1. Decomposition by Business Capability

Based on the core functionalities of an e-commerce platform for second-hand goods, we have identified and decomposed the system into the following independent microservices. Each service owns its data and logic.

Business Capability	Microservice Name	Responsibility	Owned Data (Entities)
Identity & Access	**Identity Service**	Manages user registration, authentication (Login), and profile management.	Users, Roles, Tokens
Catalog Management	**Catalog Service**	Manages product listings, categories, and inventory status (New/Old).	Products, Categories, Images
Buying & Checkout	**Order Service**	Handles shopping cart, order placement, and order history.	Orders, OrderDetails, Cart
Notifications	**Notification Service**	Sends emails/SMS for order	Templates, Logs

		confirmation and status updates.	

2. Service Contracts

To ensure loose coupling, services interact only via standardized RESTful APIs. Below are the API definitions for the key services.

2.1. Catalog Service API

This service exposes endpoints for browsing products (The functionality implemented in Lab 3).

Method	Endpoint	Description	Request Body	Response
GET	/api/v1/products	Get list of all available products	N/A	200 OK (List JSON)
GET	/api/v1/products/{id}	Get details of a specific product	N/A	200 OK (Product JSON)
POST	/api/v1/products	Add a new second-hand item	{name, price, status...}	201 Created

2.2. Order Service API

This service handles the purchasing process.

Method	Endpoint	Description	Request Body	Response
POST	/api/v1/orders	Place a new order	{user_id, items:[]}	201 Created
GET	/api/v1/orders/{id}	View order status	N/A	200 OK

3. System Communication Design (Thiết kế giao tiếp hệ thống)

In the Microservices architecture, ShopSphere uses a hybrid communication strategy:

1. **Synchronous Communication (HTTP/REST):** Used for client-facing interactions where immediate feedback is required.
    - *Example:* When a user views the product list, the Frontend calls the **Catalog Service** via HTTP GET. The user waits until the data is returned.
2. **Asynchronous Communication (Message Queue):** Used for backend processing to decouple services and improve performance.
    - *Example:* When an order is placed successfully in **Order Service**, it publishes an event OrderPlaced to a Message Broker (e.g., RabbitMQ). The **Notification Service** consumes this event to send an

email. This ensures the user doesn't have to wait for the email to be sent before seeing the "Order Success" screen.

4. C4 Model - System Context

The following diagram illustrates the high-level interaction between the ShopSphere system and external actors.