

MICROSERVICES OR MONOLITHIC ARCHITECTURE?

Table of contents

- **I. INTRODUCTION**
- **II. THEORETICAL BACKGROUND**
 - **1. Monolithic architecture**
 - **2. Microservices**
- **III. PRACTICAL EXPERIENCE**
 - **1. Monolithic architecture**
 - **2. Microservices**
- **IV. COMPARISON**
- **V. CONCLUSION**

I. INTRODUCTION

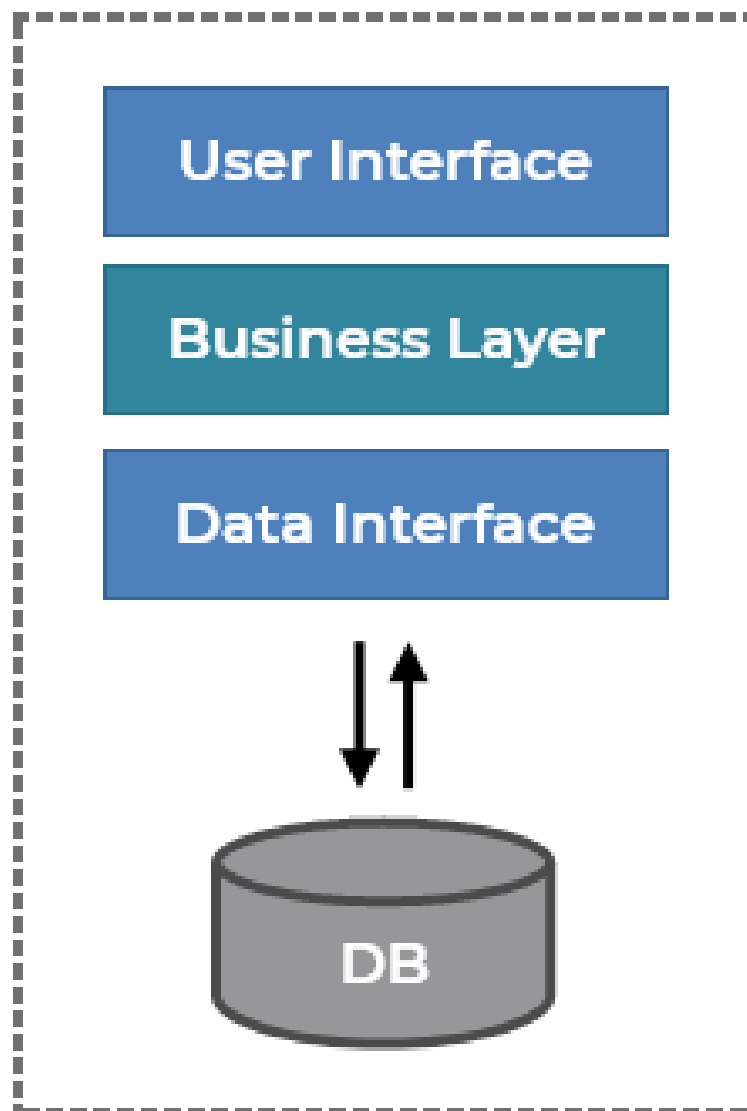
Microservices, a term that any person who is currently working in IT field, would have heard of at least a few times. So what exactly are people talking about, where are they moving to microservices architecture from, and why people choose microservices?

My article, despite being relatively short - just like my experience, should give you a brief look about both the trend - microservices and the past - monolithic architecture.

II. THEORETICAL BACKGROUND

1. Monolithic architecture

Monolithic Architecture



Monolithic architecture common diagram

Definition

"A monolithic architecture is a traditional model of a software program, which is built as a unified unit that is self-contained and independent from other applications. The word "monolith" is often attributed to something large and glacial, which isn't far from the truth of a monolith architecture for software design. A monolithic architecture is a singular, large computing network with one code base that couples all of the business concerns together. To make a change to this sort of application requires updating the entire stack by accessing the code base and building and deploying an updated version of the service-side interface. This makes updates restrictive and time-consuming."

Chandler Harris - Microservices vs. monolithic architecture

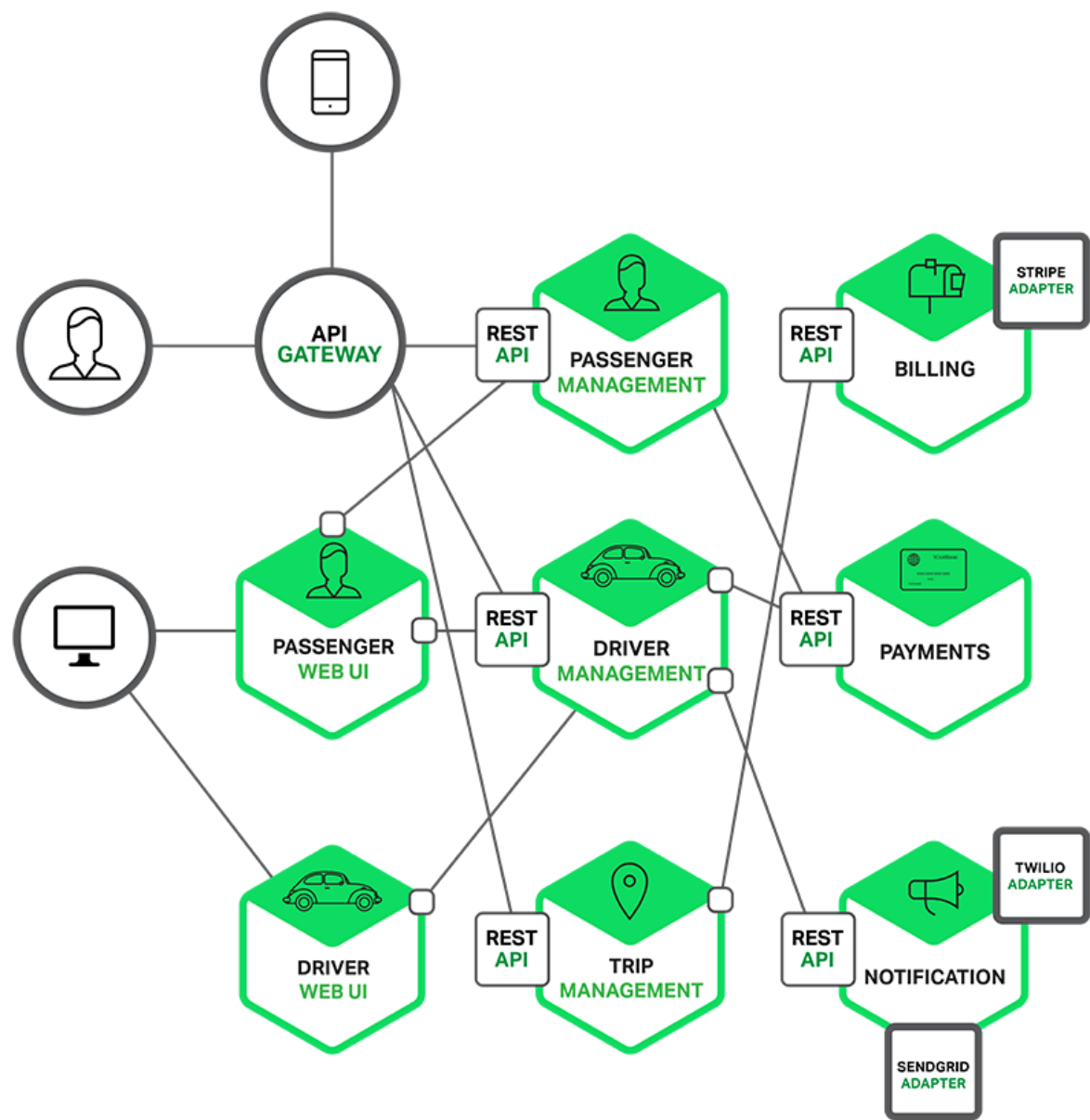
"The monolithic architecture is considered to be a traditional way of building applications. A monolithic application is built as a single and indivisible unit. Usually, such a solution comprises a client-side user interface, a server side-application, and a database. It is unified and all the functions are managed and served in one place."

Romana Gnatyk - Microservices vs Monolith: which architecture is the best choice for your business?

=> Main points about monolithic architecture and monolithic applications:

- Monolithic architecture is considered to be the traditional way to build applications.
- An application is built as a single unit.
- All functionalities are developed in a single codebase and they are usually tightly coupled.

2. Microservices



A common diagram for microservice architecture

Definition

"Microservices are an architectural and organizational approach to software development where software is composed of small independent services that communicate over well-defined APIs. These services are owned by small, self-contained teams."

Amazon - What are microservices?

"Microservices are an architectural approach to building applications. As an architectural framework, microservices are distributed and loosely coupled, so one team's changes won't break the entire app. The benefit to using microservices is that development teams are able to rapidly build new components of apps to meet changing business needs"

Red Hat - What are microservices?

=> **Main points about microservices architecture:**

- A system is now composed of multiple small independent services that communicate over well-defined APIs
- Each microservice can be developed by a small team.
- Microservices in a system are loosely coupled, each mind their own business.

III. PRACTICAL EXPERIENCE

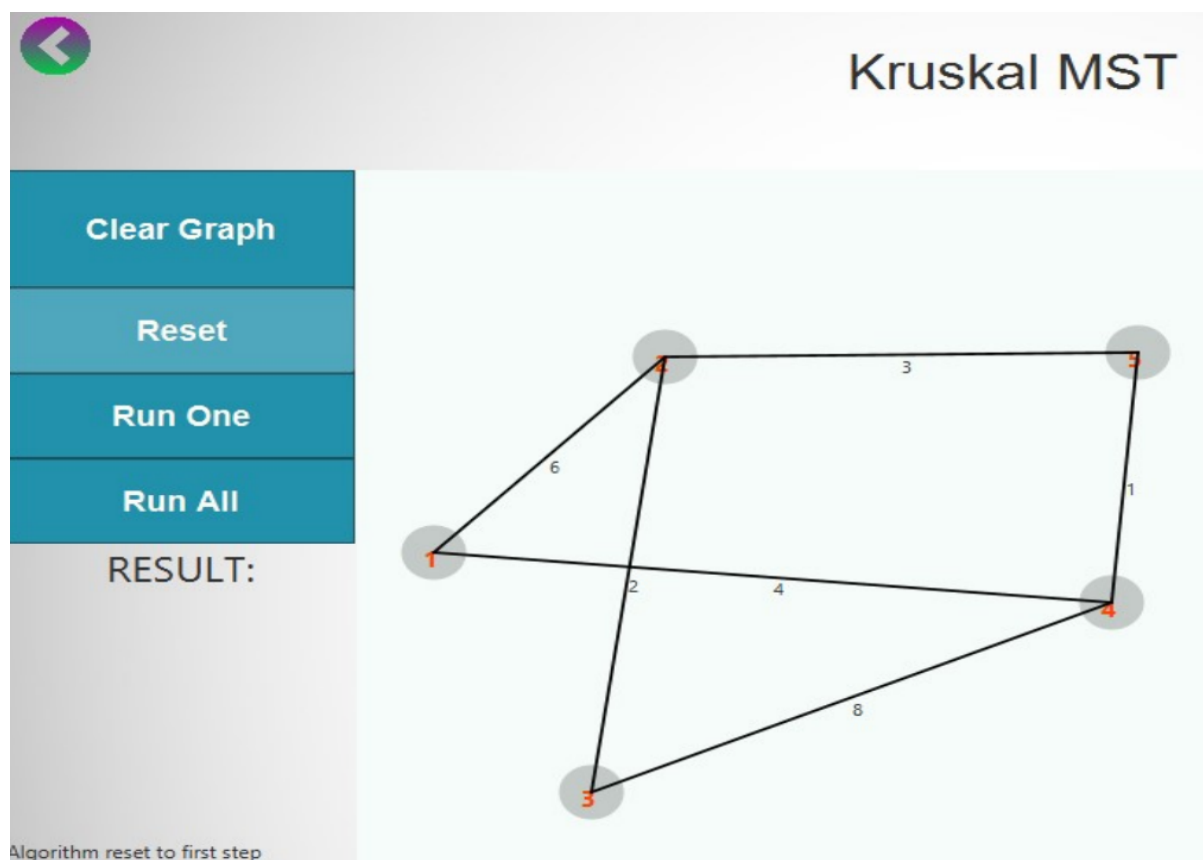
For the past few years, I have participated in several projects that use those architectures. Even though my experience is not as much as any senior developers, I still believe that it can be helpful at least for my own and hopefully it can help anyone who reads this article.

1. Monolithic architecture

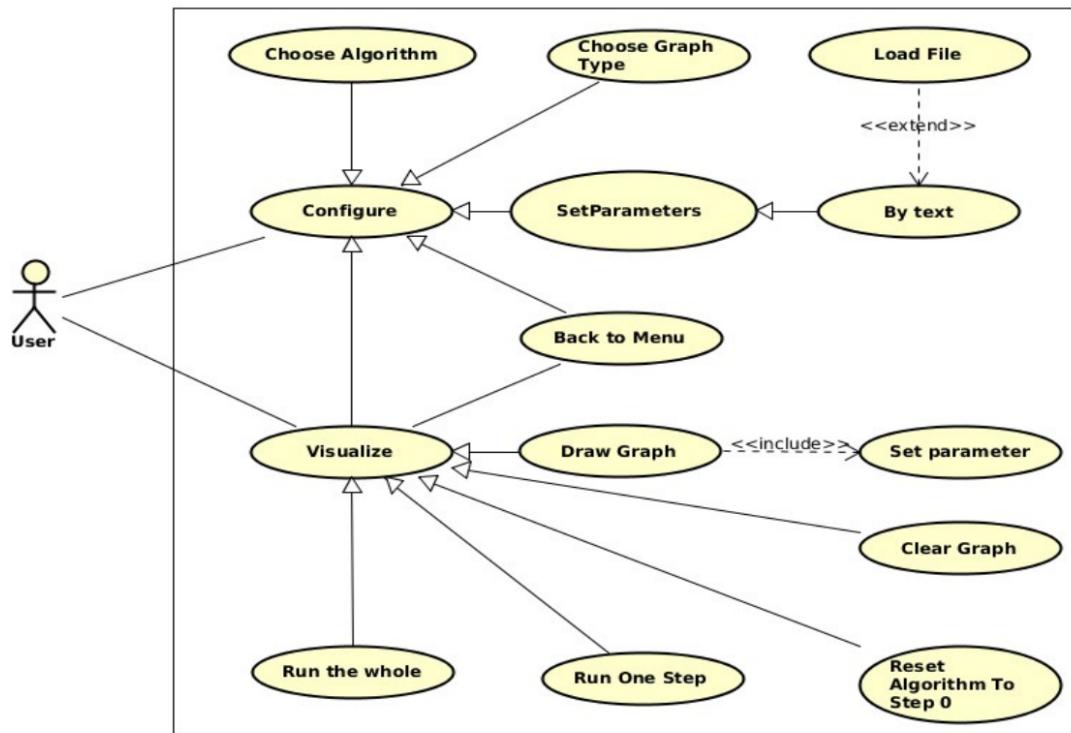
Every projects I participated in that used monolithic architecture are classroom-level, but they still give me some invaluable experience.

1.1. First Project - Working example

For the first project, my team and I developed an application that visualize graph algorithms:



UI



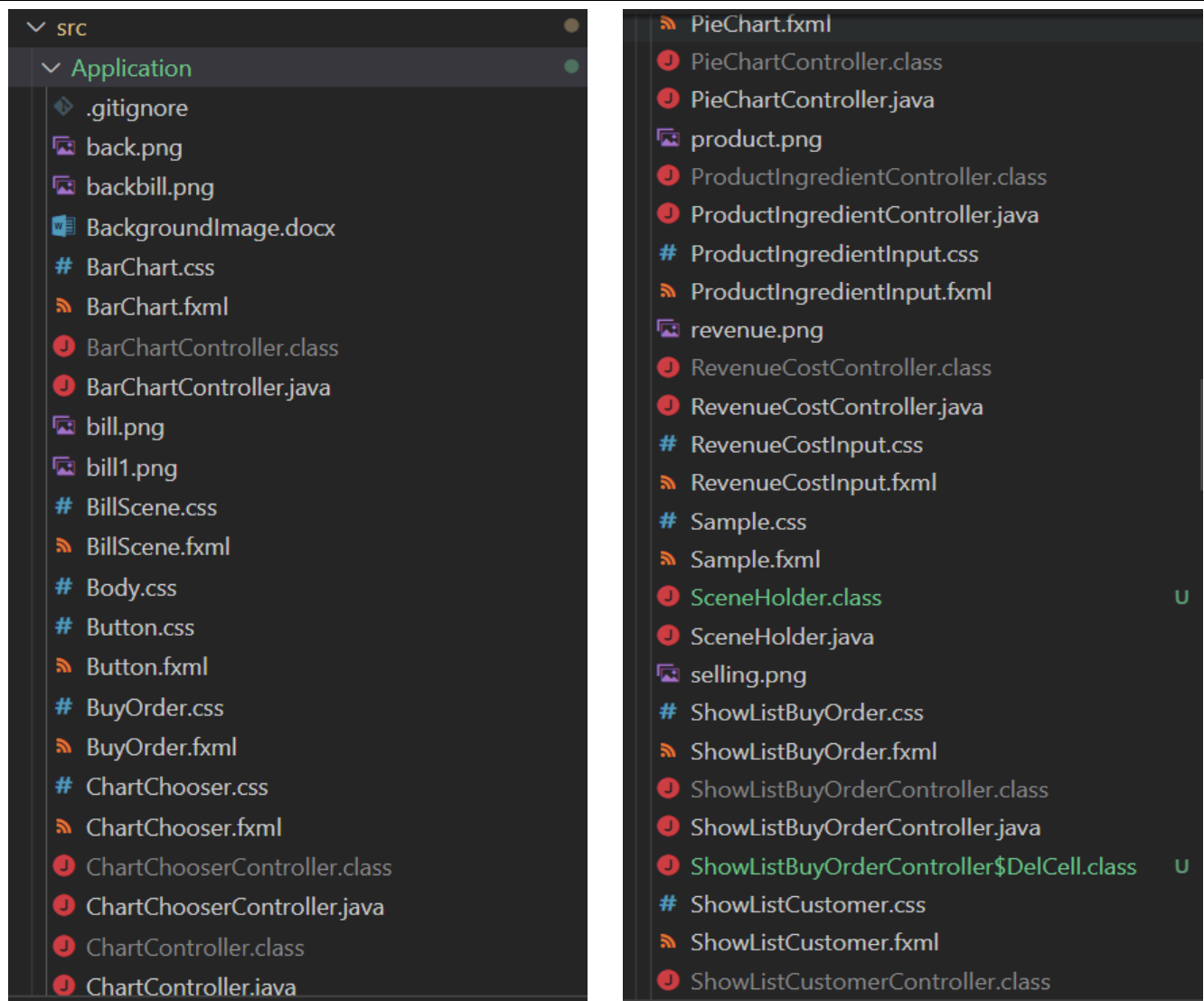
Use Case Diagram

This application did not require any database to run, so it was still fine for us to put everything in one single codebase and got away with it. The whole project was written in Java, so the User Interface was not very fancy, but the app worked.

1.2. Second Project - Bad practice

The honeymoon with monolithic architecture ended when I participated in the second project - still classroom-level, but for this one we needed to use a database and a friendly UI, and this time, things were messy.

Our naive team did what IT college students usually do when they do a project at the university: Bundle everything in one single codebase - Frontend, Backend, Authentication,... EVERYTHING! and we took system designing process for granted



A single package

Literally, everything was there, from CSS files, fxml files (kinda like HTML, but worse) to Java code

2. Microservices

Life with monolithic architecture was hard, but that does not mean microservices would be a pinky world.

2.1. First Project - Bad Practice

I was very excited to participate in a project that used microservice architecture, especially when I saw the GitLab organization page which had like 20+ repositories. I was naive, still...

Projects

New project

Your projects 26

Starred projects 0

Explore projects

Explore topics

Pending deletion

Filter by name...

Updated date

All

Personal

Y	[redacted] backend-jhi	Developer	★ 0	🔗 0	🔗 0	🔗 0	Updated 3 months ago
Y	[redacted] angular	Developer	★ 0	🔗 0	🔗 0	🔗 4	Updated 3 months ago
Y	[redacted] python-service	Developer	★ 0	🔗 0	🔗 1	🔗 0	Updated 3 months ago
Y	[redacted] alert job	Developer	★ 0	🔗 0	🔗 0	🔗 0	Updated 4 months ago
E	[redacted] data_recruitment	Developer	★ 0	🔗 0	🔗 0	🔗 0	Updated 4 months ago
Y	[redacted] frontend-jhi	Developer	★ 0	🔗 0	🔗 0	🔗 0	Updated 4 months ago
Y	[redacted] graph-apollo	Developer	★ 0	🔗 0	🔗 0	🔗 0	Updated 4 months ago

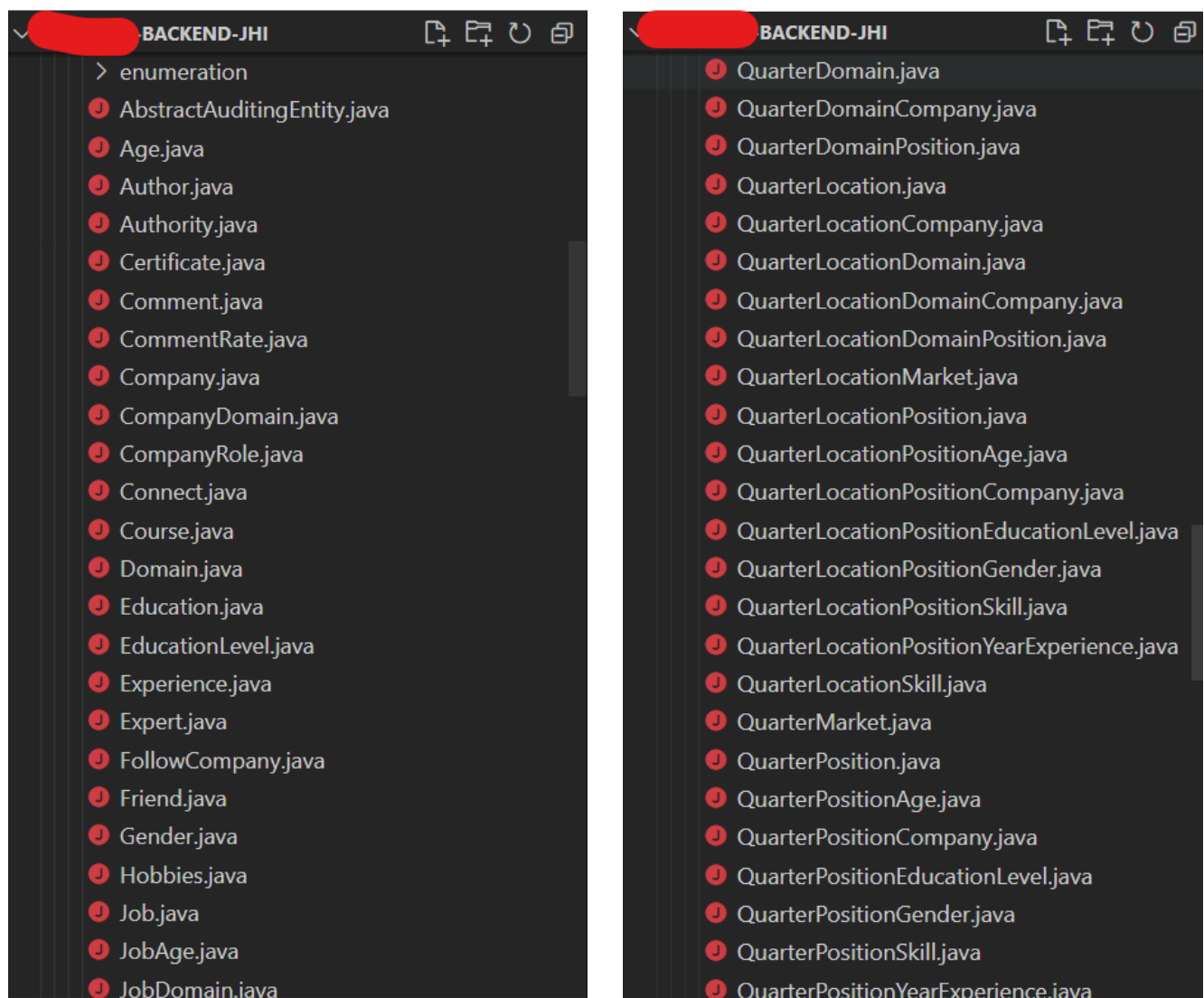
GitLab organization page - 26 repositories

Despite having 20+ repositories, there still exists several services that are not really "micro", for example:



ERD with 80+ entities

One single "microservice" that managed 80+ tables in a database which made me - a new team member found it very difficult to understand the code. Also, with 80+ entities, it is completely impossible to have any kind of UML diagram or document that can explain the application, especially for a miserable fresher like me.



80+ Java classes for entities

20+ of them have the names "Quarter..." and were used for statistical queries. They should appear in another microservice instead.

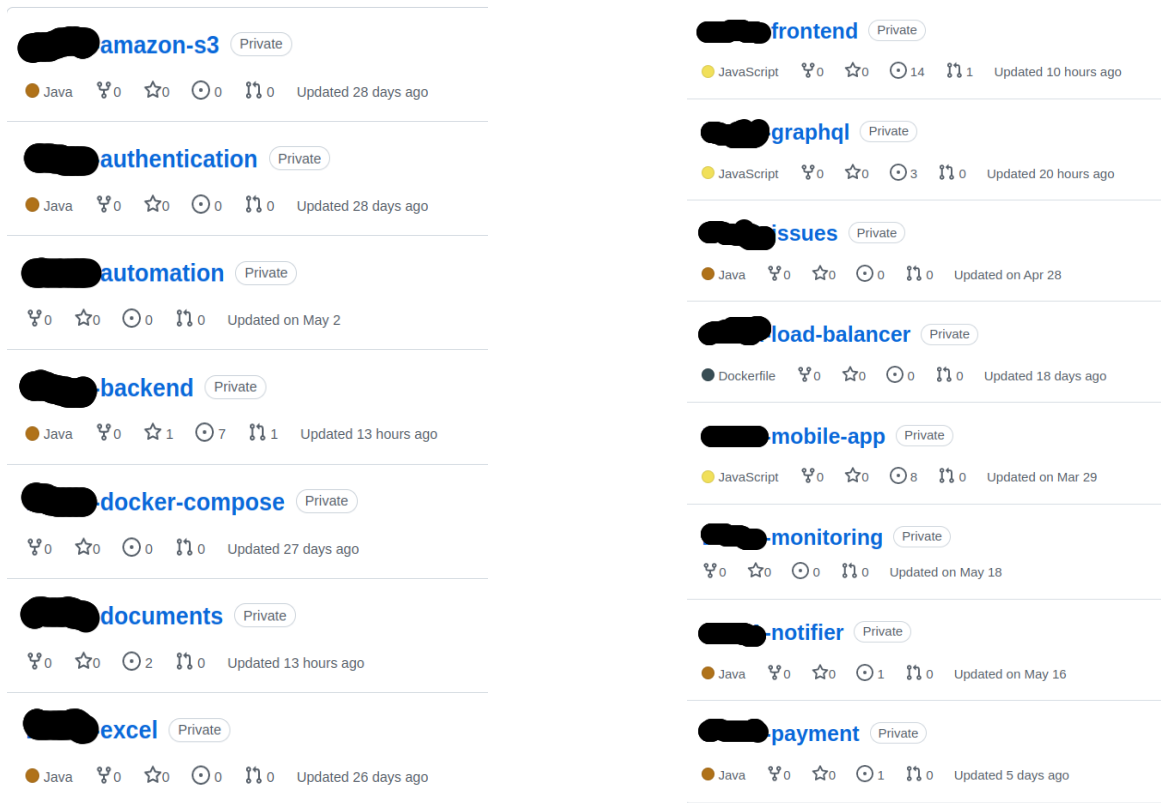
Looking back at the diagram still gives me nightmares, but for now, after having a better understanding of the architecture, I am pretty confident that if I am able to redesign this project, I would do a lot better than the one I had to work with before.

Despite having 80+ tables, as we all can see from the ERD, those entities are loosely coupled, so it should be totally fine to separate them into several more microservices, at least for authentication and statistics-related entities.

Even though the first experience was not very good, I still find the elegance in this architecture and then apply it to two other projects that I participate, one as a leader and one as the only developer (my graduation research project).

2.2. Second and Third Project - Working example

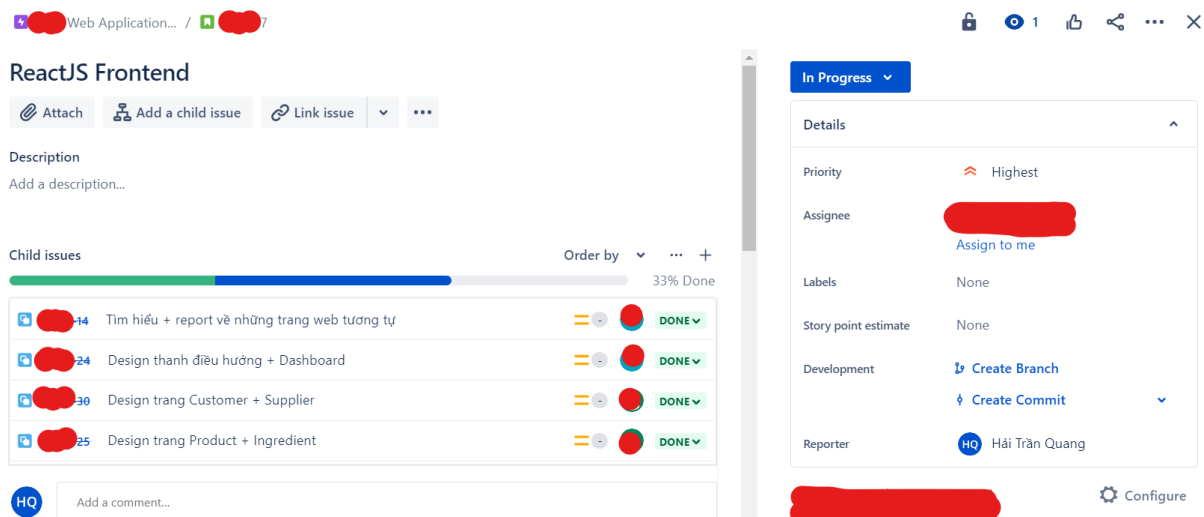
The second one, which I led:



List of repositories

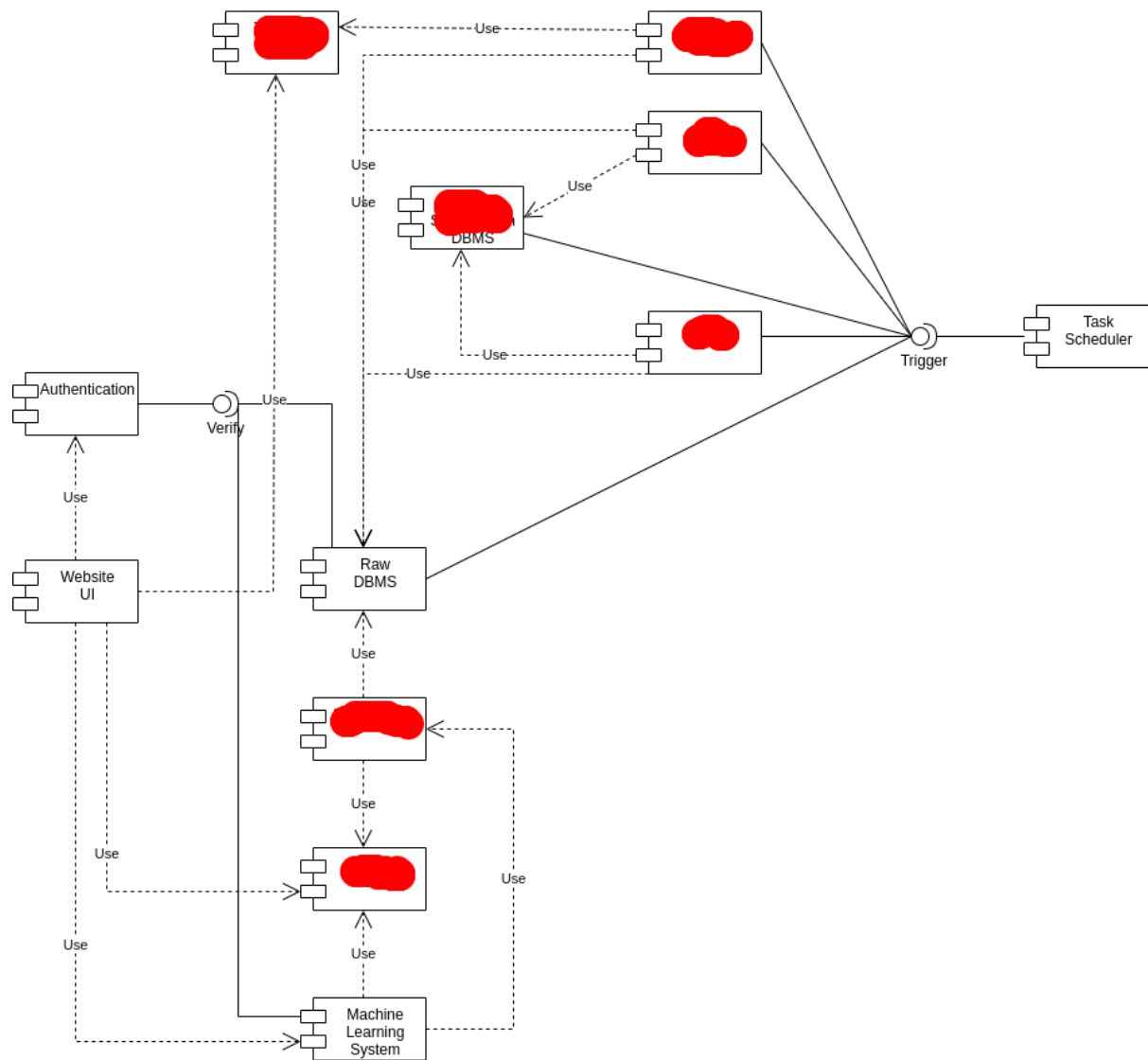
Each microservice can be developed by one single developer, and he/she can use whatever language that he/she feels confident about.

Also, for microservice projects that are developed by a team, communication becomes very important (in my case there are 10 people), so we need to use a different way to distribute the tasks and track the progress



Jira

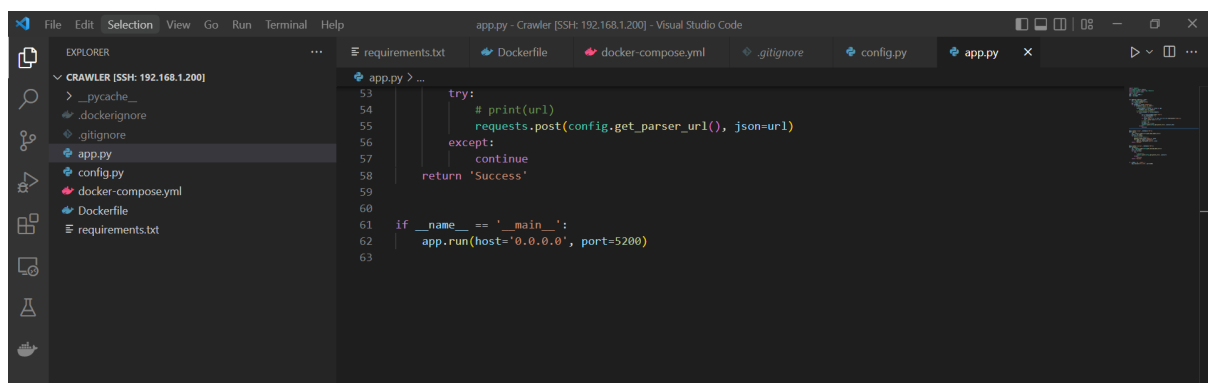
The third one - my graduation research project:



Component Diagram

This project, even though is being developed by me alone, things are still going pretty well.

Each microservice is pretty small in terms of codebase, so any new member should be able to understand and contribute after a short time.



An example - Only two files and the code is relatively short

IV. COMPARISON

Monoliths	Microservices
Single codebase	Multiple services - multiple codebases
Scaling by adding replica nodes with the whole application	Each service can be scaled independently
Difficult to understand for new team members	Easier for new members to start working on a small service
Barrier to new technologies	Newer technologies
Faster communication between functions	Slow communication between services*
Usually use only one programming language	Flexible in terms of language
Easy for testing	Difficult to test (needs to create mock services to test one service)
Easy to trace errors and exceptions	Tracing errors and exceptions is difficult
Easier deployment (single packaged application)	Difficult to deploy (have to run all microservices to make the system function as expected)
Updating a function requires redeploying again the whole system	Only need to redeploy the updated service
Lower initial costs	Higher initial costs

* Microservices usually use communication via network interfaces while Monoliths application functions communicate internally. Read/Write speed of hardware (RAM, SSD or HDD) should always be faster than communication on the Internet

V. CONCLUSION

Both architectures have their own strengths and drawbacks. But we can only utilize their strengths with **a clear system design and a cohesive way to manage and communicate between team members**, otherwise **no architectures** can save **a badly managed and poorly designed project**.

For monolithic projects:

- Be very careful when choosing monolithic architecture for any project. This architecture should only be applied to small and simple applications that serve a very narrow purpose.
- If an application reaches a point where the codebase starts to become confusing, it's a good sign to move to microservices architecture.

For microservices projects:

- Pay attention to the size of those microservices, otherwise they will be too large which make the architecture's benefits disappear.
- It is known as a "luxury" solution which would be very suitable for large projects. However, as it inherits the characteristics of "Divide and Conquer" motto, small teams or even individuals are still able to implement microservices as long as their system design process returns a clear picture about those components.