cloudshelf

Technical Exercise

As a Junior-Medium level full-stack engineer, you will be contributing to both the frontend and the backend of our product.

This exercise is no different: you can choose to complete the frontend exercise, backend exercise or both!

We suggest that you follow the following time constraints for this exercise:

- If you only work on the frontend: 3 hours.
- If you only work on the backend: 3 hours.
- If you work on both frontend and backend: 6 hours.

Regardless of which side you choose to work on, please include the following:

- At least one unit test
- A readme document which describes how to run your project and briefly explains the main features
    - If you cannot complete the exercise in the suggested time limit, please provide a short technical description which includes the information of what's missing, how you would have built it, and your thought process of how you came up with your solution

And ensure that you write your solution using **typescript** (not javascript!)

Remember: we are trying to get a feel for how you work as a developer. If you find there is a feature that you are really struggling with, work on the other features first and then come back to it - don't let it take up all your time.

Take note that we use `yarn` and not `npm` across all of our projects.

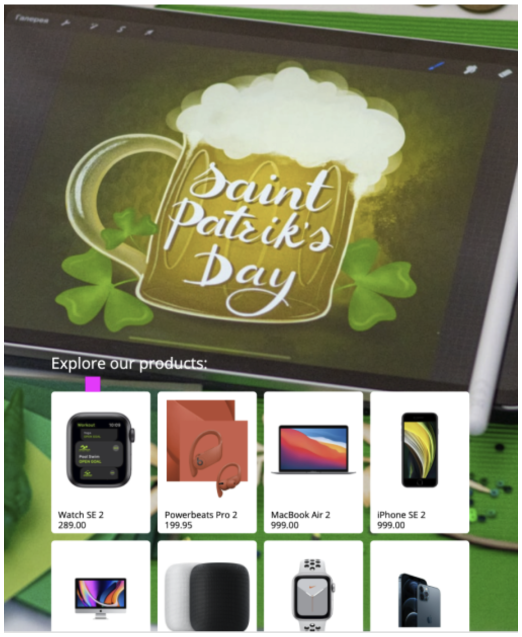
Frontend

You have been provided with a basic Next.JS/React/Typescript application. Your task is to display a list of products retrieved from the Shopify Storefront GraphQL API in a responsive grid. A graphql codegen script has been included and can be invoked by running `yarn run codegen`. You should continue to follow the object-oriented programming approach set out in the initial project.

The requirements are as follows:

- The page should initially display 10 products
- When the user scrolls, and the end of the grid is reached, more products should be loaded
- Each grid item should include a product picture, the product name, and its price
- When an item is selected (clicked), the details of that product should be displayed (including the HTML description)
- The grid should be responsive, ideally to work nicely on desktop and mobile devices
- Do not write any mutations or modify any data in shopify. Queries only!
- Do not use a UI Framework.

An example design for the grid is provided below.



The purpose of this exercise is not to build the most aesthetically pleasing user interface, but to demonstrate your qualities as a software engineer to produce a well architectured, easy to extend, testable, frontend application .

Hints

- Try to take advantage of NextJS and Server Side Rendering

- Latest version of Chrome will be used for testing
- Responsivity can be tested with Chrome's developer tools
- API Documentation: https://shopify.dev/docs/storefront-api/getting-started
- Use the following credentials for the API:
  Shopify domain: `dev-cloudshelf.myshopify.com`
  Storefront Token (Access Token): `e78aab9ceb653dc9c6748c32da2bc114`
- If you decide to build the backend, you can use your own API instead of Shopify's to fetch the products and their details. You will need to modify the codegen configuration to support this.

Backend

You have been provided with a NestJS backend project that exposes a GraphQL API for managing products.

Assuming you are only completing the backend section of this exercise, it is completely acceptable to run your GraphQL queries and mutations through Apollo Studio.

You should continue to follow the Object Oriented Programming approach set out by the code you have been provided, if new functionality is required you should demonstrate a good level of abstraction, and adhere the SOLID principles to produce easily maintainable and testable code.

In the context of this project a Product should consist of the following properties:

- A name
- A price
- A description
- At least one image (images can be stored as a URL)

Please complete the following tasks:

1. Ensure products can be created
2. Ensure products can be deleted
3. Ensure products can be retrieved as a paginated list
4. Ensure a single product can be returned
5. Ensure all mutations can only be run by an authenticated user
6. Ensure the user of any given endpoint is returned a relevant error if a problem occurs.

The following credentials can be used to connect to your dockerized database with TypeORM

- Host: `localhost`
- Port: `5432`
- Schema: `cloudshelf-junior-exercise`
- User: `cloudshelf`
- Password: `cloudshelf`
- Entity Directory: `dist/**/*.entity.ts`

Hints

- The NestJS documentation is usually good, and will be helpful if you have no experience with this framework.
- A docker-compose file has been included in the project and can be found in the `/tools/docker` directory; this will configure redis and postgres for you.
- You can start the project by running `yarn start` in your terminal.
- You can ensure the project is correctly running by visiting `http://localhost:3100/hello-candidate` in your browser of choice.
- Authentication has not been provided for you.

- It is your responsibility to ensure that the project correctly follows the specification outlined above, you should not assume all the code that has been provided to you is correct.