Architecture Decisions

In this lesson, we'll study some key decisions and at what architecture level, micro or macro, they should be taken.

We'll cover the following

Microservices provide technological isolation. Therefore, it is possible to extend the concept of micro and macro architecture to **technical decisions**.

For deployment monoliths, these decisions, inevitably, must be implemented globally.

So, only for microservices, technical decisions can be made *within* the framework of macro or micro architecture. However, some decisions have to be part of the macro architecture. Otherwise, the integration will be compromised.

Micro and macro architecture decisions

Decisions can be taken in the context of **either micro or macro architecture**. Let's discuss each.

Programming languages, frameworks, and infrastructure

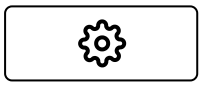
Programming languages, frameworks, and infrastructure can be defined for each **microservice** individually at the **micro architecture**.

- Then each microservice can also be implemented with a different language.
- Technology, such as the application server, that best suits the specific problems of each microservice can be used.

Programming languages, frameworks, and infrastructure can be defined uniformly for all microservices in the **macro architecture**. This is useful if:

- a company's technology strategy allows only certain technologies
- therefore, only developers with knowledge in certain technologies are hired

Database



At first glance, this decision seems to be comparable to the decision concerning the programming languages, frameworks, and infrastructure but databases are different because:

- They store data.
- The loss of data is usually unacceptable.
  - Therefore, there must be a backup strategy and a disaster recovery strategy for a database.
  - Setting these up for many different databases requires considerable effort.

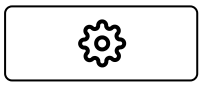
Micro: Each microservice can also have its own instance of the database. If databases were defined at the **micro architecture**.

- A crash of one database will cause only one microservice to fail which makes the entire app **more robust**.
- However, the **higher effort involved**, especially concerning operation, is an argument against individual instances.

Macro: To avoid needing many different databases, the database can be defined as part of the **macro architecture** for all microservices.

- Even if the database is defined in the macro architecture, **multiple microservices must not share a database schema**. That would contradict the bounded contexts (https://www.educative.io/collection/page/10370001/6518081205567488/4953894968885248).
- The domain model in the database schema would be used by several microservices. This would couple the microservices too strongly. Even with a unified database, the microservices must have separate schemata in the database.

User interface



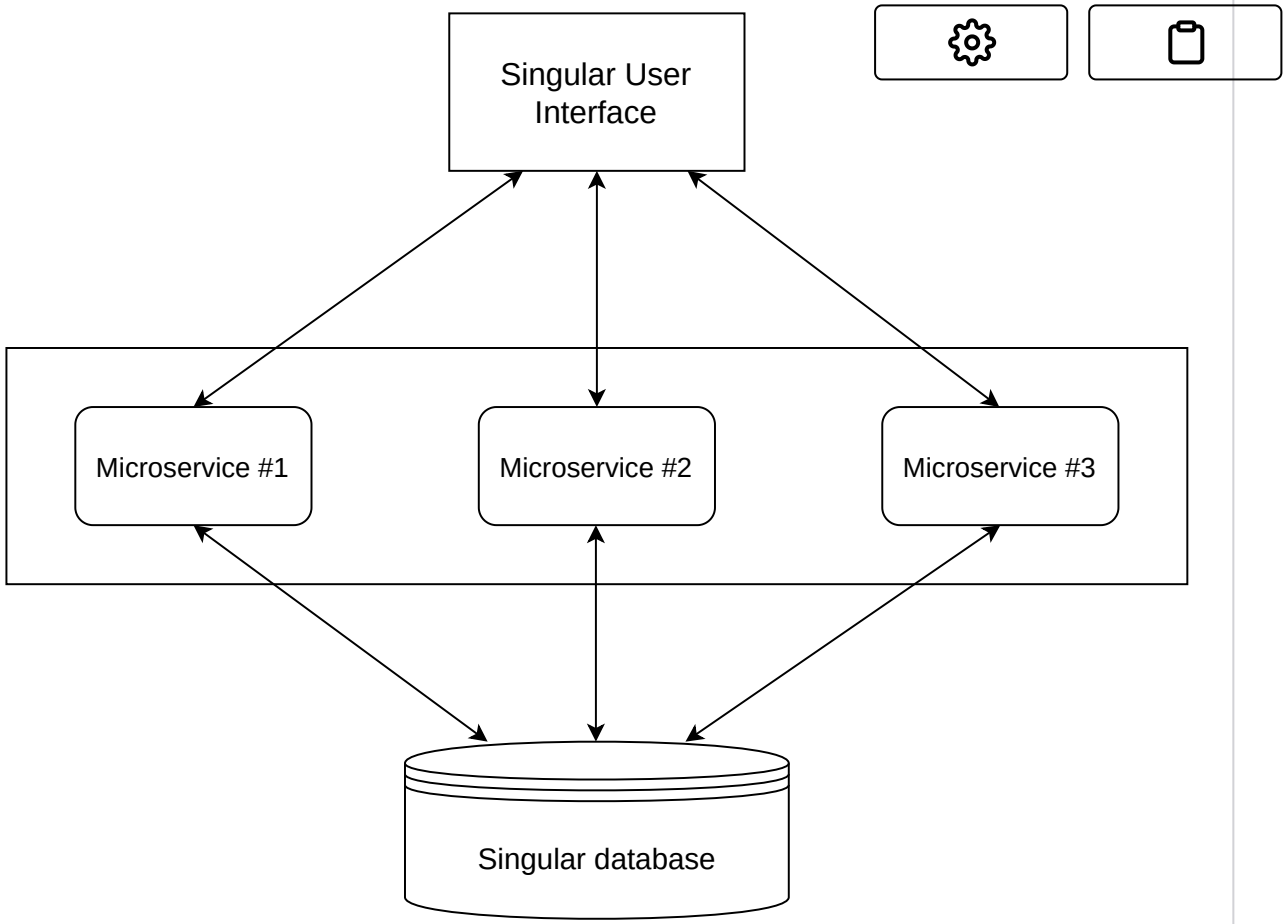
If microservices have their own user interface (UI), the *look and feel* of microservices can be a micro or macro architecture decision.

Micro: sometimes a system has **different types of users** (back office and customers, for example) with **different requirements** for the UI which are often incompatible with a uniform look and feel. A micro architecture decision for the UI is suitable in this case.

- Often there are concerns that a microservice level decision **will cause inconsistencies** in the look and feel; however, the UI can also **diverge in a monolithic system**. Hence, defining appropriate style guides and artifacts is the only way to achieve a consistent look and feel for large systems, regardless of the use of microservices.

Macro: Often a system should have a uniform UI; therefore, the look and feel must be a macro architecture decision.

- Shared `CSS` and `JavaScript` are often not enough to ensure a common style of the UI of all microservices since uniform technical artifacts can be used to implement very different types of user interfaces. Therefore, a **style guide must become part of the macro architecture**.



The UI and the database are a macro architecture decision

1 of 2

Documentation

It may be necessary to standardize the *documentation*.

Micro: The documentation should be part of the micro architecture if the same team will build and maintain the microservice.

- A certain level of documentation makes it easier to later hand the microservice over to another team.
- It may also be necessary to document certain aspects of the microservices in a uniform manner.

- For example, for security reasons, some systems need to keep track of the libraries used in the microservices. In the case of a security vulnerability in a specific library, it is then possible to identify which microservices need to be fixed.

Macro: Of course, the decision about the documentation can also be part of the macro architecture.

- Standardized documentation can provide an overview of the system and the dependencies between microservices.

Typical macro architecture decisions

There are some decisions that must always be taken at the level of macro architecture. Ultimately, all microservices together should result in a coherent system. This requires some standards.

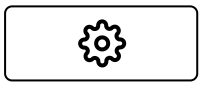
Communication protocol

The *communication protocol* of the microservices is a typical macro architecture decision.

- Only if all microservices provide a **uniform interface**, for example, a REST interface or a messaging interface, can they communicate with each other coherently.
- In addition, the data format must be **standardized**. It makes a difference whether systems communicate with JSON or XML, for example.

If the communication protocol was a **microservice decision**, i.e., a different communication channel between each microservice, **a coherent system will not exist** and will disintegrate into islands that communicate

with each other in different ways.



Authentication

With *authentication*, a **user proves their identity**. This can be done with a password and a username, to name a common example.

Since it is unacceptable for the user to re-authenticate with every microservice, the entire microservice system should use a **single authentication system**. The user then enters a username and password once and can then use any microservice.

Integration

Integration testing technology is also a typical macro architecture decision. All microservices must be tested together, so they must run together in an integration test. The macro architecture must define the necessary prerequisites for this.

Typical micro architecture decisions

Certain decisions should be taken for each microservice individually. Therefore, they are typically part of the micro architecture.

Authorization

The *authorization* of the user determines what a user is **allowed to do**. The authorization should be done in the respective microservice.

Which user is allowed to initiate what action, i.e., authorization, is part of the domain logic, and therefore belongs to the microservice like the other domain logic.

If this was decided at the macro architecture, the domain logic would be implemented in a microservice itself, but the decision about **which part** of the domain logic is available to which user would be made centrally, which is difficult, especially with complex rules.

- For example, if orders up to a certain upper limit can be triggered by certain users, authorization, concrete upper limits, and possible exceptions belong to the microservice **order**.

Authentication assigns the user roles used in authorization.

- For example, a microservice can define which actions a user with the role of *customer* can trigger and which actions a user with the role of *call center agent* can trigger.

Testing

The *testing* can be different for each microservice. Even the tests are ultimately part of the domain logic.

In addition, there may be different non-functional requirements for each microservice.

- For example, one microservice can be particularly performance-critical, whereas another is more safety-critical.

These risks must be covered by an individual focus in the tests.

Since the tests can be different, the **continuous delivery pipeline** is also different for each microservice. It must include the relevant tests. Of course, the technology for the continuous delivery pipeline can be standardized.

- For example, each pipeline can use a tool like `Jenkins`. What happens in the respective pipelines, however, depends on the respective microservice.

Jenkins

To Summarize

The following table shows the typical micro and macro architecture decisions:

Micro or Macro	Micro Architecture	Macro Architecture
Programming Language	Continuous Delivery Pipeline	Communication Protocol
Database	Authorization	Authentication
Look and Feel	Tests of the Microservice in Isolation	Integration Tests
Documentation		

QUIZ

Z

1

Suppose that it has been decided to use a REST interface between microservices for communication. What sort of technical decision could this have been?

- A) Strictly macro
- B) Strictly micro
- C) Could have been either, it’s difficult to say.

Submit Answer

Question 1 of 3
0 attempted

Reset Quiz

In the next lesson, we’ll discuss some factors that influence the operation of applications.

Back

Next

Strategic Design & Common Patterns

Operation: Micro or Macro Architecture?

Mark as Completed

Report an Issue