More on Inconsistencies

In this lesson, we'll further explore the CAP theorem.

We'll cover the following

Are inconsistencies acceptable?

As per the last lesson, **the inconsistency of an asynchronous system is inevitable unless you want to give up availability**.

It is therefore important to know the requirements for consistency, which requires some skill. Customers want a reliable system, data inconsistency seems to contradict this. That’s why it is important to know what happens when the data is temporarily inconsistent and whether this really causes problems.

After all, the inconsistencies should usually disappear after a few seconds. Besides, **certain inconsistencies can even be tolerable** from a domain perspective.

- For example, if goods are listed days before the first sale, inconsistencies are initially acceptable and must only be corrected when the goods are finally being sold.

If **inconsistencies are not acceptable at all, asynchronous communication is not an option**.

This means that synchronous communication must be used with all its disadvantages. If the tolerance for temporarily inconsistent data is not known, this can lead to a wrong decision regarding the communication variant.

Repairing inconsistencies

Guaranteed order of events

In the simplest case, the inconsistencies disappear as soon as all events have reached all systems. However, there may be exceptions. An example:

- After registration, a customer receives an initial credit balance.
- A microservice receives the event for the initial balance, but it has not yet received the registration of the user.
- The microservice cannot credit the initial balance because the customer has not been created in the system.
- If the registration of the customer arrives, later on, the initial balance would have to be executed once again.

This problem can be solved when **the order of events can be guaranteed**. In this case, the problem will not arise in the first place. Unfortunately, many solutions cannot guarantee the order of events.

Event sourcing

Event sourcing can also help. This allows the microservice to always **reconstruct its state from the events**. Therefore, the state could be discarded and recreated from the events as long as those are available without any gaps.

Extending domain logic

It is also possible to extend the domain logic. In that case, if the event for the initial balance is received before the registration, a new customer object is created, but the object is marked as invalid as long as the data from the registration is missing.

The rest of the logic would need to handle these invalid customers. That **might make the business logic quite complex**.

An error might be hard to spot because now there are so many states that an object might have. Therefore, **this solution should probably be avoided**.

QUIZ

1 A system cannot be asynchronous if _____ is completely unacceptable.

A) cross OS incompatibility

A) cross-DB incompatibility

B) network partitioning

C) inconsistency

Submit Answer

Question 1 of 2
0 attempted

Reset Quiz

In the next lesson, we'll look at some other challenges.

← Back

Challenges: Inconsistencies & CAP Th...

Next →

Other Challenges

Mark as Completed

Report an Issue