Problem Statement for a Mock Exam: Poppo's Recommendation System

Poppo, a (fictional) convenience store chain that has been in business since the early 1980s, is experiencing a gradual decline in customers at several locations due to shifts in consumer behaviour.

In an effort to move towards a digital retail model, you have been asked to create a recommendation system powered by artificial intelligence (AI). This system will be fueled by a large amount of data on customer preferences that Poppo has gathered over the course of 35 years. The main requirement for this system is adaptability, meaning that the database of consumer preferences and the AI algorithms should be separate from one another. This ensures that any updates or modifications to these components can be done independently.

Based on the above information, your tasks are the following:

1. Compose a system with a maximum of 10 microservices/components via a structural diagram (e.g. component diagram). Elaborate how and why your composition can achieve the adaptability requirement above.
2. Elaborate which microservices communication pattern(s) help achieve the above adaptability requirement. Illustrate a case where one of the selected communication pattern(s) can help with the above adaptability requirement via a behavioural diagram *(e.g. communication diagram or sequence diagram)*.
3. Elaborate on your deployment decisions and testing strategies that support the above adaptability requirement.
4. Explain how and why your cloud migration strategies support the above adaptability requirement.

* Q1:Decomposition : Did you use Event Storming or Domain-Driven Design to decompose the system? Have you considered “Born to Die” or “Born to Grow” when decomposing them? Any design pattern that helps with the problem statement’s requirement?

* Q2:Communication : Did you use Request & Response, Event-Driven, Through Data or a combination of them? Any design pattern that helps with the problem statement’s requirement?

* Q3: Deployment :How did you use the repository/-ies (Mono or Multi-repo)? Did you use a Nested Container? How? If so or if not, why? How did it help with the problem statement’s requirement?

* Q4: Cloud Migration Strategy/Strategies :Which cloud services did you use (PaaS, FaaS, IaaS)? What types of cloud services did you use (Public, Private, Hybrid or Multi)? How did you migrate to them (Re-platform, Re-factor, Retire, Retain, etc.)? How did it help with the problem statement’s requirement?

Decompose as Microservice

Based on User,

1. Customer Data : To stores and manage customer data(preferences, purchase and search history)

Based on Recommendation,

2. AI Recommendation Algorithm : To generate personalized recommendation based on customer data

Based on Product,

3. Product Catalog & Inventory: Stores information about available products and their inventory status. No need for encryption as product details are for public display.

Based on Interaction,

4. Behavior Tracking : Tracks user behavior (click, view, search, purchase) and collect feedback to improve system

Based on Customer Notification,

5. Recommendation Delivery & Notification: Delivers personalized recommendations and sends notifications to users (special offers or product updates).
6. Front-end: To display info, Product info and Transaction, and interact with the Users.

So by using the DDD concept we can manage the structure of each microservice and it can be independently developed without affecting others.

Diagram

Born to Grow (Most Independent + can be improved/scale + long-term)

Customer Data
This service will grow in importance as it gathers more data about customer preferences, purchases, and search history, etc. It's essential to the system's adaptability, as more data leads to more personalized and accurate recommendations.

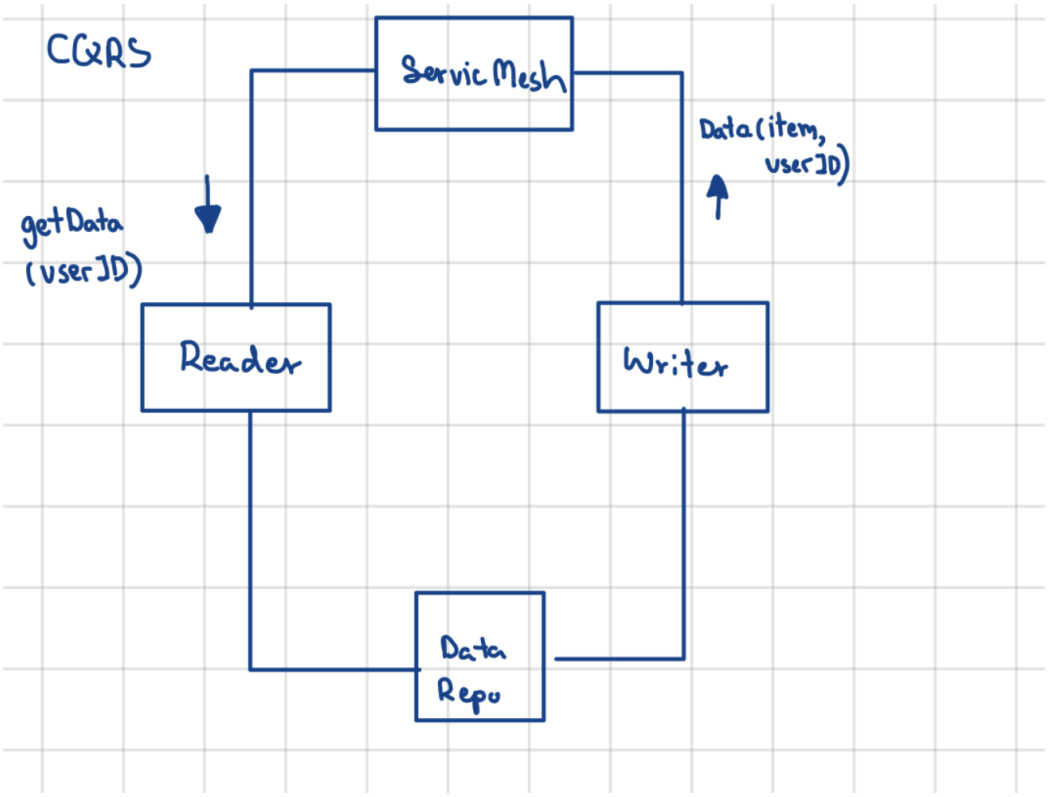
AI Recommendation Algorithm
It’s can be upgraded by adding other algorithms or the more model, the more accuracy

Recommendation Delivery & Notification
Increased customer engagement will demand scalability, especially during peak times.
Ex .During Black Friday, the system sends personalized offers in real-time to thousands of users. This ensures that the platform remains operational under heavy load.

Born to Die
Behavior Tracking
The system only tracks user behavior during a single session (while the user is active on a webpage or app), this tracking session becomes irrelevant once the user leaves the platform.

CQRS Pattern : Separating reads and writes makes the system more **adaptable**, allowing independent scaling and updates. Ensures that Data Repo operations are fast, secure, and scalable. Moreover, other parts of the system are still adapted as needed.
Read Model (User Data Repo Reader) : To fetch user data (preferences, purchase history, search history)

Write Model (User Data Repo Writer): To validate and store new user data or updates (like creating new accounts or updating preferences or data).



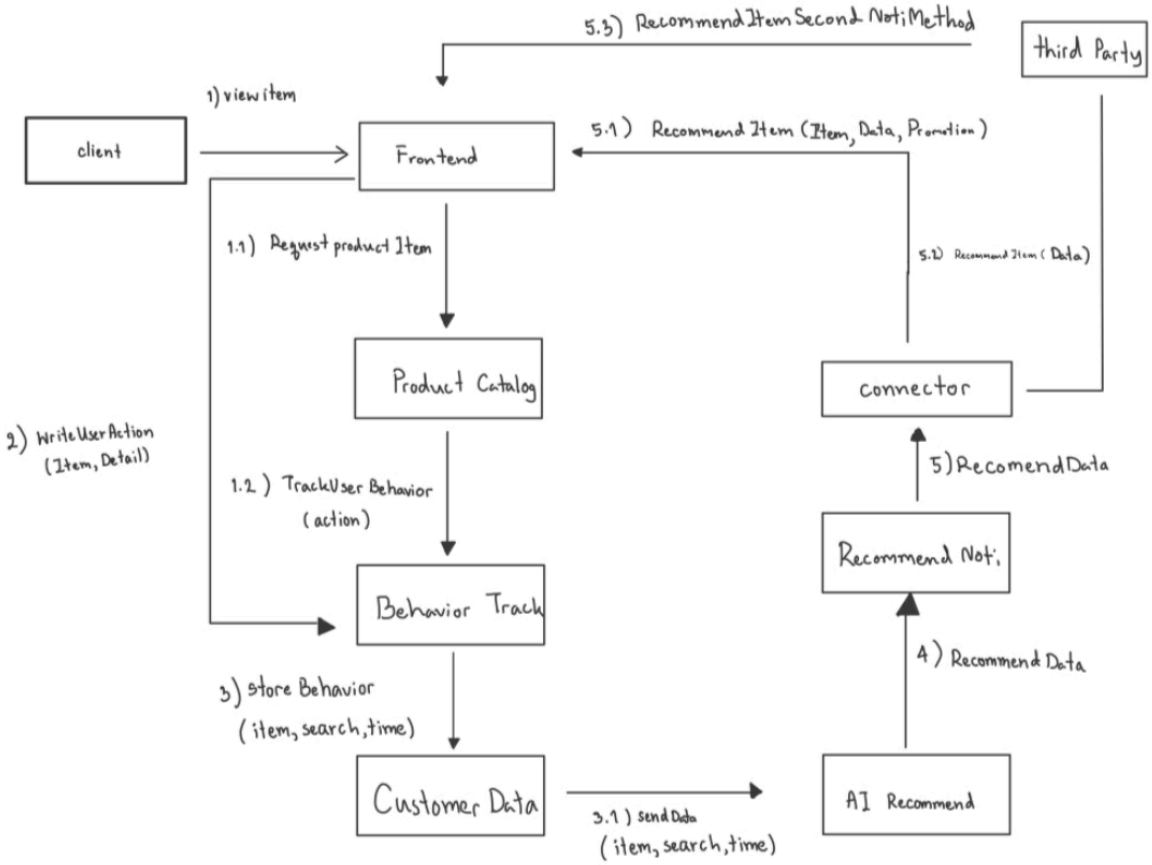
Communication

Request & Response: When the Recommendation Delivery Service calls a third-party notification provider, the system adapts dynamically to service failures. If the provider is unavailable, the Circuit Breaker triggers a fallback to alternate communication methods (like email), ensuring continuous operation.

Through Data: The AI Recommendation Engine processes user behavior data asynchronously, allowing the system to continue generating recommendations even if the Behavior Tracking service is delayed or temporarily unavailable. This decoupled communication ensures the system remains adaptable under different conditions.

Handling Failures with Circuit Breaker and switch to other notification method

- When the third-party notification provider is unavailable (returns a 503 error), the Circuit Breaker stops requests temporarily, preventing system overload.
- Ensures that the system remains operational even when some parts are down.
- If the notification provider is down, the system can quickly switch to alternate notification methods (email instead of SMS) without manual intervention.


5.3) Recommend Item Second Noti Method
third Party
1) view item
client
Frontend
5.1) Recommend Item (Item, Data, Promotion)
1.1) Request product Item
5.2) Recommend Item (Data)
Product Catalog
connector
2) Write User Action
(Item, Detail)
5) Recomend Data
1.2) Track User Behavior
(action)
Recommend Noti.
Behavior Track
4) Recommend Data
3) store Behavior
(item, search, time)
Customer Data
AI Recommend
3.1) send Data
(item, search, time)

Deployment

Pattern Multi-repo

It allows for the independent development and scaling of microservices, making them easier to manage and modify by isolating services such as the AI algorithm, Behavior Tracking, and Notification Service. Each service can have its own policies to reduce cross-service vulnerabilities or conflict bugs when updating the service, ensuring that changes made to one service, such as AI algorithms, do not affect others. Additionally, the AI algorithms will not directly impact the database or customer data, ensuring that each data component can be independently updated without interfering with other services or causing disruptions.

No Monorepo?

it keeps all services in one place, which can make it harder to manage updates for each service separately. For example, if we change something in one part, like an AI algorithm, it might accidentally affect others.

Docker k8s + Nested container

I plan to deploy the system using containers (Kubernetes). Containers provide isolated environments for each microservice, ensuring that each service can run independently and it can be adaptable later.

Weak

Using nested containers increases the complexity of the system. Managing and troubleshooting nested containers can become a difficult part for developers.

Unit Testing: Testing Individual Components

Test Case 1 (Customer Data Service)

- Type : Stub
- By using predefined static data, we can test the core behavior of the Customer Data service (CRUD operations) without dependency on external systems.If changes are needed (like adding new fields to user profiles), the service can be updated independently and tested in isolation, ensuring it adapts without disrupting other parts of the system.
- Goal: The Customer Data Service can evolve (e.g., storing more detailed customer data) without impacting other services.

Test Case 2 (AI Recommendation Service)

- Type:Mock
- Mocking the Customer Data Repo interaction ensures that the AI Recommendation Service can function correctly without requiring real customer data.
- This enables flexibility to update the AI algorithms independently, such as adding new models or algorithms, without affecting the Customer Data Repo.
- Goal: The AI service can be enhanced or modified over time, improving accuracy without interfering with other services

2. Integration Testing: Ensuring Inter-Service Communication

- **Test Case 1 (Request & Response – Notification Service)**:
  - Type: Mock
  - Mock: Simulate the third-party notification provider to test how the Recommendation Delivery Service behaves when the provider is unavailable (returns a 503 error).
  - Goal: Ensure the Circuit Breaker stops sending requests and triggers a fallback mechanism (switch to email).

- **Test Case 2 (Through Data – Behavior Tracking to AI Recommendation)**
  - Type: Stub
  - By providing **p**redefined behavior data, the test ensures that the AI Recommendation Engine can generate recommendations without depending on real-time tracking data.This makes the system flexible to operate even if the Behavior Tracking service experiences delays or downtime.
  - Goal: The system can continue functioning with previous data, adapting to service availability issues.
- **Test Case 3 (Through Data - When update service)**:
  - Type: Stub
  - Stub: Updated or Modify service
  - Goal: When updating service it not affect to other.

Services Recommended for Nested Containers:

1. Customer Data Repository

- Why: Since this microservice handles sensitive customer data, including preferences, purchase history, and personal information, it should be run in a nested container to enhance security. The nested container will provide an additional layer of isolation from the rest of the system.
- Adaptability: While security is increased, adaptability may be slightly reduced because updates to the Customer Data service might require more careful management. However, the security benefits justify the tradeoff in this case.

2. **Behavior Tracking**

- Why: Since this service collects and stores behavior data (like views, clicks, purchases) and can be resource-intensive, a nested container can help manage the data flow and limit potential breaches.
- Adaptability: As this service does not require real-time interaction with other services, placing it in a nested container will not significantly affect overall system adaptability.

Cloud Migration

PaaS

PaaS allows us to focus on building and deploying our platform without worrying about managing the underlying infrastructure. So it like a platform to deploy the application but we can still modify or update our app without any limitation so

Kubernetes clusters can run our microservices efficiently, with auto-scaling and self-healing features and modification which reach the adaptability.

Focus on developing and managing microservices independently without worrying about infrastructure.

- Automatically scales services up or down based on real-time demand, ensuring the platform remains accessible during peak traffic (e.g., Black Friday sales).
- Example: Creates new instances of the Recommendation Delivery Service when traffic spikes.

No SaaS and LaaS

LaaS

- Offers full control but requires a lot of effort to manage servers, networking, and scaling.
- Complexity for a microservices-based system since we need to handle manual orchestration, monitoring, and scaling.

SaaS

- Limits customization offers pre-built functionality which is hard to modify and contrast with our goal(reachability).

Cloud Service both Public and Private

Public

Product Catalog & AI Recommendation Algorithm: since it don't handle sensitive data and require high scalability during peak times (Black Friday sales).

Private

Customer Data Repository : Store sensitive customer information and personal data on a private cloud to ensure privacy and regulatory compliance

Replatform Migration

Re-platforming is suitable for microservices systems, as it allows us to gradually migrate and evolve services independently. So we can also re-factor some services over time to enhance modularity and scalability, ensuring the platform stays flexible and adaptable.