PayEngine Interview Coding Test

Introduction

PayEngine is a payment processing platform for SAAS providers. Our customers can use and integrate PayEngine's payment processing capabilities in a variety of different ways from minimum coding with Web-Components to a fully API-based integration and everything in between. This flexibility allows us to support almost any technology stack our customers decide to use with no changes to our products.

Objective

The objective of this mini-project is to help us understand your level of experience across the various layers of a full-stack environment. The project mimics what PayEngine's customers do to integrate onboarding and payment processing capabilities using PayEngine following our guides located at https://docs.payengine.co/payengine-developer-docs and https://docs.payengine.co/payengine-api-reference/.

Partners typically focus on the following three areas:

1. Merchant onboarding: collecting the key user data (**for this exercise you only need to implement this step of the workflow**).
2. Transacting: processing credit card and ACH payments.
3. Reporting: viewing payment history, details, and other summary graphs.

Project Structure

The frameworks expected to be used within the respective layers are the following:

- Server: nodejs, express, knex, axios
- SPA: react, axios
- Database: postgres
- **Bonus**: containerization: docker and docker-compose (this will also make it easier/faster for us to run your app when we review it)

The following is an example structure project structure of folders/files for this project. :

- **/public**
- **/src**
    - **App.js:** A default file created by CRA. This is the main file that includes the Components/Onboarding react component created by you.
    - **index.js:** A default file created by CRA
    - **/client**: Contains your client-side React SPA.
        - **/components**
            - **App.js:** Should show onboarding form using our web component. e.g.:
                - `<pay-engine type="boarding" ...>...</pay-engine>`
                - See our documentation: (https://docs.payengine.co/payengine-developer-docs/merchant-onboarding/embedded onboarding-ui)

        - **NOTE:** This component should contain the main logic of the client-side. You will need to embed conditional logic to retrieve user data from your server-side/db implementation to either create a new merchant or retrieve an existing one in order to load the application. You may choose to create a merchant once and then hardcode the ID in your application, or you might create extra options to create new merchants, maintain a list, switch between them, etc. (we recommend hardcoding first, then making it more versatile later).
    - **/server:** This is expected to be a clean nodejs project, utilizing ExpressJS in addition to other frameworks mentioned below)
        - **/migrations:** This should at least include one migration using knex, which should create a `merchants` table that will at the very least hold the merchant's name, email, and PayEngine merchant_id.
            - NOTE: You may choose to seed the table with a merchant row (especially if you are hardcoding the id in the onboarding widget).
        - **/api, /services, /etc…** You can arrange the project as you see fit. We recommend keeping it simple to start so that you can finish an MVP, then extend it as you wish.
- **dockerfile:** This is expected to be a multistage dock build process. It should first build the react project, and then include those file as part of express static files to be served. Feel free to look and follow any online guides for best practices for "containerizing a nodejs + react applications"
- **docker-compose.yml:** This is the system that will run the postgressdb and also the nodejs+reactjs container that was built above. Feel free to use any online resources for best practices on structuring the system. At a minimum, it's expected that this will include a reference to two containers as previously mentioned.

Goals.

1. Application must use PayEngine's onboarding widget to provide a merchant's onboarding flow.
2. Application must demonstrate use of npm libraries such as express, knex, and axios at a minimum.
3. Application must use Node.js and Express to developer supporting services.

Other Constraints and Recommendations

1. For the front-end of the application, developers must use React.
2. The application should use standard Javascript/npm libraries like Axios and Express.
3. PayEngine relies heavily on containerization, so please demonstrate your skills by creating a containerized app. This means that you can ultimately run the entire project.
4. It is recommended to keep it simple and standard so we only focus on how you write code. Having a nice UI and graphics is not a requirement.
5. Use Postgres.
6. Minimal migration demonstration should be implemented using KnexJS.
7. GitHub or Bitbucket should be used to deliver your project with history intact. **NOTE: You must make the repository private, and when you are ready to share it please send an invite to the reviewer (recruiting team will specify the correct email).**
8. If you prefer TDD as a development approach, it will be a great skill to demonstrate on this project.

Important Links

1. API Docs https://docs.payengine.co.
2. Developer's Documentation https://docs.payengine.co/payengine-api-reference.

3. PayEgine sandbox https://staging-sandbox.payengine.dev.

An account will be provided to you so you can create your own API secret and public key, you can also see your onboarded merchants.