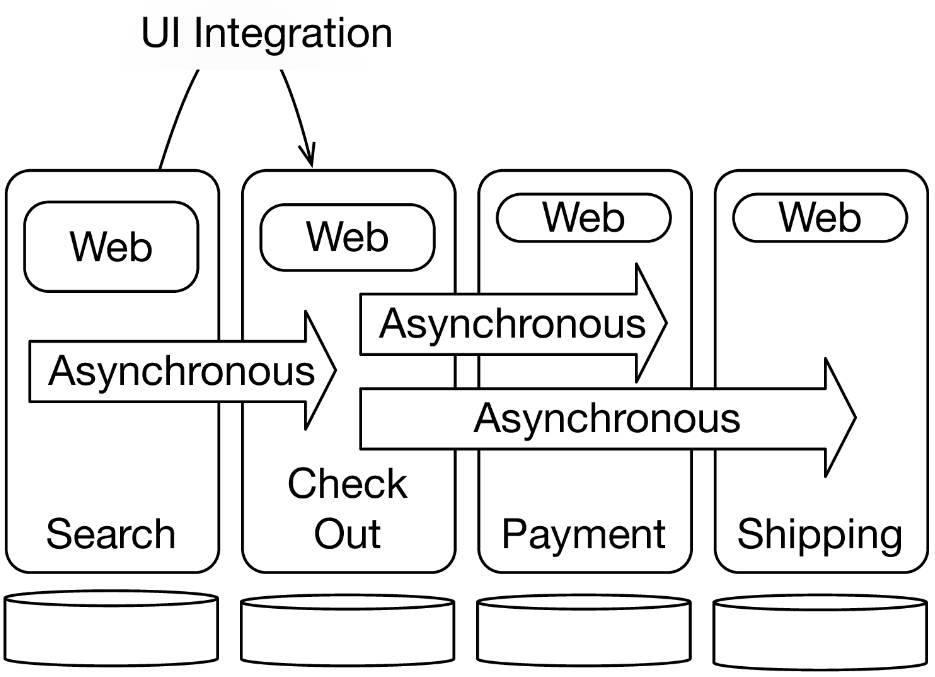
An Example

In this lesson, we'll look at an example of an SCS.

WE'LL COVER THE FOLLOWING

E-commerce example



Example of an SCS Architecture

The drawing above demonstrates how an e-commerce system divided into the bounded contexts **search, check-out, payment, and shipping** can be implemented with SCSs.

1. One SCS implements the **search** for products.

2. At **check out**, a filled shopping cart is turned into an order.

3. **Payment** ensures that the order is paid and provides information about

the payment.

4. **Shipping** sends the goods to the customer and offers the customer the possibility to obtain information regarding the state of the delivery.

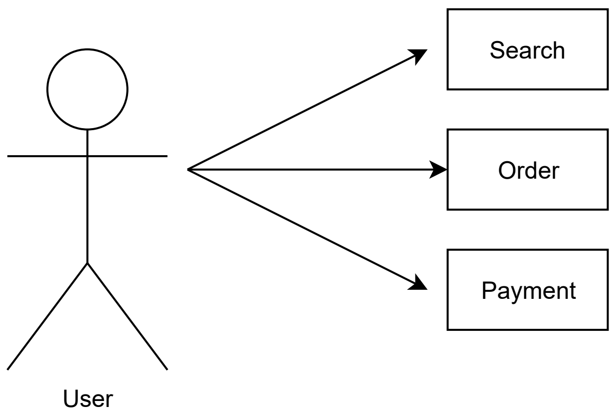
Each SCS implements a bounded context with its own database, or, at least, one schema in a common database. In addition to the logic, each SCS also contains the web UI for the respective functionalities.

An SCS does not necessarily have to implement a bounded context, but this achieves a high degree of independence of the domain logic.

Communication

A user sends an HTTP request to the system. The HTTP request is then usually processed by a single SCS because all the necessary data is available in the SCS.

This is good for performance and also for resilience. The failure of one SCS does not lead to the failure of another SCS, because the SCS does not call any other systems when processing the requests. Slow calls of other SCSs via the network are also avoided, which promotes performance.



The user communicates directly with all SCSs. They generally do not communicate with each other.

Of course, the SCSs must still communicate with each other, they are, after all, part of an overall system. One reason for communication is the **lifecycle of an order**.

1. The customer searches for products in **search**
2. Orders them in **check out**
3. Pays them in **payment**

4. Tracks them in **shipping**

When moving from one step to the next, the **information about the order must be exchanged between the systems**. This can be done **asynchronously**. The information does not have to be available in the other SCSs until the next HTTP request; therefore, **temporary inconsistencies are acceptable**.

In some places, close integration seems to be necessary. This means that *search*, for example, must display not only the search results, but also the contents of the shopping cart.

However, the shopping cart is managed by *check out*. UI integration can be a solution to challenges like these. In that case, the *check out* SCS can decide how the shopping cart is displayed, and the representation will be integrated in the other SCSs. Even changes to the logic rendering the shopping cart will be implemented only in one SCS, although many SCSs will display the shopping cart as part of their web pages.

QUIZ

1 Is it possible to display a component of an SCS in other SCSs?

COMPLETED 0%

1 of 2

In the next lesson, we'll study the differences between self-contained systems and microservices.