software architecture

LAB 4 REPORT

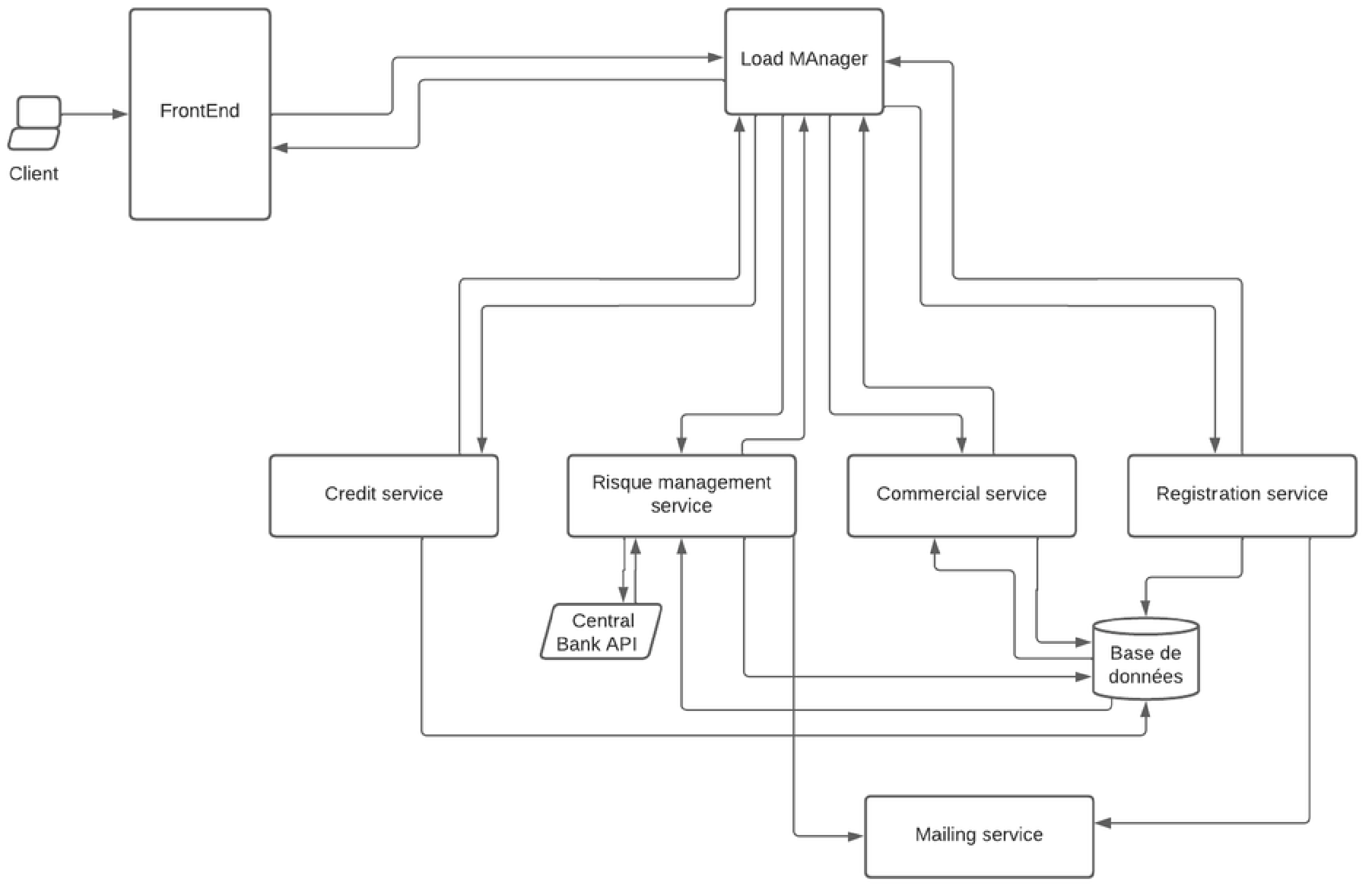
Microservices Architecture

Group GL4/1

- Hadil Helali
- Raoua Trimech
- Soulaima Kahla

- **Questions**

- **choose and explain the appropriate architecture for this loan management module.**



In this architecture , we chose to implement 6 microservices :

- Load Manager
- Credit Service
- Risque management Service
- Commercial Service
- Registeration Service
- Mailing Service

When a client submits the form after filling it out with personal information and required documents , a request is sent to the Load Manager.
The load manager then redirects it to the registration service that handles the form and an email is sent (by the mailing service) to confirm the registration.
Once registered , the load manager calls the commercial service to check the eligibility of the borrower (aka client) and calculates the initial score.

This score is sent to the load manager that passes it to the risk management service to calculate the final score and decides if the client is eligible or not for the loan.
Either way , an email is sent to the user to inform him about the decision.
If the loan is approved , the load manager calls the credit service that elaborates the credit agreement to be signed and the amortization table which can be downloaded by the client.

- **As a homework, implement few use cases [TBD in class] by explaining your choice of technologies as well.**

To implement this architecture , we chose to use **angular** for the frontend , **nestjs** and **redis**.
NestJS can be a good choice for building microservices for many reasons :

1. Modular and Scalable Architecture: NestJS follows a modular architecture, which makes it easy to break down your application into small, independent modules, each responsible for a specific task. This makes it easy to scale your application horizontally, by adding more instances of the same module to handle more traffic.
2. Dependency Injection: NestJS comes with built-in support for dependency injection, which makes it easy to manage dependencies between different modules. This is important when building microservices, where different modules need to communicate with each other.
3. Built-in Support for Async Programming: Microservices are inherently asynchronous, and NestJS provides built-in support for async programming, making it easy to handle async operations like network requests and database queries.
4. TypeScript Support: NestJS is written in TypeScript, which provides strong typing and compile-time type checking, making it easier to catch errors before they make it into production.

As for **Redis** , we have plenty of reasons as to why use it :

1. Fast Data Access: Redis is an in-memory data store, which means that it can access data quickly. This is especially important in a microservices architecture where services need to communicate with each other quickly and efficiently.
2. Scalability: Redis can be scaled horizontally, which means that it can handle an increasing amount of data and requests by adding more instances of the Redis database. Caching: Redis can be used as a cache to reduce the number of requests made to other microservices or databases. This can improve performance and reduce the load on other services.
3. Pub/Sub Messaging: Redis supports pub/sub messaging, which allows microservices to communicate with each other asynchronously. This can be useful when services need to be notified of changes in data or events in the system.
4. Data Structure Support: Redis supports various data structures like strings, hashes, lists, sets, and sorted sets, which can be useful for storing and processing data in different formats.

To sum up , Using **Redis** and **NestJS** together in a microservices architecture can provide several benefits. For example, Redis can be used to store frequently accessed data, reducing the load on databases and improving performance. NestJS can be used to develop and maintain the microservices, providing a robust and scalable infrastructure for the application.
Additionally, **Redis** and **NestJS** can be used together to enable efficient communication between microservices, which is critical in a distributed system.

You can find the source code in these github repositories :

fronend : https://github.com/SoulaimakH/loan-management-frontend
backend : https://github.com/SoulaimakH/loan-management-module-ArchLog-