CS 373 Group 14 -
Central Valley Farm Aid
Phase 3 Technical Report

About

Rural and small family farmers inherently face many threats to the viability of their establishments, ranging anywhere from infrastructure to financial constraints. Our website helps rural farmers in the Central Valley region of California identify where they can sell their crops in farmers' markets based on their location. It also shows them what non-profit organizations can provide assistance based on their location.

Questions we want to help answer:

1. Where can rural and small family farmers go to sell their products in nearby farmers' markets?
2. What non-profit organizations can help rural and small family farmers sell their products based on location?
3. How can the farmers get started or receive assistance in selling their crops?

Tech Stack

- Frontend - ReactJS, Bootstrap, Jest, Selenium, Amazon EC2
- Backend - Python with Flask, SQLAlchemy, Unittest, Amazon EC2
- Database - MySQL, Amazon RDS
- API Documentation - Postman
- Version Control - Git, Gitlab, Docker
- AWS - hosting our server on EC2 instances

Models

Location (counties and cities)

~58 counties in California from whom we can gather farming, agricultural markets, and non-profit data. Included data points:

1. Location Name
2. Established Date
3. Area
4. County Seat
5. Population Size
6. Map Image
7. Crops produced and sold in the location
8. Images

Non-profits

~129 in California specialize in helping the agricultural industry, with a focus on local businesses.

1. Nonprofit name
2. EIN
3. URL
4. Donation URL
5. City, State, Zipcode
6. Mission statement
7. Eligiblity, deductibility, status
8. Established date
9. Latitude/Longitude
10. Images

Farmers' markets

~107 Farmers' markets are held across California on which we can gather information and connect to non-profits.

1. Market name
2. Address
3. Organization
4. Description
5. Location (latitude and longitude)
6. Contact info
7. Location descriptions
8. Images
9. Website

10. Wheelchair accessible
11. FNAP status
12. Production methods

Data sources

https://www.usdalocalfoodportal.com/fe/fregisterpublicapi/

https://charityapi.orghunter.com

https://quickstats.nass.usda.gov/

https://developers.google.com/maps/

Postman API Documentation

Hosting URL: https://www.centralvalleyfarmaid.me/

- Acquired from *Namecheap*

All URL calls are defined in the Postman Documentation. (Link to Documenter View)

This API is utilized to securely connect the backend of the Central Valley Farm Aid (Group 14) project to the front end by setting up API calls that the front end can call to collect information to display on the website. Find the GitLab repo here.

API Unit tests: Find unit tests for backend on postman, they test to see if the response is formatted correctly and if the status code is OK.

Backend

The backend parses information from the database, which contains information gathered from several APIs regarding Farmers' Markets, Locations, and Nonprofits.

- https://www.usdalocalfoodportal.com/fe/fregisterpublicapi/
- https://charityapi.orghunter.com
- https://quickstats.nass.usda.gov/
- https://developers.google.com/maps/

The backend Flask server is hosted with the use of EC2 and the upload of the backend Docker image to the instance. We utilized WSGI to run our Flask server. SQLAlchemy was used to set up the models based on the tables in our MySQL database. The Flask endpoints invoke sessions on the database connection, which then query from the respective tables. The endpoints then process the query data and return a JSON response. Marshmallow was used for

schemas in order to deserialize input data to app-level objects. Testing was done on endpoint functionality using an in-memory SQLite database as well as calls to endpoints to ascertain response status and accuracy.

Phase 2:

Frontend

Frontend: The frontend is running a ReactJS app that currently only has 14 pages: the home page, about page, pages for each model, and 3 instances for each model. We are using react-bootstrap for components such as the NavBar and Cards. We used react-router-dom to connect our pages on react with a URL path.

We structured the front like this:

src/
components/
pages/
utils/
…

The components folder holds files for all the reusable components. An example of a reusable component is the card for all the team members. You can iterate through an array of member objects and pass each member into the component for it to display a card of the team member with their respective data.

The pages directory has all the pages associated with a route, such as “url/” for home, “url/about” for about, etc.

The utils folder holds any classes for utility. We currently only have a utility class for GitLab API calls, but we might also add API calls to our backend here.

The frontend was easier to integrate with the backend since we did templating in phase 1 already. In order to allow pagination, we used useEffect() and setState() in order to remember the page the user is currently on and to navigate to and change pages. To display Google Maps data in the frontend, we use an iframe and pass in a URL to the Google Maps API with parameters of the instance.

Pagination

Pagination was implemented in the website. We set up the website’s ability to paginate in the backend with specifications for the per_page limit and page number. In order to grab a specific

page, the parameters can be specified in the API call, which returns the number of instances within the corresponding page. The APIs are called in the frontend using axios based on what page the user is currently on.

Database

Hosting
We decided to use a MySQL database for its popularity and support with AWS and Python libraries. We created an RDS (Relational Database) instance of type MySQL in the same VPC (Virtual Private Cloud) as our EC2 instance for easy communication. We also configured this database so it is open to all connections (from EC2 and otherwise). To test scripts for “stuffing” this database, we downloaded MySQL Workbench locally and connected to the RDS instance over HTTPS. Making this connection allowed us to confirm and visualize that the scripts we ran worked, and in the long run, we can query data from our API’s in case we have issues displaying/ transforming it on our website.

Data collection
The mentioned APIs above are used to populate the frontend. In order to do that, the data is queried in a Python script, where it is cleaned, formatted, and finally inserted into our database with a SQL query. There are custom queries for each API in order to get the correct set of data for our purpose of aiding Central Valley California farms. One can find the exact calls made detailed in the *External API Calls* collection in the Postman documentation.

You can find this code in the idb/backend/data_collection.py script, where the steps for querying, processing, and inserting the API data can be found divided into functions. One can find the independent APi documentation in the API website links above, to change the API calls one can customize the calls on Postman and replace the existing API call in the python script. Some APIs require several API calls to gather all required data as certain parameters are not mergeable.

Phase 3:

Frontend

The frontend now has a search page to search from all three different models. This is mostly reused code from Phase 2, but getting all the results based on the search query instead of using pagination. We went with this design approach as usually, search results should not be more than a handful, especially with the limited amount of data we are using. In addition to the search

page, each model also has a search bar/function and drop downs that allow the user to sort or filter the data.

Phase 3 was a pretty basic phase for the frontend. There was some trouble with rendering the cards with the sorting/filtering/searching, but other than that it was okay. All that was needed was to follow the API documentation.

Most of the data was handled with react states to hold values to sort, filter, and search with. You would then add them as a dependency in a useEffect() hook which will re-call the API once a parameter is changed.

Backend

The backend now has relevant searching, filtering and sorting for all models. This functionality is directly used by the backend to power the front end features.

Data engineering and management
Most, if not all, of our data handling and engineering is done through SQLAlchemy on the backend side. We don't directly change any of the tables in mySQL, but work needed in terms of handling searching, filtering, ordering, and more is done through SQLAlchemy.

Database

We have a final_instance_name_data table for every instance, where it contains data sources from our data collection efforts that we directly use in our backend. One thing worthy of mentioning is that we have FULLTEXT indexing enabled on a select few columns for instances to help handle relevance searching on the backend side

---

User Stories

Phase 1 User Stories

Markets Near My Location

This user story is asking to be able to view markets that are near the user. This will be a dynamic attribute that we can filter so it will be in a later phase.

Participate in Markets

This user story is asking for a way to be able to register at a farmers market. Our instance of farmer's markets includes contact information such as email, phone numbers, and URLs, so it is (technically) already implemented.

Local Farming Laws

This user story wanted to know the local laws of each region to find out the pesticide and GMO usage in farms in that region (???). This user story is out of scope and not from the perspective of our target audience. It provides no help to our target audience so we will not be implementing it.

Markets In Most Need of Support

This user story is also from the perspective of someone wanting to buy produce from a farmers' market and not our target audience of farmers. Despite this, it might be possible to find this information later by sorting farmers' markets by the number of vendors (which might not be correlated to needing aid).

Immigrant Ran Markets

This user story asks to be able to search what markets are run by certain immigrants. Since farmers' markets are run by multiple people, this data most likely does not exist. We can, however, search the name and description of farmers' markets for keywords, so we have asked them for a list of types of immigrants that they would like to search for. They have not responded yet.

Phase 2 User Stories

Crop Searching

This user story asks for the feature of identifying specific locations that support various crops. As this relies on a searching and grouping functionality, we intend to implement this story in phase 3.

Directions to Market

This user story asks for a visual form of media through the addition of a map service such as Google Maps. This has been implemented in the models through the use of the Google Maps API, which gives an outlook on the location of each instance.

Filter Functionality

This user story identifies a feature that would be nice to have in the markets page: sorting based on number of vendors present. As this relies on a filtering functionality, we plan to implement this in phase 3 in order to accommodate the request.

Nonprofit Criteria

This user story is interested in the addition of nonprofit aid criteria. Our API is rather limited in terms of providing the requirements one needs to meet in order to receive aid from certain organizations. In addition to the contact information, we do, however, include information about the organizations through short descriptions and website links, which hopefully provides a sufficient alternative.

Location Photos

This user story asks for more visuals within our locations model. We have implemented this feature in the model page as well as each of the instances of the locations.

Challenges

One of the largest issues we've encountered throughout this phase has been the data within our APIs. We have had to change our stance with regards to the data usage for each model multiple times. In the frontend, there were some problems concerning the integration of other APIs such as Google Maps API. We had some issues with backend hosting in EC2, and there was a lot of work in getting our Docker issues resolved. For RDS, there were many issues with hosting our MySQL database that led to us utilizing another AWS account and redo-ing the process.

Phase 3 User stories

Customer Reviews

This user story asks to include full worded reviews of these farmer's markets in the instance pages to help them get more context. We are currently planning to implement this next phase.

Mobile User View

This user story asked to fix some of the maps on some farmer's market instance pages that would get cut off on the mobile view of the site. We were able to fix this.

Dynamic Search Results

This user story asks to be able to search a city in the locations page to show the corresponding county as well as being able to show farmers markets and nonprofits being returned in the search if a city was contained in their address. The first request is not possible for us to implement as our apis do not give us enough data to match the city with the county. However, the second request is implemented in our full text search.

Farmers Market Hours/Days Open

This user story asks to have the opening times of a farmers market to be listed on the instance page. This is planned to be implemented in phase 4.

Most Profitable Crops

This user story asks to be able to sort by how profitable each crop is in each location. We don’t have sufficient data to implement this.