Engineering Assessment

My approach to completing the activity.

Aaron Peter Samuel 7/8/2024

SDLC

High Level Steps

- **Planning**
  - After brainstorming and identifying the approaches, languages, and frameworks I would use, I created and prioritized user stories, which represent the shortest path to providing an MVP.
- **Development**
  - After Inspecting the outputs from the data source, I decided that an API would provide a more consistent and reliable interface for building a UI or CLI.
  - I chose **Express** as the framework for the API component which only needed to be read-only for the needs of the MVP.
  - Created blueprints & sketches of UI presentation & main functionality
    - I leveraged React along with the Material UI library to accelerate the development of a reactive and responsive **User Interface**
  - Continuously adjust the plan, adding/removing, updating, and completing stories
    - Identify what is needed vs. nice to have. Continuously adjust priorities
- **Build**
  - With each component functioning locally as expected, I began working on containerization. I developed Dockerfile(s) and used pure Docker commands to validate that layers compile as expected and containers run as intended.
- **Deploy**
  - With each Dockerfile written and tested, I began thinking about how to package and deploy this.
  - I wanted to avoid awkward dependencies or steps on either side resulting from packaging and publishing our containers to a registry.
    - For example, I decided to use docker-compose as the orchestration environment to avoid storing, transferring, or exposing tokens (e.g., docker hub, AWS).
    - The end-user can use a docker-compose build to get up and running quickly.
    - This can easily be extended to a Kubernetes deployment containing one internal service and one external service (NodeIP or LoadBalancer)
  - Docker Compose is free, is available when you install Docker, and has multi-platform support.
- **Operate & Monitor**
  - In this case, I looked at the console logs, ensured components were rendered as expected, and tested various conditions. As lessons were learned, I submitted fixes, minor refactors, etc.

Minimum Viable Product

Achieving the MVP

- Create a read-only **API** to fetch, transform, and return food trucks from our data source *sfdata*.
- Create a **UI** that allows users to filter, browse, and search for food trucks in the San Franciso area.
    - Reactive & Responsive
        - Form inputs drive
            - Data Grid, Visualizations & Map features
    - Data Visualizations
        - Gain insights about similar food categories or truck vendors
    - Provide Geolocation & Mapping
        - Use the browser location feature to calculate the distance between the user and the food truck(s).
- Create a **CLI** that provides a subset of functionality for finding food trucks.
- All components should be executable in a *container-orchestrated* runtime.

Languages & Frameworks

Development Decisions

- I chose **Node** as the core language for each project component. It is a single-threaded object-oriented language with excellent performance characteristics and high support and guidance from a large user community across several operating spaces.
  - I leveraged the Express framework to accelerate the development of the API, which is very minimal and read-only.
  - I coupled the Express API with React to achieve the MVP's UI component. I envisioned a responsive, easily maintainable SPA (single-page app).
    - I used the Material UI component library to accelerate UI design and development
  - I've used Commander many times in professional settings to quickly and consistently provide CLI tooling, so this was my go-to choice for completion of the CLI component.

Constraints

Limitations

- Open & Free
  - Users/Developers should be able to access and contribute to this for free
    - This ruled out using Google Maps, etc.
    - This directed my focus on docker/docker-compose as the runtime environment for these components. Docker is very accessible and virtually free.
- Secretless
  - Users/Developers should not need to leverage tokens or secrets to use these components.
    - The complexity of adding secrets needs to be balanced against the benefits of features it may provide access to.
      - This ruled out using the Yelp SDK to find reviews

Areas For Improvement

How Can This Be Improved

- Improve packaging
  - Build and publish our **UI** and **API** modules as npm packages, we could use GitHub package functionality to publish the artifacts to respective private repositories
  - Build and publish our container images to a private registry
- Complete the review component
  - This was an intentional limitation that resulted from the avoidance of introducing secrets/sensitive materials
- Login functionality
  - I feel like some value is lost If users are not recognized in this system, able to comment, leave reviews and otherwise enhance the truck data. This was a stretch goal and I could not think of how to complete this within the constraints I set.
- Think a little more about the CLI
  - As an afterthought, I questioned if there would be value added from using the CLI as another entry point to the API
    - I battled with this as there is also value in decoupling these two
    - In any event, I am including methods from the api in the cli project, which is an anti-pattern and can cause confusion.