**Solution: Monoliths and Microservices**

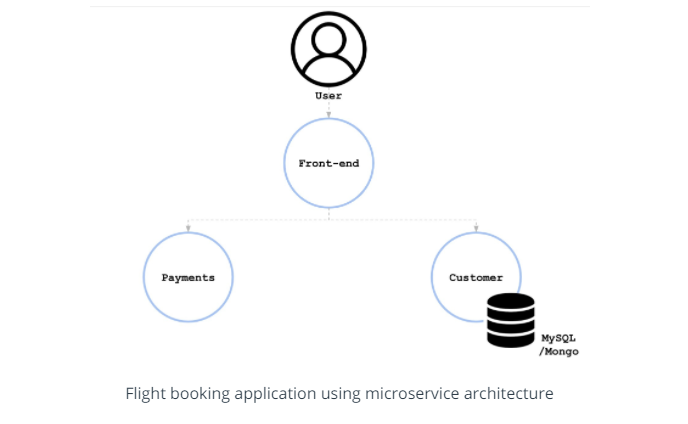
**Summary**

Given the scenario, it is paramount to choose an architecture that would be **replicable and scalable**. For example, if thousands of customers access the payment service in the same timeframe, then this particular service should be scaled up. In a monolith architecture, scaling up creates a replica of everything, including front-end and customer services, in addition to the payment service. This will also consume more resources on the platform, such as CPU and memory, and takes longer to spin up.

On the other side, a microservice is a lightweight component that requires fewer resources (CPU and memory) and less time for provisioning.

For this example, a microservice-based architecture is chosen, based on considerations that the application is a central booking system for multiple airlines, that implies a high load. The main components are:

* Front-end - entry-point for the user, where they will choose their airline or choice
* Customer - requires a database (MySQL or Mongo) to store the customer details
* Payments - to implement PayPal and Debit based operations



Additionally, the "payments" microservice is capable of handling multiple payment systems. Interaction with the PayPal interface and management of debit card APIs are fundamentally different. The "payments" component is a monolith that can be divided into multiple parts.

Payments:

* PayPal - handling all the PayPal payments
* Debit - handling all the debit card payments

