MICROSERVICES OR MONOLITHIC ARCHITECTURE?

Author: Tran Quang Hai

Table of contents

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

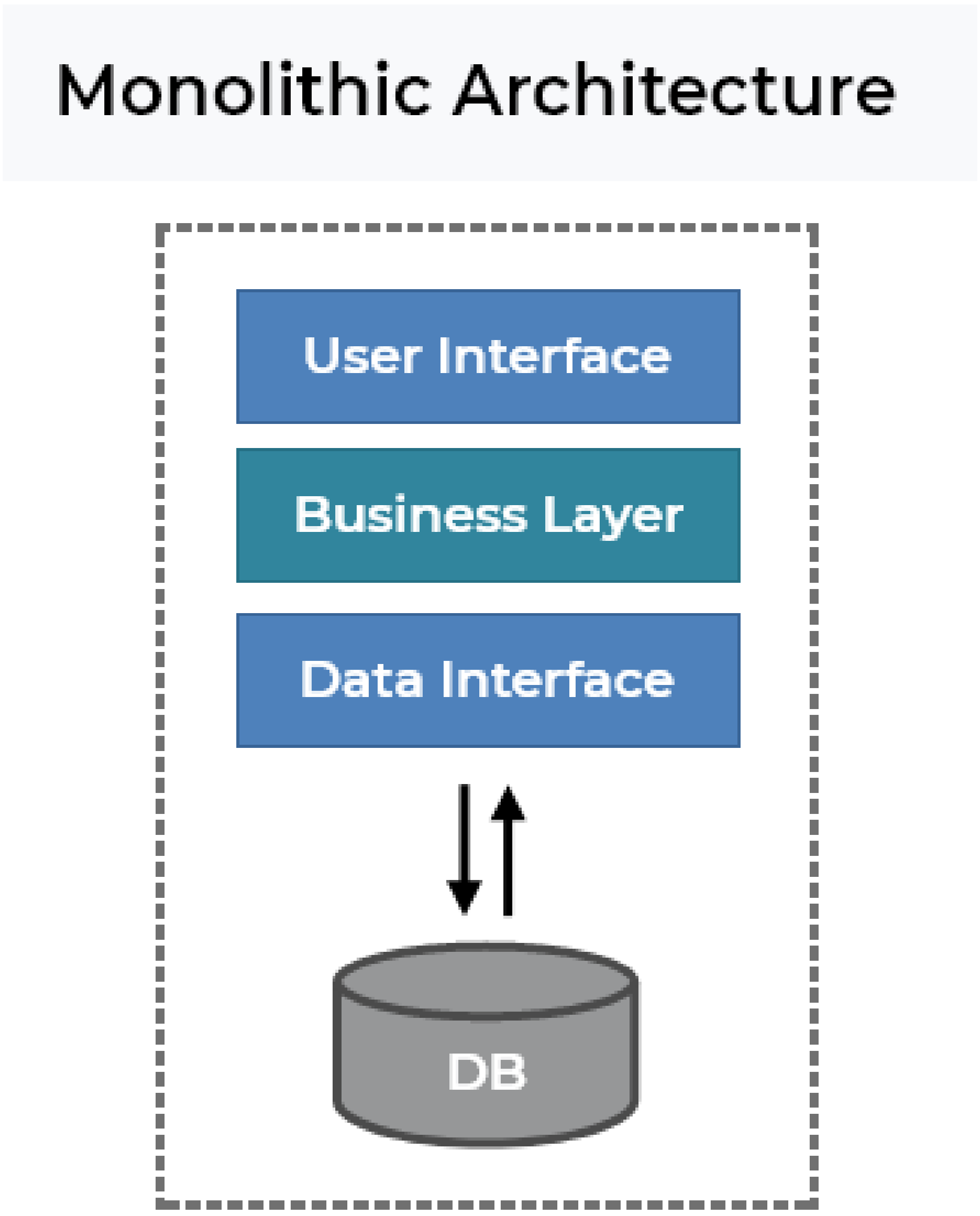
I. INTRODUCTION

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 common diagram

Source: Monolithic & Microservices Architecture - Henrique Siebert Domareski

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?

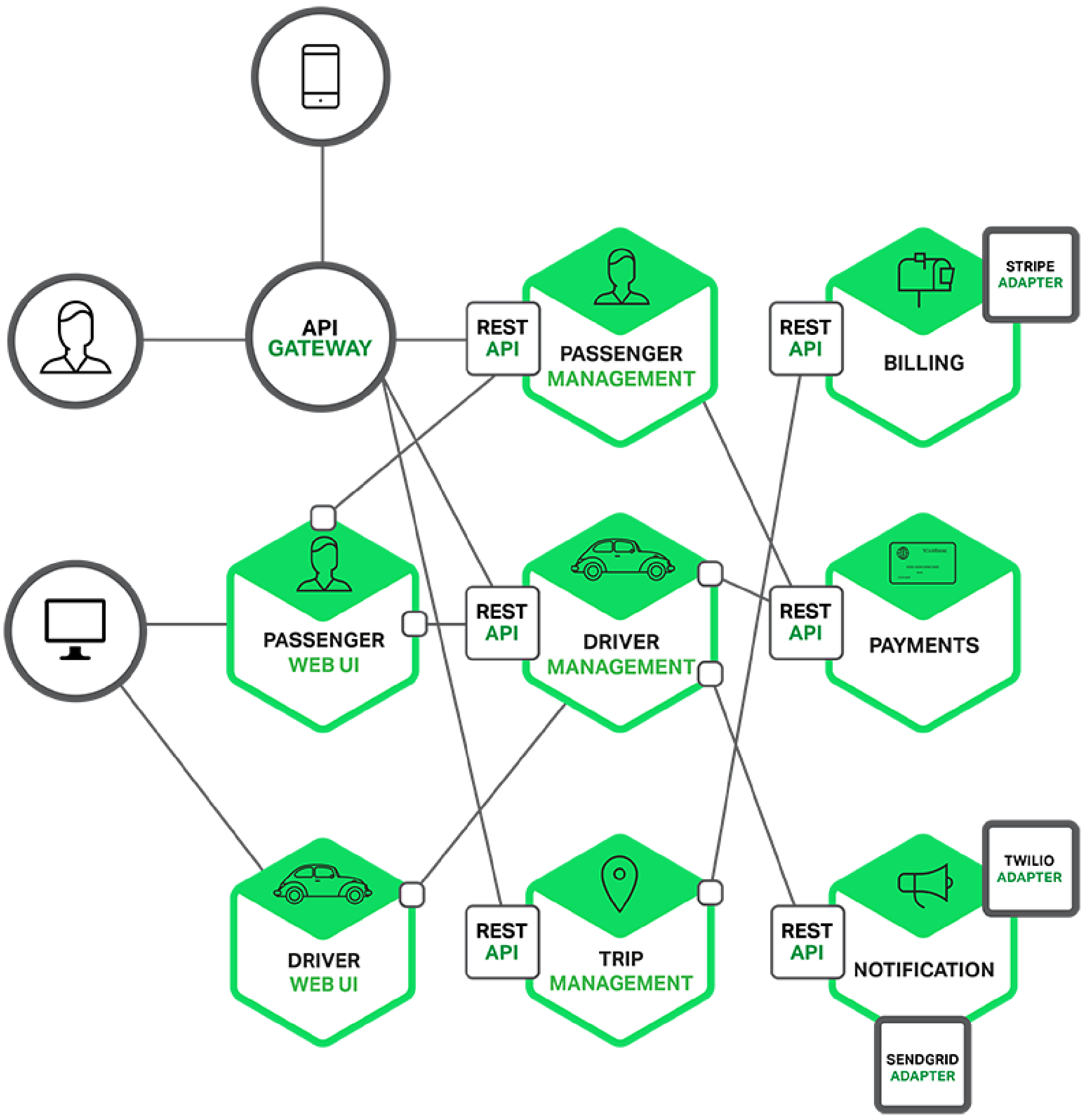
=> **Main points about monolithic architecture and monolithic applications:**

- 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. Source: microservices.io

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

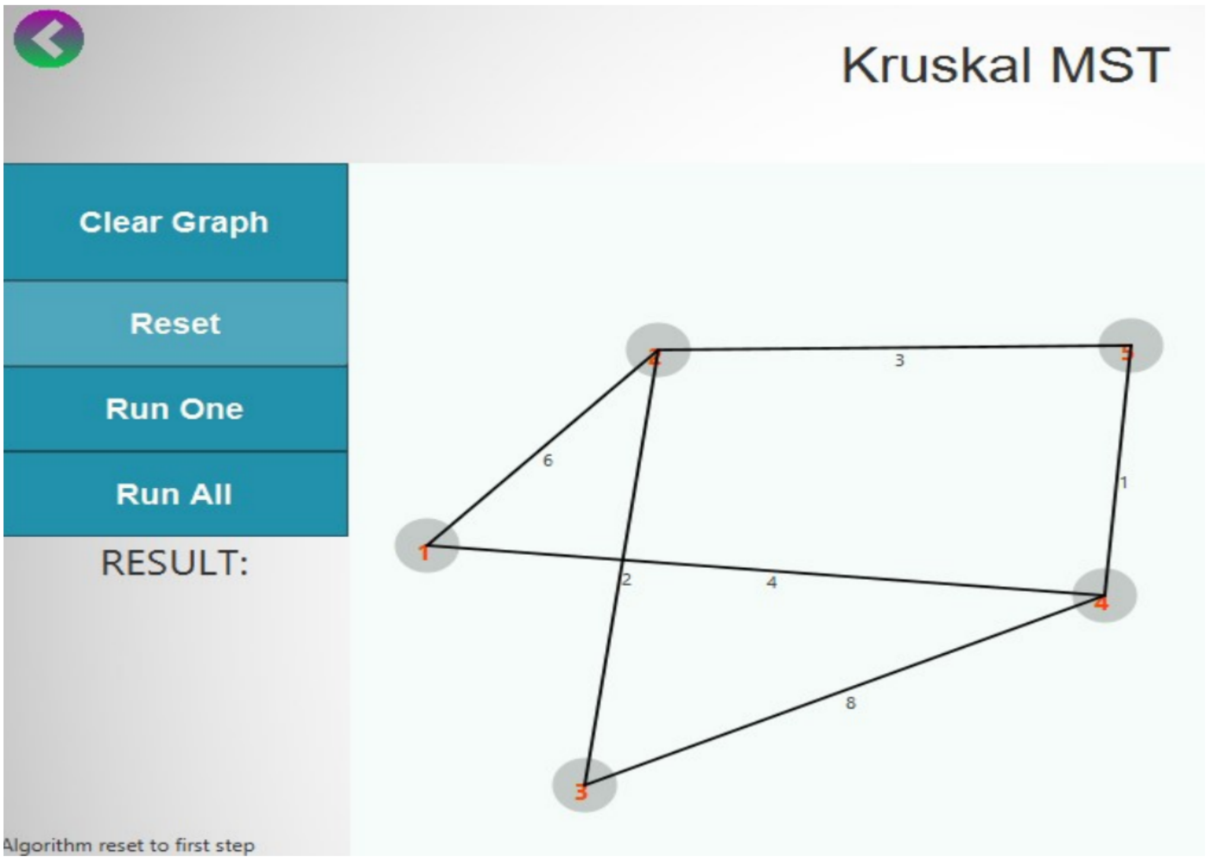
For the past few years, I have participated in several projects that use those architectures. Even though my experience is not as much as any senior developers, I still believe that it can be helpful at least for my own and hopefully it can help anyone who reads this article.

1. Monolithic architecture

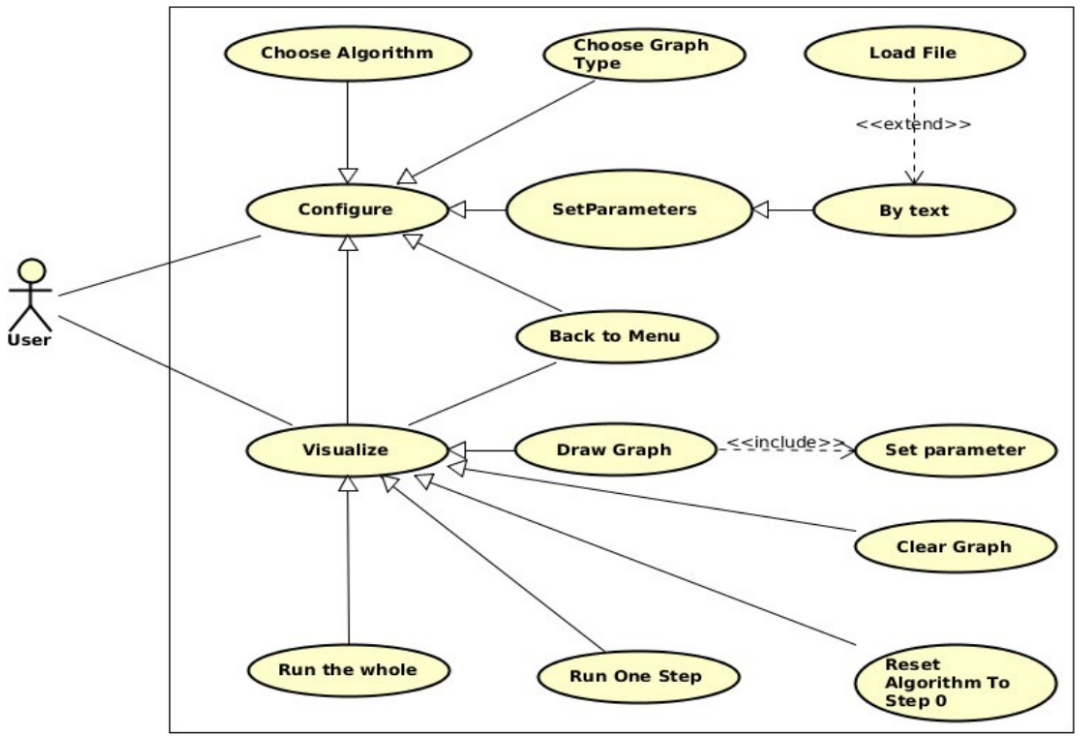
Every projects I participated in that used monolithic architecture are classroom-level, but they still give me some invaluable experience.

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

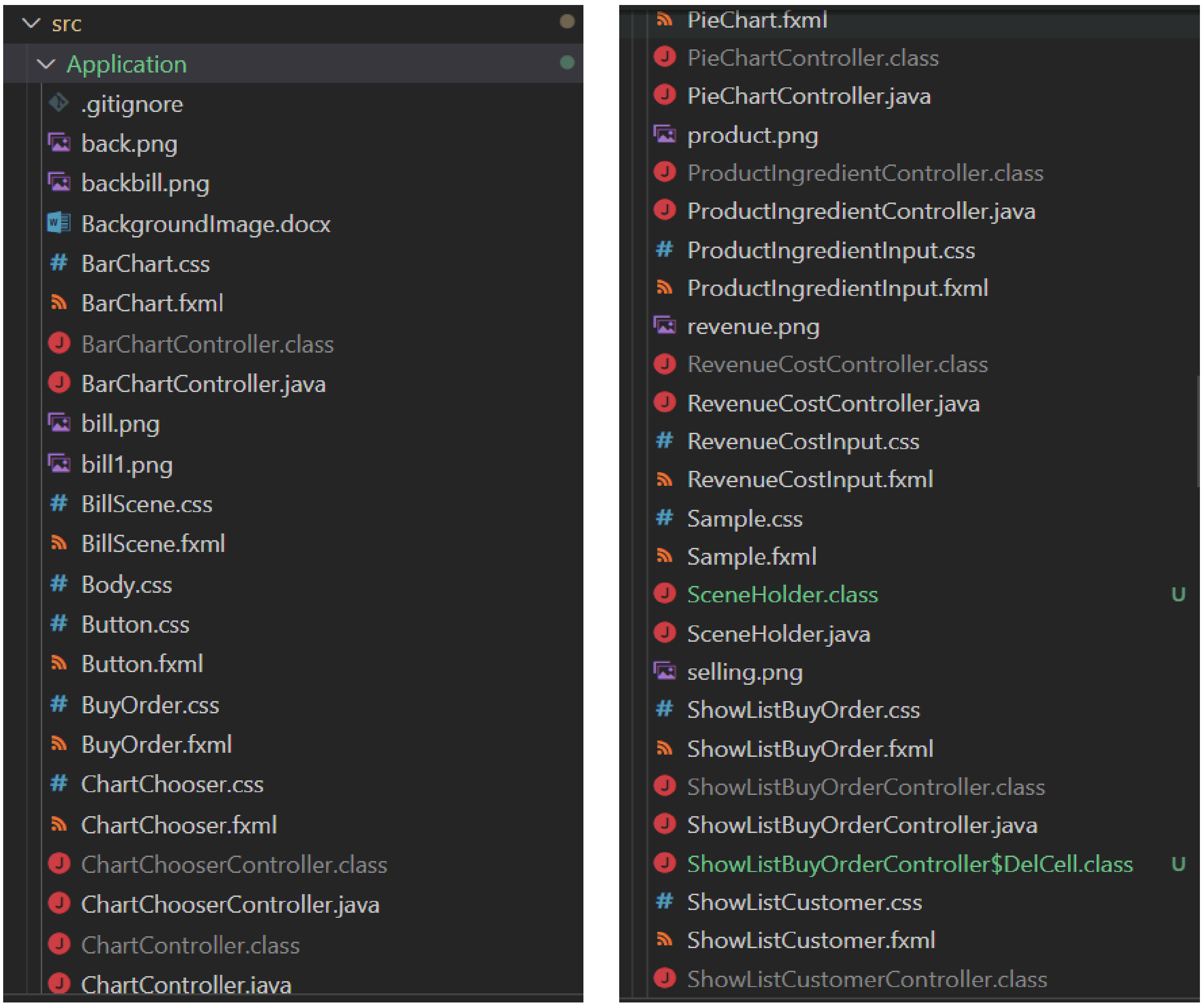
This application did not require any database to run, so it was still fine for us to put everything in one single codebase and got away with it. The whole project was written in Java, so the User Interface was not very fancy, but the app worked.

However, the project only required us to illustrate 3 algorithms. If the requirements had been wider (10 - 20 algorithms maybe?), it would have been a lot more difficult to expand.

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



```
src
  Application
    .gitignore
    back.png
    backbill.png
    BackgroundImage.docx
    BarChart.css
    BarChart.fxml
    BarChartController.class
    BarChartController.java
    bill.png
    bill1.png
    BillScene.css
    BillScene.fxml
    Body.css
    Button.css
    Button.fxml
    BuyOrder.css
    BuyOrder.fxml
    ChartChooser.css
    ChartChooser.fxml
    ChartChooserController.class
    ChartChooserController.java
    ChartController.class
    ChartController.java
```

```
    PieChart.fxml
    PieChartController.class
    PieChartController.java
    product.png
    ProductIngredientController.class
    ProductIngredientController.java
    ProductIngredientInput.css
    ProductIngredientInput.fxml
    revenue.png
    RevenueCostController.class
    RevenueCostController.java
    RevenueCostInput.css
    RevenueCostInput.fxml
    Sample.css
    Sample.fxml
    SceneHolder.class                              U
    SceneHolder.java
    selling.png
    ShowListBuyOrder.css
    ShowListBuyOrder.fxml
    ShowListBuyOrderController.class
    ShowListBuyOrderController.java
    ShowListBuyOrderController$DelCell.class       U
    ShowListCustomer.css
    ShowListCustomer.fxml
    ShowListCustomerController.class
```

A single package

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

With this level of organization, there were no surprise that the project became a mess.

- Completely impossible to isolate errors

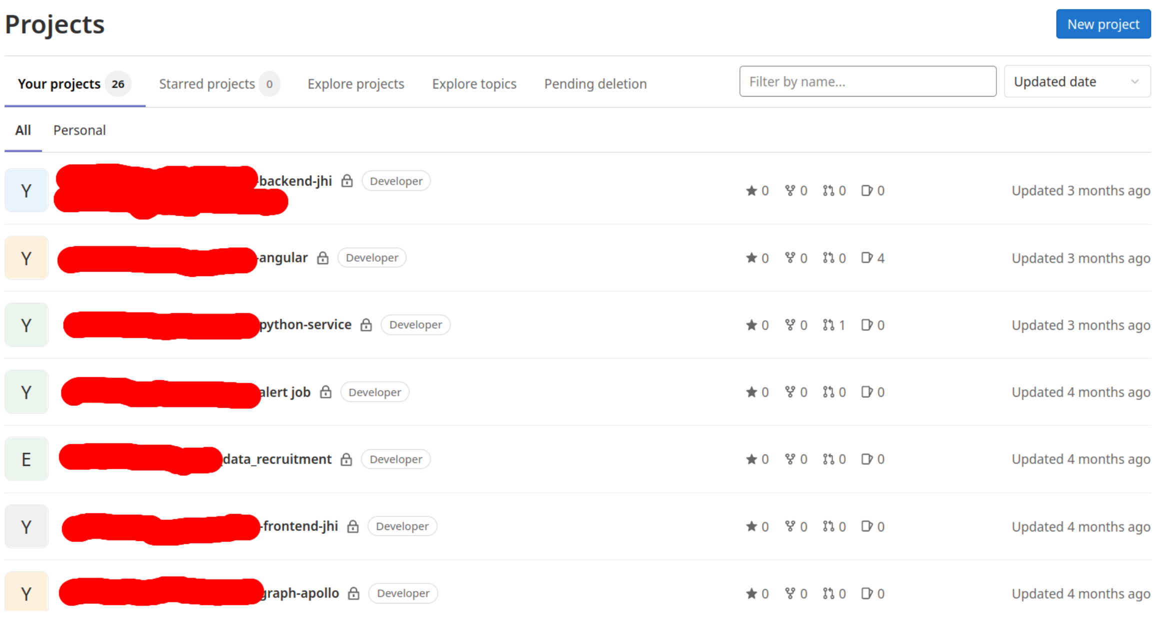
- Git workflow became difficult to manage

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 start-up 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

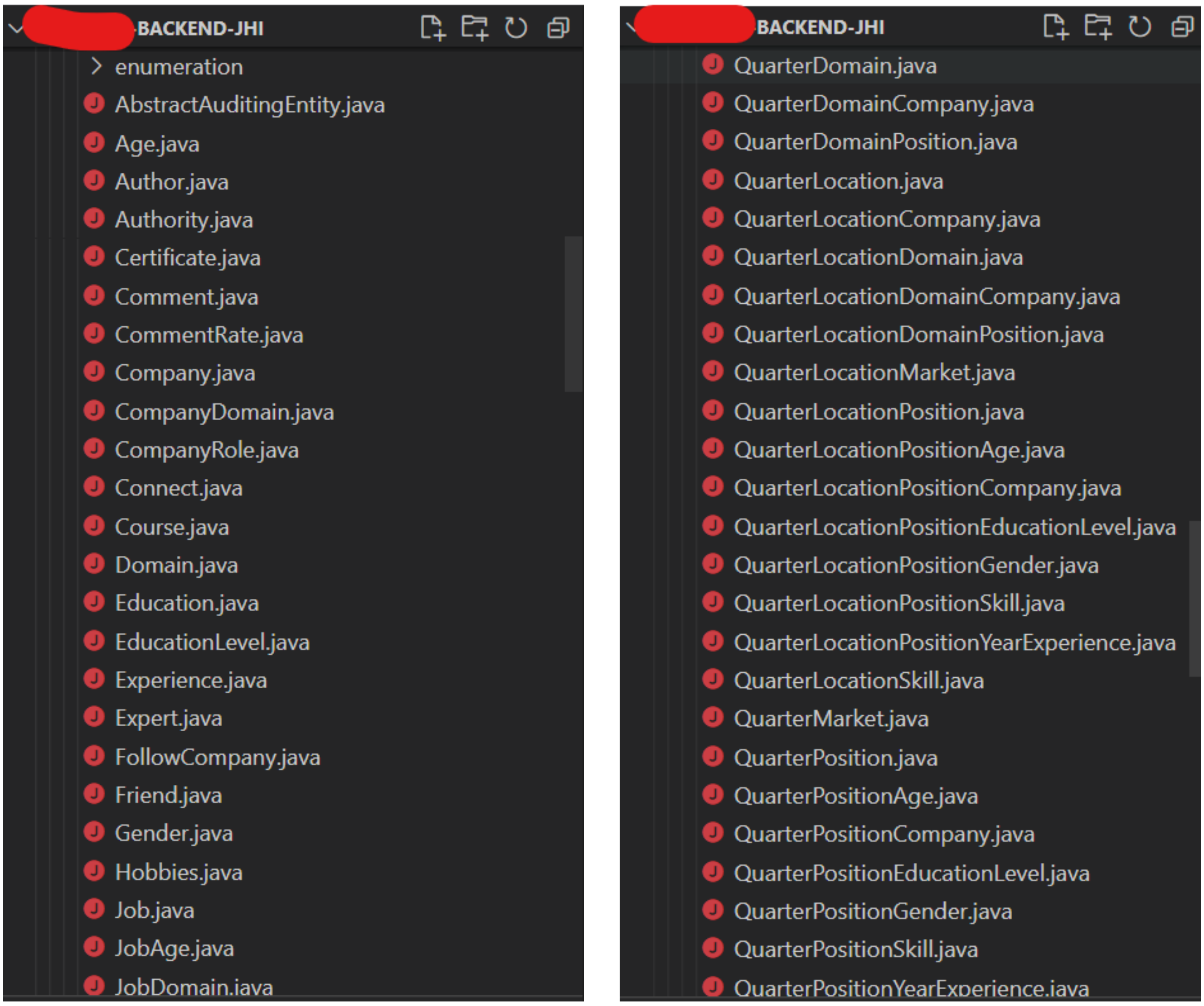
Y	[redacted]backend-jhi	Developer	★ 0	0	0	0	Updated 3 months ago
Y	[redacted]angular	Developer	★ 0	0	0	4	Updated 3 months ago
Y	[redacted]python-service	Developer	★ 0	0	1	0	Updated 3 months ago
Y	[redacted]alert job	Developer	★ 0	0	0	0	Updated 4 months ago
E	[redacted]data_recruitment	Developer	★ 0	0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



```
[redacted]-BACKEND-JHI
  > enumeration
  AbstractAuditingEntity.java
  Age.java
  Author.java
  Authority.java
  Certificate.java
  Comment.java
  CommentRate.java
  Company.java
  CompanyDomain.java
  CompanyRole.java
  Connect.java
  Course.java
  Domain.java
  Education.java
  EducationLevel.java
  Experience.java
  Expert.java
  FollowCompany.java
  Friend.java
  Gender.java
  Hobbies.java
  Job.java
  JobAge.java
  JobDomain.java
```

```
[redacted]-BACKEND-JHI
  QuarterDomain.java
  QuarterDomainCompany.java
  QuarterDomainPosition.java
  QuarterLocation.java
  QuarterLocationCompany.java
  QuarterLocationDomain.java
  QuarterLocationDomainCompany.java
  QuarterLocationDomainPosition.java
  QuarterLocationMarket.java
  QuarterLocationPosition.java
  QuarterLocationPositionAge.java
  QuarterLocationPositionCompany.java
  QuarterLocationPositionEducationLevel.java
  QuarterLocationPositionGender.java
  QuarterLocationPositionSkill.java
  QuarterLocationPositionYearExperience.java
  QuarterLocationSkill.java
  QuarterMarket.java
  QuarterPosition.java
  QuarterPositionAge.java
  QuarterPositionCompany.java
  QuarterPositionEducationLevel.java
  QuarterPositionGender.java
  QuarterPositionSkill.java
  QuarterPositionYearExperience.java
```

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.

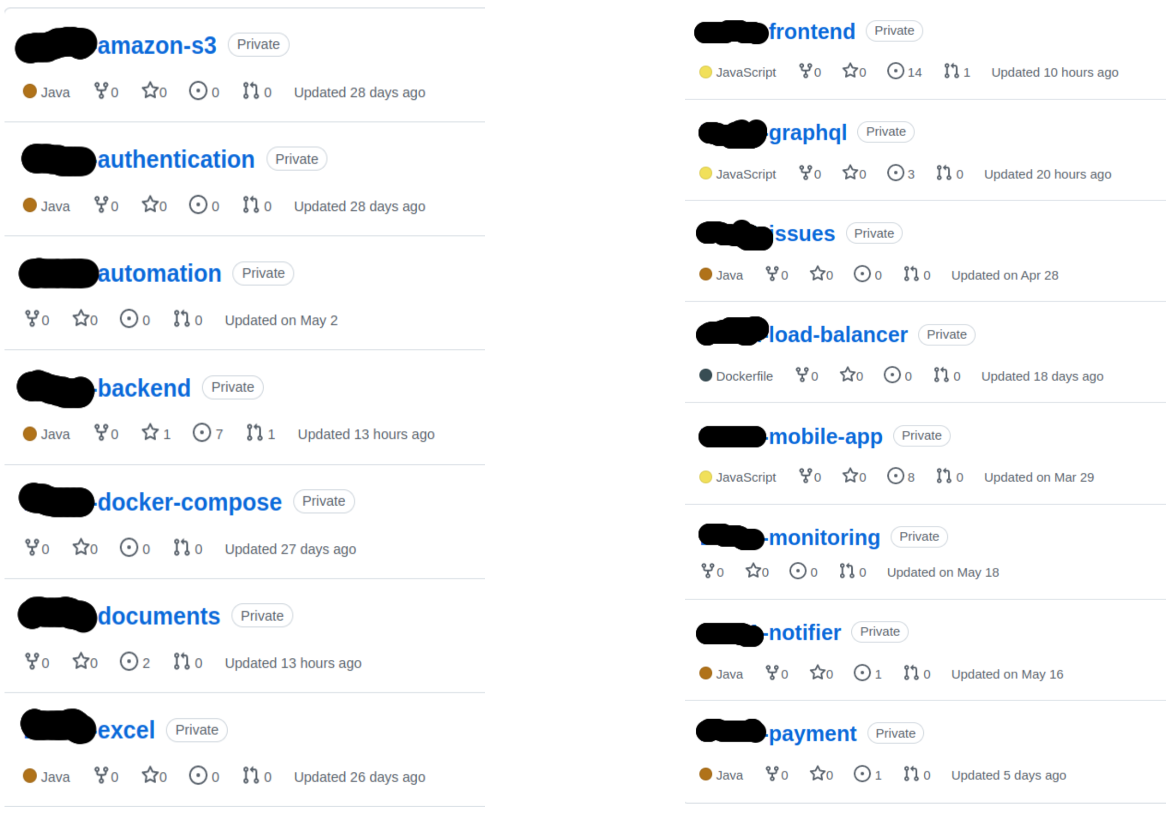
Looking back at the diagram still gives me nightmares, but for now, after having a better understanding of the architecture, I am pretty confident that if I am able to redesign this project, I would do a lot better than the one I had to work with before.

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:



[redacted]amazon-s3 Private
Java 0 0 0 0 Updated 28 days ago

[redacted]authentication Private
Java 0 0 0 0 Updated 28 days ago

[redacted]automation Private
0 0 0 0 Updated on May 2

[redacted]backend Private
Java 0 1 7 1 Updated 13 hours ago

[redacted]docker-compose Private
0 0 0 0 Updated 27 days ago

[redacted]documents Private
0 0 2 0 Updated 13 hours ago

[redacted]excel Private
Java 0 0 0 0 Updated 26 days ago

[redacted]frontend Private
JavaScript 0 0 14 1 Updated 10 hours ago

[redacted]graphql Private
JavaScript 0 0 3 0 Updated 20 hours ago

[redacted]issues Private
Java 0 0 0 0 Updated on Apr 28

[redacted]load-balancer Private
Dockerfile 0 0 0 0 Updated 18 days ago

[redacted]mobile-app Private
JavaScript 0 0 8 0 Updated on Mar 29

[redacted]monitoring Private
0 0 0 0 Updated on May 18

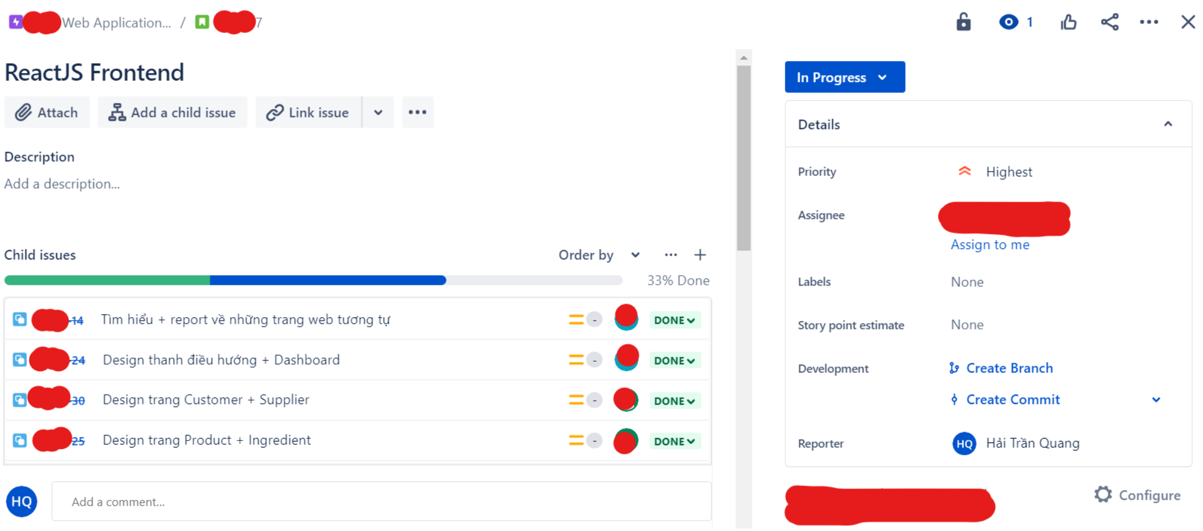
[redacted]notifier Private
Java 0 0 1 0 Updated on May 16

[redacted]payment Private
Java 0 0 1 0 Updated 5 days ago

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, and Jira was the chosen one.



[redacted]Web Application... / [redacted]7

ReactJS Frontend

Attach | Add a child issue | Link issue | ...

Description
Add a description...

Child issues — Order by — 33% Done

[redacted]14	Tìm hiểu + report về những trang web tương tự	[redacted] DONE
[redacted]24	Design thanh điều hướng + Dashboard	[redacted] DONE
[redacted]30	Design trang Customer + Supplier	[redacted] DONE
[redacted]25	Design trang Product + Ingredient	[redacted] DONE

HQ Add a comment...

In Progress

Details

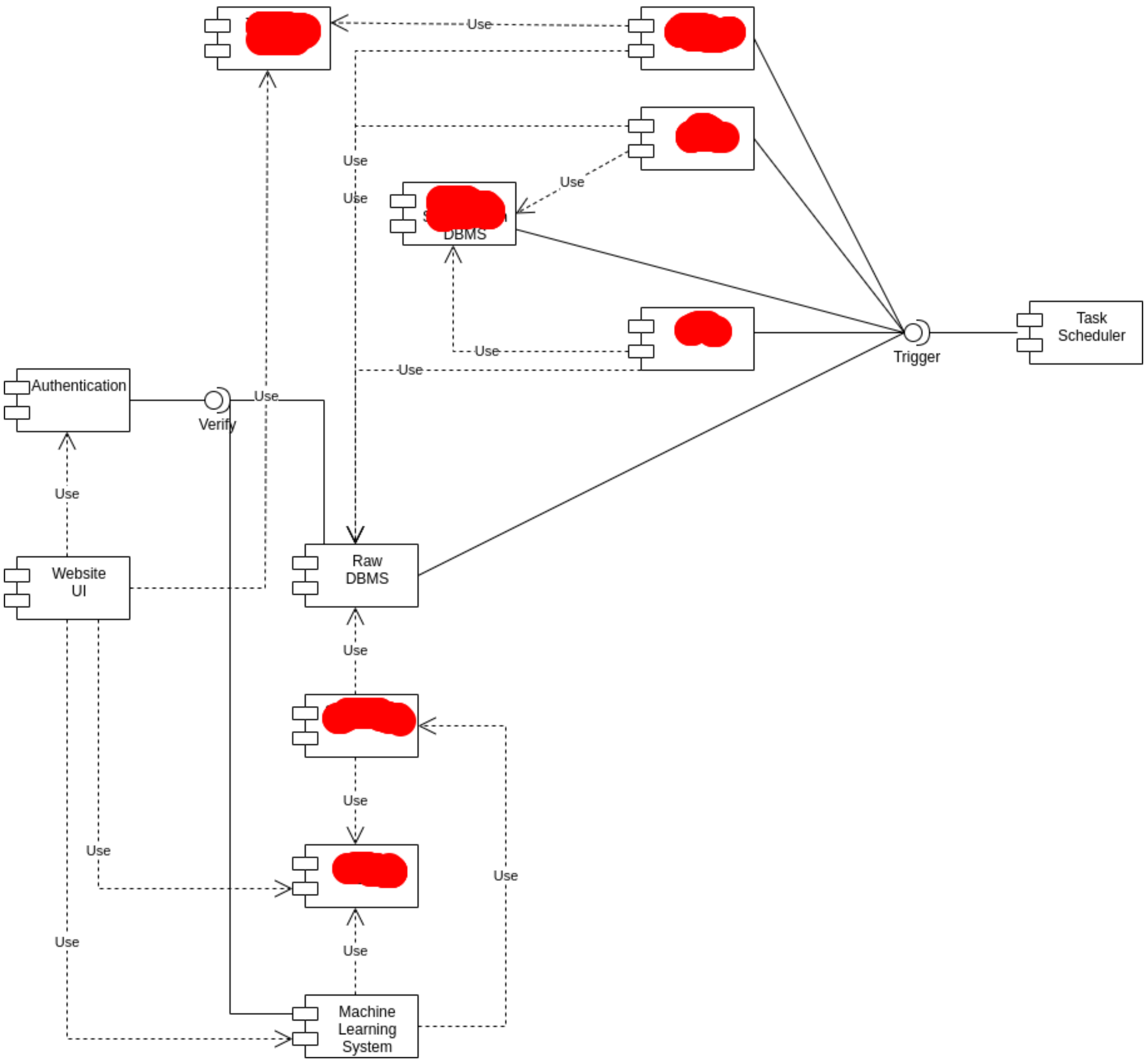
Priority	Highest
Assignee	[redacted] Assign to me
Labels	None
Story point estimate	None
Development	Create Branch Create Commit
Reporter	HQ Hải Trần Quang

[redacted]

Configure

Jira EPIC

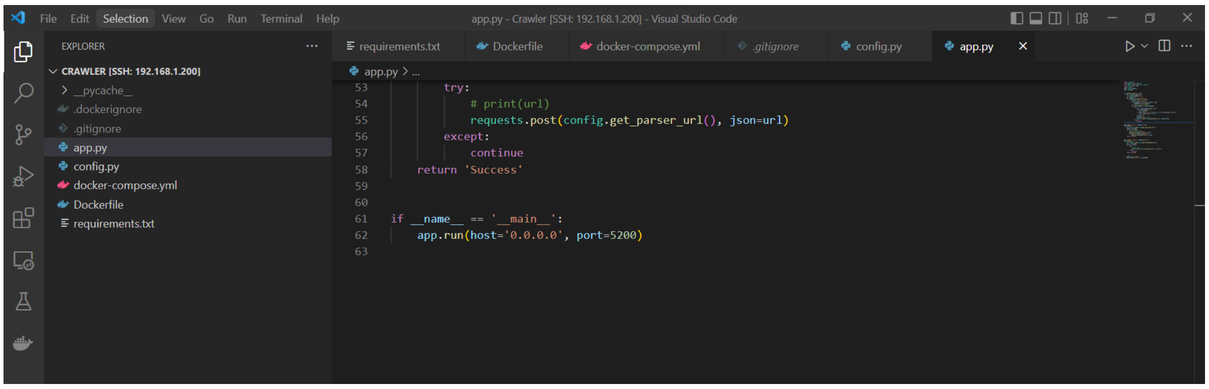
The third one - my graduation research project:



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.



```python
        try:
            # print(url)
            requests.post(config.get_parser_url(), json=url)
        except:
            continue
    return 'Success'


if __name__ == '__main__':
    app.run(host='0.0.0.0', port=5200)

```

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

Why do those boundaries between services exist so that we can divide them?

Let's revisit a famous and familiar concept, Separation of Concerns, which was first coined by Dijkstra:

"Let me try to explain to you, what to my taste is characteristic for all intelligent thinking. It is, that one is willing to study in depth an aspect of one's subject matter in isolation for the sake of its own consistency, all the time knowing that one is occupying oneself only with one of the aspects. [...]

It is what I sometimes have called "the separation of concerns", which, even if not perfectly possible, is yet the only available technique for effective ordering of one's thoughts, that I know of. This is what I mean by "focusing one's attention upon some aspect": it does not mean ignoring the other aspects, it is just doing justice to the fact that from this aspect's point of view, the other is irrelevant. It is being one- and multiple-track minded simultaneously."

Edsger W. Dijkstra, 1974, "On the role of scientific thought"

"From this aspect's point of view, the other is irrelevant", ask this question when facing a sub-problem that needs to be resolve. For example:

- Do we need to know anything about a store's customers when we authorize the owner of it? No? Then let the authentication API and customer information API belongs to two different microservices.

- Do we really need to know anything about user's information to show them a news on a news site (vnexpress for example)? No? Then let the user's details API and news API belongs to two different microservices.

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 application	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 or SSD) should always be faster than communication over the Internet.

***** Even though deployment for monoliths is easier, it requires downtime for the whole system for an update. Rolling updates can be performed, but still a larger portion of computing power needs to be "down" for an update. E.g: With 4 identical servers, we only have 75% computing power left during update time with monolithic structure, while for microservices, it would only affect one single service and one single server at a time with rolling updates.

***** If we take opportunity cost into consideration, it would be favourable for microservices. Although monoliths are easier to deploy, that does not mean those applications can be deployed frequently as each redeployment requires a longer testing process. Faster updates => More new features come out and more bugs are fixed.

V. REMARKS

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.

- Monolithic architecture also limits the choice about technology stack, including programming languages => frameworks and tools. Narrow range of technology also means you can only hire developers with very specific experience, and it can also limit their potential to contribute.

- 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.

- A "microservice" can still reach a point where it becomes "macro", and that's when we need to divide it into multiple smaller services.

- 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.

- It is relatively rare to see a project moving back to monolithic architectures after applying microservices.

- Microservices architecture is more prone to security threats since there are more servers to pay attention to and developers also need to configure access matrix for each service properly