Background:

Imagine you are working on an e-commerce platform with products and customers managed by other teams which you can read from an API (See OpenAPI spec in *`ControlPlane - e-commerce Interview - API.json`*). Customers and products can change over time, outside of your control (updating addresses, updating prices, descriptions, etc.). You're responsible for building the purchase flow for the platform.
There's also a Shipment API to create a new shipment of the product to the customer's home.

For simplicity, customers pay for purchases with a credit balance that is pre-assigned, but it's your responsibility to keep track of this balance throughout their purchases.

Description:

In **Typescript**, please design and implement a simple e-commerce server that exposes the following operations:

- Grant/deduct credit balance
- Get a customer's credit balance
- Purchase a product
    - As part of this operation, call CreateShipment to send the customer the product they just bought.
    - If the CreateShipment call fails, the purchase should not be saved, customer credit should not be deducted and the caller should get an error message.
- List product purchases
- Refund a purchase (fully or partially)

Please also design the data structure for the purchase and credit entities (and any other additional entities you see fit) and manage them end-to-end (CRUD).

For your testing, I recommend you mock the API that serves you Customers, Products and Shipments.

Notes:

- Please use types in your TS code. You don't have to use ESlint, but try to include return types on functions and generally avoid `any`.
    - API requests and responses should have explicitly defined interfaces.
- Since this system handles billing data, consider the data model's auditability and the importance of historical record keeping. For example: the ability to tell when credit balances changed and for what reason.
- If any additional components are required to run your code (databases, message brokers, etc.), please include a docker or docker compose file to run them in a container.
    - Assume node.js, npm, yarn and docker are installed
- Your thought process matters. If you have to ignore or sidestep something you'd normally address, please explain why you did and how you'd approach it in real life.

- If you implement the Customer, Product and Shipment APIs to make testing easier, please remember it's an external API and not an internal part of your system.
    - Reading Customer and Product data - for example in the "Purchase a product" call - should be done over an API call, not directly from your persistence layer
    - Creating shipments should be done using an API call, not directly into your persistence layer

Bonus:

- Implement a simple internal react application (feel free to use ClickUI) for a customer service rep to:
    - Navigate through customers' purchases
    - Refund purchases
    - Manage customers' credit balance
    - This might require implementing additional APIs
- Cache customers and products to improve performance without compromising accuracy of details consumed.
- Include some e2e tests
- Handle promo codes