Coding Challenge
Backend API Call

Introduction

This challenge is an opportunity to demonstrate proficiency in the skills we utilize here at Presto. It's understood that there are multiple solutions to the problem, so please document your development flow and thought process in a well organized README.

Expectations

Organize your code inside a git repository. When finished, commit and push your code and email us a link to your repository.

README

Include the following items in your README:

- Description of the problem and solution.
- Reasoning behind your technical choices.
- Trade-offs you might have made, anything you left out, or what you might do differently if you were to spend additional time on the project.

Commit History

Use whatever development workflow works best for you. If your solution is small enough and a single commit is justified, that's fine; we just ask that you keep your commit history as coherent as possible.

Task

Utilizing your language of choice, design a RESTful web service API call that will return a list of menu items for a specific restaurant.

Specifics

For this API call, the front end will target path `/restaurant/:restaurantId/item`.

```
public Observable<Response> execute(Request request) {
      ...
      URL url = new URL("https://api.presto.com/restaurant/1/item");
      HttpURLConnection con = url.openConnection();
      ...
}
```

The expected response will be parsed directly from JSON to a list of Item objects.

```
...
execute(request).subscribe(response -> {
      Type listType = new TypeToken<List<Item>>(){}.getType();
      List<Item> items = new Gson().fromJson(response, listType);
});
...
```

Requirements

This challenge is centered around the concept of communicating throughout the stack from the backend, so each part of that communication stack is required:

1. Handle the request from the front end
2. Connect to and query from a Postgres database instance
3. Send the response to the front end

The data model structure is up to you, it's assumed that the frontend will mimic your object layout for seamless parsing. The only requirement is that the `Item` object must contain a list of modifiers (ex. Side item, Topping, Sauce). Hint - modifiers can also have modifiers (think dressing choices for side salad).

Scalability is paramount. The code you build should not end up being a bottleneck down the road (will this have to be rebuilt when it's being hit 100x as frequently?).