Front-end Engineering Challenge

Onsite Coding Challenge

Background

At City Storage Systems, we assist in the production and fulfillment of online food orders. In the process of creating and delivering these orders, some common workflows involve processing and displaying large amounts of order events — when orders are placed, prepared, delivered, etc. These orders and events can be displayed on interfaces used by cooks, delivery drivers, in-house engineers, and more. You will be building a (simulated) system to help facilitate this.

Challenge Prompt

Build an in-browser application that displays Orders in real-time as they are delivered from a provided server API, and allows users to search their Orders by price.

During the deep dive following the challenge, we'll check out your real-time simulation in action, collaboratively explore your code, and augment, fix, and iterate on your solution together.

See the following *Challenge Details* section for full specifications.

Challenge Details

User Interface

The UI should have at least basic styling - we aren't looking for a masterpiece, but DOM elements need *at least* minimal styles applied. Please do this yourself instead of relying on off-the-shelf UI component libraries (see *Third Party Libraries* section). The exact UI and UX implementation and aesthetics is open-ended, but consider the overall usability of the interface when constructing it. Your app should meet the following general requirements:

- Display all received Orders in a tabular presentation
    - Orders should display all relevant user-facing data
- A search input should be displayed above the Orders table

Receiving Order Events

You should have received an email containing a small Node.js server. For this coding challenge, Order Events will be pushed to your application by the provided NodeJS server using Socket.IO. You may modify the server during development if you deem necessary, but you may not slow down the rate of order delivery in any way**.** A quick run of ***`npm i`*** and ***`npm start`*** in the extracted directory should start streaming Order Events on ***`localhost:4000`***.

Order Events & Orders

The Order Events are emitted from the server in batches every second. As the server runs, multiple Order Events will potentially share the same ID, denoting they represent the same Order at different points in time. An Order Event will have the following format:

```
{
   "id": "fbd68485",                 // Order ID, not event ID
   "event_name": "CREATED",          // Current order event state
   "price": 1026                     // Price in cents
   "item": "Al pastor tacos",
   "customer": "Jasmine Carroll",
   "destination": "23788 Cook Groves Apt. 743, Cooperberg, CA 94603",
}
```

As orders are ingested, if an existing order with the same ID is already rendered on the page, its data should be updated. For example, if an order with ID 2 already exists as a row in a table, and another order event with that same ID arrives the existing row should be updated to reflect the data from the new order event.

Search

Users should be able to search their received Orders by dollar amount, which should display the matching Orders as well as the count of matching Orders. Searching should happen as a user types; it should not require a dedicated click/enter-key to execute the search. Both the Order results and count should change in real-time as new orders are delivered via the Socket.IO connection.

When a price search query is entered, you must display Orders that match the specified price in dollars as well as a count of the number of matching Orders. The matching heuristic is up to you.

3rd Party Libraries

Usage of third-party libraries is acceptable, but there are a few rules:

- React, Vue, Angular, and other similar application frameworks are allowed
- CSS processors and CSS-in-JS libraries are allowed
- Renderable UI components from other libraries are **not allowed**. If it renders pixels to screen for you, do not use it. This includes css-utility libraries that are opinionated about UX and style choices but do not necessarily provide components (e.g., tailwind CSS).
- Libraries that provide common algorithmic functionality are also **not allowed**. These would include most utility libraries that work around vanilla JS (e.g., lodash, jquery, etc).
- If you have questions on a specific lib, ask your interviewer before using.

Be prepared to explain and defend your library choices.

Deliverables

At the end of the allotted time, we're looking for a web application running on your machine that meets all of the above criteria. You must use either Javascript or Typescript for this challenge. You are welcome to use any frameworks, IDEs, and openly-available project bootstraps, or none at all. Do your best to deliver a quality solution that shows off your abilities and point of view.

What We Look For

This challenge is meant to help us see your best code and to allow you to showcase your judgment when addressing an engineering challenge. When we evaluate the challenge, we look at:

- How focused and successful you are on meeting the requirements
- The robustness of your application architecture, including state management
- The optimality and correctness of your data structures and algorithms
- The performance of your application
- The overall UX/UI of the app