Project Template

Objective

Build a small application E-Commerce SaaS which consists of the following elements:

- Front-End built in preferably React (Typescript)
- Back-End REST API

Very Important: Any and all technologies, languages and frameworks can be used (this includes ChatGPT). However, keep in mind we will be reviewing your code and will interview you in order to see whether or not you understand the solution.

PLEASE read this document carefully, for questions please contact Gerrit (gerrit@andilesolutions.com)

Frontend Goals

The following features must be included in your web application:

- Must be able to add customers
- Dashboard containing a list of products
  - All products must be listed on this page, however, they must also be paginated.
  - Products must be able to be deleted from here in bulk (selectable products)
- Product details page
  - This is a page to view details for a single product
  - On this page you must be able to edit and remove the product
- Basket Page & Checkout Page
  - The basket page must contain all products that a customer has decided to “wish list” before confirming the purchase
  - The Checkout page contains the order confirmation
- Past Orders
  - Any orders done in the past must be saved for historical purposes

- These orders must be searchable

API Goals

- You must have the following REST endpoints:

Endpoint	Method	Body	Response
`product`	POST	Product Body	Product Body
`remove-products`	POST	string[] (a list of ids)	200 OK
`products`	POST	(optional) string[] (a list of ids)	`Product[]`
`product/{id}`	GET	NONE	Product Body
	PUT	Product Body	Product Body (Updated)
`order`	POST	Order Body	Order Body
`orders`	POST	{ ids str[] (optional) customerId str (opt) }	`Order[]`
`order/{id}`	PUT	Order Body	Order Body (Updated)
	GET	NONE	Order Body
`customer`	POST	Customer Body	200 OK
`customer/{id}`	DELETE	NONE	200 OK

Structure Bodies

1. Product

```
{
  "id": string,
  "name": string,
  "description": string,
  "price": number
}
```

2. Order

```
{
  "id": string,
  "paid": boolean,
  "customerId": string,
  "products": string[] (list of ids)
  "total": float
}
```

3. Customer

```
{
  "id": string,
  "name": string,
  "email": string,
}
```

Outcomes

1. FrontEnd UI that can interact with the Backend API
2. Backend API must communicate with both UI and a Database system **must be running on port 5000** in order to work with testing by default.
    a. The provided postman collection will be used for testing purposes (feel free to expand on this if you want.)
    b. **IMPORTANT:** Note this includes unit testing of your services/functions.

Extras

- UI Authentication
- CI/CD
- Hosting of application
- Etc.

What am I being reviewed on?

- Quality over quantity.

- While it’s always great to have more features and go the extra mile, generally in production it’s a lot more important to have a clean and well designed product. This means that clean code and properly implemented patterns will count more in your favour than making sure you have every last feature implemented.
- Pattern usage
- Cleanliness of code
- Complexity and Readability
- **Pro Tip:** In production you can always take a day or two extra to implement a proper solution, but technical debt could result in unreadable code and having to redo the implementation further down the line.

Solution Submission

1. Provide links to both the Front- and Back-End repositories.
2. Include steps to run these applications (this includes the steps required to set up your database).
3. Also include steps to run the testing.