Wine Data Lake Web Application

Setup & Instructions

Last Updated: August 10th, 2022

Overview

The **Wine Data Lake** web app has three primary components: a frontend web client, a backend API, and a postgresQL database. Each of these components can be configured and deployed separately or together. Instructions in this document will focus on deploying the frontend and backend together on a single service, Google App Engine.

Goals

1. **Setup the Database:**
   Host a copy of the Wine Data Lake postgresSQL database.
2. **Deploy the Application:**
   Using Google App Engine, deploy the application to the cloud.

Evaluation Environment

An evaluation environment will be live from August 10th, 2022 to September 17th, 2022. It has been deployed to the below address using the developers' personal cloud accounts. After September 17th, the web application and database will be spun down and the web application *will no longer be live*. By this period, the project should have been migrated to the company's cloud accounts and/or will need to be rebuilt from the source code.

https://winedatalake.wl.r.appspot.com

The Database

Predecessor Project Overview (Techsheets PDF Scraper)

The preceding team created a Python Web Scraper that receives a PDF input and outputs postgresQL INSERT commands to the terminal. This allowed the team to build a database of wine information related to the uploaded techsheets. The program itself does not build the database; it creates commands to insert data into the database.

Database Challenges & Considerations

Winesheet PDFs contain heterogeneous data and there is no standard industry format. They often contain artistic design choices (like the winery owner's signature passing through a block of data) that are utilized to differentiate the product from others'. While a human being can easily read these stylized PDFS, the design choices and inconsistent formatting make parsing winesheet PDFs an incredibly difficult task.

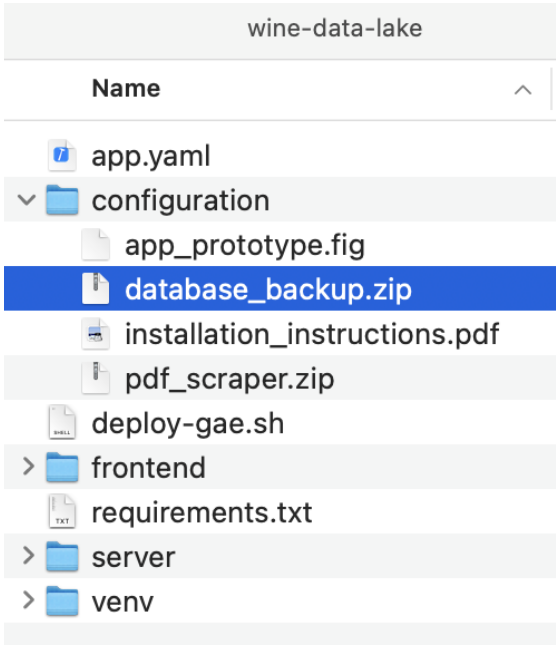
The ability of a PDF parser to accurately parse data from PDFs is limited - particularly without a user to check the data and confirm its accuracy. As such, it's important to note that the data in the database is not necessarily "clean." Due to the sensitive nature of winery branding, cleaning the database should be prioritized for the future.

Hosting & Setting Up the Database

The database can be hosted on any postgresQL platform. Here are some options for hosting the database:

- https://www.elephantsql.com/
- https://www.heroku.com/postgres
- https://cloud.google.com/sql/docs/postgres/create-instance
  https://aws.amazon.com/rds/postgresql/

A backup of the database is stored in the `configuration/` folder of the application. Alternatively, you could build the predecessor project's PDF Scraper, which is also included in the `configuration/` directory.



The Frontend Client

Technologies Used

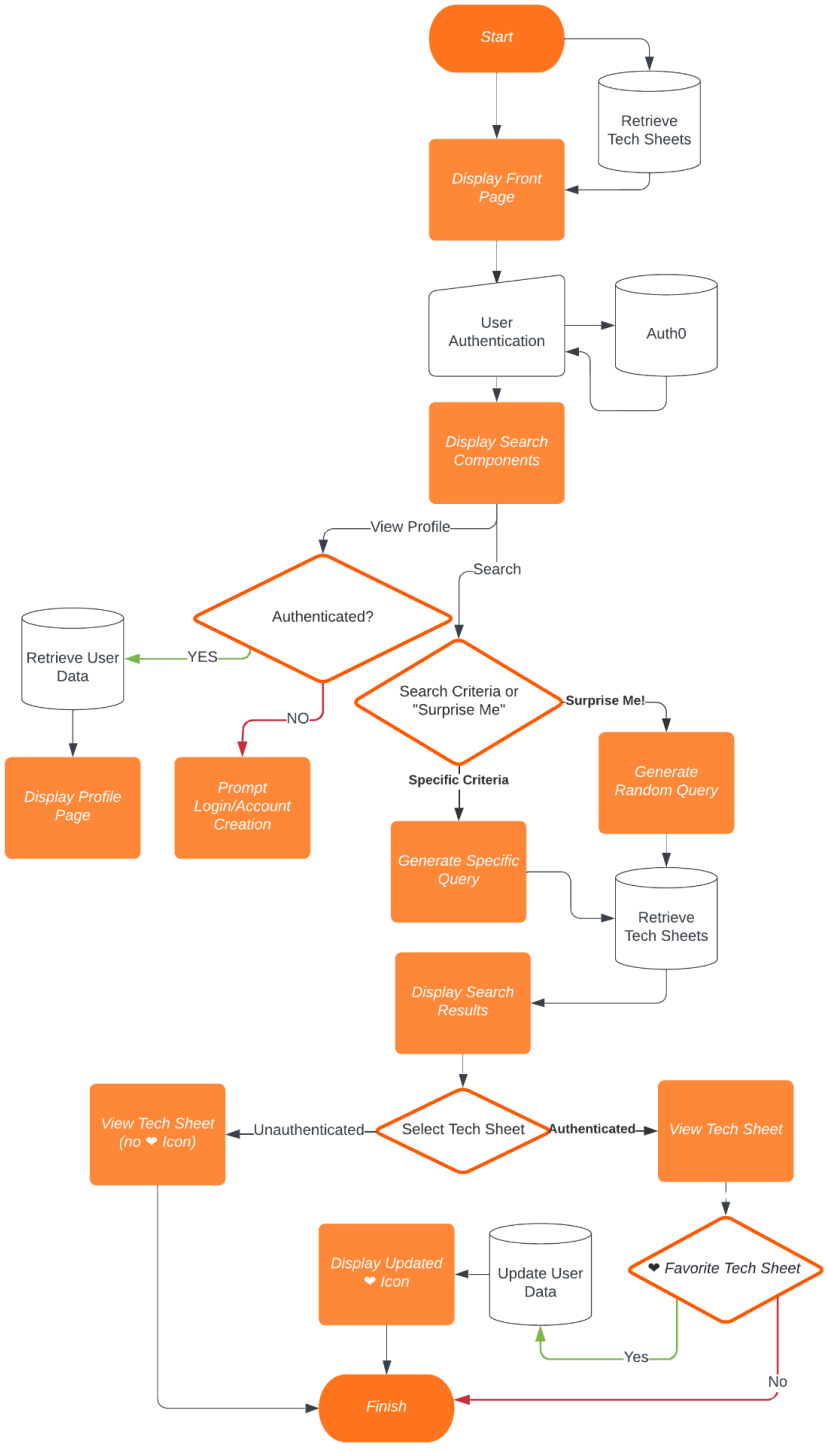
The frontend client is a React app built using the create-react-app bundler. It is written in JavaScript, JSX, HTML, and CSS. The application is "built" using Node.js and NPM and, once built, the application can be found in the project's `build/` directory.

Dependencies for the frontend can be found in the `frontend/package.json` file; some of the more important dependencies are:

- axios
- react-router
- auth0-react

Flow & Structure

The frontend contains four major experiences: a landing page, a search results page, a techsheets page, and a profile page. The landing page is the application's entry point. It contains several clickable thumbnails, a prompt for authentication, and a search box. The search results page displays the search results and contains a sidebar that allows a user to manipulate search data. The techsheets page allows the user to view the techeet data, preview the original techsheet, and download the original PDF to their computer. If authenticated, a user can "favorite"/"star" a given techsheet. Lastly, if authenticated, a user can view their profile and see techsheets they've marked as favorites.

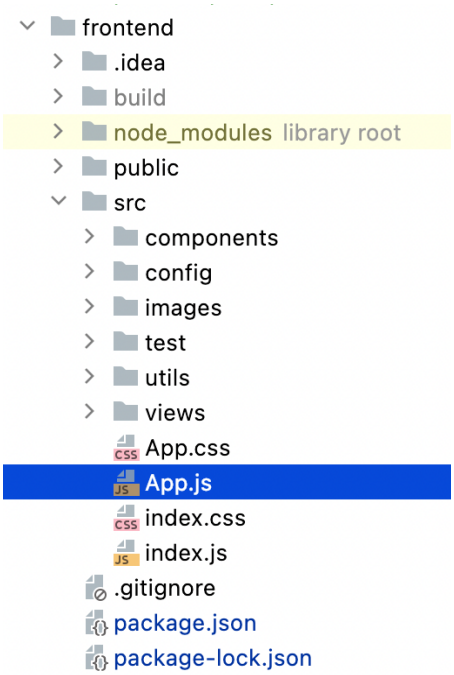

Start
Retrieve Tech Sheets
Display Front Page
User Authentication
Auth0
Display Search Components
View Profile
Search
Authenticated?
YES
Retrieve User Data
NO
Display Profile Page
Prompt Login/Account Creation
Search Criteria or "Surprise Me"
Surprise Me!
Specific Criteria
Generate Random Query
Generate Specific Query
Retrieve Tech Sheets
Display Search Results
View Tech Sheet (no ❤ Icon)
Unauthenticated
Select Tech Sheet
Authenticated
View Tech Sheet
❤ Favorite Tech Sheet
Update User Data
Display Updated ❤ Icon
Yes
No
Finish

Locating Assets & Resources

Frontend assets are all located in the `frontend/` directory. If the app has been built, `build/` contains the built application (static files). `public/` contains image assets, logos, and icons. `src/` contains all the pieces used to build the application, from .css to React components.

The `src/` file contains several additional directories.

- `components/` contains tiny pieces of the application, like the navigation bar
- `images/` contains icon sets and other UI/UX elements
- `config/` contains all the settings for the Auth0 instance
- `utils/` contains helper function
- `views/` contains each individual page that the user can navigate to
- `index.js` is the entrypoint for the application and handles routing
- `app.js` loads the main “template” for the site

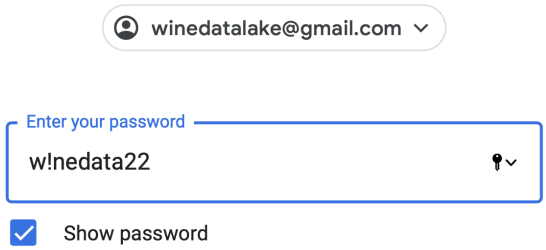


Having trouble?

- Ensure that Node.js and NPM are both installed on your local machine.
- From the `frontend/` directory, run `npm install` to install the dependencies referenced in `package.json`.

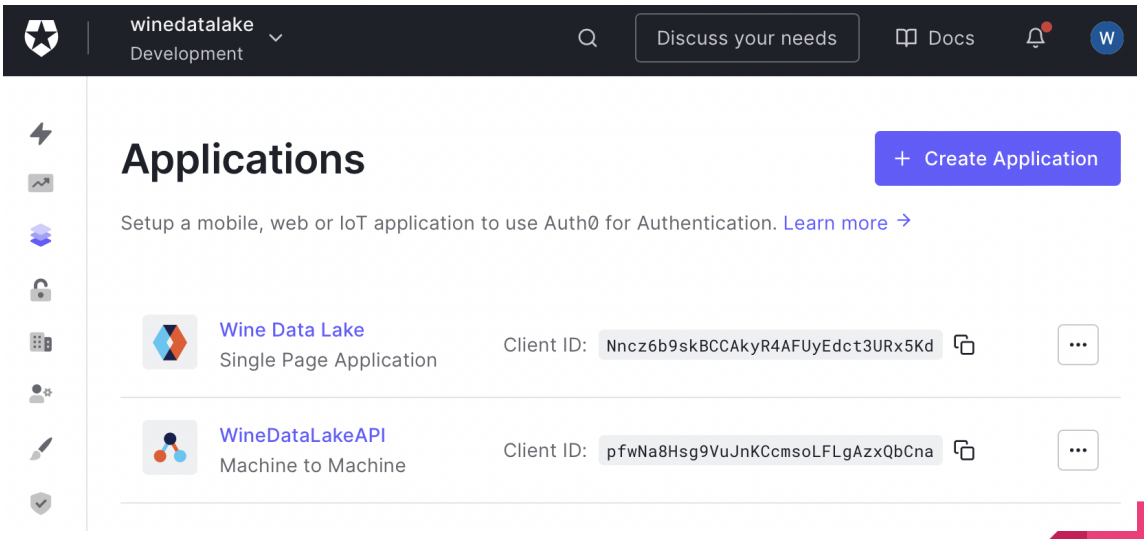
Auth0 Configuration & Setup

Auth0 has been configured on a Free Tier using dummy gmail credentials. On the frontend, Auth0 routes the user to a login/signup page, authenticates them, and returns them back to the site with a token. This token proves to the frontend that the user signed in and can access their profile data.



On the frontend, Auth0 is configured in `frontend/configurations/auth_config.json`. You can sign into Auth0 and see where the corresponding information is stored to get a better idea of how the API integrates with the Wine Data Lake application.

```
{
 "domain": "winedatalake.us.auth0.com",
 "clientId": "Nncz6b9skBCCAkyR4AFUyEdct3URx5Kd",
 "secret":
"7TuyECcDnsACqDk8E98AVF4PD4Pf9QJ4Bm6q5asgYKOd0FqR1C3t4xoSrwskc9ko",
 "audience": "aF3LnZv!&W@CB*@JZhTR8k7ZPT3gBGqvNdGmyJLspA#9T6hLJx59&pvAZ6",
 "scope": "read:users",
 "appOrigin": "https://winedatalake.wl.r.appspot.com",
 "apiOrigin": "https://winedatalake.wl.r.appspot.com"
}
```



Building the React App with *create-react-app*

Launching a React application involves building it. Building a React app is similar to compiling a program… except that the output is avaScript that's been optimized for deployment on the internet and is completely unreadable.

1. Via the terminal, navigate to the `frontend/` directory.
2. Enter `which node` and `which npm` to confirm that Node.js and NPM are both correctly installed.
3. Run `npm install` to install the frontend dependencies.
4. Use `npm run start` to view a development preview of the site on http://localhost:3000.
5. Execute `npm run build` to build a production version of the React application in the `build/` directory.
6. The `build/` folder can be deployed to a platform of your choice.

The Backend API

JWT Decoding w/ Auth0

Auth0 is also integrated into the backend. In the API, when a request to a protected endpoint comes in with an authorization token (JWT), Auth0 and the JWT verification library `jose` are used to verify that the token is valid. You can see the Auth0 set up in the Flask application's entry point: `api.py`. The helper functions used to verify a token can be found in `verify_jwt.py` (based on Auth0's Quick-Start code).

```
oauth = OAuth(app)
DOMAIN = 'winedatalake.us.auth0.com'
ALGORITHMS = ["RS256"]
CLIENT_ID = 'Nncz6b9skBCCAkyR4AFUyEdct3URx5Kd'
CLIENT_SECRET = 'aF3LnZv!&W@CB*@JZhTR8k7ZPT3gBGqvNdGmyJLspA#9T6hLJx59&pvAZ6'

auth0 = oauth.register(
    'auth0',
    client_id=CLIENT_ID,
    client_secret=CLIENT_SECRET,
    api_base_url="https://" + DOMAIN,
    access_token_url="https://" + DOMAIN + "/oauth/token",
    authorize_url="https://" + DOMAIN + "/authorize",
    client_kwargs={
        'scope': "openid profile email",
    },
)
```

When a user submits a JWT to the API as a means of authentication, the user can be identified by the Auth0 subscriber id contained in the token. **No other user-identifying information is shared via token**.

```
payload = verify_jwt(request) # Verifies the token
user_auth = payload["sub"]    # Pulls out the Auth0 subscriber ID
```

App Deployment to Google App Engine

Building the React App with *create-react-app*

Launching a React application involves building it. Building a React app is similar to compiling a program… except that the output is avaScript that's been optimized for deployment on the internet and is completely unreadable.

7. Via the terminal, navigate to the `frontend/` directory.
8. Enter `which node` and `which npm` to confirm that Node.js and NPM are both correctly installed.
9. Run `npm install` to install the frontend dependencies.
10. Use `npm run start` to view a development preview of the site on http://localhost:3000.
11. Execute `npm run build` to build a production version of the React application in the `build/` directory.
12. The `build/` folder can be deployed to a platform of your choice.

Google Cloud Setup

1. Create a Google Cloud account.
   https://console.cloud.google.com/
   Note: The account won't be usable until Google has confirmed your method-of-payment.
2. Install the Google CLI tools by following this tutorial:
   https://cloud.google.com/sdk/gcloud
3. Use the below tutorials to *create a Google App Engine project* and *enable billing*.
   a. https://cloud.google.com/appengine/docs/standard/nodejs/building-app/creating-project

b. https://cloud.google.com/appengine/docs/legacy/standard/python/console

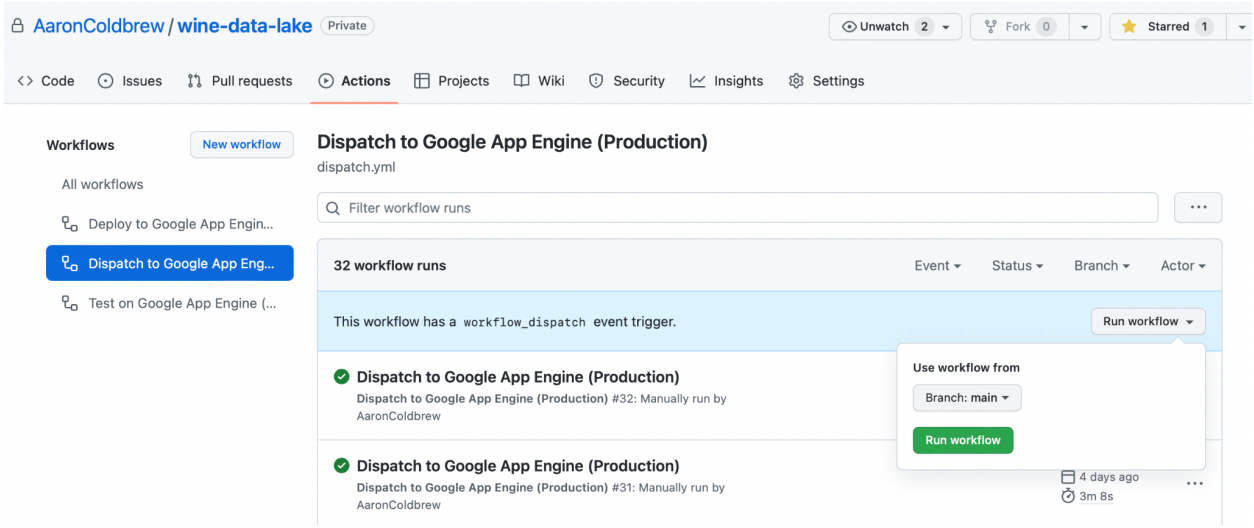
Deploy the GAE App

4. Use the terminal to navigate to the project directory. Type the following command follow the prompts to authenticate yourself: `gcloud auth application-default login`
5. If this fails, the Google CLI tools may not be installed correctly. Retry Steps #2 and #3.
6. Type the following commands and replace [PROJECT_NAME] with the name of your project. This should walk you through some various configuration tasks.
    a. `gcloud config set project [PROJECT_NAME]`
    b. `gcloud config set compute/region`
7. Deploy the application to your Google Cloud GAE project by following the below tutorial. https://cloud.google.com/appengine/docs/standard/nodejs/building-app/deploying-web-service
8. Google App Engine will use the associated .yaml file to configure your application. You can read more about .yaml files here: https://cloud.google.com/appengine/docs/legacy/standard/python/config/appref

Deploying via Github Actions Workflows

In addition to the workflows provided here, the project includes several GitHub actions workflows that rely on Github Secrets. These workflows located in the `.github/workflows` directory. The `main` workflow dispatches the application to App Engine when a PR is successfully merged into main/origin. The `dispatch` workflow deploys a branch of the repository to GAE on command. Lastly, the `test` workflow creates a "version" of the application on a non-production address. You can learn more about setting up Github Actions here and configuring secrets here:

- https://github.com/features/actions
- https://docs.github.com/en/actions/security-guides/encrypted-secrets



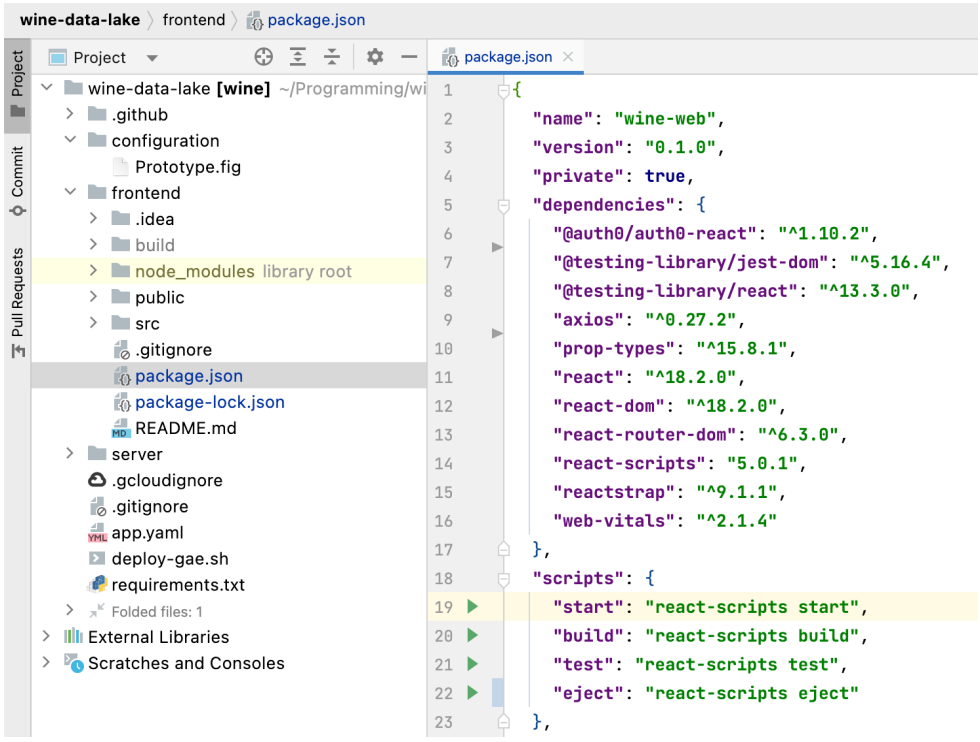
Troubleshooting

The application won't load!?

- Google App Engine requires all applications to be deployed on Port #8080.
  If you change this port during development, it must be changed back for deployment.

- Google App Engine automatically distributes files into Cloud Storage. Unfortunately, it cannot handle any filenames with non-ASCII characters. If you're still getting errors or certain assets aren't loading, confirm that project file names don't include non-ASCII encoded characters (ex: Unicode).

- When deploying locally, if the frontend doesn't seem to be updating, check to make sure that the `build/` folder is up-to-date and consider using one of *create-react-app's* builtin development tools. The package.json file contains some prebuilt commands to make development easier; you can call `npm run start` to preview a given build.



Future Considerations

Conversion to no-SQL

Conversion to no-SQL might be beneficial as the project grows. Currently, keeping an instance of the database running 24/7 is overkill in terms of resource utilization. A no-SQL database might help minimize costs and remove complexity.

Data Cleaning

The database contains misspellings, and heterogeneously formatted data. A team needs to look at the stored data and decide if the parser is meeting the project's requirements.

Admin Accounts & "Upload a Techsheet" Flow

As an alternative to cleaning the data, the parser could be integrated with the web application to create an "Upload a Techsheet" experience. An approved user could submit a techsheet, have it run through the PDF parser, and then have the data returned to them for human verification. This would allow wineries to feel confident that the data entering the database was accurate. It would also might help keep the data clean.

Thumbnail Images vs. PDF Previews

During the "Upload a Techsheet" experience, it might make sense to ask the user for a thumbnail. Feedback indicates that the preview images for the techsheets are too small! People can't see the text and find that confusing. An image of the bottle or a label or even the winery's logo might be a better thumbnail.

Add User Generated Reviews

Users generating reviews would help us complete the objective of sharing “tasting notes” or “keywords” with the user and would open up more complicated searches.

Refine Search

The search results don’t parse the search strings very finely. This could be improved upon.

Implement KPI/User Metrics

Our team ran out of time before implementing KPI/User Metrics. This would be an excellent next step.

Design UI to Display Other Datasets

The database contains more data than is displayed. Currently, the techsheets are displayed. How does the company envision displaying varietal data unconnected to specific techsheets? There is an opportunity to expand the type of data contained within the database and reconsider how users search for new wines.

Credits

Project Sponsor

A big “Thank you!” to Jim Cupples of *All the Farms* for trusting us with your idea.

Developers

- Aaron Thompson • thompsaa@oregonstate.edu
- Marilyn Leary • learym@oregonstate.edu
- Brice Kaszubinski • kaszubib@oregonstate.edu

Developed for the Oregon State University Summer Capstone of 2022.