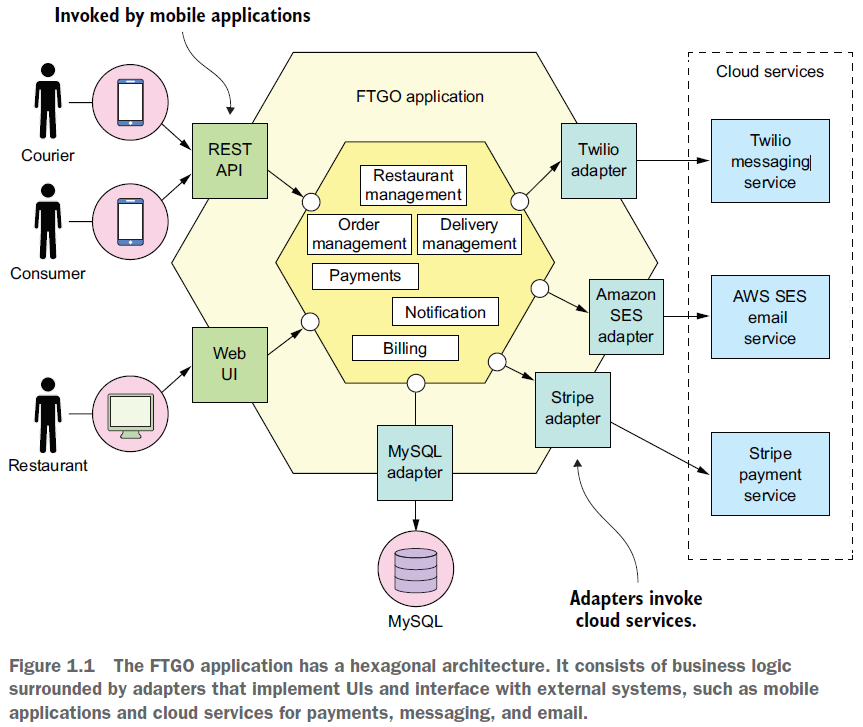
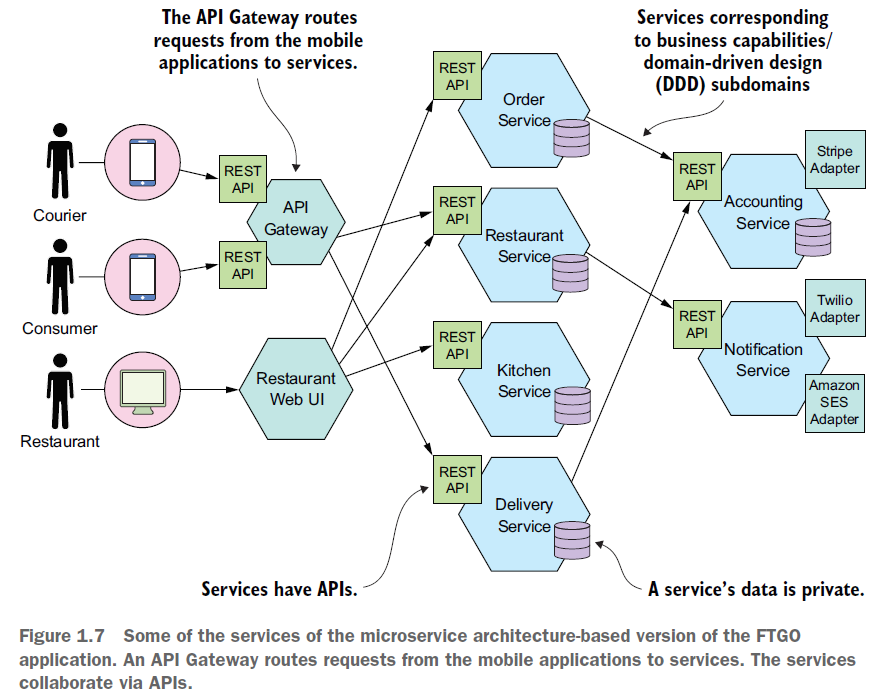
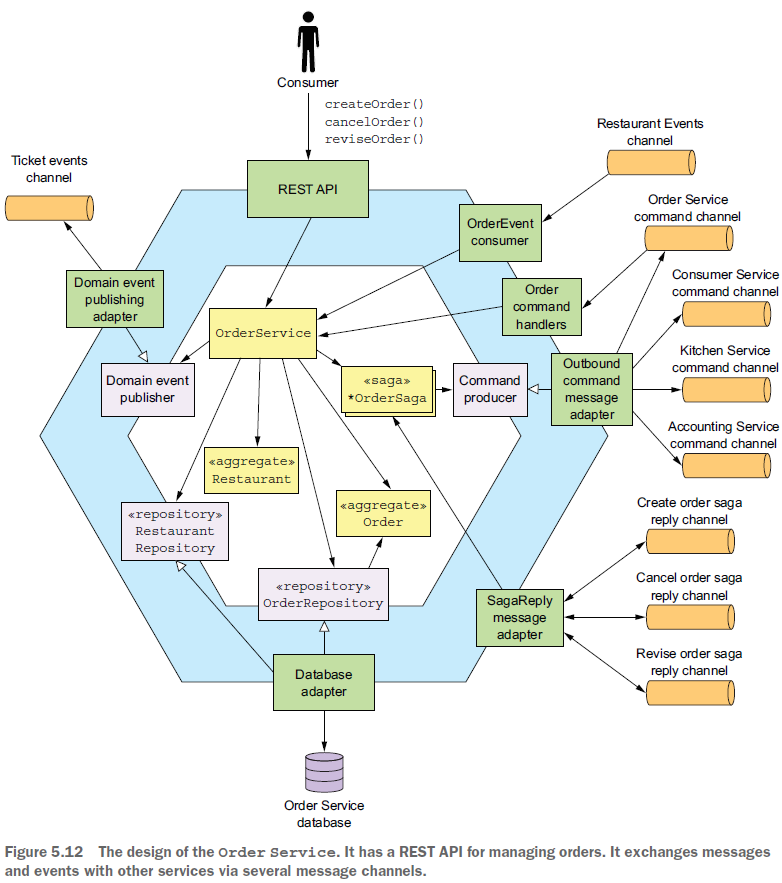
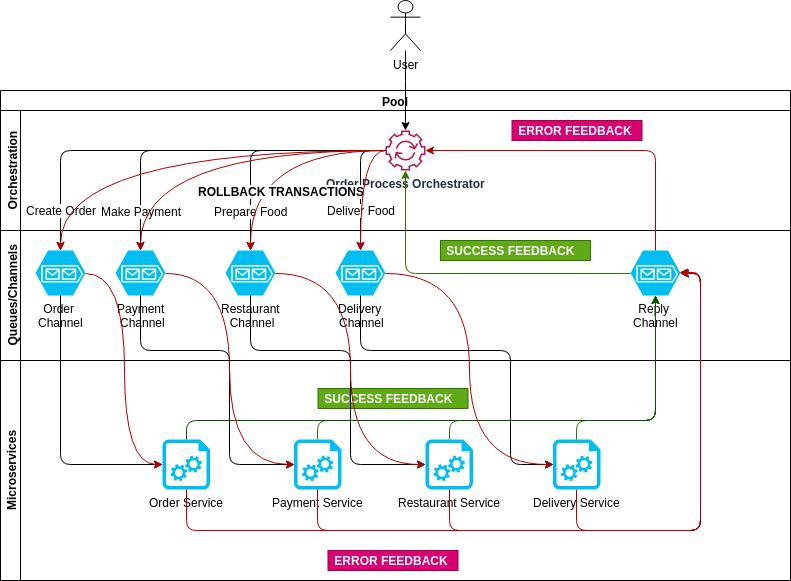
<https://github.com/SonNXP/ftgo-application>

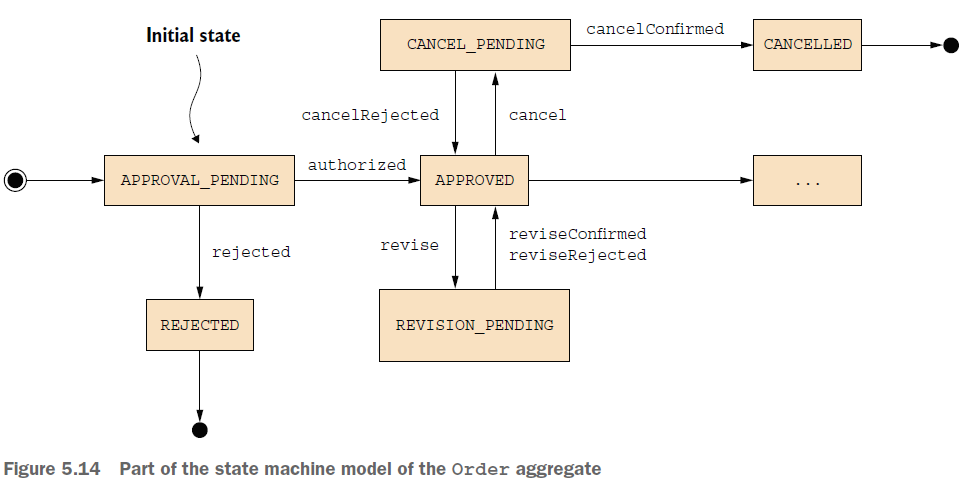




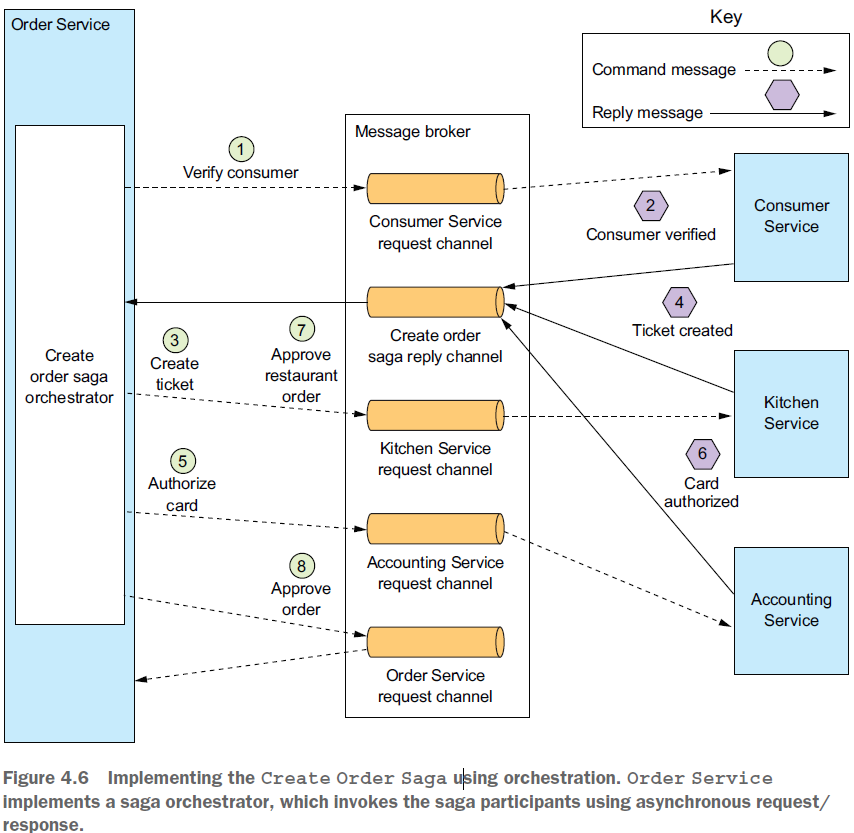


REST API: @Controller or @RestController





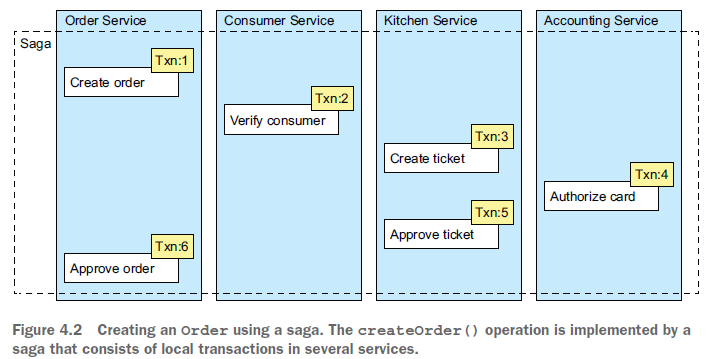
Similarly, other Order Service operations such as revise() and cancel() first change the Order to a pending state and use a saga to verify that the operation can be performed. Then, once the saga has verified that the operation can be performed, it changes the Order transitions to some other state that reflects the successful outcome of the operation. If the verification of the operation fails, the Order reverts to the previous state. For example, the cancel() operation first transitions the Order to the CANCEL\_PENDING state. If the order can be cancelled, the Cancel Order Saga changes the state of the Order to the CANCELLED state. Otherwise, if a cancel() operation is rejected because, for example, it’s too late to cancel the order, then the Order transitions back to the APPROVED state.



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>



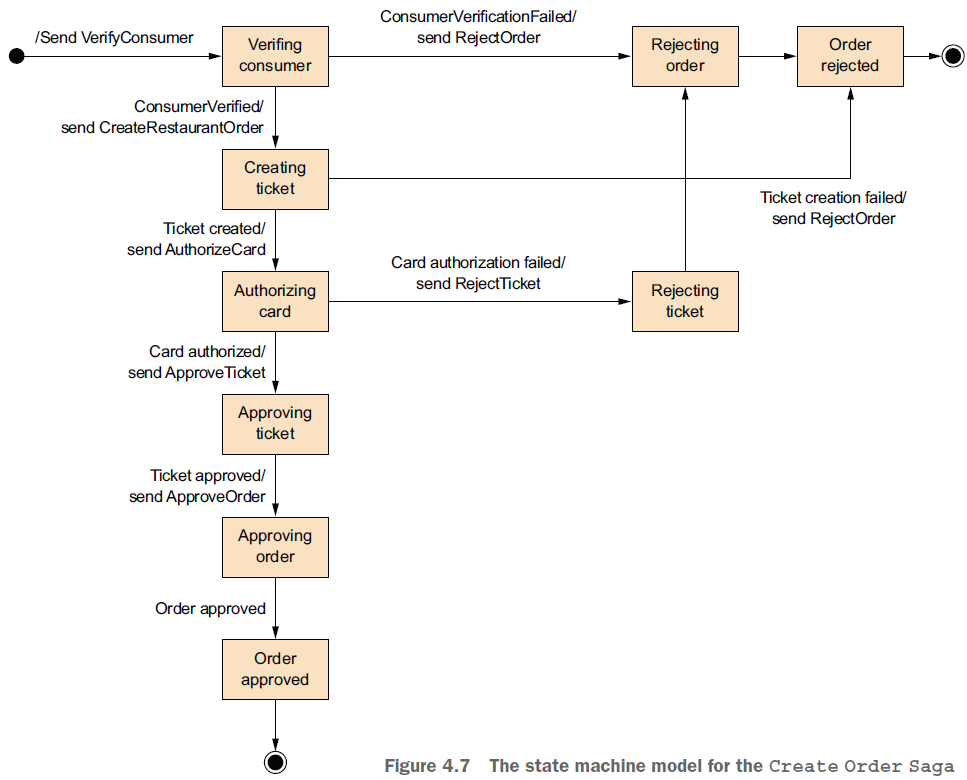


Figure 4.7 shows the state machine model for the Create Order Saga. This state

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

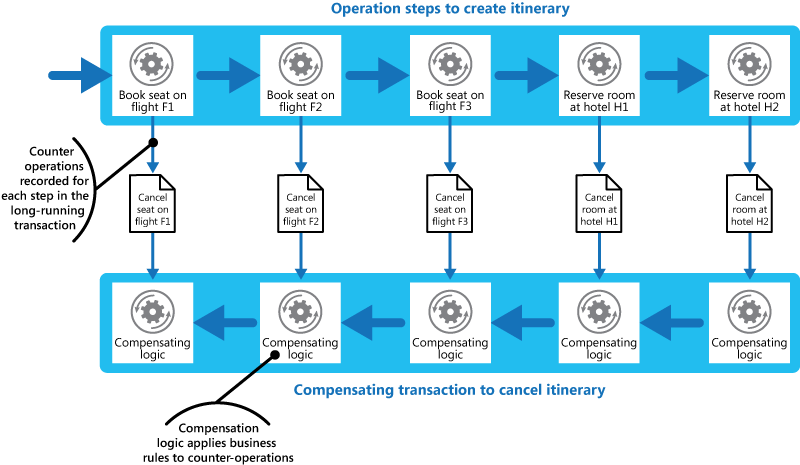
the participants.

### [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)

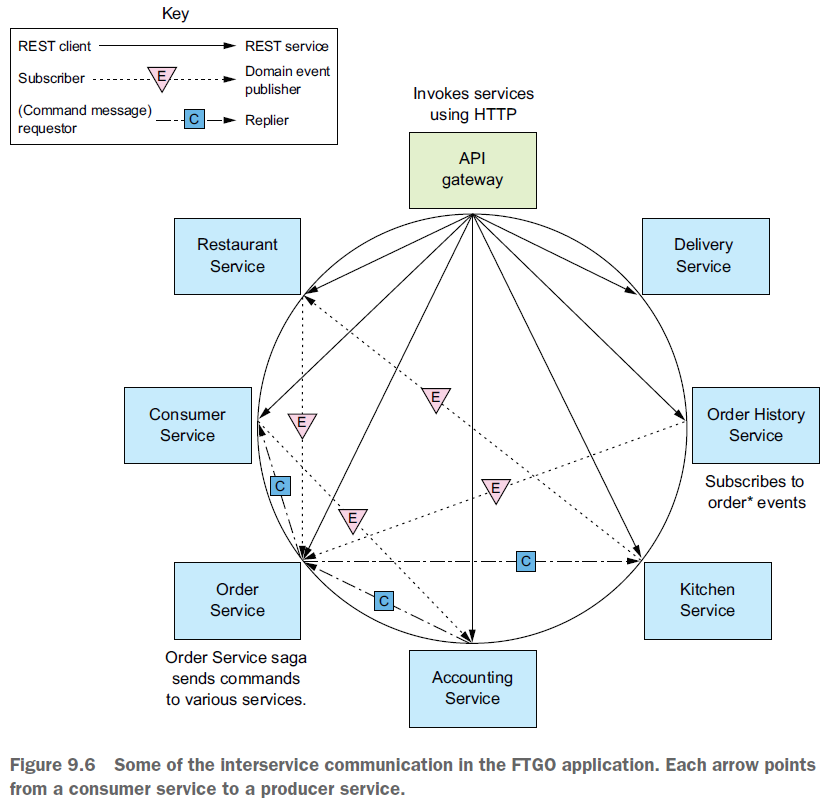
Let’s continue with our previous example of a food delivery system. To implement the SAGA design pattern, we need a central orchestrator called Order Orchestrator. The orchestrator can be a process manager that receives the initial order request. Its core responsibilities are.

* 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.

Revise:

The revise() method is called to initiate the revision of an order. Among other

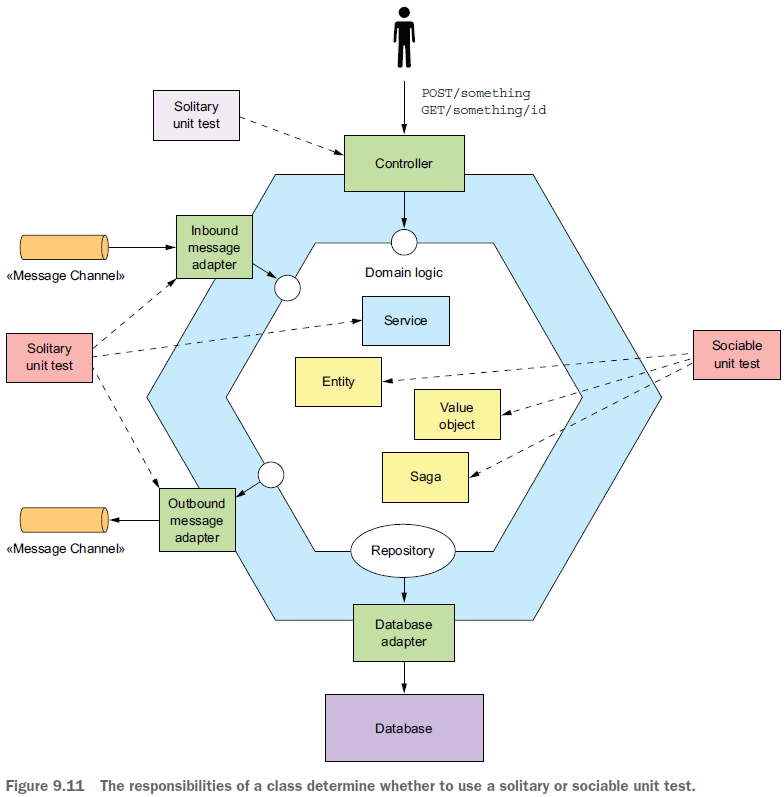
* things, it verifies that the revised order won’t violate the order minimum (check internal logic of this service first) and changes
* the state of the order to REVISION\_PENDING. Once Revise Order Saga has successfully
* updated Kitchen Service and Accounting Service, it then calls confirmRevision()
* to complete the revision.



There are two types of unit tests (https://martinfowler.com/bliki/UnitTest.html):

 *Solitary (đơn độc) unit test*—Tests a class in isolation using mock objects for the class’s dependencies

 *Sociable (hòa đồng) unit test*—Tests a class and its dependencies. Detail in OrderTest.java. These kind of unit test does NOT use mocks, stubs. It makes real needed (dependencies) objects with real values for testing and use AssertEquals to check.



 *Solitary (đơn độc) unit test*—Tests a class in isolation using mock objects for the

class’s dependencies

 *Sociable (hòa đồng) unit test*—Tests a class and its dependencies. These kind of

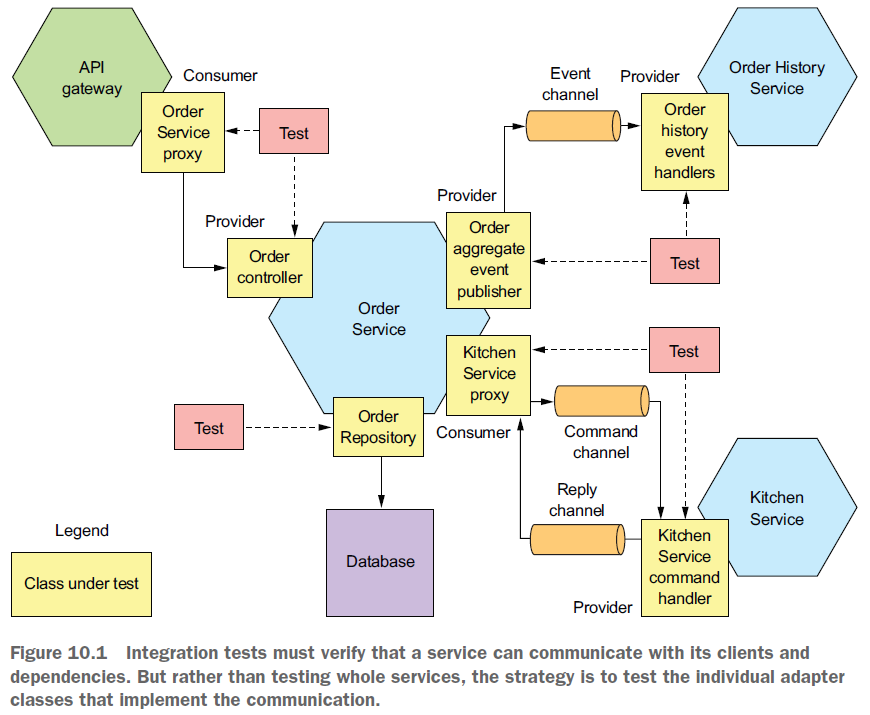
unit test does NOT use mocks, stubs. It makes real needed (dependencies) objects, concrete objects

with real values for testing and use AssertEquals to check.

<https://www.baeldung.com/spring-reflection-test-utils> Set private field, cal private methods

Unit test for Rest API can use @WebMvcTest, MockMvc

Test with server (TomCat) @TestRestTemplate



Run Unittest for Order service: <https://github.com/SonNXP/ftgo-application/tree/master/ftgo-order-service>

Some unit test is not SpringBootTest. It uses Junit only. JUnit is a unit testing framework for Java programming language.

## **Features of JUnit**

* JUnit is an open source framework, which is used for writing and running tests.
* Provides annotations to identify test methods.
* Provides assertions for testing expected results.
* Provides test runners for running tests.
* JUnit tests allow you to write codes faster, which increases quality.
* JUnit is elegantly simple. It is less complex and takes less time.
* JUnit tests can be run automatically and they check their own results and provide immediate feedback. There's no need to manually comb through a report of test results.
* JUnit tests can be organized into test suites containing test cases and even other test suites.
* JUnit shows test progress in a bar that is green if the test is running smoothly, and it turns red when a test fails.

Run gradlew.bat test

With SpringBootTest: it makes Test App Context for each file so don’t need MAIN() to run

<https://junit.org/junit5/docs/current/user-guide/> Nếu ko dùng @SpringBootTest thì xem Run Test

**JUnit 5 = JUnit Platform + JUnit Jupiter + JUnit Vintage**

// import org.junit.Test;

import org.junit.jupiter.api.Test;

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

Build the services using this command:

./gradlew assemble

about ./gradlew assemble: Assembles the outputs of this project

run test on each unit test file: error

java.lang.NoClassDefFoundError: org/junit/runner/manipulation/Filter

**Step by step**:

1. Temporary: Build the Spring Cloud Contracts using this command:

E:\Documents\Books\Java\Projects\ftgo-application>gradlew buildContracts

BUILD SUCCESSFUL in 9s

1 actionable task: 1 executed

1. Build the services using this command:

gradlew assemble

All services build with error:

Task :ftgo-restaurant-service:compileJava UP-TO-DATE

Errors occurred while build effective model from C:\Users\Admin\.gradle\caches\modules-2\files-2.1\com.sun.xml.bind\jaxb-core\2.2.11\db0f76866c6b1e50084e03ee8cf9ce6b19becdb3\jaxb-core-2.2.11.pom:

'dependencyManagement.dependencies.dependency.systemPath' for com.sun:tools:jar must specify an absolute path but is ${tools.jar} in com.sun.xml.bind:jaxb-core:2.2.11

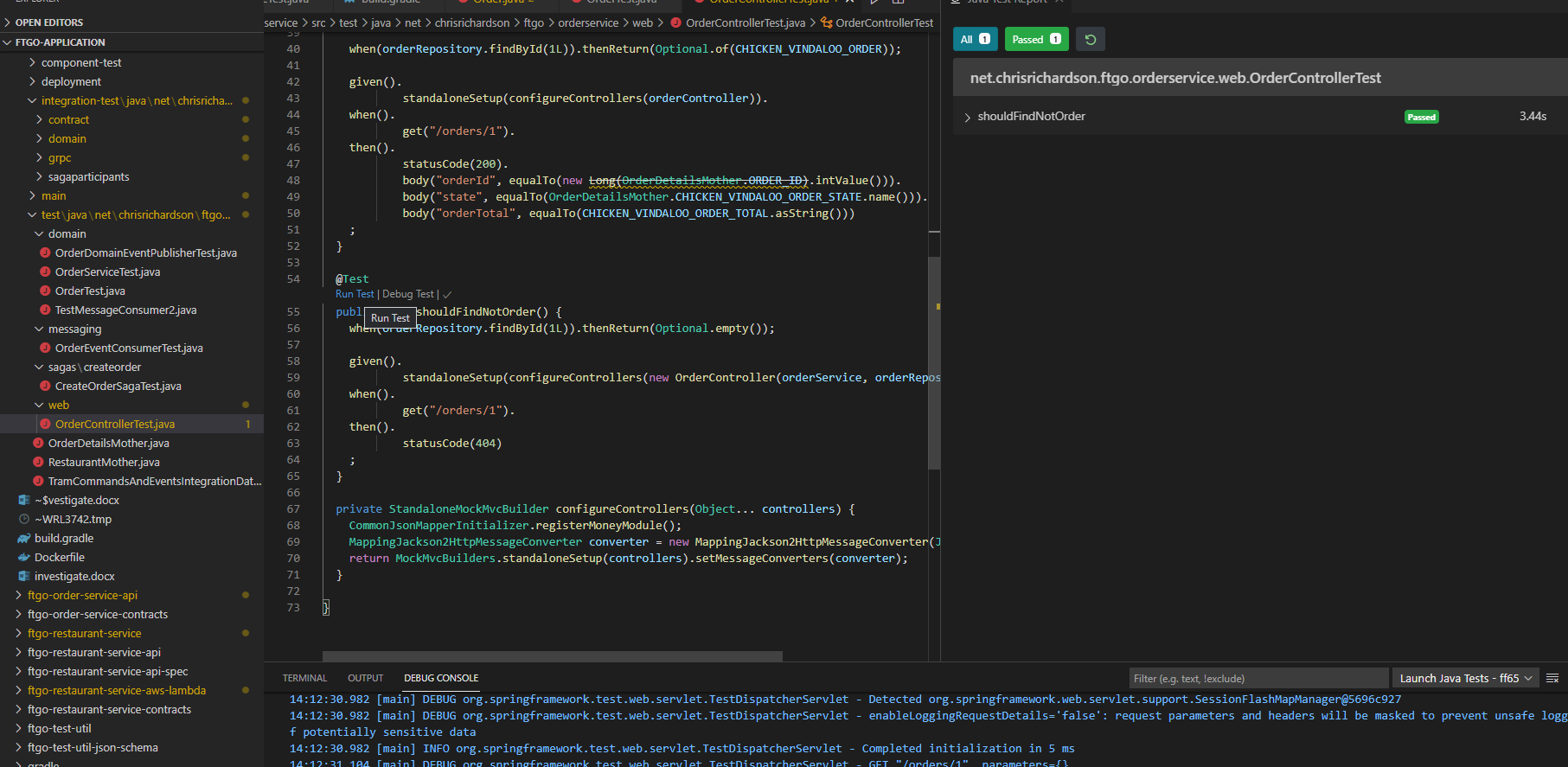
Errors occurred while build effective model from C:\Users\Admin\.gradle\caches\modules-2\files-2.1\com.sun.xml.bind\jaxb-impl\2.2.11\2d4b554997fd01d1a2233b1529b22fc9ecc0cf5c\jaxb-impl-2.2.11.pom:

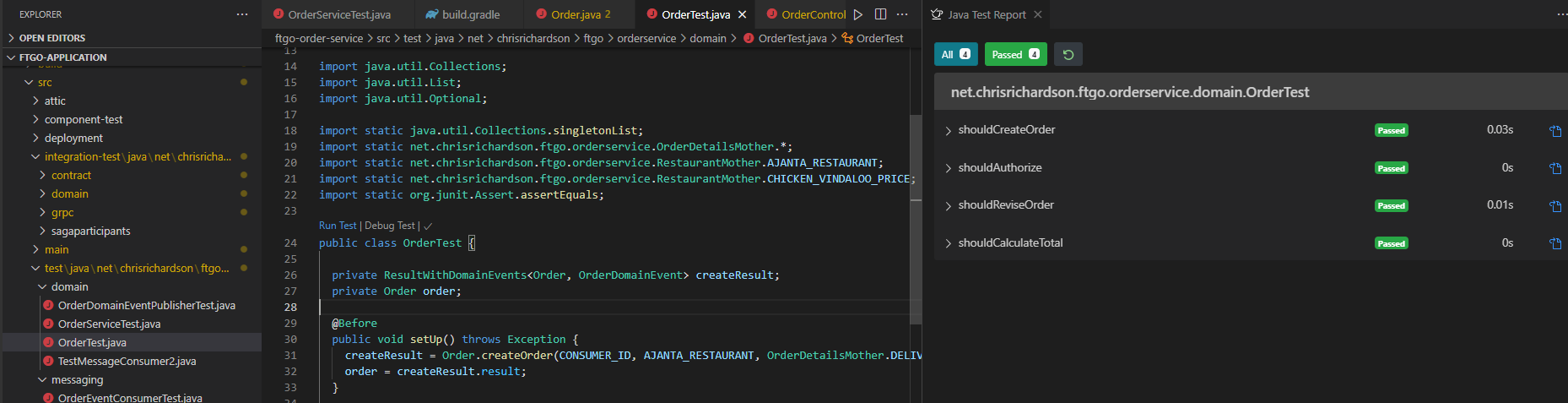
'dependencyManagement.dependencies.dependency.systemPath' for com.sun:tools:jar must specify an absolute path but is ${tools.jar} in com.sun.xml.bind:jaxb-impl:2.2.11

<https://stackoverflow.com/questions/58782854/java-11-with-gradle-must-specify-an-absolute-path-but-is-tools-jar>

Fixed: build OK (8min)

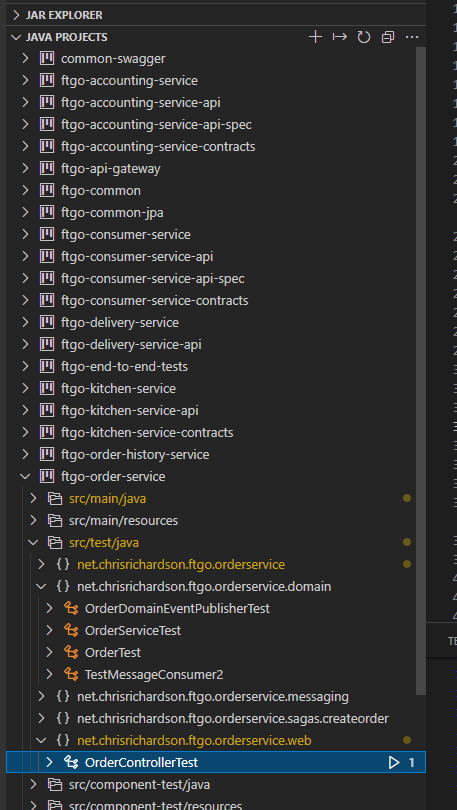
Run a unit test: just click on Run Test | Debug Test. But some files this command doesn’t show?





At the beginning, has an error:

java.lang.NoClassDefFoundError: org/junit/runner/manipulation/Filter

 Click on Triangle to Run Test

Mở riêng folder FTGO-ORDER-SERVICE thì ko ra Java Projects, ko show Run Test ở các test files.

<https://code.visualstudio.com/docs/java/java-testing>

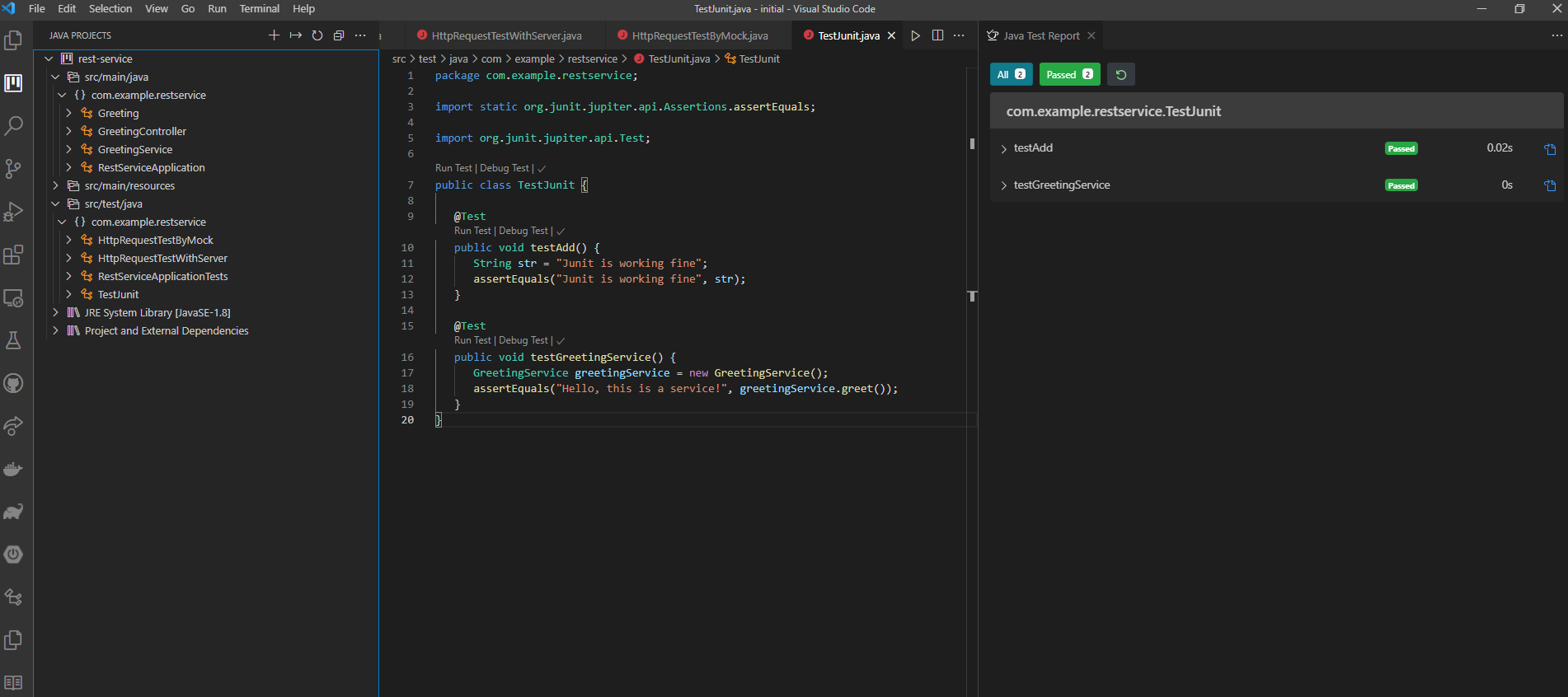
To open Java test Report, after run test, just click: Run Test | Debug Test | v (HERE)

This way run for Junit test, using Java Test Runner

Run for SpringBootTest also (Structure of project must be correct). Ex: <https://github.com/SonNXP/gs-rest-service> and

<https://github.com/SonNXP/gs-rest-service/tree/master/initial>

Note: If you cannot see the CodeLens in your editor, please refer to this [issue comment](https://github.com/microsoft/vscode-java-test/issues/470#issuecomment-444681714) as a workaround. <https://github.com/microsoft/vscode-java-test/issues/470#issuecomment-444681714>



Run Integration test: do the same UT. But error, need Database server run first. Create

ftgo\_order\_service

schema, error reason change BUT still don’t have correct tables.

Make DB for this app:

Mysql folder

compile-schema-per-service.sh

for schema in ftgo\_accounting\_service ftgo\_consumer\_service ftgo\_order\_service ftgo\_kitchen\_service ftgo\_restaurant\_service ftgo\_delivery\_service;

do

  user=${schema}\_user

  password=${schema}\_password

  cat >> /docker-entrypoint-initdb.d/5.schema-per-service.sql <<END

  CREATE USER '${user}'@'%' IDENTIFIED BY '$password';

  create database $schema;

  GRANT ALL PRIVILEGES ON $schema.\* TO '${user}'@'%' WITH GRANT OPTION;

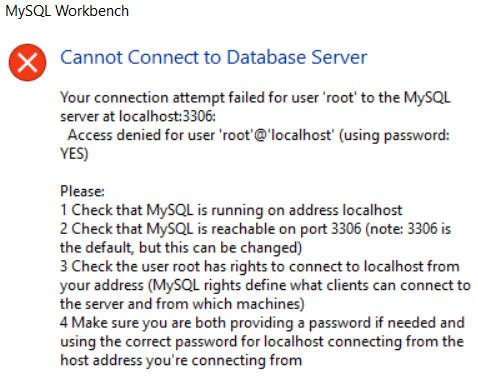
  USE $schema;

END

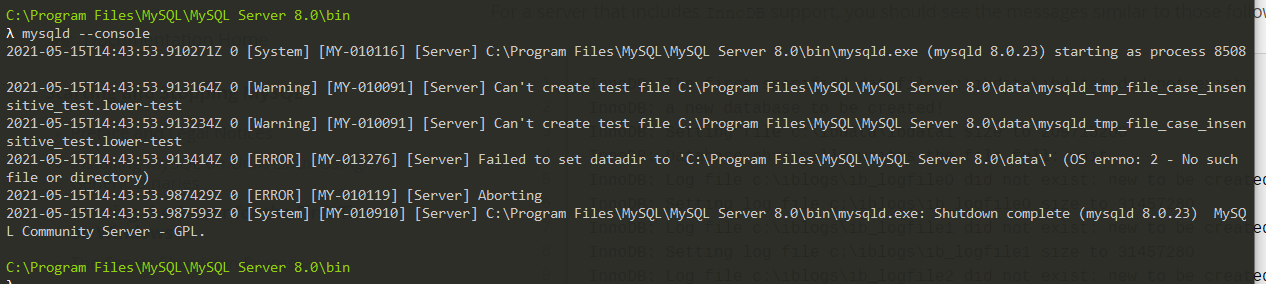
    cat /docker-entrypoint-initdb.d/template >> /docker-entrypoint-initdb.d/5.schema-per-service.sql

done

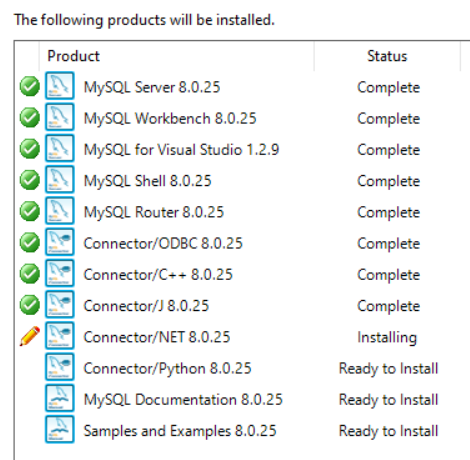
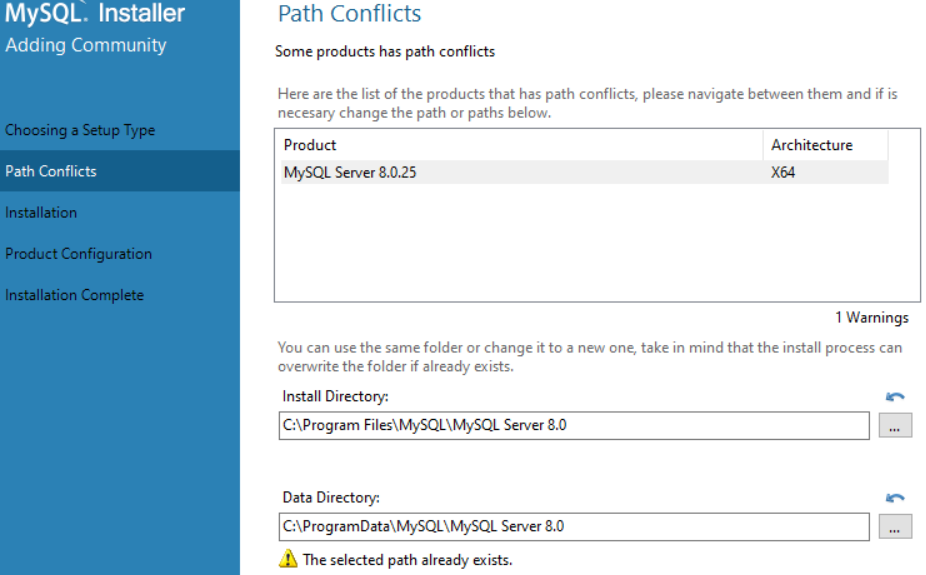
<https://stackoverflow.com/questions/11754781/how-to-declare-a-variable-in-mysql>

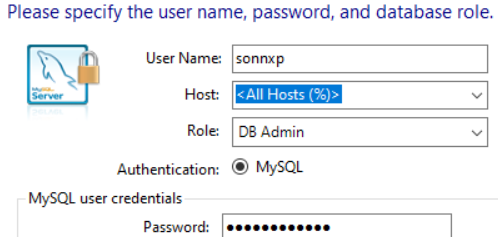
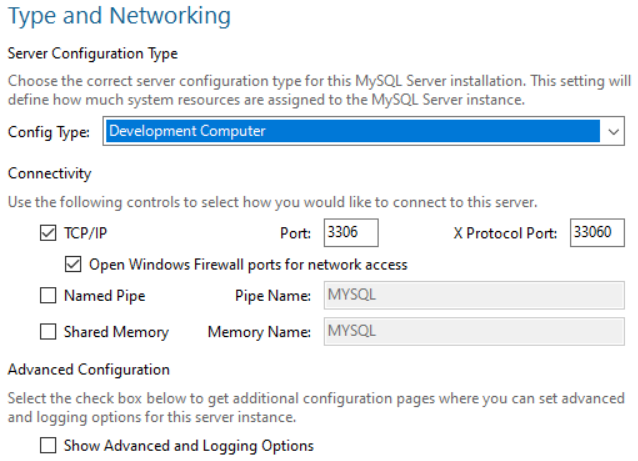


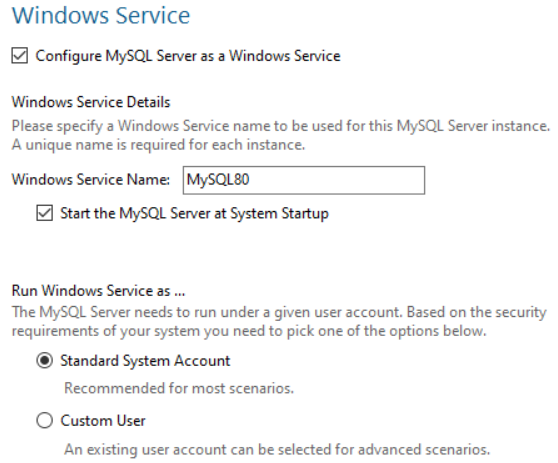
Chua start MySQL server? FIXED by reinstall all MySQL



Re-install MySQL: use MSI file to uninstall all and then reinstall.







<https://www.onlinetutorialspoint.com/mysql/install-mysql-on-windows-10-step-by-step.html>

Khong bi loi nua, fixed.

<https://www.baeldung.com/spring-data-jpa-generate-db-schema>

Copy to application.properties: 2 types: create or/and drop, see link

spring.jpa.properties.javax.persistence.schema-generation.scripts.action=create

spring.jpa.properties.javax.persistence.schema-generation.scripts.create-target=create.sql

spring.jpa.properties.javax.persistence.schema-generation.scripts.create-source=metadata

and then RUN the project: mvn spring-boot:run (this RUN when have no notes\_app database -> still generate). Error when RUN: o.h.engine.jdbc.spi.SqlExceptionHelper : Unknown database 'notes\_app'

will have automatically: (example project: E:\Documents\Books\Java\Projects\easy-notes)

create table notes (

    id bigint not null auto\_increment,

    content varchar(255),

    created\_at datetime not null,

    title varchar(255),

    updated\_at datetime not null,

    primary key (id)

) engine = InnoDB

Run whole ftgo-application. gradlew :composeUp when Docker is running

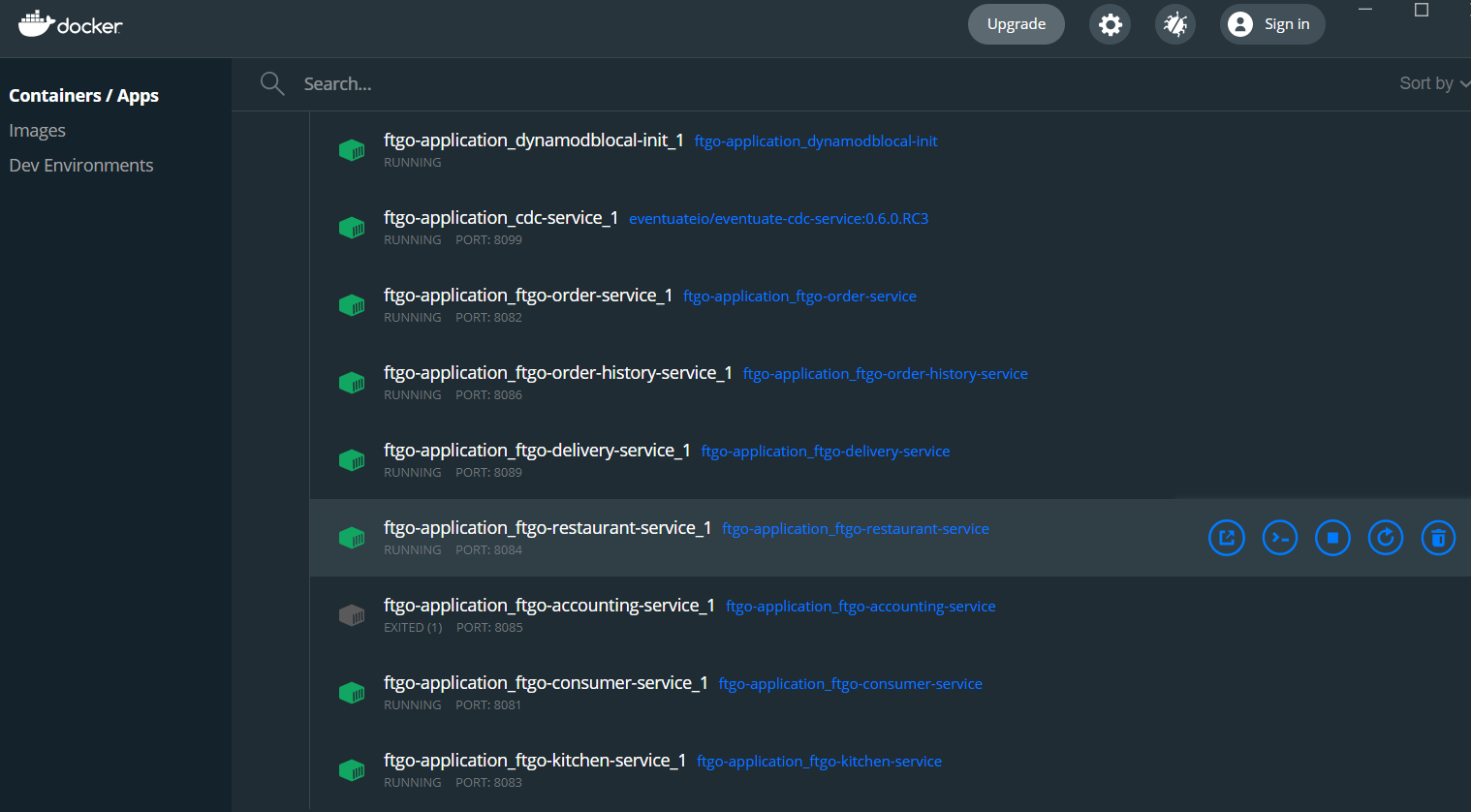
ERROR: for mysql Cannot start service mysql: Ports are not available: listen tcp 0.0.0.0:3306: bind: Only one usage of each socket address (protocol/network address/port) is normally permitted.

Encountered errors while bringing up the project.

Shutdown MySQL is running and run again: gradlew :composeUp

Fixed by: shutdown mysqld.exe that occupt port 3306.

<https://stackoverflow.com/questions/64307077/docker-compose-only-one-usage-of-each-socket-address-protocol-network-address/64310265>

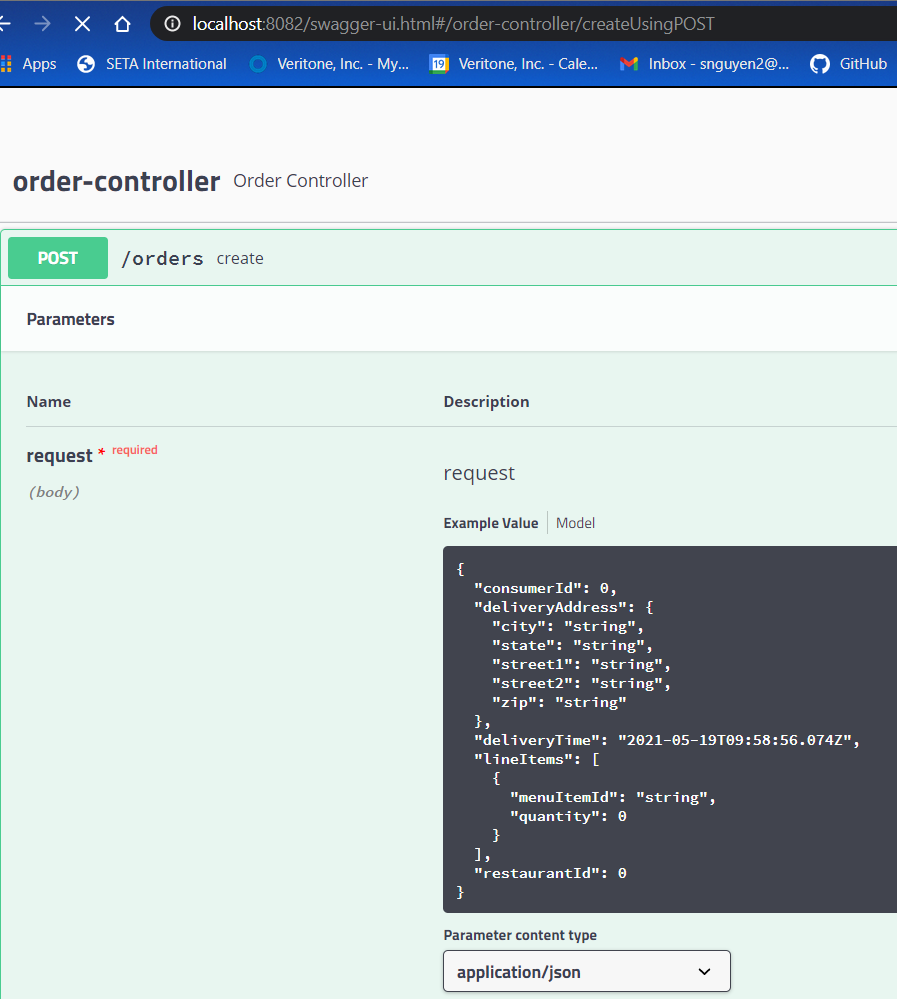
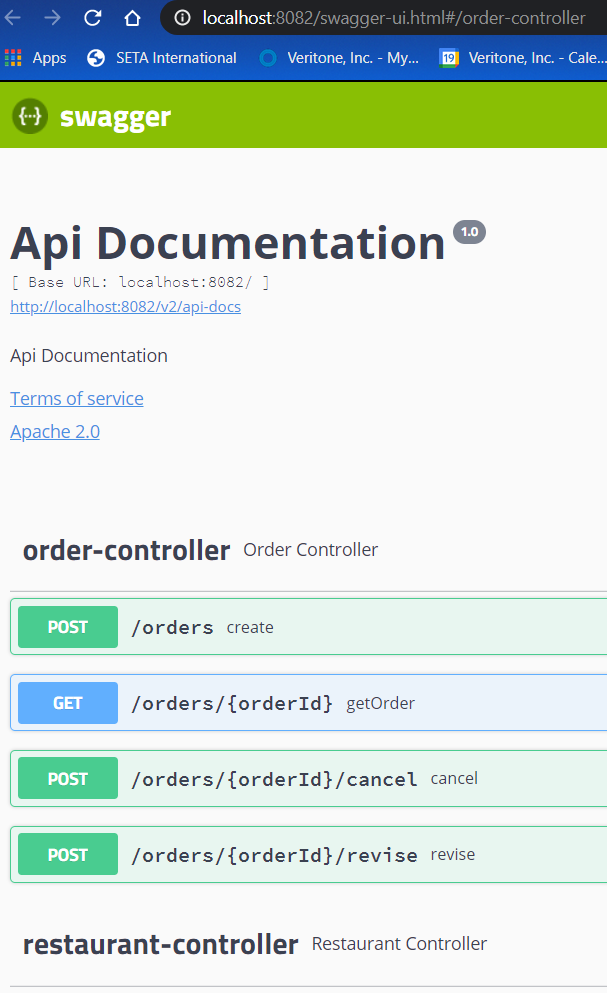


If has:

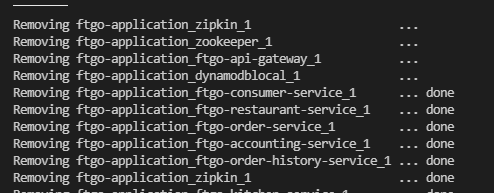
Waiting for ftgo-order-history-service\_1 to become healthy (it's unhealthy)

Waiting for ftgo-order-history-service\_1 to become healthy (it's unhealthy)

Click START on container in Docker



Run for several minutes,



All containers lost.