Final Report

Equifod Group C

Riley Comer Jordan Colledge Joonsik Kim Jordan Ribbink

Table of Contents

Handoff guide

Overview

It is important to understand that the Equifood App is comprised of multiple sub-projects which are all distributed as a part of the monorepo. This is an NX monorepo and all syntax for interfacing with these projects (i.e. running particular actions) will be according to the NX standards.

The three applications that exist are currently **Equifood Customer**, **Equifood Merchant**, and **Equifood API**.

Additionally, some shared libraries exist for code-sharing between these apps.

All code is strongly-typed and written in TypeScript for the benefits of static analysis and error prevention, as well as a more convenient developer experience.

Frontend

There are two frontend apps for this project.

1. The customer facing app
2. The merchant/restaurant facing app

NOTE The frontend requires an `.env` file to be created (use `example.env` as reference)

Customer App The client-facing frontend for the project (React Native) can be found in the `apps/customer-app` directory.

The app is supported as an IOS/Android application.

NOTE The Equifood Customer App requires an `.env` file to be created (use `example.env` as reference)

Merchant App The merchant-facing frontend for the project (React Native) can be found in the `apps/merchant-app` directory.

The app is supported as an IOS/Android application.

NOTE The Equifood Merchant App requires an `.env` file to be created (use `example.env` as reference)

Backend

The backend for the project (NestJS) can be found in the `apps/equifood-api` directory.

Within the backend/API an admin portal endpoint is exposed for the administartor users (`/admin`). For instance, if your API is running on http://localhost:3333, then your admin portal would exist at http://localhost/admin.

NOTE The backend requires an `.env` file to be created (use `example.env` as reference)

Shared

Shared types for the frontend and backend can be found in the `lib/api-interfaces` directory.

These types are shared between all of the TypeScript apps within this project for easy code reuse,

Additionally, UI components, auth flows, and various other hooks/providers are shared in the `lib/ui-shared` directory. These are shared between the two React Native apps (Equifood Customer & Equifood Merchant).

Usage

Installation & Configuration

1. Install Node JS v16. This task can optionally be completed using Node Version Manager.
2. Ensure you have C/C++ build tools installed on machine (i.e. gcc). Bettersqlite uses node-gyp and is dependent on these. Visual C++ Redistributable may be required for users on Windows. If you encounter issues with compiled binaries, please consider running the `npm run rebuild` command.
3. Run `npm install`
4. Create a configuration file `.env` in the `apps/equifood-api`, `apps/equifood-customer`, and `apps/equifood-merchant` directory (use `example.env` for reference)

Development

Apps can be served using `nx serve APP_NAME` command. Currently available apps are `equifood-api`, `equifood-customer`, and `equifood-merchant`.

Remember, for either the customer or merchant application to functional, the backend must be running and properly configured in the respective `.env` file.

Currently, there are 3 apps that can be launched:

1. Equifood Customer: `npx nx run equifood-customer`
2. Equifood Merchant: `npx nx run equifood-merchant`
3. Equifood API: `npx nx run equifood-api`

Testing

1. Run `npx nx test PROJECT_NAME` in order to run all tests for the desired project. If more information is needed, please refer to the NX documentation.

It is noted that unit tests are somewhat minimal in the project as it would have been difficult to write a thoroughly tested application in the very limited time provided. Some features (algorithmically complex) are tested, whereas large parts of the frontend are not. Ultimately, mocking numerous API requests and updating these as the app evolved would have been very tiresome and detracted from development.

It would be beneficial to test backend features, not only using unit tests, but also using manual testing techniques. There are some aspects concerning security which should be very closely monitored (i.e. behaviour with regards to uploads, just about anything in the `AuthModule` and especially all social logins, etc.).

While we have tested for many edge cases in terms of the behaviour of the app, it would be important to conduct an independent QA deep dive in order to ensure that the app performs as expected.

Documention

Relevant documentation for this project exists in the `documentation` folder

Within this folder all technical specifications for the software, IP agreements, configuration instructions, and any other pertinent information can be found.

These are mostly written in markdown, however some exists in the form of PDF files.

Further Work

While this is a reasonably functional app, there exists some components which might make it a poor candidate for the App Store or Google Play in the current state (or just features that still need further implementation).

Any future work that we have foreseen or ran out of time to get to exists as an issue in our GitHub repository. While most of the further work can be found here, we thought it could be helpful to shed more detail on a few special considerations.

UI Work

Some of the UI has not been polished (i.e. `ItemListScreen` on Merchant App, `MerchantScreen` on Customer App, other aspects which can quickly be sighted in a quick review).

Database Work

The API has been generally developed with `better-sqlite3` in mind. While the database type can in fact be configured in the `.env`, it would be prudent to ensure the code complies with a modern, scalable database such as `MariaDB`. Some data types may not be supported on such databases, or data types that are encoded as VARCHAR/TEXT in `better-sqlite3` may have beter alternatives in other databases. `better-sqlite3` was deployed for the relative ease of use within a NodeJS environment for quick development, but better alternatives exist.

File System Work

The file system undoubtedly needs a bit of an overhaul. While a large part of the infrastructure exists for placing uploads inside of something like an S3 bucket or other cloud provider, it is currently only configured for local storage. A small amount of work would need to be done in order to ensure that these files are outsourced somewhere accessible somewhere with adequate redundancy, space, and the ability to access without bottlenecking the API.

Testing

First and foremost, it would be important to stress test this app using a sufficient number of users in order to ensure it scales well. Beyond this, the test coverage of this application could probably improve (especially with regards to the more complex backend situations). This was also mentioned in a segment above. There are a few security considerations that also need to be tested/implemented if this app was going to be deployed to the market. Any of the considerations we have had exist in our GitHub repository.

Required maintenence/upkeep

None! We have no third-party subscriptions or logins. Everything is hosted locally as of now and this can be deployed to a third party server/provider as needed for your use case.

Note: You will need to generate Google Sign-In and Facebook Sign-In tokens for the social logins. These will need to be configured in the `.env` file of the Equifood Customer App.

Relevant Resources

Link to GitHub repository: https://github.com/jribbink/equifood

Link to Promo Video: https://github.com/jribbink/equifood/blob/master/docs/promo-video.mp4