Save Farmers

Team: Save Farmers

Dillon Pikulik

Eric Vo

Nick Burlakov

Eastern Washington University

March 17, 2025

Version 5.0

Github: https://github.com/Sanmeet-EWU/cscd-488-490-project-save-farmers

Figma:

https://www.figma.com/design/shMiHT8N7REikxoaUDINU9/Save-Farmers?node-id=0-1&t=NkgSp0bFDW9XTU2Q-

Demo Video: Save Farmers Demo

Web App: https://blue-beach-0555dc31e.6.azurestaticapps.net/

Table of Contents

List of Figures

1 Abstract

Small farmers lack access to a dedicated platform for selling their crops, leading to difficulties in selling produce before it spoils. This forces them to accept low prices, pay for costly storage, or rely on middlemen who take a share of their profits. To address this issue, we have developed the Save Farmers app, a marketplace that enables farmers to list and sell their crops directly to consumers, giving them more control over their sales while reducing waste and increasing earnings.

The app is built using the Ionic framework and is compatible for both Android and iOS. Its backend was developed using ASP.NET and hosted on Microsoft Azure. Key features include easy crop listing, post editing, and access to buyer and seller contact information. Additionally, an admin panel provides insights into user activity, post history, and general app analytics.

By eliminating middlemen, the Save Farmers app helps farmers sell their produce faster and at fairer prices. Future enhancements will include payment integration, real-time notifications, and an in-app messaging system to streamline transactions. We are proud of our achievements and remain committed to improving the livelihoods of farmers worldwide.

1.1 Problem Statement and Motivation

Small farmers do not have access to a platform tailored specifically for selling their crops. As a result, they often struggle to sell their produce before it spoils, forcing them to accept lower prices, invest in costly storage solutions, or rely on middlemen who reduce their earnings. These challenges not only affect farmers' income but also contribute to food waste and inefficiencies in the agricultural market.

Our goal is to provide a dedicated digital marketplace where farmers can list and sell crops directly to consumers, eliminating unnecessary intermediaries and ensuring fairer pricing. By developing a user-friendly and accessible platform, we aim to empower farmers with greater control over their sales, reduce post-harvest losses, and enhance their financial stability. The Save Farmers app bridges this gap by offering a seamless way for farmers to connect with buyers, ultimately improving market access and supporting sustainable farming practices.

2 Introduction

2.1 Purpose and Scope

This document provides a overview of the Save Farmers App, a marketplace designed to address the challenges farmers face in selling their crops efficiently. The purpose of this document is to outline the system's objectives, define its scope, clarify specialized terms, and provide references for the tools and frameworks used in the project's development.

The Save Farmers App aims to simplify the process of selling crops by connecting farmers directly with buyers. This document serves as a guide for understanding the system's design, functionality, and future directions while providing a basis for collaboration among stakeholders.

The project encompasses the design and development of a mobile application that facilitates the posting and searching of crops. It targets two primary user groups: farmers (sellers) and buyers. The app's core features are user authentication, crop postings, search and filter functionalities, and profile management. External integrations, such as messaging or payment systems, are not part of the initial development phase but are considered for future enhancements.

2.2 Definitions, Acronyms, and Abbreviations

1. **Save Farmers App**: The system under development connects farmers and buyers.
2. **LTIMindtree**: The project's sponsor organization.
3. **ER Diagram**: Entity-relationship diagram used to design the database schema.

3 System Overview

3.1 Product Perspective

The Save Farmers App fits within the broader ecosystem of agricultural technology solutions aimed at improving farmer livelihoods. It addresses inefficiencies in traditional crop marketing processes, such as delayed sales and spoilage, by offering a centralized platform where farmers can post their crops and buyers can search for them. The app draws inspiration from platforms like Facebook Marketplace and Central Dispatch but tailors its functionality to the unique needs of the agricultural sector. It operates without a built-in payment system or in-app messaging.

3.2 Product Features

The app includes the following features, aligned with its functional requirements:

- User Login (FR-1): Allows farmers and buyers to create and manage accounts.
- Crop Posting (FR-2): Enables farmers to list their crops for sale with details like price, quantity, and location.
- Crop Search and Filters (FR-3, FR-6): Buyers can search for specific crops and apply filters to help their results.
- Profile Page (FR-4): Users have dedicated profiles displaying past posts and analytics about those posts.
- Post Editing (FR-5): Farmers can update existing posts to ensure accurate information for buyers.
- Viewing the Listings (FR-3): Detailed crop listings provide essential data to help buyers make informed decisions.
- Admin Account (FR-7): This account will only be for the admins of the app. This account will be able to view analytics about everything on the app.
- Analytics (FR-8): Admins and sellers will have information displayed to them about their details.
- Rating System (FR-9): Users will have an opportunity to provide feedback on their interaction with buyers and sellers, letting everyone know who is easier/more reliable to work with.

3.3 User Classes and Characteristics

- Farmers (Sellers):
    - Primary users are responsible for posting crops for sale.
    - Requires features: account creation, crop posting, and editing.
- Buyers:
    - Primary users who search for and purchase crops.

    - Requires features: search and filters.
- Admin:
    - Secondary users who can view analytics about the app.
    - Uses all features because admin of app.

3.4 Assumptions and Dependencies

- Assumptions:
    - Users have access to a smartphone with an internet connection to use the app.
    - Farmers are familiar with basic app navigation and data entry.
- Dependencies:
    - SQL: Backend database for storing user data and posts.
    - ASP.NET: Backend API connecting the front-end to the database.
    - Azure: Housing the API and database.
    - Cypress: Tool for validating app functionality during testing phases.

4 User Stories and Scenarios

4.1 User Story 1: Crop Post

U1: As a farmer, I want to post my specific crops for sale, so that I can quickly sell the crops.

Scenario 1: Successful crop post

- Given I am on the crop post page
- When I enter the crop name, price, location, time, etc
- And I click the "Post" button
- Then I should see the listing posted

Scenario 2: Unsuccessful crop post

- Given I am on the crop post page
- When I enter the crop name, price, time, etc

- And I click the "Post" button
- Then I should see the message "Missing An Input On The Post"

4.2 User Story 2: Crop Posts

U2: As a farmer, I want to see all of my posts of crops for sale, so that I can see what crops I have already posted.

Scenario 1: Having posted crops

- Given I am on the “My Crops” page
- Then I should see all of my crop postings

Scenario 2: Not having any posted crops

- Given I am on the “My Crops” page
- Then I should see no crops posted

4.3 User Story 3: Editing Crop Post

U3: As a farmer, I want to edit the post of my specific crops that are currently for sale, so that there is no miscommunication between me (the farmer) and the buyer.

Scenario 1: Successfully deleted crop post

- Given I am on the “My Crops” page
- When I find that crop I want to delete
- And I click the "Delete" button on that crop post
- Then I should see that the listing is no longer on the “My Crops” and “Search” pages

Scenario 2: Editing crop post

- Given I am on the “My Crops” page
- When I find that crop I want to edit
- And I click the "Edit" button on that crop post
- And I changed the field that needed editing

- And I click the "Save" button on that crop post
- Then I should see that the listing has changed on the “My Crops” and “Search” pages

4.4 User Story 4: Crop Search

U4: As a buyer, I want to search for specific crops by name, so that I can quickly find and purchase the crops I am interested in.

Scenario 1: Successful crop search

- Given I am on the crop “Search” page
- When I enter the crop name "Tomatoes" in the search bar
- And I click the "Search" button
- Then I should see a list of available "Tomatoes"
- And I should see their prices, quantities, and seller information

Scenario 2: No crops found

- Given I am on the crop “Search” page
- When I enter the crop name "Dragon Fruit" in the search bar
- And I click the "Search" button
- Then I should see the message "No results found for 'Dragon Fruit'"
- And I should remain on the crop search page

4.5 User Story 5: View Crop Details

U5: As a buyer, I want to view detailed information about a specific crop, so that I can make an informed purchasing decision.

Scenario 1: Viewing crop details

- Given I have searched for "Tomatoes"
- And I see a list of available "Tomatoes"
- When I click on a specific listing for "Tomatoes"
- Then I should be taken to the crop details page

- And I should see information such as price, quantity, seller location, and delivery options

Scenario 2: Incomplete crop details

- Given I am on the crop details page for "Tomatoes"
- When the seller has not provided full details (e.g., no quantity listed)
- Then I should see the message "Quantity information not available"
- And I should still be able to view other available information

4.6 User Story 6: Account

U6: As a buyer or seller, I want to be able to create and have an account, so that I can see my profile.

Scenario 1: Making an account

- Given I need to make an account
- When I click “Sign Up”
- Then I can create an account
- And login to use the app’s features

Scenario 2: Looking at account

- Given that I want to look at posts
- Then I click on the “Profile” button
- Then I will be in my account
- And I can see the posts

4.7 User Story 7: Admin Account

U7: As an admin, I want to be able to have an account, so that I can see analytics and use the app on on the app.

Scenario 1: Logging in

- Given I have the account credentials

- And Login
- Then I can use the app

Scenario 2: Admin analytics

- Given I want to look at how many accounts were made in the last month
- Then I click the accounts tab
- And months filter
- Then I can view all the accounts that are made in the last month

4.8 User Story 8: Analytics

U8: As an admin or seller, I want to be able view analytics, so that I can make calculated decisions.

Scenario 1: Admin analytics

- Given I want to look at how many accounts were made in the last month
- Then I click the accounts tab
- And months filter
- Then I can view all the accounts that are made in the last month

Scenario 2: Seller analytics

- Given I want to look at how many fruits were bought in the last month
- Then I click the fruits tab
- And months filter
- Then I can view how many fruits were bought in the last month

4.9 User Story 7: User Rating System

U9: As a user, I should be able to leave a review on buyers and sellers, so that other users could have more details in choosing a buyer or seller than just crop types, prices and experation.

Scenario 1: User rates buyer/seller

- Given I have a product to sell/buy
- And have successfully sold/bought it
- Then I can leave an optional review on how easy it was to make the transaction with the other party.

Scenario 2: User views buyer/seller reviews

- Given that I want to look at other users interactions with a particular seller/buyer
- Then I click on their profile
- And I can view all the information recorded by other users who have a transaction history with the seller/buyer.

5 Functional Requirements

Functional Requirement:	**Description/Specification:**
[FR-1(U6)] User Login	Users should be able to create an account and log in.
Priority	Level 0 (Essential)
[FR-2(U1)] Sellers Posts	Sellers should be able to make posts of the products they are selling.
Priority	Level 0 (Essential)
[FR-3(U4,5)] Seeing Posts	Any user should be able to see the posts that are made by the sellers.
Priority	Level 0 (Essential)
[FR-4(U2,6)] Profile Page	All users should have a profile page that has the posts.
Priority	Level 1 (High Priority)
[FR-5(U3)] Sellers Editing Posts	Sellers should be able to edit their existing posts.
Priority	Level 1 (High Priority)
[FR-6(U4,5)] Search Filters	All users should be able to make search filters to find what they are looking for easier and faster.
Priority	Level 2 (Medium Priority)
[FR-7(U7,6,5,4,2)] Admin Account	This account will only be for the admins of the app. This account will be able to view analytics about everything on the app.
Priority	Level 1 (High Priority)
[FR-8(U8)] Analytics	Admins and sellers will have information displayed to them about their details.
Priority	Level 1 (High Priority)
[FR-9(U9)] User Rating System	Users should be able to rate their experience with sellers/buyers.

Priority	Level 2 (Medium Priority)

6 Use Case Diagrams and Use Cases

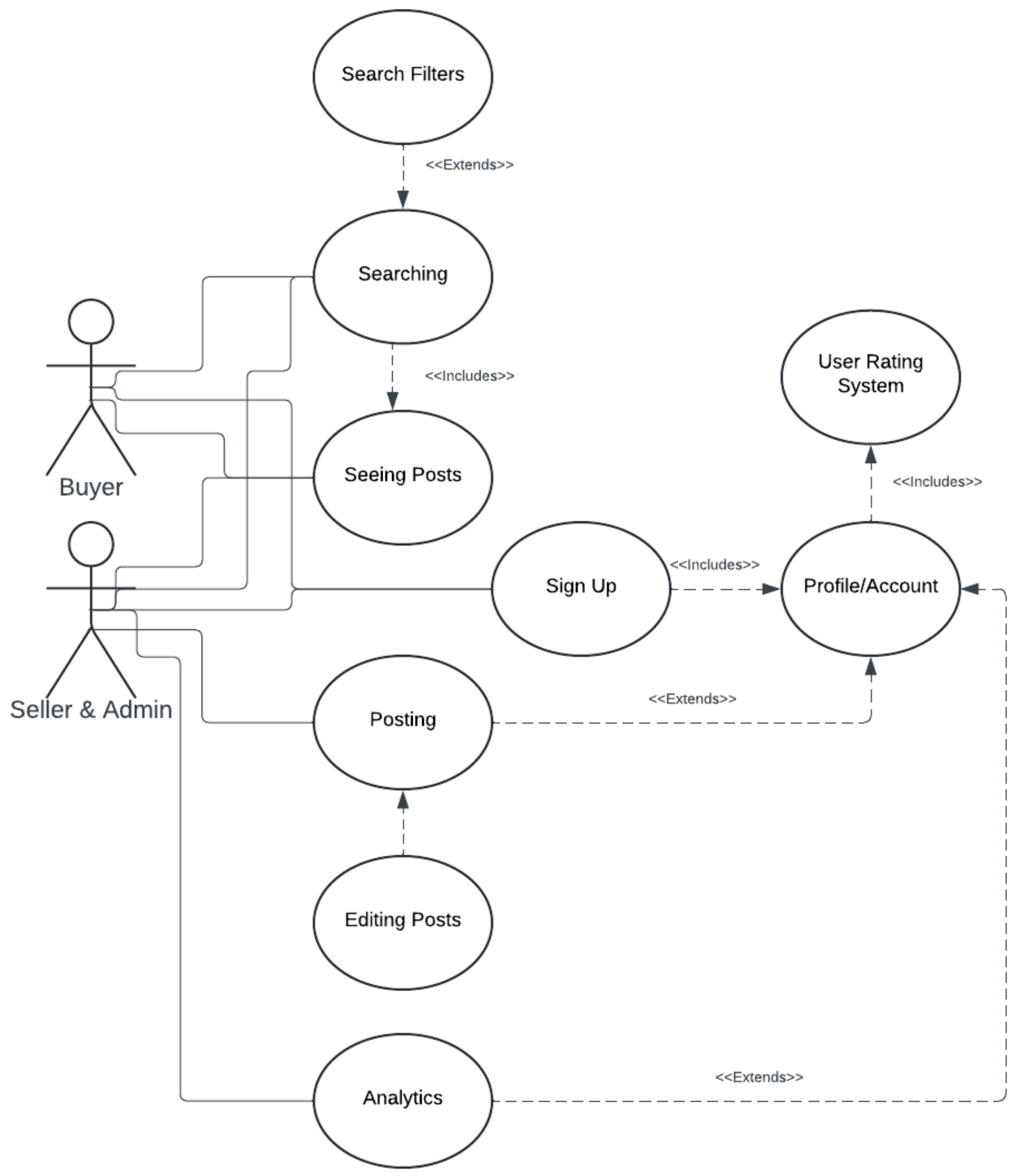


Fig. 1 (Use Case Diagram)

6.1 Use Case 1: Sign Up

- Actors: Seller and Buyer

- Preconditions: None
- Postconditions: Will have an account
- Main Flow:
  - Click Sign Up
  - Puts in information about self
  - Now has an account
- Alternative Flow: If not all information is provided, then an account can not be made.
- Related Requirements: FR-1

6.2 Use Case 2: Profile/Account

- Actors: Seller and Buyer
- Preconditions: Sign Up
- Postconditions: Can post and use the account
- Main Flow:
  - Click Profile Button
  - Now they can see their profile
  - And they can use the feature of the app
- Alternative Flow: Without having the profile they can't use the many different features of the app.
- Related Requirements: FR-1, FR-4

6.3 Use Case 3: Posting

- Actors: Seller
- Preconditions: Has Account
- Postconditions: Can post their product
- Main Flow:
  - Click "Post"
  - Fill in information
  - Then click "Submit"
- Alternative Flow: If they don't fill out all of the information the post will be incomplete and won't post.
- Related Requirements: FR-2, FR-4

6.4 Use Case 4: Searching

- Actors: Seller and Buyer
- Preconditions: Has Account
- Postconditions: Can search

- Main Flow:
    - Click Search
    - Will see all available products
    - And can search within them
- Alternative Flow: If no products are available, you won't see anything.
- Related Requirements: FR-3

6.5 Use Case 5: Search Filters

- Actors: Seller and Buyer
- Preconditions: Has Account
- Postconditions: Can search with filters
- Main Flow:
    - Click Search
    - Click Filter
    - Make Filter
    - Click Apply
    - Now they will have their search filtered
- Alternative Flow: If no posts are made, you won't see anything.
- Related Requirements: FR-3, FR-6

6.6 Use Case 6: Seeing Posts

- Actors: Seller and Buyer
- Preconditions: Has Account
- Postconditions: Can see sellers' posts
- Main Flow:
    - Click Posts
    - Now you will see all available posts
- Alternative Flow: If no posts are made, you won't see anything.
- Related Requirements: FR-3

6.7 Use Case 7: Editing Posts

- Actors: Seller
- Preconditions: Has Account
- Postconditions: The user can edit their posts
- Main Flow:
    - Click Profile
    - Then find the post you want to edit
    - Click Edit
    - Then edit the post

    - Then Click Submit
- Alternative Flow: If you don't want to edit the post you don't have to do anything.
- Related Requirements: FR-5

6.8 Use Case 8: Analytics

- Actors: Admin and Seller
- Preconditions: Has Account
- Postconditions: Can see their analytics
- Main Flow:
    - Click Analytics
    - Then finds analytics that they are looking for
- Alternative Flow: If they don't want to not look at Analytics they can click to a different page.
- Related Requirements: FR-8

6.9 Use Case 9: User Rating System

- Actors: Buyer and Seller
- Preconditions: Has Account
- Postconditions: Can see user reviews
- Main Flow:
    - Click on profile user
    - Then finds reviews left by other users
- Alternative Flow: If user wishes to leave review then
    - Once their transaction has been concluded
    - Click on "leave review"
    - Type your experience with the other party
    - Click Submit
    - User review is logged, filtered for censorship flags and posted on other parties profile.
- Related Requirements: FR-9

7 Non-functional Requirements

Non-Functional Requirement:	**Description:**
[NFR-1] Usability	The system shall be easy to navigate for all users, regardless of technical skill level.
[NFR-2] Availability	The system shall have a consistent uptime so

	that users can trust the app and use it reliably.
[NFR-3] Scalability	The system shall utilize a modular architecture, enabling individual components to be upgraded or replaced without impacting the overall system functionality.
[NFR-4] Performance	The system shall load pages within 2 seconds under normal traffic conditions.

8 Traceability Matrix

Functional Requirement:	**Use Case:**	**User Story:**	**Priority:**
FR-1: User Login	UC-1: Sign Up	US6: As a user, I want to create an account and log in.	Level 0
FR-2: Sellers Posts	UC-3: Posting	US1: As a seller, I want to post my crops up for sale.	Level 0
FR-3: Seeing Posts	UC-6: Seeing Posts	US4, US5: As a user, I want to see all public crop postings.	Level 0
FR-4: Profile Page	UC-2: Profile/Account	US2, US6: As a user, I want to see my posts.	Level 1
FR-5: Seller Editing Posts	UC-7: Editing Posts	US3: As a seller, I want to edit my existing posts.	Level 1
FR-6:Search Filter	UC-5: Search Filters	US4, US5: As a user, I want to be able to search for specific posts.	Level 2
FR-7: Admin Account	UC-2: Profile/Account	US7: As an admin, I want to be able to have an account.	Level 1

FR-8: Analytics	UC-8: Analytics	US8: As an admin or seller, I want to be able to view analytics.	Level 1
FR-9: User Rating System	UC-9: User Rating System	US9: As a user, I want to be able to view and leave reviews on other profiles	Level 2

9 Evaluation of Existing Solution

According to our project description, I have 2 existing apps/websites that I can evaluate. Facebook Marketplace is a marketplace where any user of Facebook can post anything for sale, with exceptions. Our app will be more geared toward farmers but use the same concepts. Farmers will be able to post what they have for sale and give a price and a description of necessary information. Just like Facebook Marketplace, we will have no internal payment options, but Facebook does use the messaging app Messenger, which could be good to incorporate into our app later. This way, we could put out less public information on our app, such as phone numbers.

Another website that I will look at is Center Dispatch. That website is geared towards car transport. Central dispatch has brokers that post the cars that need to be moved and shippers that move those cars. On the pos,t the information about what vehicle, price, location, days, and contact number are provided to the shippers. The contact of the shipper to broker and payment are all done outside the website itself. Our app will look more like Central Dispatch. The implementation of payment and messaging within the app is not a priority for us at this moment.

In both of these websites/apps, there is a big issue with user authentication, spam, and resellers. In Facebook Marketplace, there is no real way to avoid this, and it is solely up to the user to trust the other. While in Central Dispatch, some ways/questions could be provided to prove the legitimacy of a shipper/broker outside of the app.

These 2 examples are going to be a big source of inspiration for our app since our sponsor has given us a lot of creative freedom. First, keeping in mind the requirements, then implementing useful features and possibly creating our own to benefit farmers.

10 System Architecture Design

10.1 Architecture Diagram

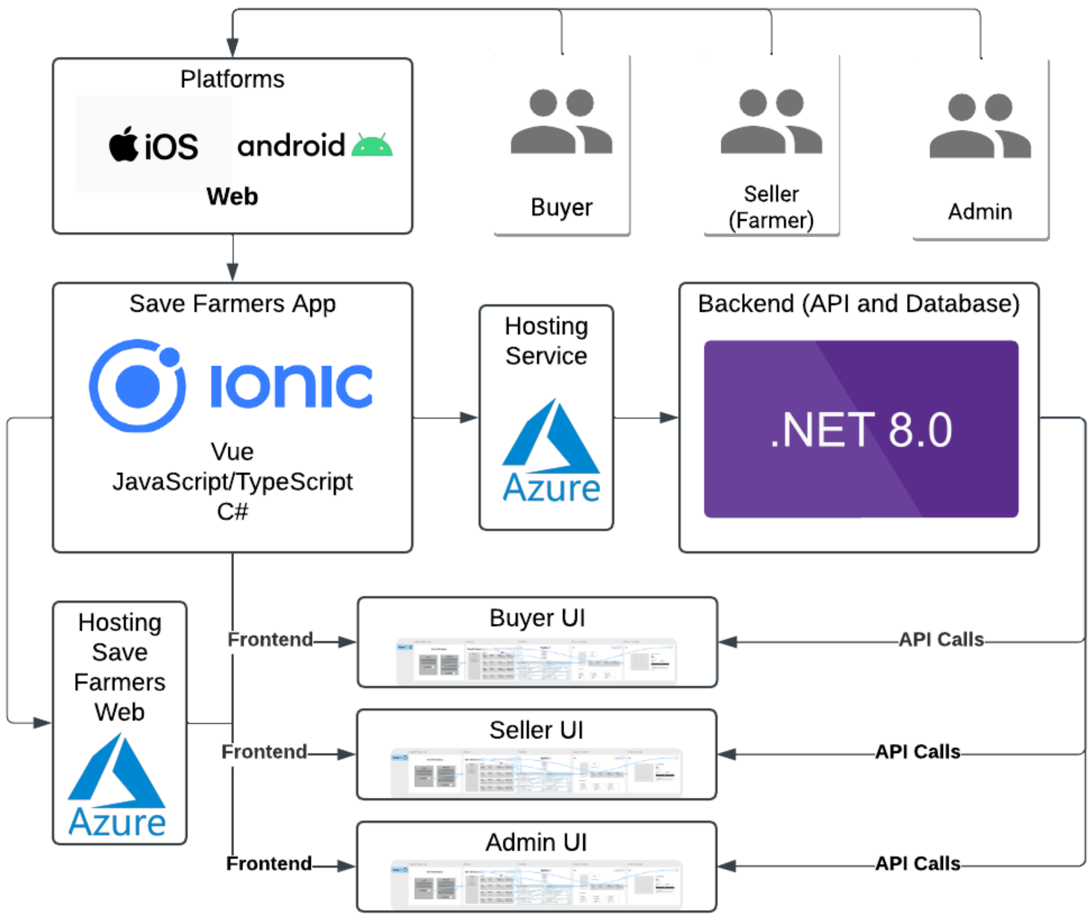


Fig. 2 (Architecture Diagram)

There are 3 types of users buyer, seller (farmer), and admin. These users can access the app through IOS, android, or the web. The Save Farmers program is either accessed from the web which is hosted from Azure or can be accessed from the app which is running locally on device. The backend of the app (API and database) are hosted on Azure. When the app's frontend makes a call to the API, for the database, it is accessing it from Azure.

10.2 Behavioral Diagram (Activity Diagram)

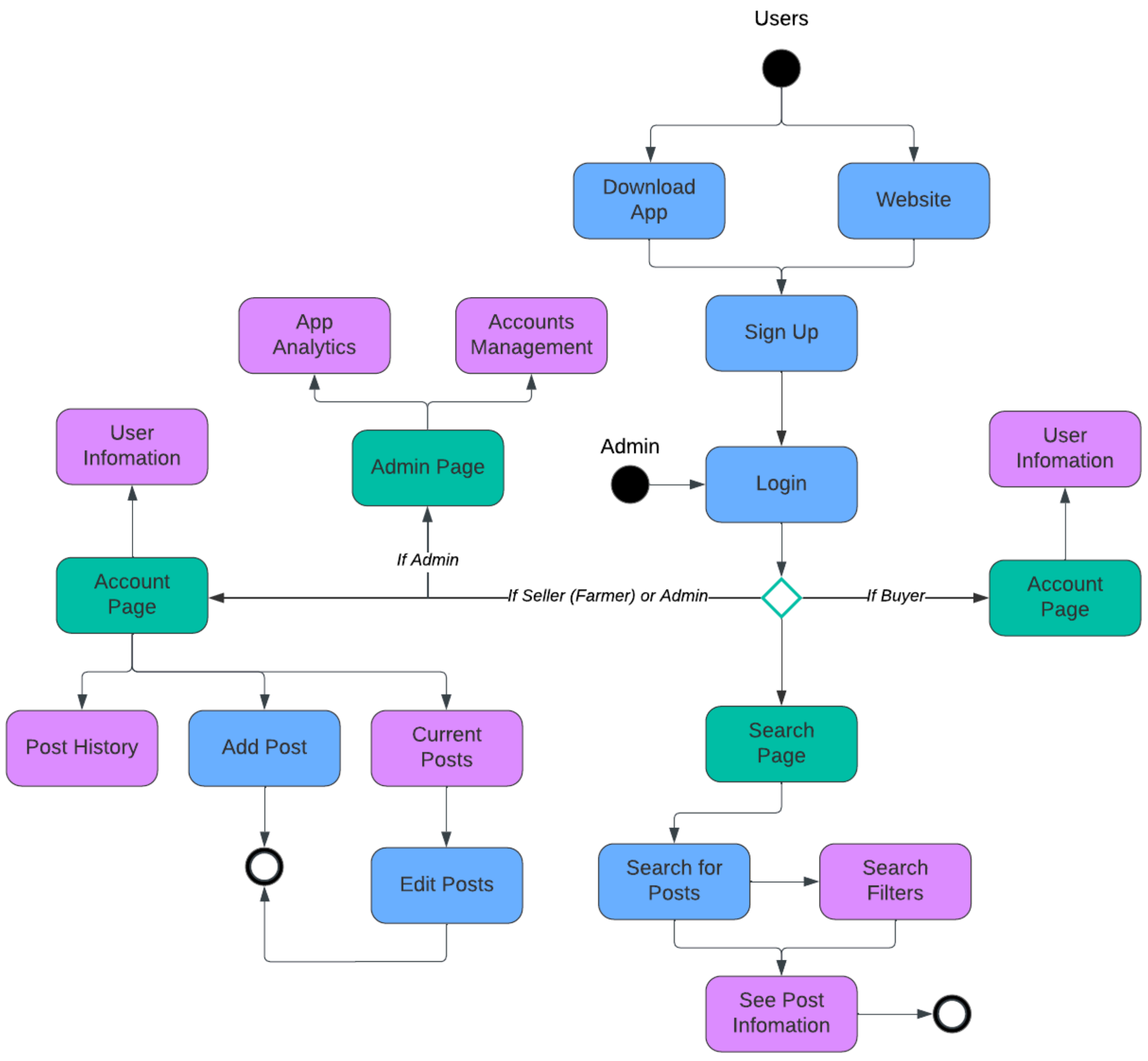


Fig. 3 (Activity Diagram)

A user will download the Save Farmers App or access it from the web and proceed to make an account. Once an account is made, the user can log in. If the user is a seller or an admin, they can access their account page. The seller and admin can also make posts and edit them. A seller or admin can see their post history and user information on the account page. Only if you are the admin do you have an admin account. The admin has a special page where they can see app analytics and manage accounts on the app. If you are a buyer, you can access your account page and see user information. All users can access the search page, search for posts, and make filters.

After a user finds the post that they want, they can contact the seller through the given information in the post.

10.3 Structural Diagram (Class Diagram)

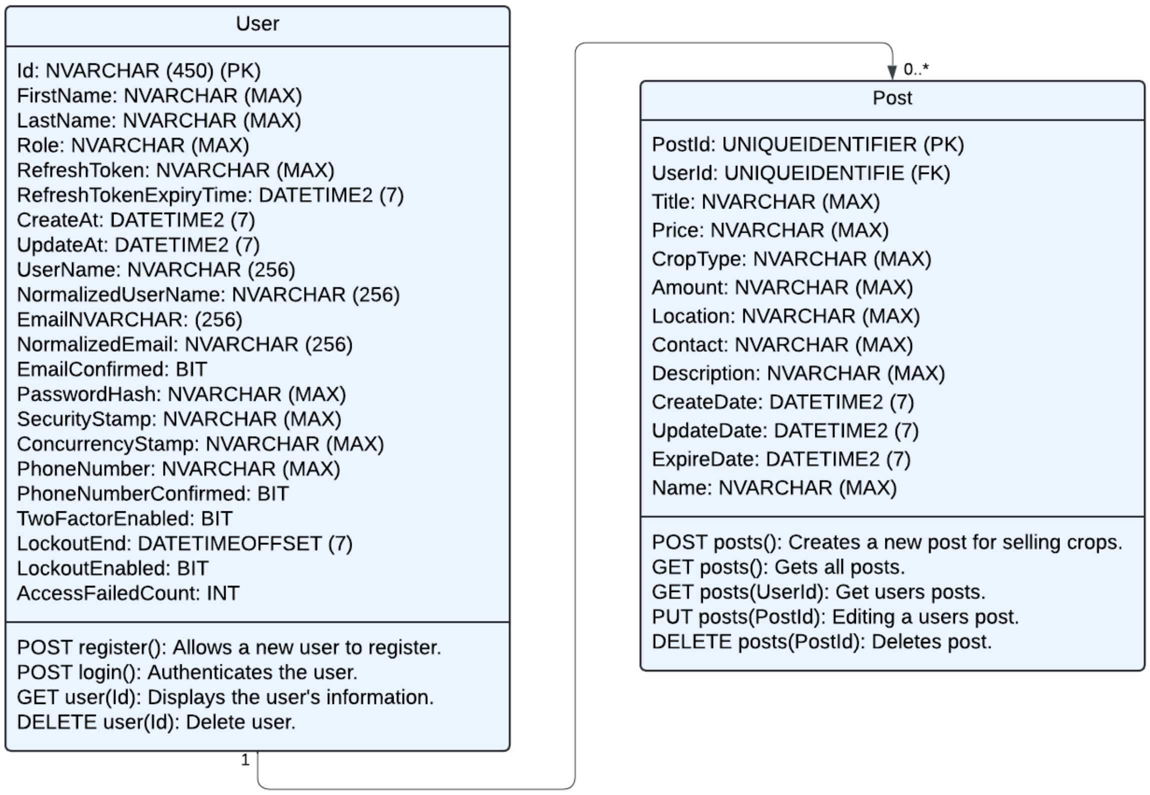

User
Id: NVARCHAR (450) (PK)
FirstName: NVARCHAR (MAX)
LastName: NVARCHAR (MAX)
Role: NVARCHAR (MAX)
RefreshToken: NVARCHAR (MAX)
RefreshTokenExpiryTime: DATETIME2 (7)
CreateAt: DATETIME2 (7)
UpdateAt: DATETIME2 (7)
UserName: NVARCHAR (256)
NormalizedUserName: NVARCHAR (256)
EmailNVARCHAR: (256)
NormalizedEmail: NVARCHAR (256)
EmailConfirmed: BIT
PasswordHash: NVARCHAR (MAX)
SecurityStamp: NVARCHAR (MAX)
ConcurrencyStamp: NVARCHAR (MAX)
PhoneNumber: NVARCHAR (MAX)
PhoneNumberConfirmed: BIT
TwoFactorEnabled: BIT
LockoutEnd: DATETIMEOFFSET (7)
LockoutEnabled: BIT
AccessFailedCount: INT
POST register(): Allows a new user to register.
POST login(): Authenticates the user.
GET user(Id): Displays the user's information.
DELETE user(Id): Delete user.
1
0..*
Post
PostId: UNIQUEIDENTIFIER (PK)
UserId: UNIQUEIDENTIFIE (FK)
Title: NVARCHAR (MAX)
Price: NVARCHAR (MAX)
CropType: NVARCHAR (MAX)
Amount: NVARCHAR (MAX)
Location: NVARCHAR (MAX)
Contact: NVARCHAR (MAX)
Description: NVARCHAR (MAX)
CreateDate: DATETIME2 (7)
UpdateDate: DATETIME2 (7)
ExpireDate: DATETIME2 (7)
Name: NVARCHAR (MAX)
POST posts(): Creates a new post for selling crops.
GET posts(): Gets all posts.
GET posts(UserId): Get users posts.
PUT posts(PostId): Editing a users post.
DELETE posts(PostId): Deletes post.

Fig. 4 (Class Diagram)

This class diagram represents a crop-selling platform with two main classes: **User** and **Post**. The **User** class manages authentication details, personal information, and security settings, while the **Post** class handles crop sale listings with attributes like title, price, crop type, location, and contact details. A **one-to-many** relationship exists, meaning a single user can create multiple posts. The diagram also defines API endpoints for user management (register, login, retrieve, delete) and post operations (create, fetch, update, delete), ensuring a structured and efficient system for listing and managing crop sales.

11 User Interface Design

11.1 Rough Draft Mockup

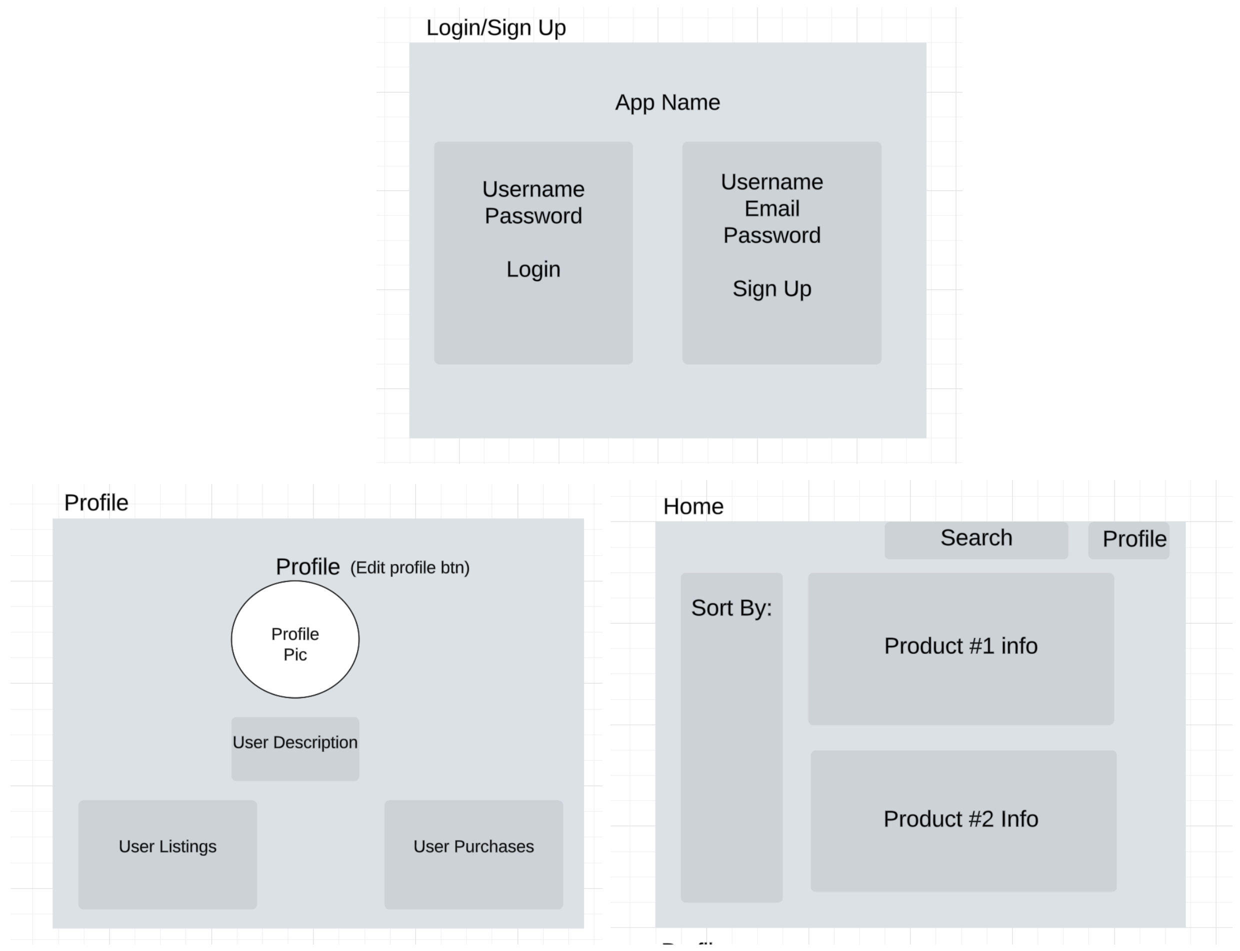


Fig. 5 (Rough Draft Mockup)

Our initial mockup was a hand drawn sketch that covered the basics; a screen to login/sign up, view listings, and view user profile. Using this basic design we digitized our first wire frame and proceeded to create a more detailed daft in Figma that would cover our most important elements.

11.2 Figma Design

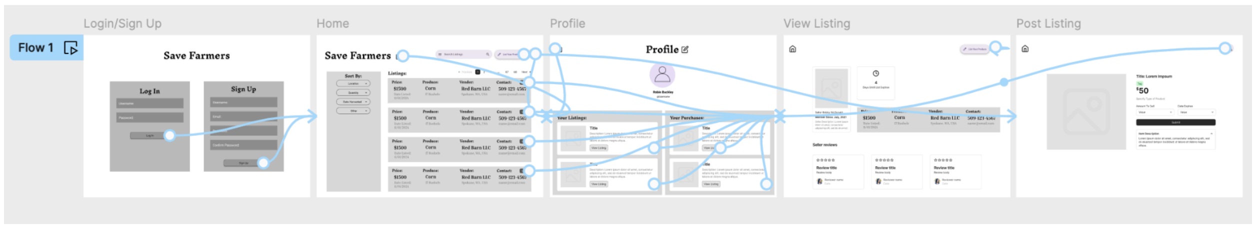


Fig. 6 (Figma Design)

To view the original file you can click here. In addition to the original screens of login/sign up, view listings, and view user profile, we also added a post listing and view listing screen.

11.3 Initial UI Features

The **Login/Sign Up** screen allows users to either log in with a username and password or sign up with additional fields like email and password confirmation. The **Log In** button directs users to the **Home** screen upon a successful login, as does the **Sign Up** button.

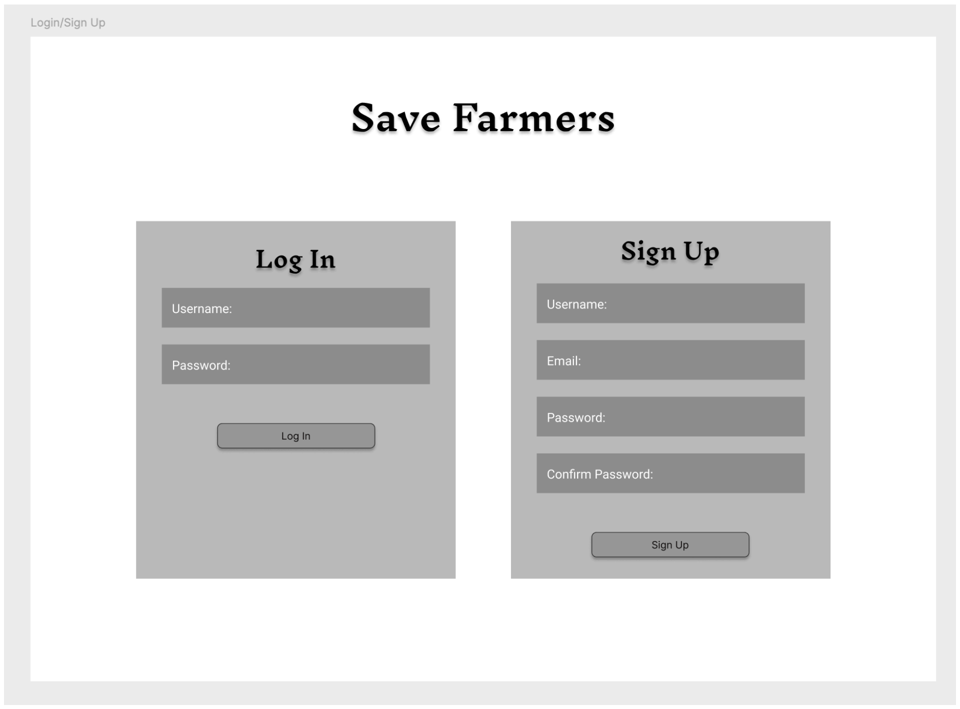
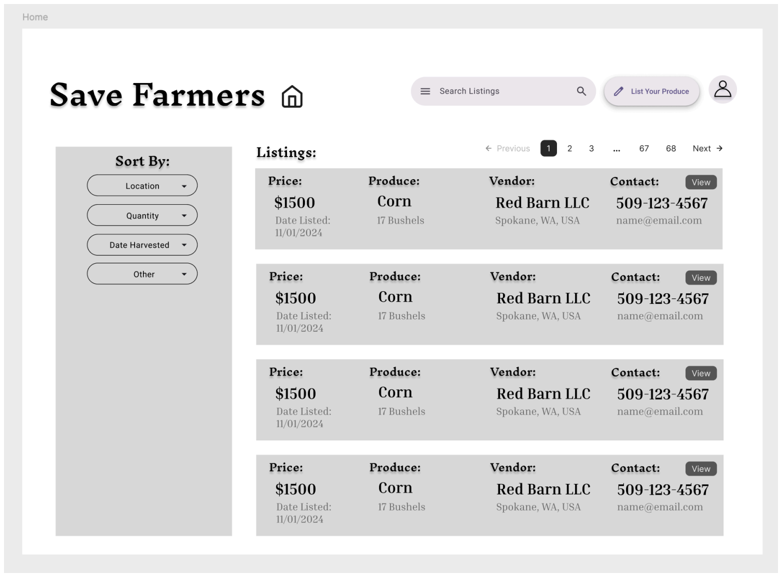


Fig. 7 (Figma Login/Sign Up)

The **Home** screen lists cards with details such as price, product type, vendor name, location, and contact info, along with options to sort listings by location, quantity, or date harvested. Users can also search listings using a search bar. Each listing card has a clickable button that takes users to the **View Listing** screen for the selected item.



Price: $1500 Date Listed: 11/01/2024 | Produce: Corn 17 Bushels | Vendor: Red Barn LLC Spokane, WA, USA | Contact: 509-123-4567 name@email.com

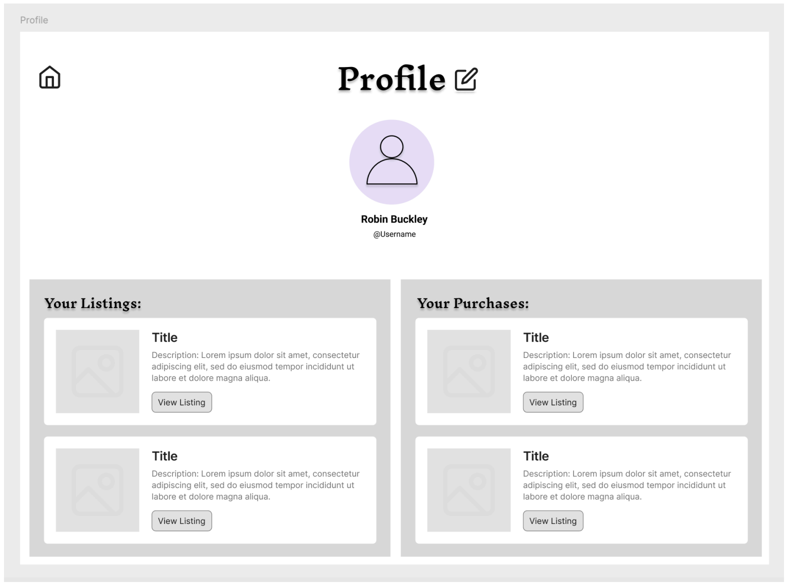
Price: $1500 Date Listed: 11/01/2024 | Produce: Corn 17 Bushels | Vendor: Red Barn LLC Spokane, WA, USA | Contact: 509-123-4567 name@email.com

Price: $1500 Date Listed: 11/01/2024 | Produce: Corn 17 Bushels | Vendor: Red Barn LLC Spokane, WA, USA | Contact: 509-123-4567 name@email.com

Price: $1500 Date Listed: 11/01/2024 | Produce: Corn 17 Bushels | Vendor: Red Barn LLC Spokane, WA, USA | Contact: 509-123-4567 name@email.com

Fig. 8 (Figma Home)

In the **Profile** screen, users see their profile picture, name, and email, along with tabs for "Your Purchases" and "Your Listings," where they can manage items they've bought or listed. The **Edit Profile** button allows users to update their information within the Profile screen. In the **Your Purchases** and **Your Listings** sections, each item includes a **View Listing** button that directs users to the **View Listing** screen for that specific product. Additionally, there is a **Post New Listing** button that takes users to the **Post Listing** screen to create a new listing.



Description: Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua.

Fig. 9 (Figma Profile)

The **View Listing** screen provides a detailed look at a product with an image, title, price, quantity, stock info, vendor location, and expiration date, as well as product descriptions and seller reviews. There is a **Back** button that returns users to the **Home** screen, as well as a profile button that redirects users to the **Profile** screen.

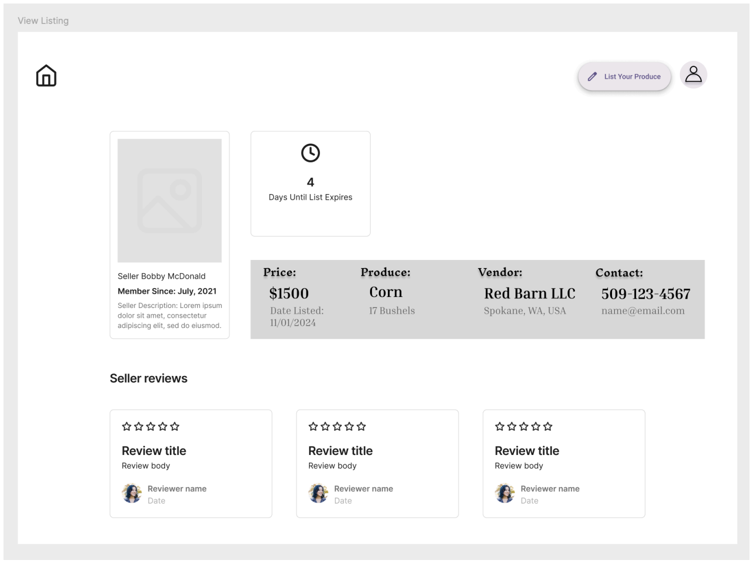


Fig. 10 (Figma View Listing)

The **Post Listing** screen allows users to create a new listing by entering a title, price, quantity, description, and uploading an image. Users can complete the listing process by clicking the **Submit** button, which publishes their listing and redirects them back to the Profile screen under "Your Listings" so they can view their newly created post.

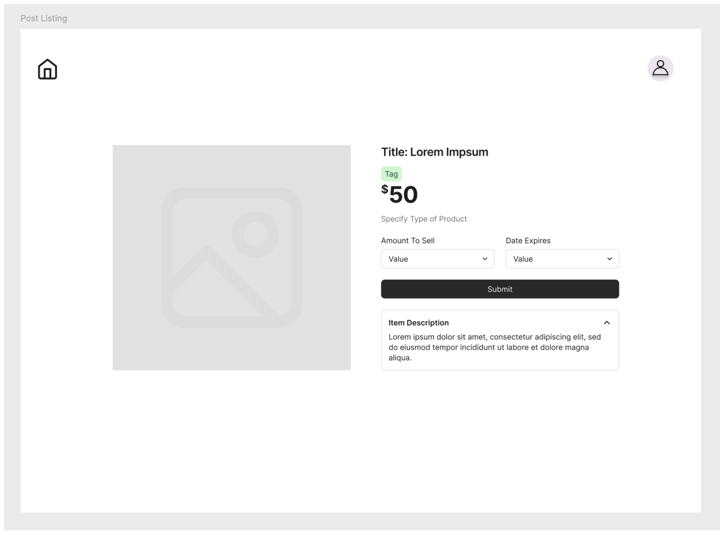

Post Listing
Title: Lorem Impsum
Tag
$50
Specify Type of Product
Amount To Sell
Value
Date Expires
Value
Submit
Item Description
Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua.

Fig. 11 (Figma Post Listing)

12 Data Design

12.1 Data Model

Entities:

- **User:** Represents an individual registered in the system.
- **Post:** Represents a posting of goods (like crops) created by users.

Relationships:

- **User to Post:** One-to-Many.

12.2 Database Schema (ER Diagram)

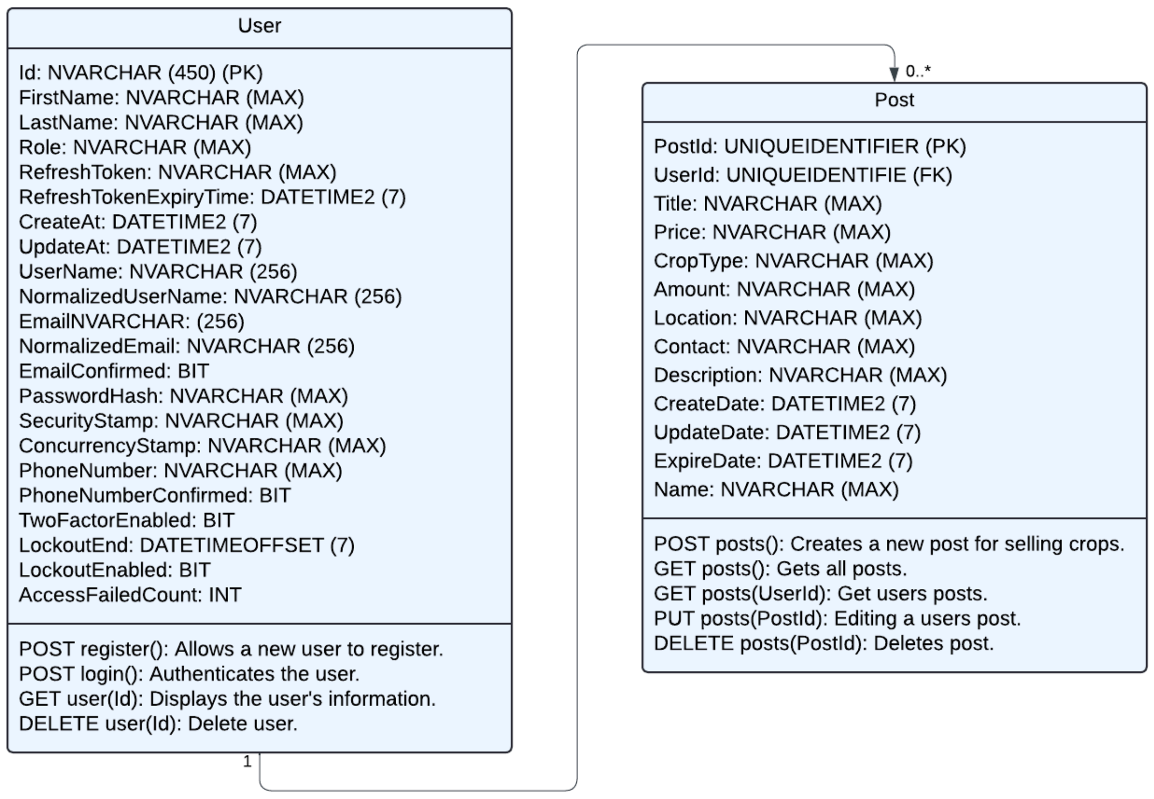

User
Id: NVARCHAR (450) (PK)
FirstName: NVARCHAR (MAX)
LastName: NVARCHAR (MAX)
Role: NVARCHAR (MAX)
RefreshToken: NVARCHAR (MAX)
RefreshTokenExpiryTime: DATETIME2 (7)
CreateAt: DATETIME2 (7)
UpdateAt: DATETIME2 (7)
UserName: NVARCHAR (256)
NormalizedUserName: NVARCHAR (256)
EmailNVARCHAR: (256)
NormalizedEmail: NVARCHAR (256)
EmailConfirmed: BIT
PasswordHash: NVARCHAR (MAX)
SecurityStamp: NVARCHAR (MAX)
ConcurrencyStamp: NVARCHAR (MAX)
PhoneNumber: NVARCHAR (MAX)
PhoneNumberConfirmed: BIT
TwoFactorEnabled: BIT
LockoutEnd: DATETIMEOFFSET (7)
LockoutEnabled: BIT
AccessFailedCount: INT
POST register(): Allows a new user to register.
POST login(): Authenticates the user.
GET user(Id): Displays the user's information.
DELETE user(Id): Delete user.
1
0..*
Post
PostId: UNIQUEIDENTIFIER (PK)
UserId: UNIQUEIDENTIFIE (FK)
Title: NVARCHAR (MAX)
Price: NVARCHAR (MAX)
CropType: NVARCHAR (MAX)
Amount: NVARCHAR (MAX)
Location: NVARCHAR (MAX)
Contact: NVARCHAR (MAX)
Description: NVARCHAR (MAX)
CreateDate: DATETIME2 (7)
UpdateDate: DATETIME2 (7)
ExpireDate: DATETIME2 (7)
Name: NVARCHAR (MAX)
POST posts(): Creates a new post for selling crops.
GET posts(): Gets all posts.
GET posts(UserId): Get users posts.
PUT posts(PostId): Editing a users post.
DELETE posts(PostId): Deletes post.

Fig. 12 (ER Diagram)

12.3 Sample Data (SQL Inserts)

Sample Data for **User** Table:

INSERT INTO [dbo].[AspNetUsers] ([Id], [FirstName], [LastName], [Role], [RefreshToken], [RefreshTokenExpiryTime], [CreateAt], [UpdateAt], [UserName], [NormalizedUserName], [Email], [NormalizedEmail], [EmailConfirmed], [PasswordHash], [SecurityStamp], [ConcurrencyStamp], [PhoneNumber], [PhoneNumberConfirmed], [TwoFactorEnabled], [LockoutEnd], [LockoutEnabled], [AccessFailedCount]) VALUES (N'16b23b37-54e3-4169-9058-c2ae475641ce', N'd', N'p', N'', N'QkUXbqbIBNEmBN6sW9ZYOn2Maxk8vKkKOeLybxU4bZs=', N'2025-03-14 00:00:11', N'2025-02-11 23:30:01', N'2025-02-12 00:00:11', N'dp', N'DP', N'dp@gmail.com', N'DP@GMAIL.COM', 0, N'AQAAAAIAAYagAAAAEPTlg7WCzJdUNpszA8aUHnliDVfskmXiqIV6IGdF6YmBz4Jf08TpGi8Z+DsUuLhQQA==', N'ME2GGDGIISPDQ4P5RK2RRCUDQY7IC5B5', N'b26f2b20-e3f0-4752-a0ed-a1900ba7f8c4', NULL, 0, 0, NULL, 1, 0)

<u>Sample Data for **Post** Table:</u>
INSERT INTO [dbo].[Posts] ([PostId], [Title], [Price], [CropType], [Amount], [Location], [Contact], [Description], [CreateDate], [UpdateDate], [ExpireDate], [Name], [UserId]) VALUES (N'e6a3d978-ece0-4a1d-a937-af922c38e68a', N'asdasdasdas', N'string', N'string', N'string', N'string', N'string', N'string', N'2025-02-05 08:40:35', N'2025-02-05 08:40:35', N'2025-02-05 08:40:14', N'string', N'3fa85f64-5717-4562-b3fc-2c963f66afa6')

12.4 Database Justification

We are using Microsoft ASP.Net Core with Swagger API tools for quick testing and visualization.

For this project, we originally went with SQLite for its simplicity and quick building. This worked for the posting aspect of the project, but when it came to making a login and connecting the website, we realized that making all of it would be very time-consuming with SQLite. SQLite was good for a quick start, but for the size of our project, we wanted to use something appropriate and give us the tools that would aid in scaling. Now, we have fully switched to using ASP.net. Also, Microsoft ASP.Net Core has a very simple way to connect itself to Azure, so possibly in the future, we could upload all this to work in the cloud.

This was not an easy switch. Writing the backend for the database was not simple and took quite a bit of time. Now that we have the backend, we have to attach it to the frontend that we already have. This part is simple but has its quarks. 2 out of the 3 team members use Macs as their primary machines, but ASP.net doesn’t work on Macs. So they are using the home desktops to work on the project. Setting up the project and the switch took some time to get working. But now, we are working on the connections to the API and fixing the problems along the way.

13 Prototype Development

13.1 Version 1.0 Implemented Functionalities

We have implemented the Login, Sign up, Profile, and Home pages. These pages do not have a backend. We have a working local database using Sqlite. This database is setup to work in Ionic. On the database branch in Github on the Profile page, there are 3 field inputs that will take those inputs, store them in the database, and output them on the same page. This database is for testing that it is working.

13.2 Login/Sign Up Pages:

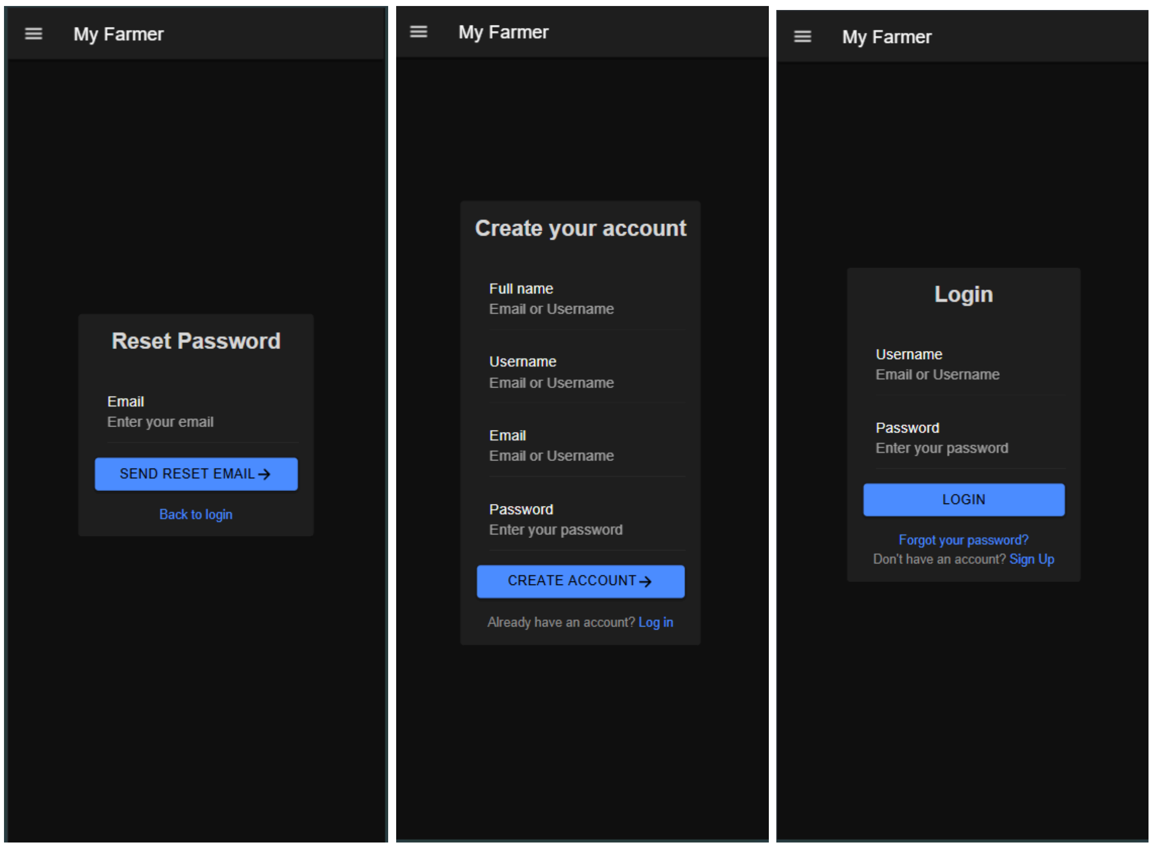
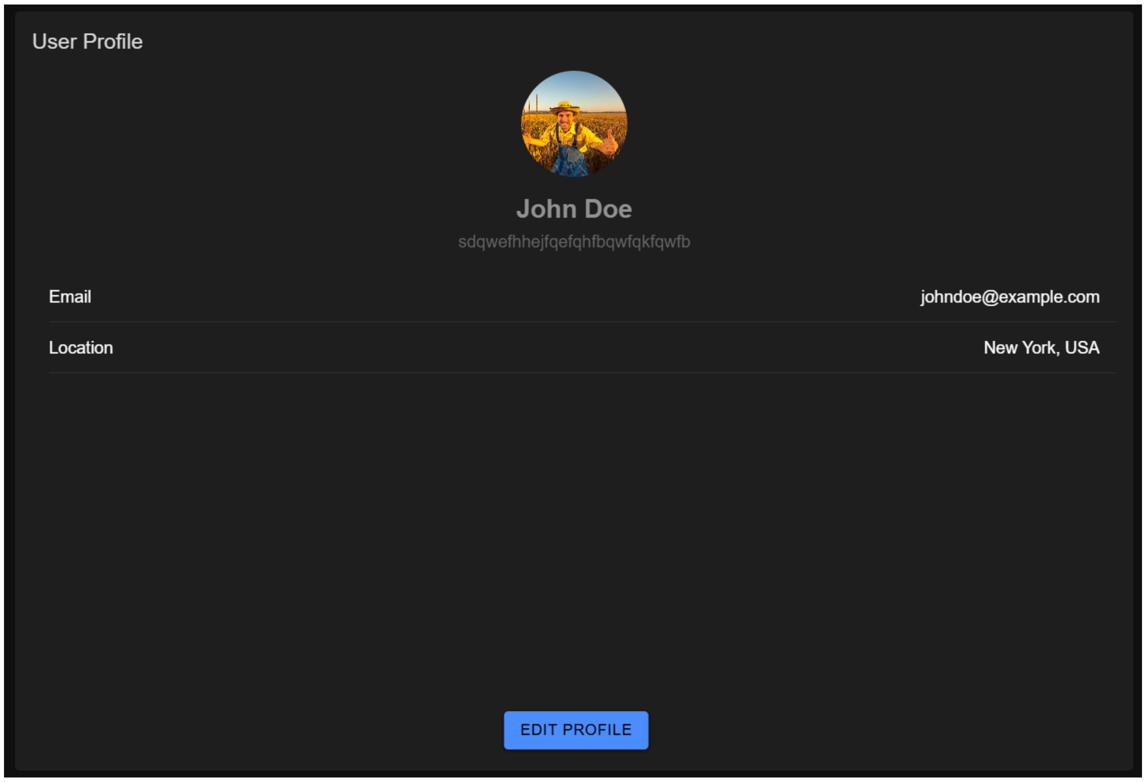


Fig. 13 (Login/Sign Up Pages)

Using the Ionic framework we made these pages. They do not reflect the full design that we are going for but are enough to help other parts of the app be created.

13.3 Profile Page:


User Profile

John Doe

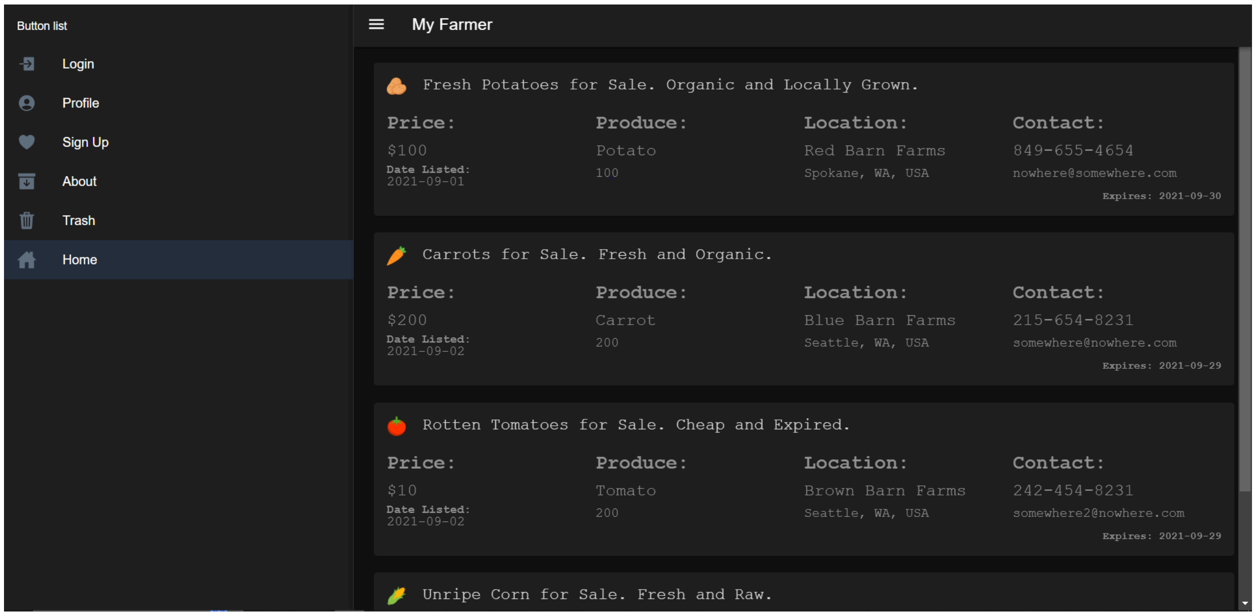
sdqwefhhejfqefqhfbqwfqkfqwfb

Email	johndoe@example.com
Location	New York, USA

Fig. 14 (Profile Page)

Using the Ionic framework we made this page. It does not reflect the full design that we are going for but is enough to help other parts of the app be created.

13.4 Home Page/User Posts Page:



Fresh Potatoes for Sale. Organic and Locally Grown.

Price:	Produce:	Location:	Contact:
$100	Potato	Red Barn Farms	849-655-4654
Date Listed: 2021-09-01	100	Spokane, WA, USA	nowhere@somewhere.com
			Expires: 2021-09-30

Carrots for Sale. Fresh and Organic.

Price:	Produce:	Location:	Contact:
$200	Carrot	Blue Barn Farms	215-654-8231
Date Listed: 2021-09-02	200	Seattle, WA, USA	somewhere@nowhere.com
			Expires: 2021-09-29

Rotten Tomatoes for Sale. Cheap and Expired.

Price:	Produce:	Location:	Contact:
$10	Tomato	Brown Barn Farms	242-454-8231
Date Listed: 2021-09-02	200	Seattle, WA, USA	somewhere2@nowhere.com
			Expires: 2021-09-29

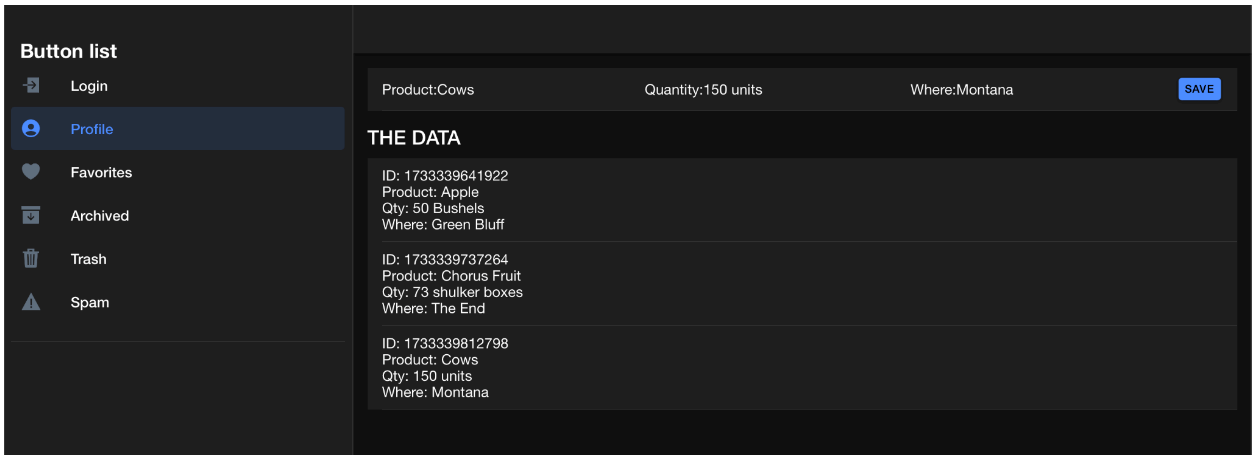
Unripe Corn for Sale. Fresh and Raw.

Fig. 15 (Home Page/User Posts Page)

This page is where we put most of our time into. We designed the posts to be easily readable and convey all the information needed. This design is the direction that we are heading into.

13.5 Sqlite Database in Ionic (DEPRECATED):

Currently, we have moved away from SQLite and built the backend on ASP.NET with an SQL database. This section just shows what we had in our first prototype.



Product:Cows Quantity:150 units Where:Montana

THE DATA

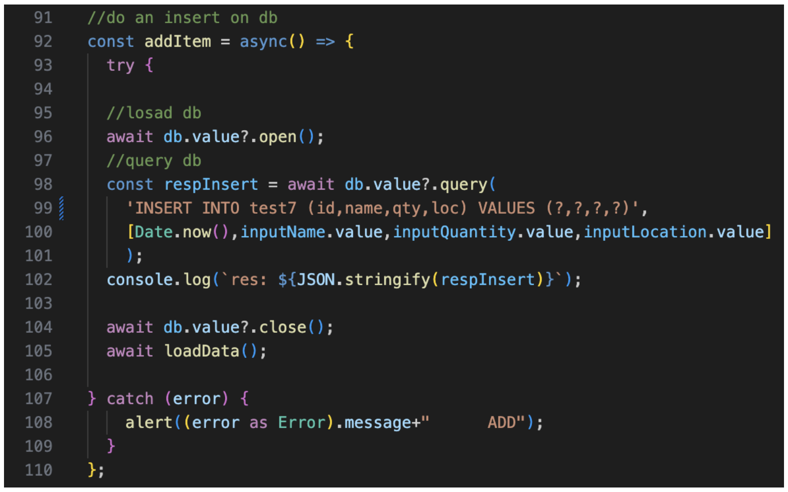
ID: 1733339641922
Product: Apple
Qty: 50 Bushels
Where: Green Bluff

ID: 1733339737264
Product: Chorus Fruit
Qty: 73 shulker boxes
Where: The End

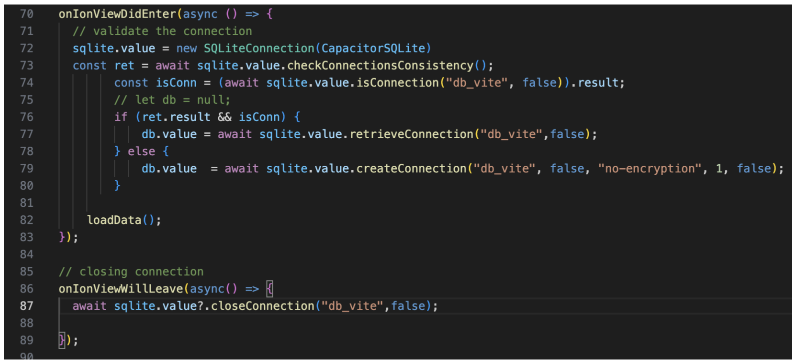
ID: 1733339812798
Product: Cows
Qty: 150 units
Where: Montana

Fig. 16 (Mock Database UI)

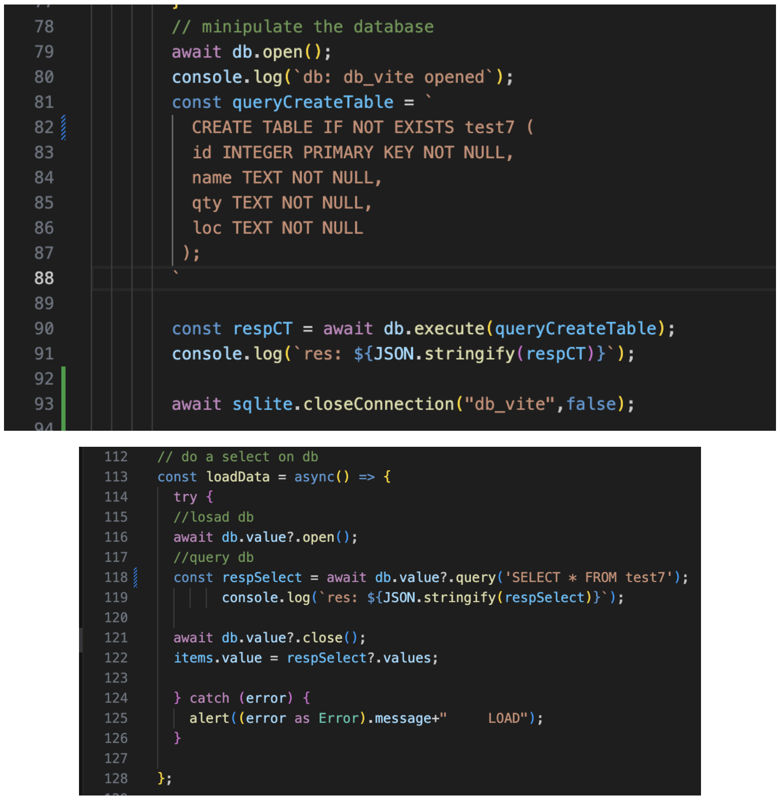
This is the current mock database input and display that we have set up for testing the functionality of the database.



```
//do an insert on db
const addItem = async() => {
  try {

  //losad db
  await db.value?.open();
  //query db
  const respInsert = await db.value?.query(
    'INSERT INTO test7 (id,name,qty,loc) VALUES (?,?,?,?)',
    [Date.now(),inputName.value,inputQuantity.value,inputLocation.value]
    );
  console.log(`res: ${JSON.stringify(respInsert)}`);

  await db.value?.close();
  await loadData();

  } catch (error) {
    alert((error as Error).message+"      ADD");
  }
};
```



```
onIonViewDidEnter(async () => {
  // validate the connection
  sqlite.value = new SQLiteConnection(CapacitorSQLite)
  const ret = await sqlite.value.checkConnectionsConsistency();
        const isConn = (await sqlite.value.isConnection("db_vite", false)).result;
        // let db = null;
        if (ret.result && isConn) {
            db.value = await sqlite.value.retrieveConnection("db_vite",false);
        } else {
            db.value  = await sqlite.value.createConnection("db_vite", false, "no-encryption", 1, false);
        }

    loadData();
});

// closing connection
onIonViewWillLeave(async() => {
  await sqlite.value?.closeConnection("db_vite",false);

});
```



```
    // minipulate the database
    await db.open();
    console.log(`db: db_vite opened`);
    const queryCreateTable = `
      CREATE TABLE IF NOT EXISTS test7 (
      id INTEGER PRIMARY KEY NOT NULL,
      name TEXT NOT NULL,
      qty TEXT NOT NULL,
      loc TEXT NOT NULL
     );
    `

    const respCT = await db.execute(queryCreateTable);
    console.log(`res: ${JSON.stringify(respCT)}`);

    await sqlite.closeConnection("db_vite",false);
```

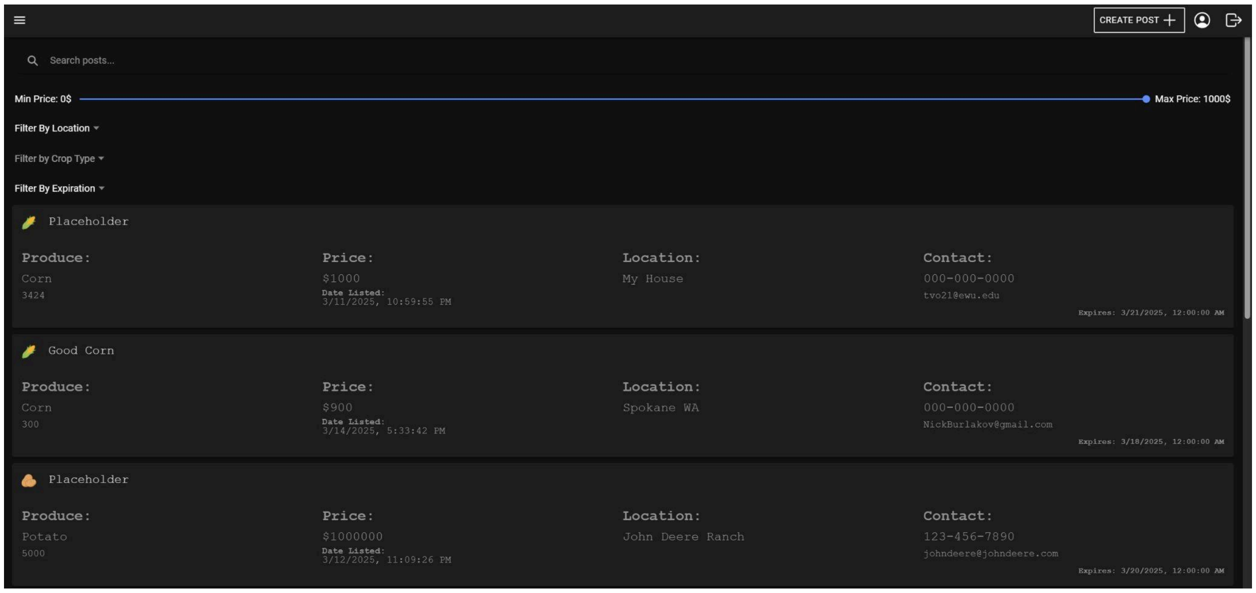
```
// do a select on db
const loadData = async() => {
  try {
  //losad db
  await db.value?.open();
  //query db
  const respSelect = await db.value?.query('SELECT * FROM test7');
        console.log(`res: ${JSON.stringify(respSelect)}`);

  await db.value?.close();
  items.value = respSelect?.values;

  } catch (error) {
    alert((error as Error).message+"     LOAD");
  }

};
```

Fig. 17 (Database Functions)

These are the functions for, loading, validating, inserting, and creating the database. There is more code that goes into the setup of the database. The database is Sqlite using Capacitor in Ionic.

13.6 Version 2.0 (Current) Implemented Functionalities

Currently, we have implemented the Login, Sign up, Profile, Admin, Create Post, Edit Post, and Home pages. All the requirements are met except one, the user rating system. These pages are connected to the API hosted on Azure with our backend. We are no longer using Sqlite in this version, we are using ASP.NET with a SQL database.



Min Price: 0$ Max Price: 1000$

Filter By Location

Filter by Crop Type

Filter By Expiration

Placeholder

Produce: Corn 3424

Price: $1000 Date Listed: 3/11/2025, 10:59:55 PM

Location: My House

Contact: 000-000-0000 tvo21@ewu.edu

Expires: 3/21/2025, 12:00:00 AM

Good Corn

Produce: Corn 300

Price: $900 Date Listed: 3/14/2025, 5:33:42 PM

Location: Spokane WA

Contact: 000-000-0000 NickBurlakov@gmail.com

Expires: 3/18/2025, 12:00:00 AM

Placeholder

Produce: Potato 5000

Price: $1000000 Date Listed: 3/12/2025, 11:09:26 PM

Location: John Deere Ranch

Contact: 123-456-7890 johndeere@johndeere.com

Expires: 3/20/2025, 12:00:00 AM

Fig. 18 (Home Page v2.0)

On this page, the user will be able to see their active posts and others' posts that are active. Here, everyone will be able to contact the sellers by their post information. You are able to search through the posts using the search bar and/or the filters.

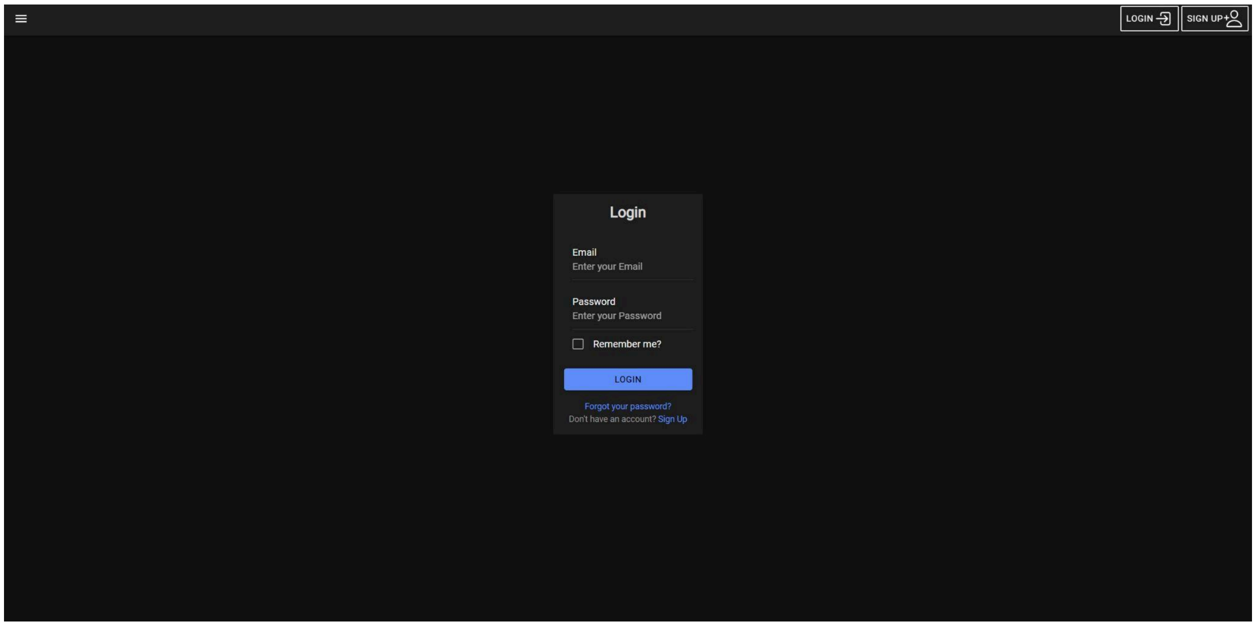
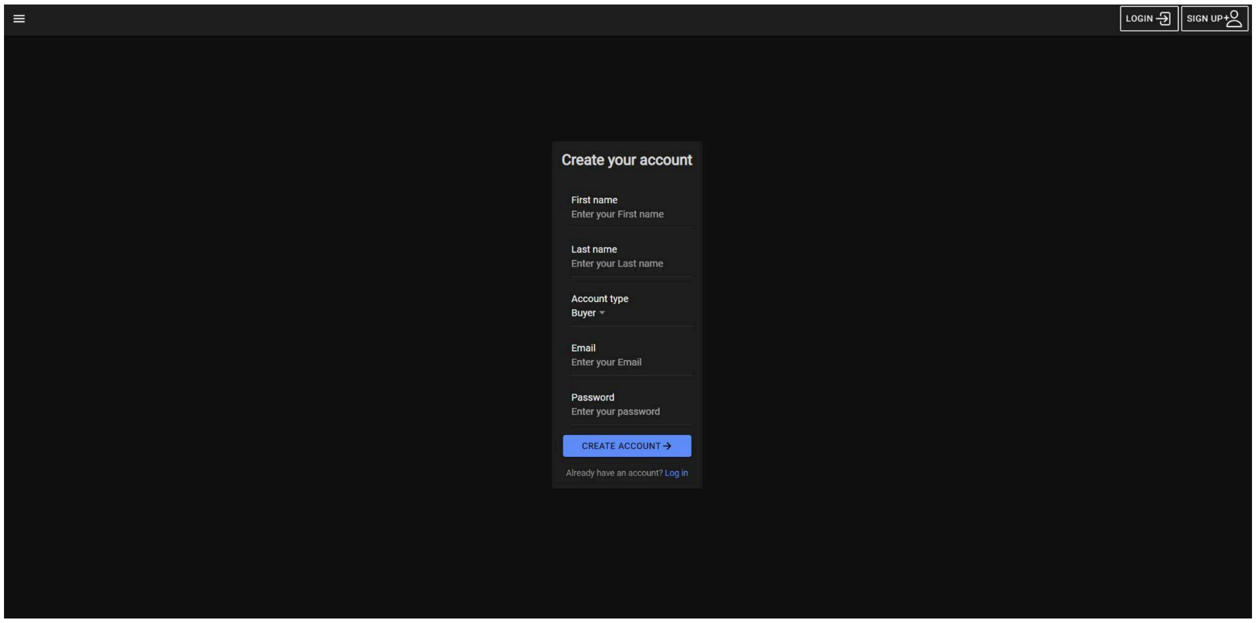


Fig. 19 (Login Page v2.0)

On this page, you can log in after making an account or use a previously created account.



Create your account

First name
Enter your First name

Last name
Enter your Last name

Account type
Buyer

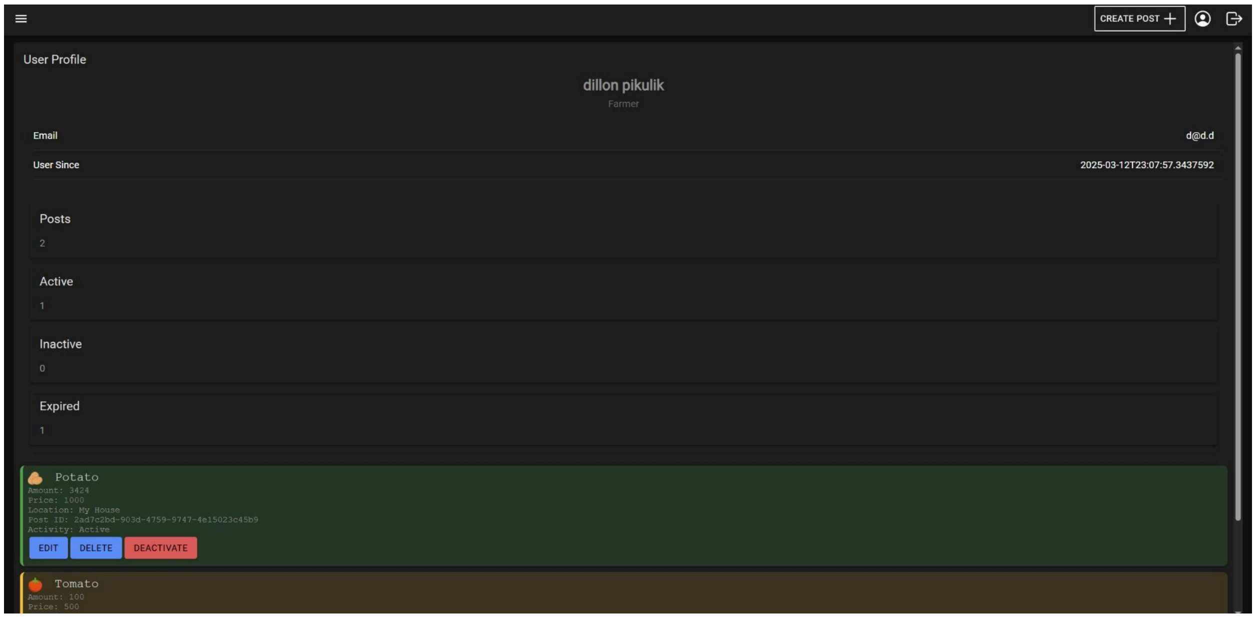
Email
Enter your Email

Password
Enter your password

Already have an account? Log in

Fig. 20 (Sign Up Page v2.0)

On the signup page, you can make an account. Use a correctly formatted email and a password that is 9 characters long and consists of letters and numbers.



User Profile

dillon pikulik
Farmer

Email d@d.d
User Since 2025-03-12T23:07:57.3437592

Posts
2

Active
1

Inactive
0

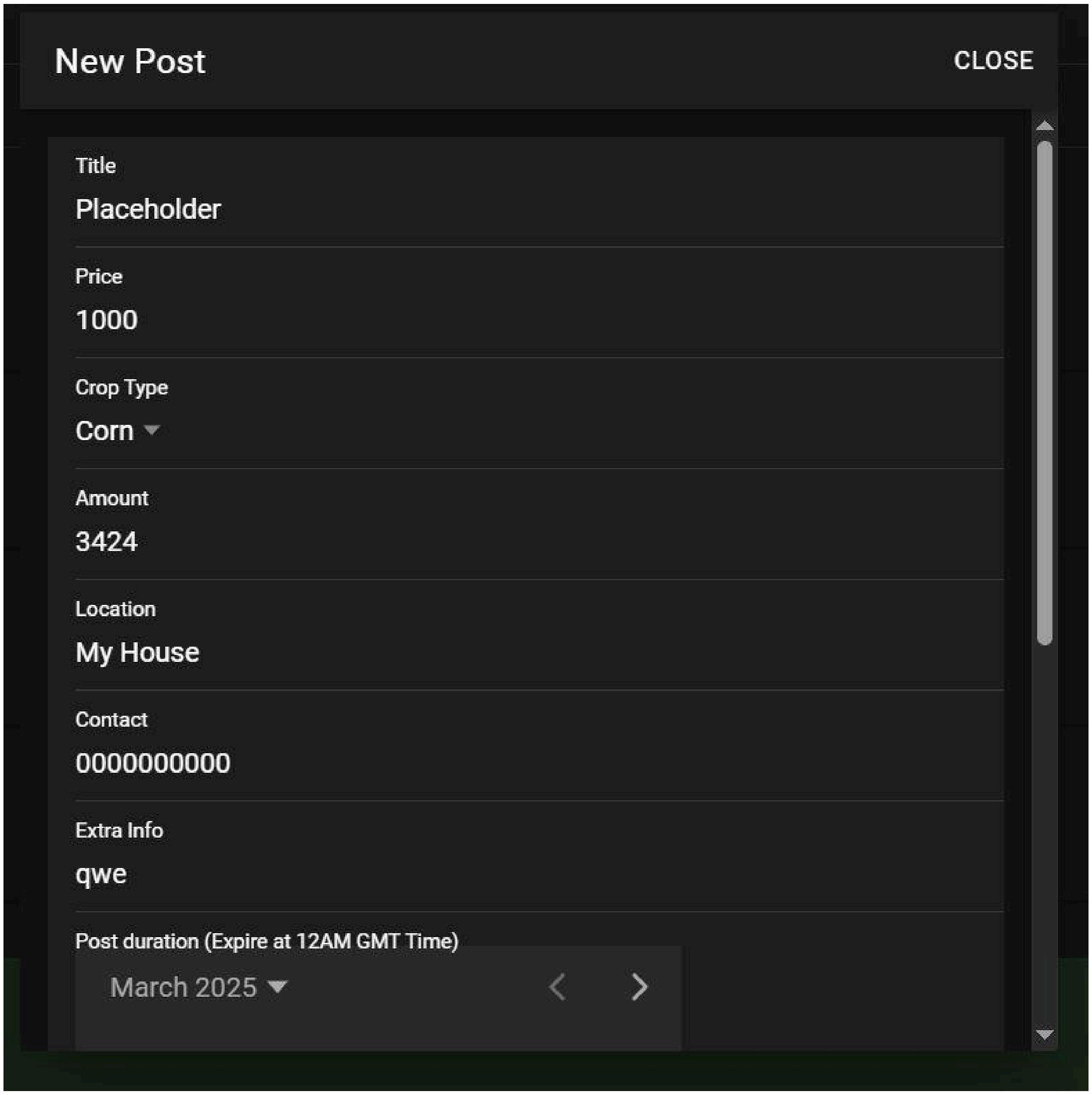
Expired
1

```
Potato
Amount: 3424
Price: 1000
Location: My House
Post ID: 2ad7c2bd-903d-4759-9747-4e15023c45b9
Activity: Active
```

```
Tomato
Amount: 100
Price: 500
```

Fig. 21 (Profile Page v2.0)

On the profile page, you can see your name, role, email, and the date you made your account. Most importantly, you can see your posts and edit, delete, and/or activate/deactivate them.



Title
Placeholder

Price
1000

Crop Type
Corn

Amount
3424

Location
My House

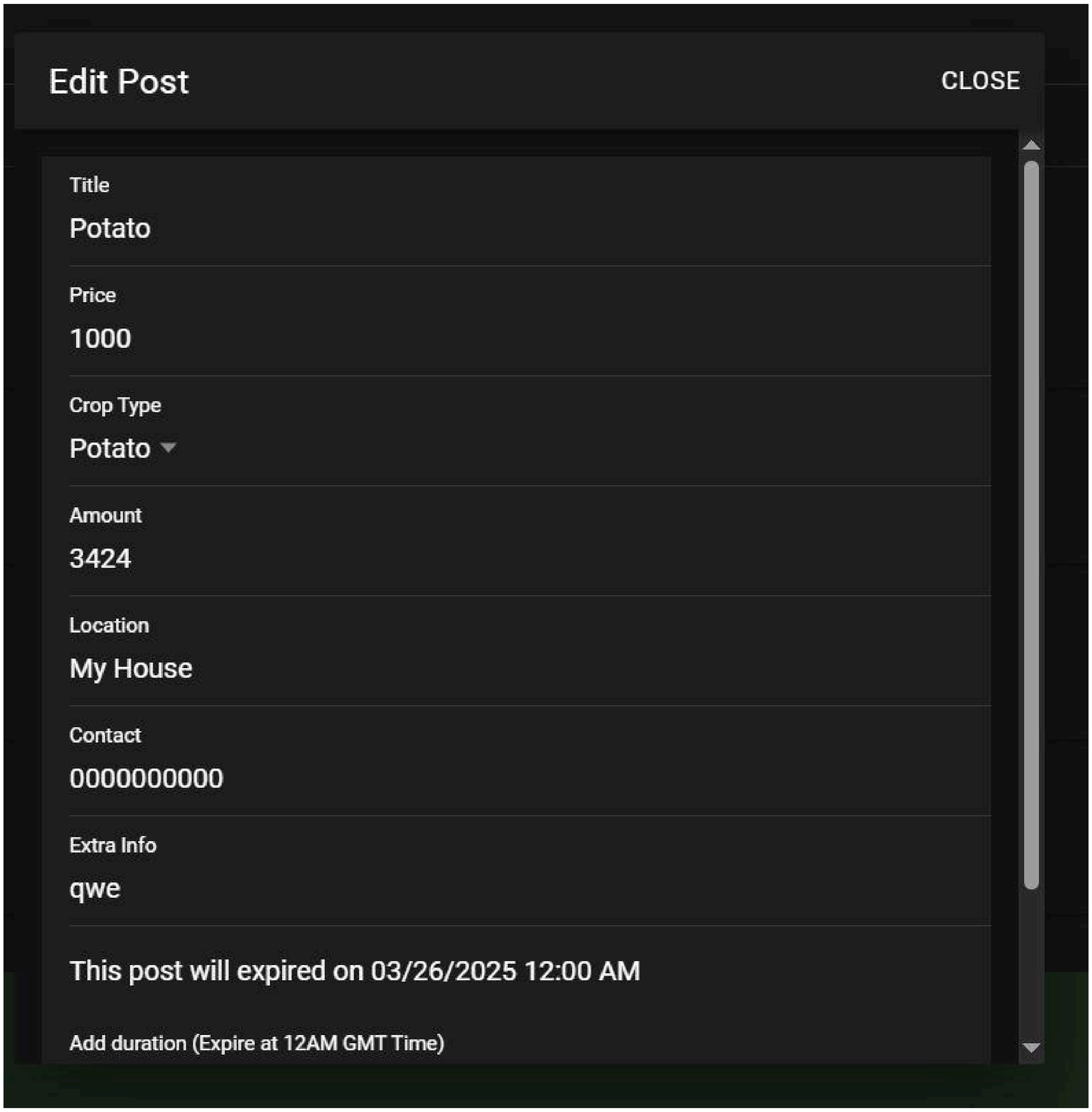
Contact
0000000000

Extra Info
qwe

Post duration (Expire at 12AM GMT Time)
March 2025

Fig. 22 (Create Post Window v2.0)

On the create post window, you can create a post by filling in the information about it.



Edit Post

CLOSE

Title
Potato

Price
1000

Crop Type
Potato

Amount
3424

Location
My House

Contact
0000000000

Extra Info
qwe

This post will expired on 03/26/2025 12:00 AM

Add duration (Expire at 12AM GMT Time)

Fig. 23 (Edit Post Window v2.0)

On the edit post window, you can edit the post that you created from your profile page.

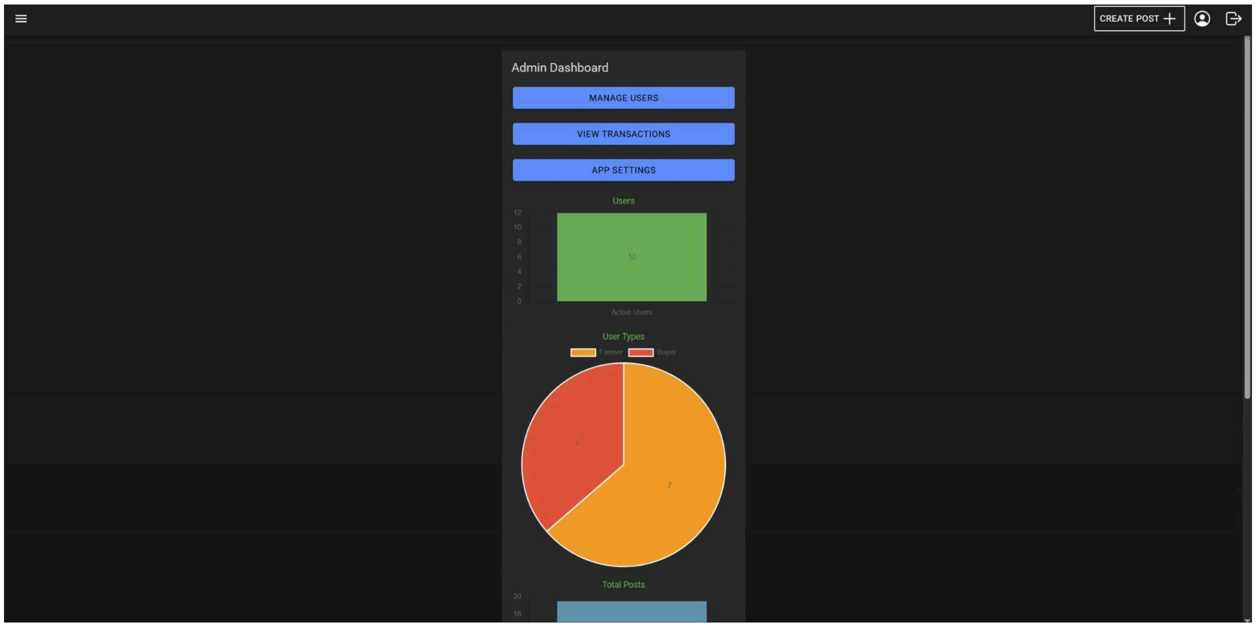
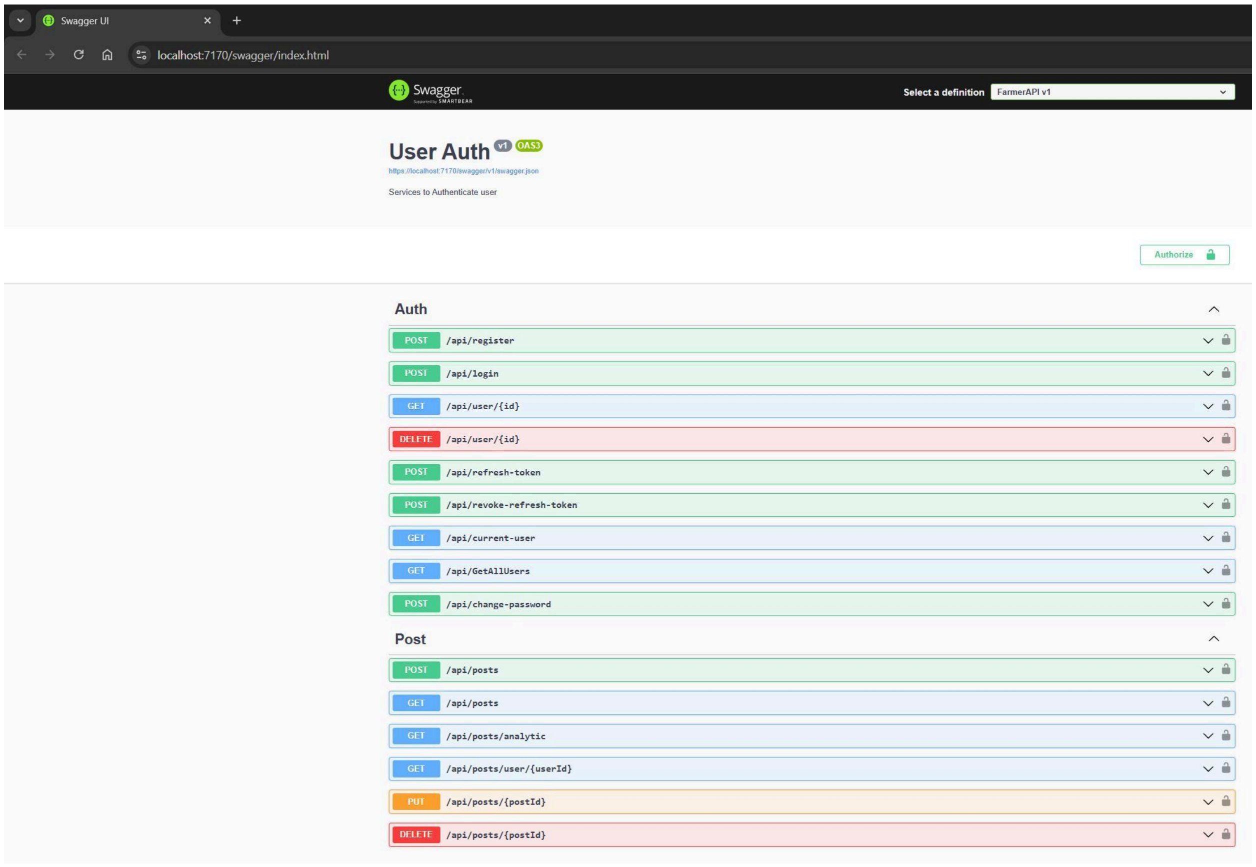


Fig. 24 (Admin Page v2.0)

Only the admin has access to the admin page and can see analytics about the app. Here, the admin can also delete users.



User Auth v1 OAS3

https://localhost:7170/swagger/v1/swagger.json

Services to Authenticate user

Auth

```
POST    /api/register
POST    /api/login
GET     /api/user/{id}
DELETE  /api/user/{id}
POST    /api/refresh-token
POST    /api/revoke-refresh-token
GET     /api/current-user
GET     /api/GetAllUsers
POST    /api/change-password
```

Post

```
POST    /api/posts
GET     /api/posts
GET     /api/posts/analytic
GET     /api/posts/user/{userId}
PUT     /api/posts/{postId}
DELETE  /api/posts/{postId}
```

Fig. 25 (Swagger View of the API Backend)

This is our API with all of the calls that we can make to access the information that we need. This API is made using ASP.NET. All endpoints are encrypted, and only valid users can access the information.

14 Testing and Validation

14.1 Test Plan

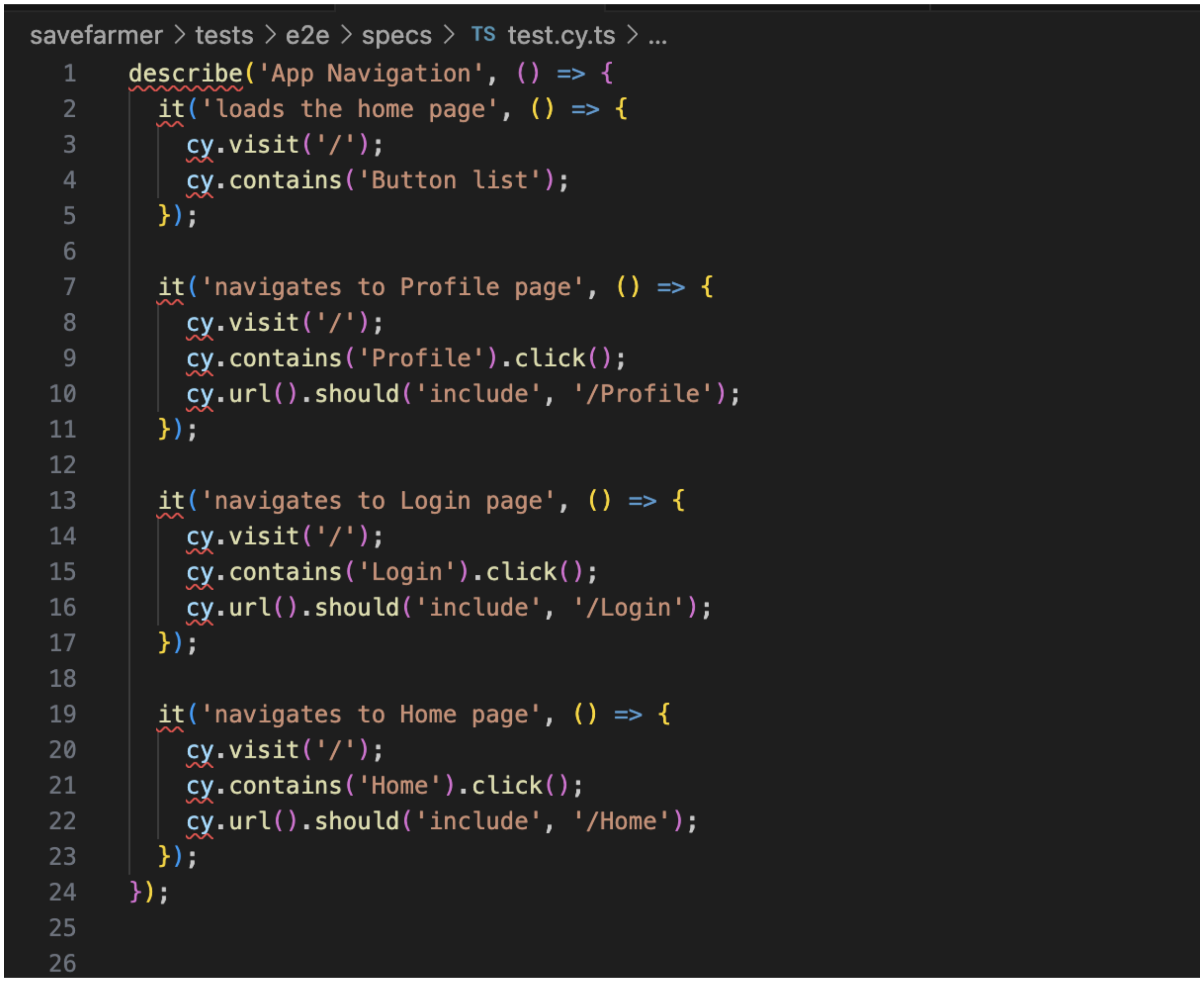
Our testing methodology focused on verifying the core functionalities of the app through end-to-end testing via Cypress. We determined that this early into development, the key features that needed testing were navigation and page rendering.

Currently, due to issues in the deployment of apps and Azure, we have removed all testing from the app. If we have the time in the future, we will figure the issue out, add the test back, and write more.

14.2 Test Cases and Results (DEPRECATED)

NOTE: This section is **deprecated** because of a compatibility issue with our **ASP Backend**.

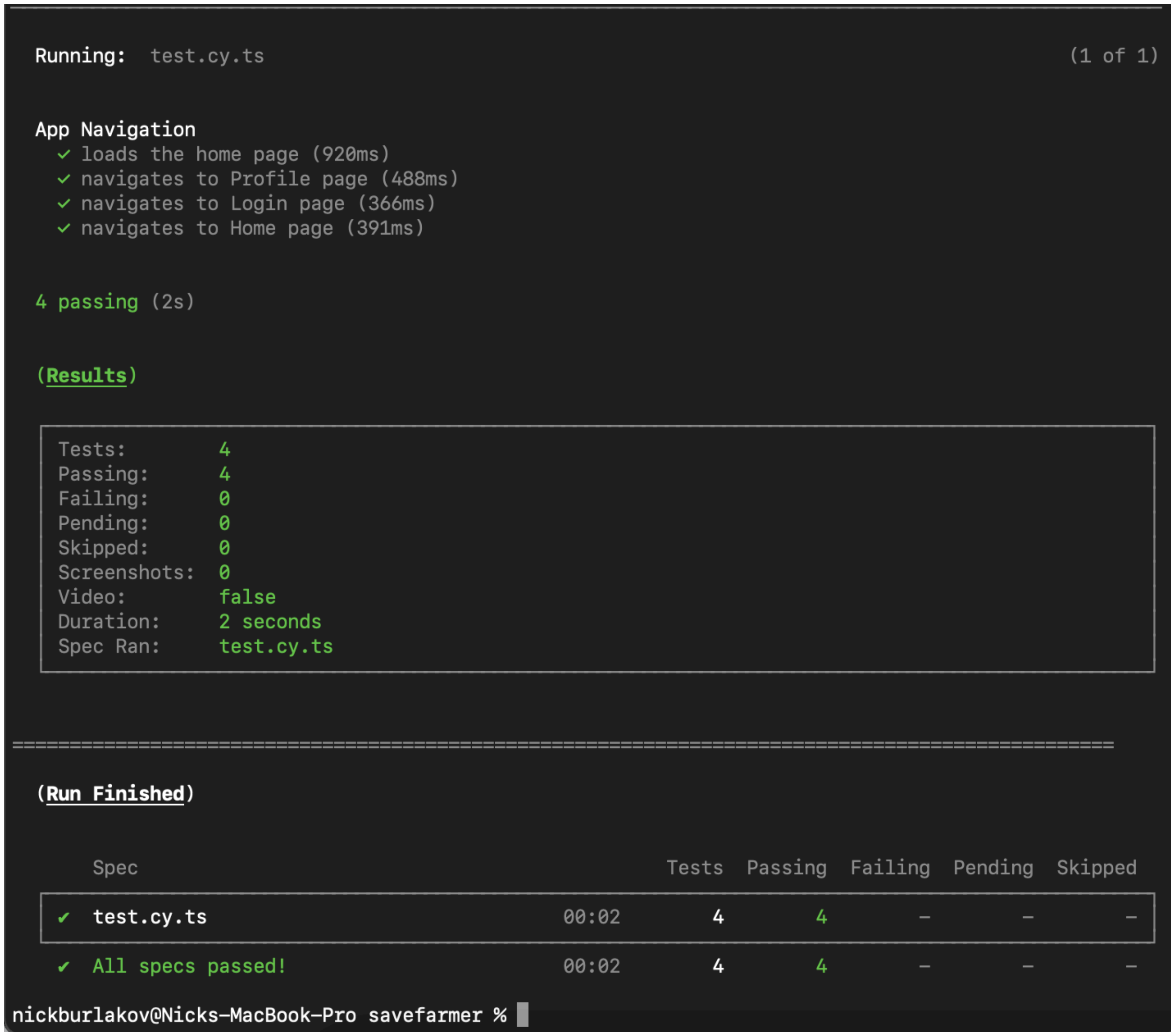
The tests focus on validating the navigation functionality of the app, ensuring that users can smoothly move between different pages. The App Navigation test suite includes the following scenarios:



```
describe('App Navigation', () => {
  it('loads the home page', () => {
    cy.visit('/');
    cy.contains('Button list');
  });

  it('navigates to Profile page', () => {
    cy.visit('/');
    cy.contains('Profile').click();
    cy.url().should('include', '/Profile');
  });

  it('navigates to Login page', () => {
    cy.visit('/');
    cy.contains('Login').click();
    cy.url().should('include', '/Login');
  });

  it('navigates to Home page', () => {
    cy.visit('/');
    cy.contains('Home').click();
    cy.url().should('include', '/Home');
  });
});
```

Fig. 26 (Test Cases)

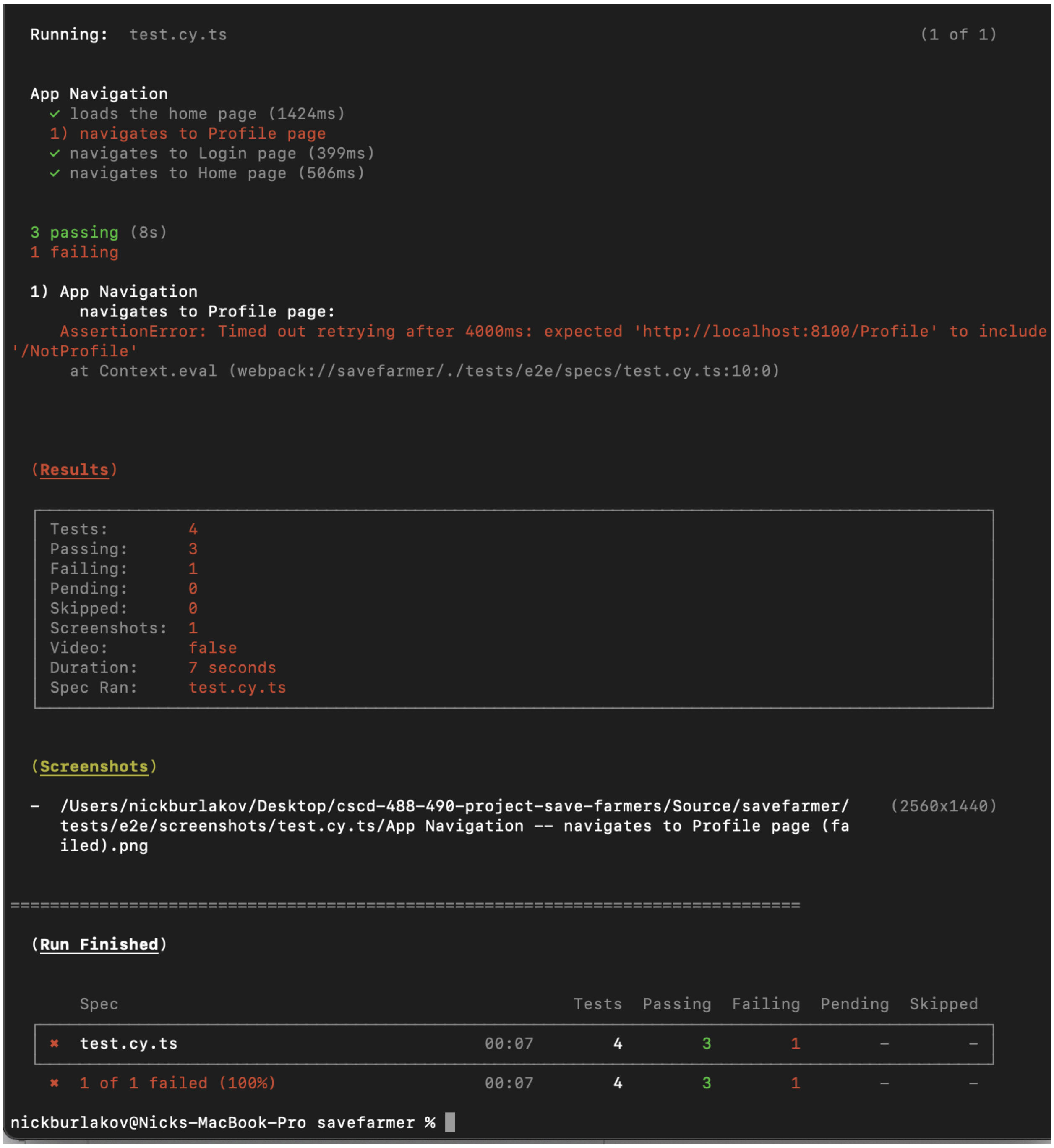
1. **Loads the Home Page**: Verifies that the app's home page is accessible and the "Button list" header is rendered.
2. **Navigates to Profile Page**: Confirms that clicking the "Profile" button correctly redirects to the /Profile page.
3. **Navigates to Login Page**: Ensures that the "Login" button redirects to the /Login page as expected.
4. **Navigates to Home Page**: Checks that the "Home" button leads back to the /Home page.



```
Running:  test.cy.ts                                                                    (1 of 1)


App Navigation
  ✓ loads the home page (920ms)
  ✓ navigates to Profile page (488ms)
  ✓ navigates to Login page (366ms)
  ✓ navigates to Home page (391ms)


4 passing (2s)


(Results)

┌──────────────────────────────────────────────────────────────────────────────────────────────────┐
│ Tests:        4                                                                                  │
│ Passing:      4                                                                                  │
│ Failing:      0                                                                                  │
│ Pending:      0                                                                                  │
│ Skipped:      0                                                                                  │
│ Screenshots:  0                                                                                  │
│ Video:        false                                                                              │
│ Duration:     2 seconds                                                                          │
│ Spec Ran:     test.cy.ts                                                                         │
└──────────────────────────────────────────────────────────────────────────────────────────────────┘


================================================================================================


(Run Finished)


     Spec                                         Tests  Passing  Failing  Pending  Skipped
┌──────────────────────────────────────────────────────────────────────────────────────────────────┐
│ ✔  test.cy.ts                          00:02        4        4        -        -        - │
└──────────────────────────────────────────────────────────────────────────────────────────────────┘
  ✔  All specs passed!                   00:02        4        4        -        -        -
nickburlakov@Nicks-MacBook-Pro savefarmer %
```

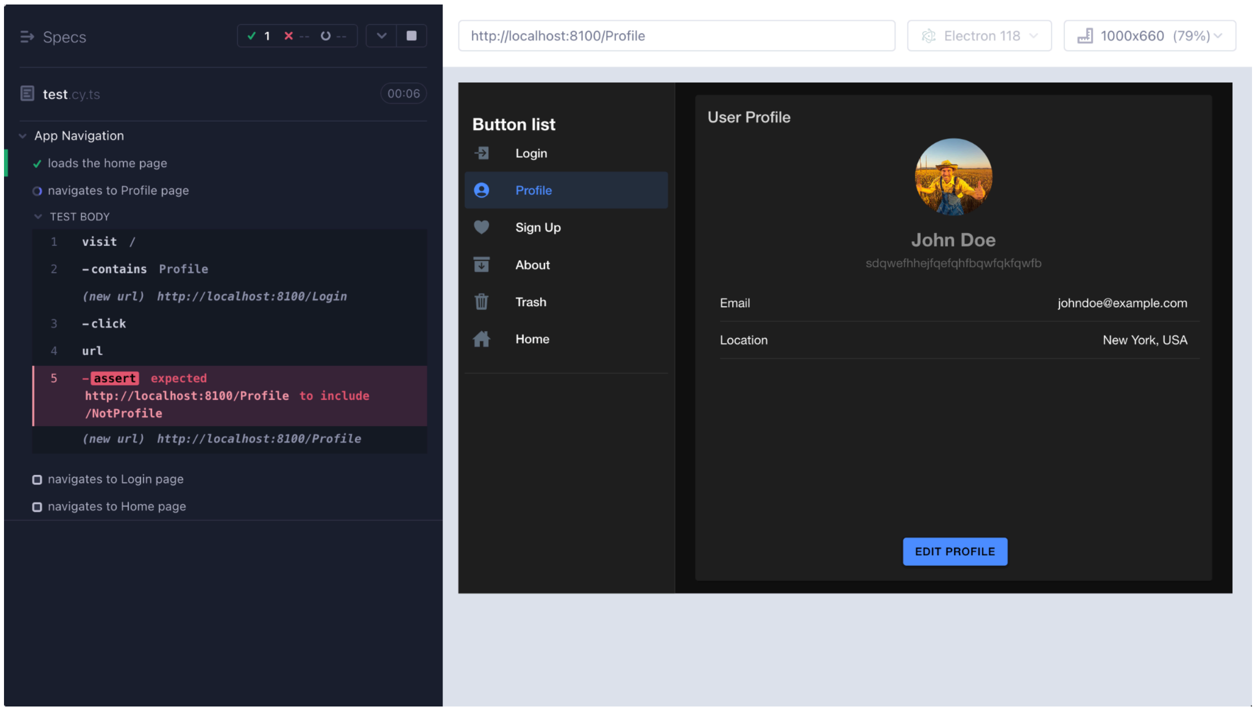
Fig. 27 (Successful Test Validation)

These tests confirm proper functionality of the navigation menu and ensure the app responds correctly to user interactions. If a test were to fail, Cypress would notify us of the failed case and generate a screenshot of the error from a user perspective, making debugging easier.



```
  Running:  test.cy.ts                                                                      (1 of 1)


  App Navigation
    ✓ loads the home page (1424ms)
    1) navigates to Profile page
    ✓ navigates to Login page (399ms)
    ✓ navigates to Home page (506ms)


  3 passing (8s)
  1 failing

  1) App Navigation
       navigates to Profile page:
     AssertionError: Timed out retrying after 4000ms: expected 'http://localhost:8100/Profile' to include
 '/NotProfile'
      at Context.eval (webpack://savefarmer/./tests/e2e/specs/test.cy.ts:10:0)




  (Results)

  ┌────────────────────────────────────────────────────────────────────────────────────────────────┐
  │ Tests:        4                                                                                │
  │ Passing:      3                                                                                │
  │ Failing:      1                                                                                │
  │ Pending:      0                                                                                │
  │ Skipped:      0                                                                                │
  │ Screenshots:  1                                                                                │
  │ Video:        false                                                                            │
  │ Duration:     7 seconds                                                                        │
  │ Spec Ran:     test.cy.ts                                                                       │
  └────────────────────────────────────────────────────────────────────────────────────────────────┘


  (Screenshots)

  -  /Users/nickburlakov/Desktop/cscd-488-490-project-save-farmers/Source/savefarmer/     (2560x1440)
     tests/e2e/screenshots/test.cy.ts/App Navigation -- navigates to Profile page (fa
     iled).png


================================================================================

  (Run Finished)


       Spec                                              Tests  Passing  Failing  Pending  Skipped
  ┌────────────────────────────────────────────────────────────────────────────────────────────────┐
  │ ✖  test.cy.ts                               00:07        4        3        1        -        - │
  └────────────────────────────────────────────────────────────────────────────────────────────────┘
    ✖  1 of 1 failed (100%)                     00:07        4        3        1        -        -

nickburlakov@Nicks-MacBook-Pro savefarmer %
```

Fig. 28 (Failed Test Validation)



```
Specs                    ✓ 1  ✕ --  ○ --
test.cy.ts                              00:06
App Navigation
  ✓ loads the home page
  ○ navigates to Profile page
    TEST BODY
      visit  /
      -contains  Profile
      (new url)  http://localhost:8100/Login
      -click
      url
      -assert  expected
       http://localhost:8100/Profile to include
       /NotProfile
      (new url)  http://localhost:8100/Profile
  □ navigates to Login page
  □ navigates to Home page
```

Fig. 29 (Failed Instance Snapshot)

The test results highlight how important automated testing is in ensuring a seamless user experience. By validating the navigation functionality of the app, we can confirm that key interactions work as intended. The failed test for the Profile page underscores the value of Cypress's detailed reporting and snapshot features, which make it easier to pinpoint and resolve issues. These tests provide confidence in the app's ability to handle user navigation effectively, while also emphasizing the need for ongoing testing to maintain functionality and user satisfaction.

15 Deployment and Maintenance

Currently, both front and back ends are deployed on Azure. The front end is on a static web app and the back end on a web app with an SQL database.

15.1 Frontend Deployment - Web

Assuming you have a properly set up Azure account with an active subscription, follow these steps:

1. Fork the project to your personal GitHub repository.
2. Navigate to your Azure main menu and click on **Create a resource**.
3. Search for **Static Web App** and click **Create**.
4. In the **Deployment details** section, select **GitHub** and link it to your account.

5. For **Organization**, select your account name and choose the correct repository for the project.
6. For **Build presets**, select **Vue.js**.
7. In the **App location** field, enter: ./Source/savefarmer.
8. In **Deployment configuration**, select **GitHub**.
9. Click **Create**.

This process should automatically generate a GitHub workflow and deploy your web app to the Azure-provided link.

15.2 Backend Deployment - API

Setting up the API in Azure

I recommend following this video for guidance, then proceeding to steps 27-30 as they are not covered in the video.

Prerequisites:

- Ensure you have an Azure account.
- Be logged into Visual Studio.

Steps to Deploy the API

1. **Open the API Project** in Visual Studio.
2. In Visual Studio, **right-click** your API project and select **Publish**.
3. Select **Azure App Service** as the target. For this project, choose **Azure App Service (Windows)**.
4. Click **Next**.
5. Select **Create a new app service** and click **Next**.
6. Select **Skip this step** for the API setup.
7. Click **Finish**.

SQL Database Configuration

1. On the **Publish** page, scroll down to the **SQL Server Database** section.
2. Click the **three dots** (•••) and select **Connect**.
3. Select **Azure SQL Database** and click **Next**.
4. Create a new database.
5. When prompted, create a **username** and **password**—remember these for later use.

6. Click **Next**, change the **Connection String Name** to "ConnectionString:DefaultConnection", and click **Next** followed by **Finish**.
7. Now, click **Publish**.

Configuring Azure Portal

1. Go to the **Azure Portal** and navigate to your **SQL Server**.
2. In **Settings**, go to **Entra ID** (formerly **Nucriseift**).
3. Click **Set Admin**, find yourself, and add yourself as an admin.

SQL Database Permissions

1. Go to the **SQL Database** in Azure and open the **Query Editor**.
2. **Allow your IP address** and log in as the admin.
3. Run the following SQL commands in the query editor:

```
CREATE USER [YourAppServiceName] FROM EXTERNAL PROVIDER;
ALTER ROLE db_owner ADD MEMBER [YourAppServiceName];
```

4. Click **Run**.

Environment Variable Configuration

1. Navigate to your **Web App** in Azure.
2. Go to **Settings** > **Environment Variables**.
3. Click **Add** and enter:
   - **Name:** JwtSetting__Key
   - **Value:** ThisIsA32CharactersLongSecretKeyEXL
4. Click **Apply**.

Final Step

- Wait about **5 minutes**, then visit your **App Service link**. Your API endpoint should now be working.

15.3 Additional Information for Deployment

Refer to the Developer Documentation for more instructions.

15.4 Maintenance

Currently, we do not see anything that needs maintaining. No dependencies should need updating because they are all running the specific version that the project needs.

Trouble Shooting

1. Can't access the app: Check your internet connection and try refreshing.
2. Listing not showing: Ensure you have submitted all required details correctly.
3. If you're having trouble editing a post: Try logging out and back in before editing again.

Contact Information

For support, contact our developers via email:

- tvo12@ewu.edu
- dpikulik@ewu.edu
- nburlakov@ewu.edu

16 Conclusion and Future Work

16.1 Summary

We are ending this project with the Save Farmers App functional and deployed. Despite all of the bottlenecks that we went through, we still persevered to a finished product.

Lessons Learned:

- Starting with a focused Minimum Viable Product ensures essential features are prioritized.
- Effective communication with stakeholders is critical for aligning technical solutions with user needs.
- Early identification of technical constraints helps streamline the development process.
- Full Stack Development is hard, and staying organized with the team and project is key.

16.2 Challenges and Resolutions

Challenges and Resolutions faced throughout our work on the project.

1. **Deployment:** We knew deploying to Azure was not an easy task, but we didn't expect it to be so hard. There were a lot of issues with the app that were not easily visible and that Azure was not happy with.
    a. **Resolution:** Simply taking the time to read the logs and exploring forums for answers. Very time-consuming, but we had no other alternatives.
2. **Platform-Specific Errors:** When the app was ready to be pushed to the device with no error, during the building process for a specific platform, there would be errors.
    a. **Resolution:** Researching the issue. Luckily, we didn't have any big issues, but Cypress testing had to be removed because of IOS. (See Section 14 **Testing and Validation)**

16.3 Future Enhancements

To expand the functionality and reach of our app, we can implement the following:

1. **Integrated Messaging System:** Adding an in-app chat feature to facilitate direct communication between buyers and sellers, reducing reliance on external channels.
2. **Payment Integration:** Enabling secure in-app payment options to streamline transactions and build trust among users.
3. **Rating and Review System:** Allowing users to rate and review sellers and buyers to enhance credibility and transparency.
4. **Localization Support:** Multilingual support and region-specific settings to cater to a diverse user base.
5. **Mobile Notifications:** Adding real-time alerts for new listings, updates to favorites, and responses to posts.
6. **Community Features:** Introducing forums or groups where farmers and buyers can share advice, tips, and collaborate.
7. **Image Support:** Images for user profiles and posts for more understanding of the user and corp that the buyer is working with.
8. **Dynamic Language:** language selection that allows users to switch languages in real time without restarting the app.
9. **Loading Page Implementation:** Adding a dedicated loading screen that appears during data fetching, ensuring users receive clear visual feedback when API requests are in progress. This improves the user experience by preventing confusion during longer load times.

17. Individual Contributions

This section highlights the specific contributions of each team member to the *Save Farmers App* project.

- **Dillon Pikulik**
    - **Role:** Team Leader, Communications, Full Stack Developer, and Deployment
    - **Contributions:**
        - Communications with the team and sponsor.
        - Backend Deployment
        - Backend Error fixing
        - Frontend Feature Development
        - App Testing
        - Documentation Developer

- **Nicholas Burlakov**
    - **Role:** UI Designer, Frontend Dev, Tester
    - **Contributions:**
        - Designed the initial user interface.
        - Conducted functional testing of the prototype to identify and resolve bugs in early development stages.
        - Provided feedback on usability and ensured the interface aligns with the user stories and target audience needs.
        - UI design
        - Written documentation

- **Eric Vo**
    - **Role:** Full Stack Developer, Ionic Framework Frontend, ASP.NET core Backend, and Deployment
    - **Contributions:**
        - UI design
        - Writing documentation
        - Designing database scheme

- Setting up Ionic
- Backend development
- Frontend Deployment

Each team member played a pivotal role in shaping the foundation of the Save Farmers App. The collective effort has set the stage for further development and success.

18. References

Gavilan, Felipe. “Deploying a Web API Into Azure with Its Database (Quick and Easy) | ASP.NET Core.” *YouTube*, YouTube, 2025, www.youtube.com/watch?v=szaK4e3L8WY.

Ionicframework. “The Cross-Platform App Development Leader.” *Ionic Framework*, ionicframework.com/. Accessed 15 Mar. 2025.

Nick WyntherNick Wynther 10811 silver badge66 bronze badges, et al. “Azure Static Web App with Vue Project Routing Not Working.” *Stack Overflow*, 1 Feb. 1966, stackoverflow.com/questions/67167681/azure-static-web-app-with-vue-project-routing-not-working/68196831#68196831.

RoseHJM. “Azure Deployment Environments Documentation.” *Microsoft Learn*, learn.microsoft.com/en-us/azure/deployment-environments/. Accessed 15 Mar. 2025.

“Testing Frameworks for Javascript: Write, Run, Debug: Cypress.” *RSS*, www.cypress.io/. Accessed 15 Mar. 2025.

Wadepickett. “ASP.NET Documentation.” *Microsoft Learn*, learn.microsoft.com/en-us/aspnet/core/?view=aspnetcore-8.0. Accessed 15 Mar. 2025.

19. Appendix