In the following pdf you'll find answers to the questions for the technical assessment mapped by question number.

1. Design a performant database structure for a relational database to store user profiles, including common data like name and age, as well as records of profile visits and user likes. Do not use Hibernate or JPA for this task:

- ☐ In each service, you'll find under main.ressources.data a sql script that is launched on application launch to create the tables.

2. Create a SQL query that retrieves all profile visitors of a user, sorted according to the most appropriate criteria.

- ☐ The answer for this will be under com.malekkamoua.meet5.userservice.repository.UserRepository (getAllVisitors(Long userId))

3. Design a Java class that models user profiles with attributes such as name, age, and additional user-defined fields.

- ☐ The answer for this will be under com.malekkamoua.meet5.userservice.repository.models: User class.

4. Implement validation mechanisms within the Java class to ensure data consistency and integrity.

- ☐ The answer for this will be under com.malekkamoua.meet5.userservice.repository.validators: validateUser()

5. Develop a RESTful API endpoint (/user/visit) to record when user A visits user B.
6. Develop a RESTful API endpoint (/user/like) to record when user A likes user B.

- ☐ The answer for both of these questions will be under com.malekkamoua.meet5.interactionservice.controller. A kafka listener is put in place in order to listen to USER_INTERACTIONS topic that dictates if the interaction is a "*Like*" or a "*Visit*" interaction

7. Implement logic to detect fraudulent activity, such as a user visiting and liking 100 users within the first 10 minutes.

- ☐ The answer for this will be under com.malekkamoua.meet5.fraudservice.service: isFraudulentInteraction()

8. Develop a method to handle bulk data insertion efficiently without relying on Hibernate or JPA.

- ☐ The answer for this will be under com.malekkamoua.meet5.userservice.repository: NotificationRepository. This is a notifications feature that listens to kafka VISIT_INTERACTIONS and LIKE_INTERACTIONS that are returned in a list. This list stores a big bulk of data.

9. Analyze the existing monolithic backend architecture

- ☐ The monolith architecture uses a single database. All data is centralized. This leads to tight coupling between components. Changes to one part of the application may require redeployment of the entire monolith. This architecture is challenging for this specific use case.

  Limited Scalability:
  - Scaling the application horizontally becomes challenging due to the need to replicate the entire monolith.

  Maintainability Issues:
  - Difficulty in maintaining and evolving a large, tightly coupled codebase.
  - Changes in one module may have unintended consequences in other parts.

  Deployment Bottlenecks:
  - Deployment of updates or changes requires downtime for the entire application.
  - Risk of deployment failures impacting the entire system.

Propose a detailed microservices architecture. Identify services such as User Service, Interaction Service, and Fraud Detection Service. Design APIs, data flow, and communication mechanisms between services.

The joint picture in Github describes a microservice architecture for this situation. There are 3 microservices:

User Service:

- ➔ Handles CRUD operations related to user data.
- ➔ Stores user notifications related to visits and likes.

Interaction Service:

- ➔ Manages user interactions such as profile visits and likes.
- ➔ Saves and retrieves interaction data between users.

Fraud Detection Service:

- ➔ Monitors user activity and detects fraudulent patterns.
- ➔ Analyzes user behavior to identify potential fraud, e.g., excessive likes and visits.

The starting point is the API gateway that redirects user requests to the designated service. In order to do that, the services are all saved in a registry. Communication between the services is based on Kafka events. All microservices are independent. Every one of them has his own managed database.

Address challenges such as data consistency, API versioning, and fault tolerance in the proposal.

In order to ensure data consistency:

- ➔ Use of transactional behavior (@Transactional)
- ➔ Use of event sourcing to maintain a log of all changes and ensure data integrity.

API versioning: use of significatif addressing:

- ➔ user service: /user
- ➔ interaction service: /interaction
- ➔ fraud service: /fraud

Versioning could also be using the release version (V1/V2..)

Fault tolerance:

- ➔ Using fallback methods that ensure that the application never goes down (Hystrix for example)
- ➔ Retry mechanism

For the Google cloud related points, due to technical blockage from my side, I didn't get the chance to do them. My credit card isn't accepted by Google.