changes between commits, so easy to debug and ensured the features merged to main are always working. Unit testing also will be performed when I complete a feature, this is to make sure the small part is functional to avoid a big crush on the project.

Heroku will be the deploy platform for this project. Heroku also related with my GitHub repo and will perform auto sync from dev branch. At first the project will deploy to Heroku as an staging app, when most of the features are completed in the project and working well in the staging app, it will then promoted to production.



Maybe feature
- Email function
- No need Sign-in for Home page (index-item-page)

To Do
- Create ERD
- Search feature in Index

Pending
- Create an wireframe (include mobile and Ipad version)

Done
- Production Deployment to Heroku
- As a new app user, I want to be able to sign up, so that I can buy and sell items.
- As an existing app user, I want to be able to sign in, so that I can buy and sell items.
- Staging Deployment to Heroku
- As a signed in user, I want to be able to sign out, so I feel safe when I close the website.
- As a buyer Amy, I want to be able to find them by category, so I can easily find the item I want to buy.