Challenge Tech Doc

Back-end

Summary

Rails was chosen for back-end development due to its ease of use and scaffolding capabilities. This made developing the API for the single endpoint `/orders/checkout` quicker. In addition to the `Order` table, a `User` table was also created.

Assumptions I took

- I assumed a fixed bag price of $5.90. If this wasn't the case, an `OrderItem` table might be necessary, which would have a `has_many` relationship with the `Order` table. This would allow for a method like `full_price` in the `Order` model to calculate total order price.

Beware of state logic and how it's transitioning.

- Order `state` and `price` are secured on the back-end. The `state` is managed by the checkout service functions, while the `price` is determined by the `quantity` sent from the client app. These fields aren't updateable by client app values and are defined in the private `_params` methods in the `orders_controller.rb`

How I mimic an error to happen 50% of the time, so we can see the error state of the flow.

- The back-end simulates an error 50% of the time, using the `CheckoutService` and the `.sample` method on an array with `success` and `failed` states. This way, the client app can handle both successful and erroneous order states.

Future Improvements

- Given more time, I would have created a `MockPaymentService` for better payment handling and an `OrderStatus` table to manage order state transitions more effectively.

Front-end

Summary

The front-end application was developed using React and Vite. Additional packages such as Axios, Formik, Yup, Tailwind, Lodash, and React Router were used to handle API calls, form validation and submission, CSS styling, and routing.

Folder Structures

The application follows a feature-based pattern. The `Components` folder contains feature-specific components and reusable `elements`. Formik-injected components are separated into a `formik` folder to adhere to the Single Responsibility Principle (SRP). The `API` folder contains endpoint files. The `Hooks` folder contains the `useSend` hook for POST requests, with more method-based hooks planned for future requests. The `Pages` folder contains main components for each route, and would be restructured into entity-specific folders as the project grows. The `Router` folder contains the `routes.jsx` file for defining routes and components. The `Utils` folder contains helper functions and constants

Tailwind Config

Colors from Figma were added to the Tailwind config, named according to context (e.g., `red` is `error`, `blue` is `normal`). A safelist was added to keep dynamically generated dividers colors during the PostCSS build.

TSConfig

The TSConfig was used to set up absolute imports, which are important for simplifying import statements and improving code maintainability.