T2A3 Two-sided Marketplace

Example data has been loaded into the app. Please feel free to log in with admin@admin.com (password: password) or create your account and listings from scratch.

The problem & purpose

In Australia, 7.6 million tonnes of food is wasted every single year -- 70% of which is perfectly edible. Most of that comes from our own homes. 5 million Australians suffer from food insecurity. We have enough food. What we don't have, is a way to get it into the mouths of our nation's hungriest people. Virtual Farmer's Market is an app that allows you to connect to people in your neighbourhood and buy and sell food to one another. From freshly baked bread, to free-range eggs from your chicken coop, to confectionary items you bought too much of at the store. The Virtual Farmer's Market enables food distribution to tackle the ever growing problem of food insecurity in Australia.

https://dashboard.heroku.com/login https://github.com/angieloux/farmers-marketplace

Functionality / features

The home page features a number of listings of various categories (i.e. poultry, meat, confectionery, fruits & vegetables). Once a user has signed up with a name, email and password, they can make a purchase, add a listing of their own, or edit an existing listing that they own. A listing must include a name, category, description and price and can optionally include an image and some feature tags. A non-logged in user can view products, but will be prompted to sign in if they want to make a purchase.

Sitemap


Sitemap

Target audience

The scope of the target audience is broad, focusing on any and all adults of any age that are interested in distributing food in a sustainable way.

Tech stack (e.g. html, css, deployment platform, etc)

The app is built on Ruby on Rails and html and is deployed on Heroku. The app is in its early stages of design and will include CSS styling down the track.

User stories for your *app*

When designing the functionality of the app, the following user stories were considered.

As a Buyer...

I want to...	**so that I can...**
Log into my account	make a purchase

I want to...	so that I can...
Browse all listings	see what food is available
View a listing	see image, features, description, make a purchase
Search & filter listings	find food I am specifically looking for
Reset my password	keep my account secure
Checkout	successfully make a purchase

As a Seller...

I want to...	so that I can...
Do all the things a buyer can do	still act as a consumer / make purchases
Add a new listing	start to make sales
Edit a listing	update / correct listing details
Add an image to a listing	make the listing more desirable for customers
Delete a listing	remove a listing from sale

Wireframes for your *app*

 Wireframes - Index  Wireframes - New  Wireframes - Show

An ERD for your *app*

 ERD

Explain the different high-level components (abstractions) in your *app*

Rails follows the MVC (Model, View, Controller) architecture, which means the app can be clearly and distinctly portioned into sections, i.e. separation of concerns (SoC).

The database is provided by PostgreSQL. The main components of the app are users, categories, listings, transactions, features and listings_features. Additionally, active_storage attachments, active_storage_blobs and active_storage_variant_records were added automatically by Rails so as to store information about attached images.

The relationships are discussed more in depth further down in this document, but this is a brief overview:

Listing model A listing belongs to a user, which can be a buyer or a seller. It includes `sold` which at a later date will be implemented as a part of the app (i.e. viewing order history, a sellers history, etc). `search_by_name` and `filter_by_category`enable the user to find what products they are after more quickly. Additionally, two data sanitization methods `remove_whitespace` and `convert_price_to_cents`make for cleaner data and a nicer user experience.

User model & listings controller & views A user model was created with Devise to provide authentication. A user interacts with the website and can take on the role of buyer or seller. As a logged in user, the

listings_controller designs the logic a user needs to make a purchase, add a listing, or edit or delete a listing (if they are the owner). The views also further restricts certain buttons from being seen by those who are not authenticated (e.g. a non-listing owner cannot see "Edit" and "Delete" buttons associated with a listing, and similarly, a listing-owner cannot see the "Buy" button when looking at the show view). A non-logged in user is able to sign up or sign in, recover their password, etc. All users, whether logged in or not, can view all the items for sale via the index page. When adding a new listing, it must have one category (i.e. dairy, meat, poultry), a description, a name, a price and, optionally some features (organic, non-GMO, etc) and one image. To make finding products easier, a user can also search for specific products by name, i.e. ("beans" would return "Refried beans") or filter by category.

Category model A listing must have a category, and a user can filter their search by category to show them only relevant products (i.e. vegetables).

Payments controller & transactions model A user can also make a transaction by clicking "Buy" on the listings 'show' view. Stripe provides this functionality.

Features model & features_listing join table A listing can also be tagged with a some features, if the poster decides (e.g. vegetarian, gluten-free, etc). In the future this will be used as a way to filter products as well, like done with category.

Detail any third party services that your *app* will use

The third party services adopted by the app are:

- **Heroku** is container-based cloud Platform as a Service that is used as a means to deploy the app to production.
- **Amazon Web Services Simple Storage Service (S3)**: AWS S3 buckets are used by the app for the purposes of storing images. Rails has built-in active storage that allows for connecting to online cloud services like S3.It is is a highly scalable tool in it can store an unlimited amount of data and therefore allows for faster image loading times. Images can either be attached by the user at the time of adding a new listing, or later on when they choose to edit a listing they own.
- The **Devise** gem is used as a means to provide user authentication within the app. It has many uses, some of which include: -- Allowing a user to sign up, sign in, or sign out. -- Storing passwords in an encrypted hash for increased security. -- Allowing the user to reset their password -- Timing out a logged in user if their session is idle for more than 30 minutes
- **Stripe** is a payments service and is used in a test capacity to provide a mock check-out process.

Describe your projects *models* in terms of the relationships (active record associations) they have with each other

The app consists of User, Category, Listing, Transaction and Feature models. A user is able to take on the role of both buyer and seller, and so there are a a number of active record associations between all of the models, detailed below.

- A user `has_many` listings and a listing `belongs_to` a user.
- `purchases` and `sales` are defined as two separate aliases for the Transaction model, linked via foreign key of `buyer_id` and `seller_id` respectively. A user `has_many` purchases and sales.
- A user also `has_many` purchased_listings `through` purchases and also `has_many` sold_listings `through` sales, both with a `source` of listing. (i.e. Rails says look for an association of listing on the

sales model [i.e. Transaction])
- A transaction `belongs_to` a seller (i.e. user) and also `belongs_to` a buyer (i.e. user).
- A transaction `belongs_to` a listing.
- A listing `belongs_to` a category and a user.
- A listing `has_one` transactions, and also `has_one_attached` image. (In future updates, this will likely be updated to allow for multiple image uploads).
- A listing `has_many` features through the feature_listing join table. A feature also `has_many` listings.

Discuss the database relations to be implemented in your application

As seen in the ERD, there are a number of database relations used within the app. A user is a person who can both buy and sell food. The database relations are shown below.data

- User & Listings: A user can have zero or many listings, but a listing must be associated only with one user (one-to-many). They are linked by user_id foreign key on the listings table.
- User & Transactions: A user, which can be either a buyer or seller, can have many transactions (i.e. purchases and sales), and a transaction is associated with a buyer and a seller (i.e. the alias for user). Transactions have a buyer_id and a seller_id which are foreign keys on the transaction table associated with the user id on the user table.
- Listing & Categories: Listings can have only one category, and the category can have zero or many listings (one-to-many). They are linked by category_id foreign key on the listings table.
- Listings & transactions: Listings can have zero or one transaction which is linked through buyer_id and seller_id foreign keys on the transaction table which is linked to the User table user id primary key. This is a one-to-many relationship.
- Features_listings is a join table that joins features and listings through their respective ids. Features_listings can have zero or many features, and features can have zero or many features_listings. Listings can have zero or many features_listings, and they can have zero or many features through features_listings. This is a many-to-many relationship.
- Active storage: Each listing can store an image. The active_storage_blobs table stores information about the file (i.e. size, content type, filename, etc). The active_storage_attachments table is related to listings by way of polymorphic join. The record_type and record_id implement this relationship, with record_id acting as a foreign key and defining a one-to-many relationship with listings. The active_storage_attachments has a foreign key of blob_id which results in a one-to-many relationship with the active_storage_blobs table. This means the active_storage_attachments is a join table that links listings to active_storage_blobs through active_storage_attachments (many-to-many).

```
ActiveRecord::Schema.define(version: 2021_12_12_123145) do

  # These are extensions that must be enabled in order to support this
database
  enable_extension "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 ["blob_id"], name:
"index_active_storage_attachments_on_blob_id"
    t.index ["record_type", "record_id", "name", "blob_id"], 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 ["key"], 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 ["blob_id", "variation_digest"], name:
"index_active_storage_variant_records_uniqueness", unique: true
  end

  create_table "categories", force: :cascade do |t|
    t.string "name"
    t.datetime "created_at", precision: 6, null: false
    t.datetime "updated_at", precision: 6, null: false
  end

  create_table "features", force: :cascade do |t|
    t.string "name"
    t.datetime "created_at", precision: 6, null: false
    t.datetime "updated_at", precision: 6, null: false
  end

  create_table "features_listings", force: :cascade do |t|
    t.bigint "feature_id", null: false
    t.bigint "listing_id", null: false
    t.datetime "created_at", precision: 6, null: false
    t.datetime "updated_at", precision: 6, null: false
    t.index ["feature_id"], name: "index_features_listings_on_feature_id"
    t.index ["listing_id"], name: "index_features_listings_on_listing_id"
  end

  create_table "listings", force: :cascade do |t|
    t.string "name"
    t.text "description"
    t.integer "price"
    t.bigint "category_id", null: false
    t.bigint "user_id", null: false
    t.datetime "created_at", precision: 6, null: false
```

```
    t.datetime "updated_at", precision: 6, null: false
    t.string "image_filename"
    t.string "username"
    t.datetime "date_created"
    t.boolean "sold", default: false, null: false
    t.index ["category_id"], name: "index_listings_on_category_id"
    t.index ["user_id"], name: "index_listings_on_user_id"
  end

  create_table "transactions", force: :cascade do |t|
    t.bigint "seller_id", null: false
    t.bigint "buyer_id", null: false
    t.bigint "listing_id", null: false
    t.datetime "created_at", precision: 6, null: false
    t.datetime "updated_at", precision: 6, null: false
    t.index ["buyer_id"], name: "index_transactions_on_buyer_id"
    t.index ["listing_id"], name: "index_transactions_on_listing_id"
    t.index ["seller_id"], name: "index_transactions_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.integer "sign_in_count", default: 0, null: false
    t.datetime "current_sign_in_at"
    t.datetime "last_sign_in_at"
    t.string "current_sign_in_ip"
    t.string "last_sign_in_ip"
    t.string "name", null: false
    t.index ["email"], name: "index_users_on_email", unique: true
    t.index ["reset_password_token"], name:
"index_users_on_reset_password_token", unique: true
  end

  add_foreign_key "active_storage_attachments", "active_storage_blobs",
column: "blob_id"
  add_foreign_key "active_storage_variant_records",
"active_storage_blobs", column: "blob_id"
  add_foreign_key "features_listings", "features"
  add_foreign_key "features_listings", "listings"
  add_foreign_key "listings", "categories"
  add_foreign_key "listings", "users"
  add_foreign_key "transactions", "users", column: "buyer_id"
  add_foreign_key "transactions", "users", column: "listing_id"
  add_foreign_key "transactions", "users", column: "seller_id"
end
```

Describe the way tasks are allocated and tracked in your project

I used Trello for project management. The backlog column contained tasks I had yet to decide whether they were important enough to include, as well as bugs and blockers. The to-do column includes everything that hadn't been assigned. Doing and done are fairly self explanatory. Find the board here: https://trello.com/b/8zUOI5au/marketplace-app