VE20-COMM: Technical Addendum

Developer's Guide and technical details for the development of project deliverables

Authors

Lucas Fernandes
Taylor Ostrum
Nathan Morin
Kavim Bhatnagar

Table of Contents

Developer’s Guide

VE20-COMM GitHub Code Repository - https://github.com/srdpt/shops/tree/bTerm20Dev

This project was developed using React 17.0.1, written in JavaScript

Project Dependencies

Git - Use most recent version
React - Use most recent version
Material-UI - 4.11.1
Leaflet - 1.7.1
React-leaflet - 3.0.0
React-burger-menu - 2.9.0
React-router-dom - 5.2.0
React-select - 3.1.1
React-scripts - 4.0.0

Installation & Setup

These steps will walk you through cloning the repository, installing dependencies, as well as running the application locally

1) Clone the Repository

   If you don’t already, please install Git

   Once inside of your desired directory

   Use either cmd or powershell on windows to run
   git clone https://github.com/srdpt/shops.git

   cd shops

   You have now cloned the repository into this new directory which can be accessed using your preferred text-editor or IDE.

2) Installing project dependencies

   While still in the shops directory, you will now have to install npm(Node Package Manager).

   Once that is complete you can run

```
npm install
```

   Inside of the shops directory in order to automatically install all of the necessary project dependencies

3) Starting the application

   You are now able to develop and test the application locally!

   In order to start the application based on the current branch you are on run

```
npm start
```

   This script will now launch the local application
   Using your preferred web browser direct it to

   http://localhost:3000/

   The application will load and can be used!

   For development purposes as you save changes to your local branch and this locally served application will immediately update with your changes!

   You are now all set to begin working on and using the application!

Project Structure

This application follows the React application project structure as seen in the figure below. This following section will outline the project structure and define all subdirectories of the project. Many of the individual files not listed here explicitly are commented in their respective files.

Shops Directory

Base directory for the Venice Shops source code. It contains two sub directories, **public** and **src**, and contains additional files

Shops Directory Files

- .gitignore: Tells Git which files and directories shouldn't be pushed to the Git repository
- README.md: Introductory page displayed on the GitHub repository
- ReactReadMe.md: Introductory file to introduce React and essential scripts that come packaged with React App
- debug.log: Logs launch errors from React-scripts
- package.json: Houses all project dependencies as well as all project metadata
- yarn.lock: Similarly helps handle project dependencies when using the Yarn package manager(A substitute to npm based on user preference)

Public Directory

Directory for all publicly accessible files when application is launched. All components are fed through index.js found inside the src directory. The application is then rendered using HTML and CSS.

Public Files

- CNAME: holds domain name that directs to application when deployed
- favicon.ico: Small site icon to display on tabs inside of web browsers
- index.css: Basic CSS file declaring sizing of site content to take up the entire viewport
- Index.html: HTML page that renders whole application in browser
- manifest.json: Declarative file to be used in application deployment
- robots.txt: text file to instruct web robots (typically search engine robots) how to crawl pages on the website

Src Directory

Directory contains all of the site's content and functionality throughout the subfolders. The files included at this directory level are default files that come with React App.

Subdirectories of Src

- components: Directory for all developed application UI components. Files serve as both controllers and views for their respective components.
- css: Directory for all project styling written in CSS
- data: Directory for all team's JSON data that is locally queried when running the application
- pages: Component Directory for each unique page the application displays
- static: Directory contains all image resources and fonts used throughout the application

Component Hierarchy

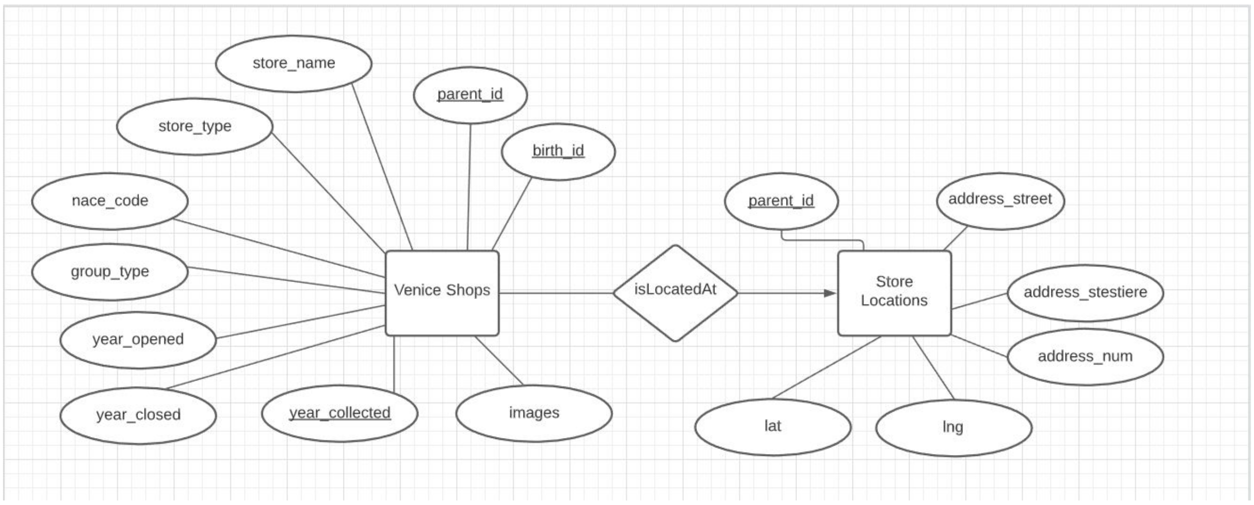
This hierarchy will define the structure of the components used in the application.

- Index.js
    - App.js
        - SiteHeader.js
            - Sidebar.js

          {Components below loaded independently using React Router}
        - MapPage.js
            - MapView.js
                - MapMarker.js
            - MapFilter.js
                - ShopTypeSelect.js
        - NotFoundPage.js

Data Cleaning Methodologies

As referenced by page 19 of our report, our team identified the essential attributes needed in order to define a single business and track that as well it's address over time. All of these attributes were included in every prior survey year. In this section we will be describing our dataset in far greater detail and how we arrived with our final product that can be found in our project files. To start, please review the Entity Relationship Diagram (ERD) our team designed to understand the relationship of our datasets.



As you can see above, Store Locations has only the parent_id as it's primary key as this ID is the unique identifier for each unique address ever surveyed in Venice. We generated this simply using a simple incrementing sequence by one through the set to ID every record. Then we also have Venice Shops, where the birth_id, parent_id, and year_collected are the primary keys. Additionally the parent_id in Venice Shops is a foreign key and references parent_id in Store Locations. Venice Shops has a one to many relationship with Store Locations, meaning that every shop record in Venice Shops has one location, but locations can have many shops. Having this relationship allows the same shop to be tracked over time as well as the list of shops that have inhabited a single location over time. Mentioned prior was our birth_id attribute, which is the ID we use to identify every unique business in Venice. Finally an important attribute to note in Venice Shops is images, which is stored as an array list of references to the images associated with a shop record.

By sorting Venice Shops records by their parent_id in ascending order, in cases of the same parent_id as well as other shop attributes being the same, particularly the store_name, then we sequenced that shop to have it’s unique birth_id. This ID process is far harder to generate using a simple sequence expression and required our team to process manually as we worked with records having missing attributes. The problem also arises from handling shops that were collected in the same year but had different parent_id’s as there are cases of stores having multiple addresses and entrances that were recorded. Processing the birth_id cannot be understated as it’s a far more laborative process due to the many edge cases. Most of this is alleviated though by the accuracy and completion of shop records data.

Our team worked to alleviate some of this work while processing the data because in most cases past teams recorded incomplete records. The varying inconsistency of records required more work on our team’s part. One case in particular were records missing longitudinal data. To work through this we worked with SerenDPT to acquire the dataset used in their iZioleti application which helps users navigate through Venice using it’s unique addressing system to provide navigation. Our team was provided a dataset that contained the longitudinal data for a myriad of addresses in the city. This helped us complete these records and informed us when designing our Store Locations dataset as we structured it similarly so that it has more applications outside of our project.

While these methodologies aided us in completing Venice Shops and Store Locations, our team also compiled the media data of all eight project teams collected in their surveys. Since images couldn’t be directly stored with our sets due to their structure and later implementation as JSON files. We have designed a naming scheme for these images as seen below.

yearCollected_sestiere_addressNumber_numberImage

This scheme can easily identify the associate record to reference and allows for records to hold multiple images if necessary. The files themselves are stored within the team’s project files hosted on Google Drive but they should ultimately be hosted using another provider in order to be served to the team’s application deliverable.

Recommendations

As referenced by page 32 of our report, our project and it’s deliverables are only the beginning. Concluding development of the initial application, the groundwork for future features and optimizations has been set. This section covers the optimizations and features our team believes should be the focus of future teams.

Before beginning to add new features, looking at the status of the application at the conclusion of our project, we first feel that the map marker generation function should be optimized and leverage using pre-stored arrays of values rather than rebuilding the same array on every change of the filter options. This will exponentially increase site performance which is incredibly important on the live production site. Additionally, updating the datasets to handle the image references from our media folders will be incredibly important when looking towards future feature implementations.

Looking at new innovations to the application, we firstly suggest team’s implement displaying the images for the selected marker in the popup. Our team unfortunately was unable to implement this feature but we were able to conduct research on fantastic libraries that would help us. We found react widget’s image carousel component to be the perfect container for displaying images for each shop record as it can handle one or many images linked to a record.

An important aspect of the application is validating the accuracy of the data being presented. In many cases records are incomplete or even inaccurate. Giving users a means to directly impact our database is incredibly useful by providing them the tools to provide feedback and more importantly their stories. We believe the inclusion of a form feature to request the application to add or modify attributes of a specific record would be incredibly useful when reevaluating the data. Submitting these changes would absolutely need to be approved and surveyed manually by the application admins before permanently being included. An extension to this is reviving a feature some past teams have attempted to capture in their surveys are oral histories from users.

Gathering anecdotes on specific shops, locations, or streets can be incredibly insightful and serve as a means to cross reference more archival data through comparison.

While data validation is a key feature to focus on in order to further increase the accuracy of the team's existing datasets, it's more important now than ever to ensure the data collection methodologies are strong so that once team's can physically survey Venice they can gather as much data as possible on the impact of COVID-19 and establish the start of studying the trends of data over the next few years. Accomplishing this requires improving the Input App developed by SerenDPT. To give some context the datasets we have produced are hosted on the CKData site that serenDPT also hosts. This is the application that our platform fetches data from. Doing this means the application pulls from the live, modifiable database. Within the CKData app every dataset hosted has access to an internal application called Input App. Users with access to this application can upload and write data to the desired dataset, but in its current state it lacks any adaptability to be able to write to our newly uploaded dataset. We strongly encourage following teams to prioritize developing this application further so that when surveys are able to be conducted in person again it can be a far more efficient process. Additionally investigating the potential for the application to be publicly accessible and have an approval process similar to the data validation feature mentioned prior can enable crowdsourcing data collection which can prove to be a powerful tool when looking to conduct analysis.

While these features are focused on improving the methods for working with data collected by the Venice Project Center, the opportunity to expand the sources of data is now available through our team's application. We encourage teams to work towards getting in contact with the Venice Chamber of Commerce in hopes to receive the datasets they track of the city. Their structures and attributes may vary but it would be an incredible supplement to the data validation features by allowing users to compare Venice Project Center data with that from the Chamber of Commerce. Further sources can utilize Leaflet's map layer features to include ways to visualize data in the city by looking to incorporate data from census tracts of the city in order to visualize the state of the population and find how that relates to shop data collected by the Venice Project Center. Similarly displaying Airbnb data in the application can prove to be useful in similar contexts to census tracts, both ways help visualize the areas most residents and tourists inhabit.

We also want to discuss a feature our team originally wanted to incorporate but unfortunately was out of the scope of our project and that was generating dynamic statistical graphs for the data sets in order to conduct data analysis. We encourage teams to utilize the D3.js library when fetching data from our database server on the CKData application so that they can conduct dynamic analysis over time in order to generate visuals based on hypotheses the team generates. These can be investigated at later points in time as more data is added to the database and is thus updating these graphics.