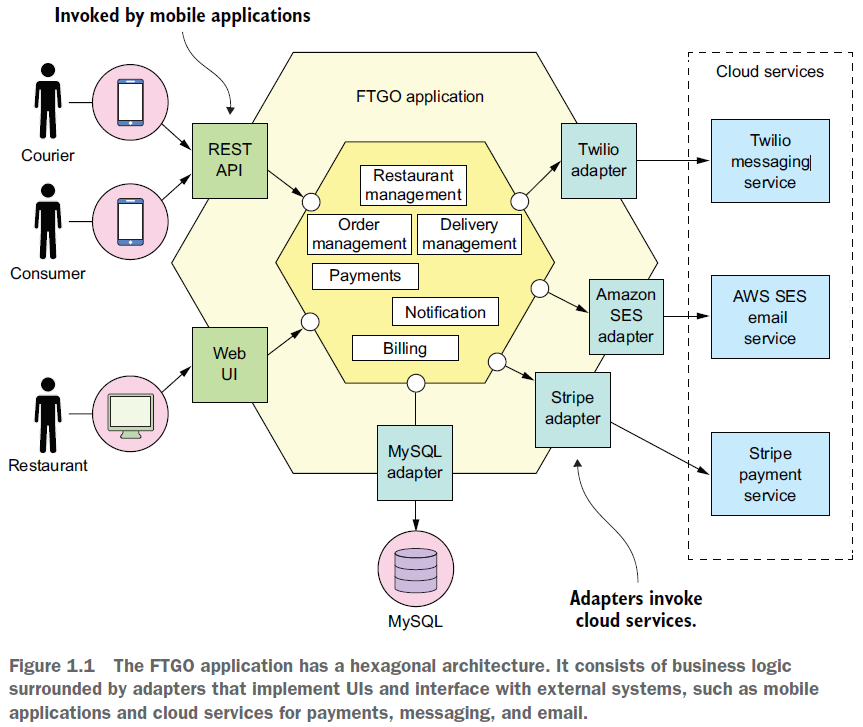
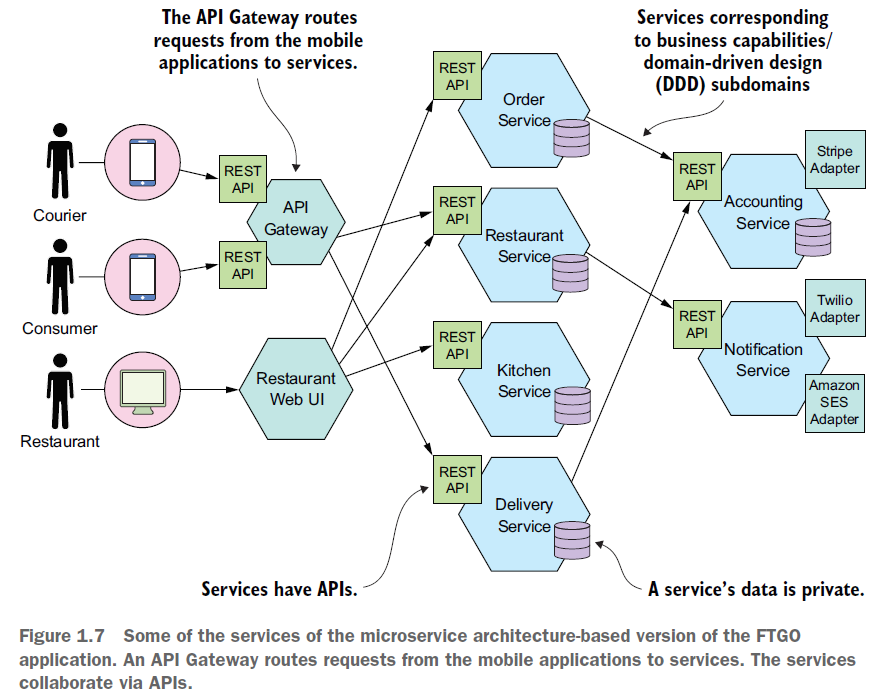
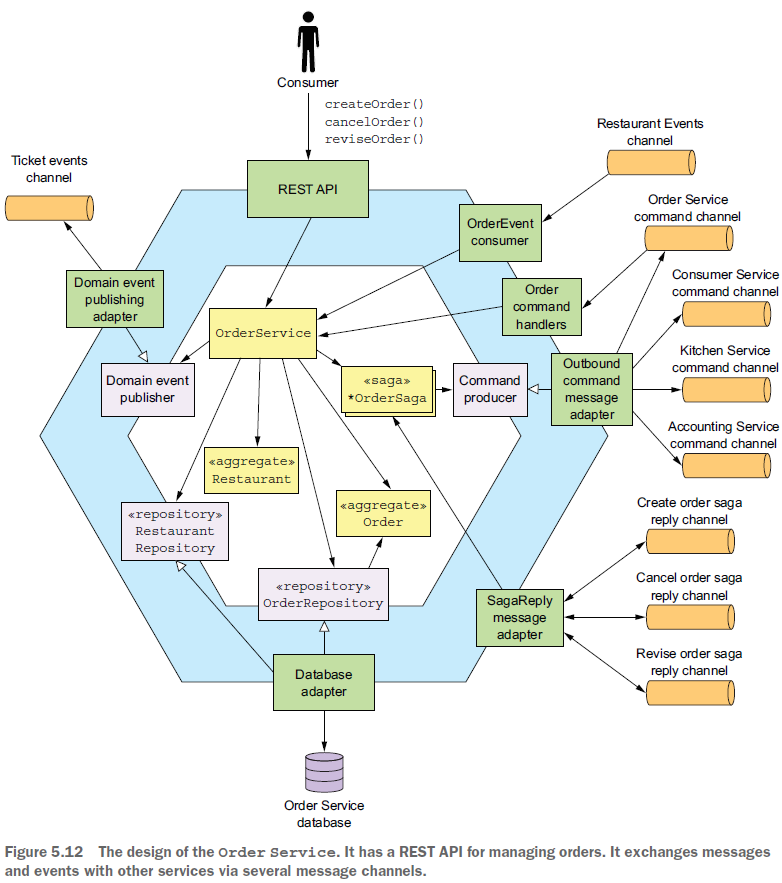
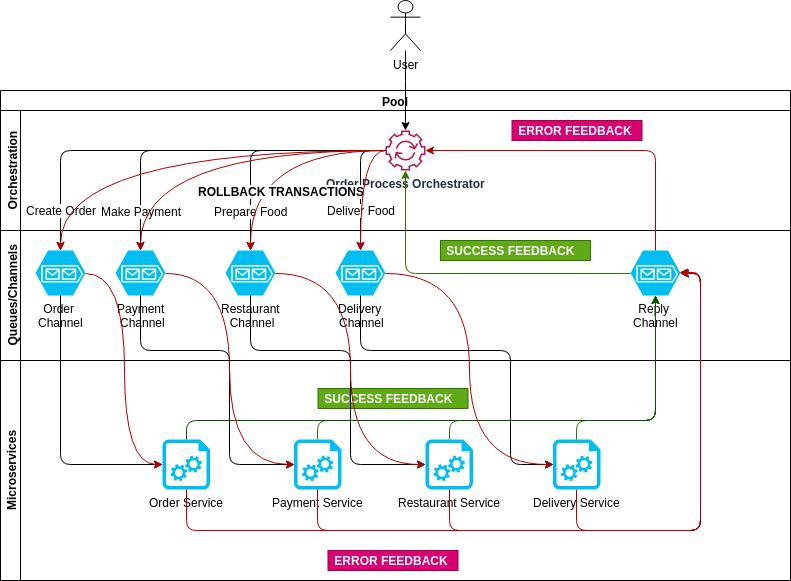
<https://github.com/SonNXP/ftgo-application>

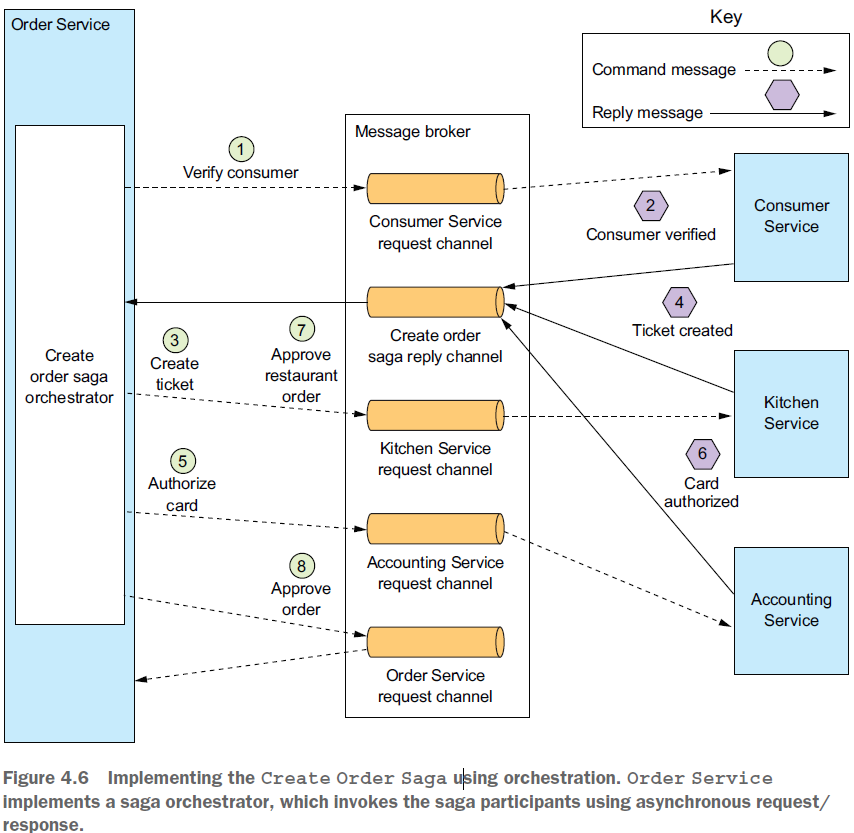






REST API: @Controller or @RestController

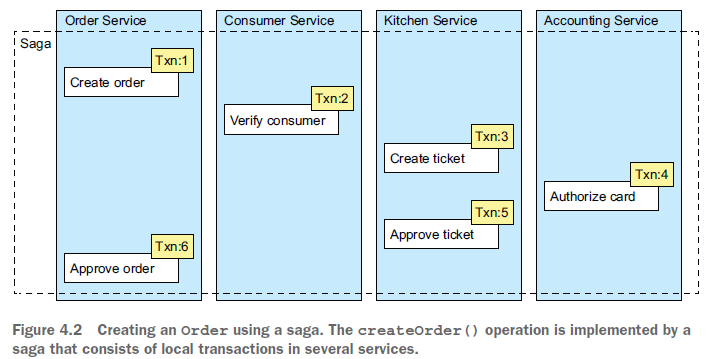




Order Service first creates an Order and a Create Order Saga orchestrator. After that,

the flow for the happy path is as follows:

<https://eventuate.io/docs/manual/eventuate-tram/latest/getting-started-eventuate-tram-sagas.html>



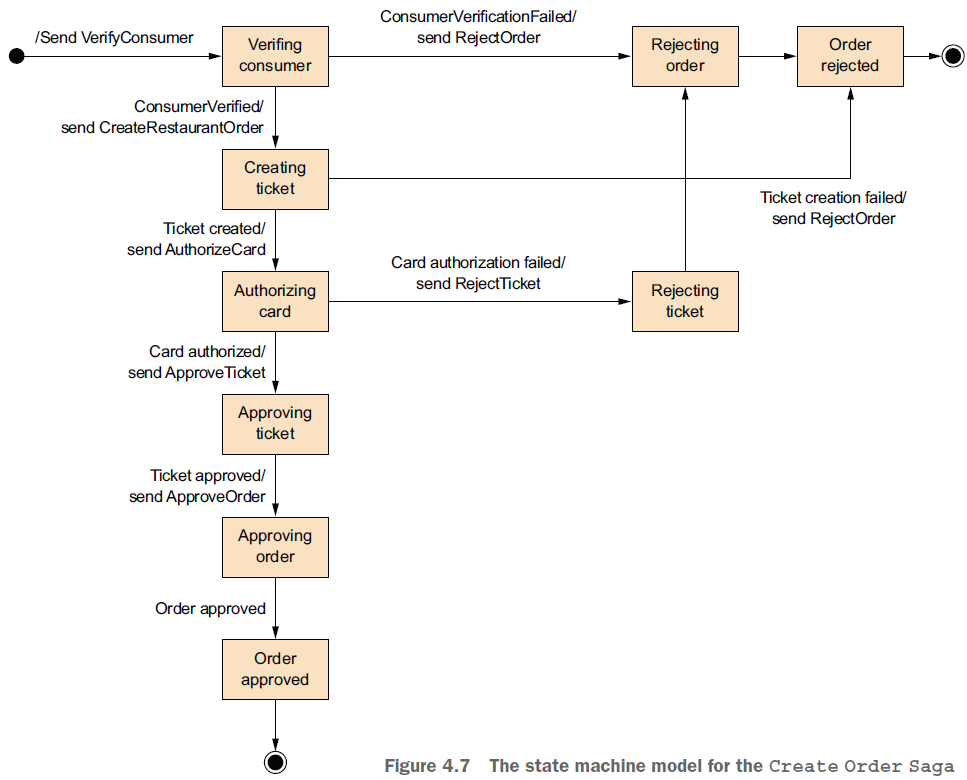


Figure 4.7 shows the state machine model for the Create Order Saga. This state

machine consists of numerous states, including the following:

 Verifying Consumer—The initial state. When in this state, the saga is waiting

for the Consumer Service to verify that the consumer can place the order.

 Creating Ticket—The saga is waiting for a reply to the Create Ticket command.

 Authorizing Card—Waiting for Accounting Service to authorize the consumer’s

credit card.

 Order Approved—A final state indicating that the saga completed successfully.

 Order Rejected—A final state indicating that the Order was rejected by one of

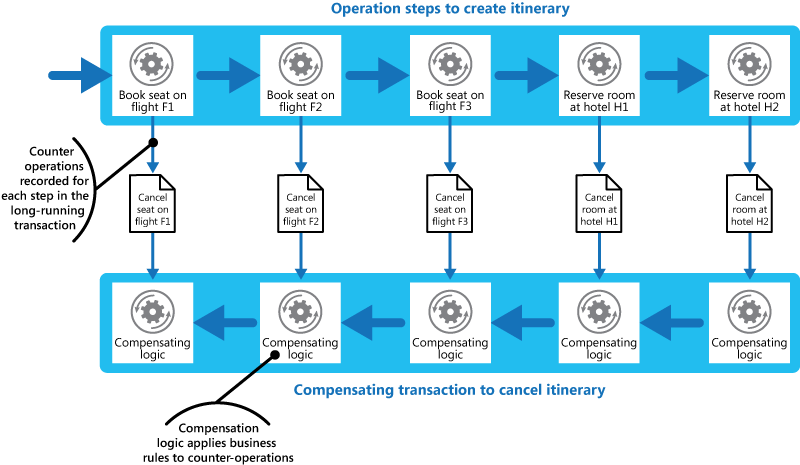
the participants.

### [Giao dịch có kỳ hạn (Forward Transaction)](http://quantri.vn/dict/details/14094-giao-dich-co-ky-han-forward-transaction)

<https://en.wikipedia.org/wiki/Compensating_transaction>

<https://docs.microsoft.com/en-us/azure/architecture/patterns/compensating-transaction>

Use this pattern only for operations that must be undone if they fail. If possible, design solutions to avoid the complexity of requiring compensating transactions.



<https://www.prakharsrivastav.com/posts/saga-orchestration-in-microservices/>

# Understanding Orchestration[⌗](https://www.prakharsrivastav.com/posts/saga-orchestration-in-microservices/#understanding-orchestration)

Let’s continue with our previous example of a food delivery system. To implement the SAGA design pattern, we need a central orchestrator called Order Orchestrator. The orchestrator can be a process manager that receives the initial order request. Its core responsibilities are.

* Receive process initiation request and call the first service.
* Listen to success or failure feedback from the currently running service.
* For successful feedback, ask the next service to proceed.
* For failure feedback, relay a message to all participation services to rollback their transactions.