Test Plan - Ordering Service - Ofri Shapira

# 

[**Requirements Tree**](#_mm215fcbrlww) 2

[**Test Cases**](#_xeyfkkaqiv1l)4

[**Messages**](#_6bvh5z390uko) 70

[Inputs & Outputs](#_u400u232rpir)65

[Invalid Inputs & Outputs](#_n3t1fayj7e2r)73

[API Requests](#_doc0i2z7qr8f) 75

[API Responses](#_mpbszi4n0h28) 77

[Logs Messages](#_2dpii2hu3m6o)79

# 

# Requirements Tree

1. Order Management
   1. Main Success Scenario
      1. [Verify that the main success scenario for creating order is valid.](#_6zmsk5rr469w)
   2. Creating Order
      1. [Verify that the user can submit an order.](#_o5fbf8l0zvu1)
   3. Canceling Order
      1. [Verify that the user can cancel his order when the order status is ‘submitted’.](#_hhr3ovyrus3a)
      2. [Verify that the user can cancel his order when the order status is ‘awaitingvalidation’.](#_9zsf1fi03w4w)
      3. [Verify that the user can cancel his order when the order status is ‘stockconfirmd’.](#_jc3s6j8jb9sa)
      4. [Verify that the user can not cancel his order when the order status is ‘paid’.](#_x795035wwmul)
      5. [Verify that the user can not cancel his order when the order status is ‘shipped’.](#_kcl1ydapyt2b)
   4. Updating Order Status to [‘shipped’](#_y5blu6xko5ze)
      1. [Verify that the service allows updating order status to ‘shipped’, from ‘paid’ status.](#_qv840xcvi86g)
      2. [Verify that the service does not allow updating the order status to ‘shipped’, from ‘submitted’ status.](#_y5blu6xko5ze)
      3. [Verify that the service does not allow updating the order status to ‘shipped’, from ‘awatingvalidation’ status.](#_i4s1nbgz6dla)
      4. [Verify that the service does not allow updating the order status to ‘shipped’, from ‘stockconfirmd’ status.](#_tmafas23r8zx)
      5. [Verify that the service does not allow updating the order status to ‘shipped’, from ‘canceled’ status](#_k9g539upt5cx).
2. Order Tracking
   1. [Verify that the ‘submitted’ order status appears at the correct state.](#_sx2uoox10n7l)
   2. [Verify that the ‘awatingvalidation’ order status appears at the correct service state.](#_20yd4du650hy)
   3. [Verify that the ‘stockconfirmd’ order status appears at the correct service state.](#_7xfsqs9ypna9)
   4. [Verify that the ‘paid’ order status appears at the correct service state.](#_tltdc2ygjpg4)
   5. [Verify that the ‘shipped’ order status appears at the correct service state.](#_8anpinwe7iss)
   6. [Verify that the ‘canceled’ order status appears at the correct service state.](#_ddiqwijlx8ve)
3. Payment Processing
   1. [Verify that the order process is continued whenever the payment process has succeeded.](#_wnfoa0k8eifl)
   2. [Verify that the order is canceled whenever the payment process has failed.](#_9y1xgpmz8eoj)
   3. [Validate that the service will cancel the ordering process when the service is on ‘confirmstock’ status and the user does not initiate any action for 1 hour.](#_pgqyuz1edzph)
4. Inventory Management
   1. [Verify that the order process continues whenever a confirmation message has been sent from the catalog service.](#_7xfsqs9ypna9)
   2. [Verify that the order is canceled whenever a rejection message is](#_l27ewp1yugqe)

[received from the Catalog service.](#_l27ewp1yugqe)

1. Security
   1. [Verify that a signed-in user is only exposed to his own orders.](#_no7t9asau2jx)
   2. [Verify that a signed-in user is only exposed to his own card types.](#_sxeymiaas7lz)
   3. [Verify that a signed-in user receives only his own orders when he tries to fetch an order by id.](#_9fptwyutk73f)
   4. [Verify that a signed-in user can only cancel his own order.](#_hhr3ovyrus3a)  
      (Additional TCs: 4, 5).
   5. [Verify that a signed-in user can only ship his own order.](#_qv840xcvi86g)
   6. [Verify that an unauthorized request to get all orders is denied.](#_6z28o8cofw27)
   7. [Verify that an unauthorized request to get an order by id is denied.](#_g18xfhyhgyko)
   8. [Verify that an unauthorized request to get all card types is denied.](#_tumpfapnj8dx)
   9. [Verify that an unauthorized request to ship order is denied.](#_ciaqmnbovykk)
   10. [Verify that an unauthorized request to cancel order is denied.](#_uytnlpwf6qly)
   11. [Verify that the service is able to perform 300 orders while DDOS attack is being simulated on the service.](#_3cfaf6tv7qvh)
2. Scalability
   1. [Verify that the service can consume 150 messages that are waiting in the queue in a maximum time of one hour.](#_kxh5vldgugk7)
   2. [Verify that the service is reading the first message in the queue in a maximum time of 3 seconds.](#_3jma485thsmq)
3. Reliability
   1. [Verify that the service is able to recover the data whenever the service crashes in between the ‘submitted’ and ‘awaitingvalidation’ states.](#_31jwrh3avjw1)
   2. [Verify that the service is able to recover the data whenever the service crashes in between the ‘awaitingvalidation’ and ‘stockconfirmed’ states.](#_vnuo8sra0xsh)
   3. [Verify that the service is able to recover the data whenever the service crashes between the ‘stockconfirmed’ and ‘paid’ states.](#_rll6ryacmw44)
4. Reporting and Analytics
   1. [Verify that the Ordering Service is logging whenever the order status is ‘submitted’.](#_2r0uhxc)
   2. [Verify that the Ordering Service is logging whenever the order status is ‘awaitvalidtation’.](#_k670wktw9wgb)
   3. [Verify that the Ordering Service is logging whenever the order status is ‘stockconfirmed’.](#_mxrecte7v7cl)
   4. [Verify that the Ordering Service is logging whenever the order status is ‘paid’.](#_7z2fjdhy7b95)
   5. [Verify that the Ordering Service is logging whenever the order status is ‘shipped’.](#_e42lh1cvafpy)
   6. [Verify that the Ordering Service is logging whenever the order status is ‘canceled’.](#_k3i9hzvo5r8y)
   7. [Verify that the Ordering Service is logging whenever the app crashes.](#_5i6z4uiwjz7)

# Test Cases

## **Verify that the main success scenario for creating order is valid.**

Test Case Purpose: Verify that the creating order functionality of the microservice is working.

Testing Environment:

* Ordering Service software
* Ordering Service database
* Basket mock
* Catalog mock
* Payment mock
* RabbitMQ
* Postman

Preconditions:

* Ordering Service software is up and running.
* The Ordering Service database is connected and running.
* The login process has already been done with the details of the default user ‘Alice’.
* All the required mocks are active, filled with the relevant test data and ready for action.
* All the services that are simulated by the mocks (basket, payment and catalog) are down.
* RabbitMQ is connected and ready for action, and all of the service queues are empty.
* Postman is connected and ready for action.

Traceability: 1.1.1

Test Type:

* Sanity
* Functional
* API

|  | **Steps** | **Expected Result** | **Test Result** |
| --- | --- | --- | --- |
| 1 | Send from the basket mock to the Ordering queue the following message: [Input#1](#_76f4h77dmzn3) | The basket queue received the following message: [Output#1](#_glbdazqfclpu) |  |
| 2 | Verify that a new order entity has been created within the ordering table, with OrderStatusID of 1. | New order entity is created within the ordering table, with OrderStatusID of 1. |  |
| 3 | Verify that the catalog queue received from the ordering service the following message: [Output#2](#_fy07el923stq). | * The catalog queue received the following message: [Output#2](#_fy07el923stq). * The OrderStatusID in the ordering table is updated to 2. |  |
| 4 | Send from the catalog mock to the Ordering queue the following message: [Input#2](#_2knja88zi6ps). | The OrderStatusID in the ordering table is updated to 3. |  |
| 5 | Verify that the payment queue received from the ordering service the following message: [Output#3](#_i2goziy2oll). | The payment queue received the following message: [Output#3](#_i2goziy2oll). |  |
| 6 | Send from the payment mock to the Ordering queue the following message: [Input#3](#_orjqb39ik4oy). | The OrderStatusID is updated to 4. |  |
| 7 | Send the following API request from Postman:  [request#](#_miqkoomeg1rc)2. | * 200 HTTP status code should be returned. * The OrderStatusID in the ordering table updated to 5. |  |

## **Verify that the user can submit an order.**

Test Case Purpose: Verify that the submitting order functionality of the service is working.

Testing Environment:

* Ordering Service software
* Ordering Service database
* Basket mock
* RabbitMQ

Preconditions:

* Ordering Service software is up and running.
* The Ordering Service database is connected and running.
* The login process has already been done with the details of the default user ‘Alice’.
* All the required mocks are active, filled with the relevant test data and ready for action.
* All the services that are simulated by the mocks (basket, payment and catalog) are down.
* RabbitMQ is connected and ready for action, and all of the service queues are empty.

Traceability: 1.2.1

Test Type:

* Sanity
* Functional

|  | **Steps** | **Expected Result** | **Test Result** |
| --- | --- | --- | --- |
| 1 | Send from the basket mock to the Ordering queue the following message: [Input#1](#_76f4h77dmzn3) | The basket queue received the following message: [Output#1](#_glbdazqfclpu) |  |
| 2 | Verify that a new order entity has been created within the ordering table, with OrderStatusID of 1. | New order entity is created within the ordering table, with OrderStatusID of 1. |  |

## **Verify that the user can cancel his order when the order status is ‘submitted’.**

Test Case Purpose: Verify that the canceling order functionality of the microservice is working on order status number 1.

Testing Environment:

* Ordering Service software
* Ordering Service database
* Basket mock
* RabbitMQ
* Postman

Preconditions:

* The Ordering Service is up and running.
* The Ordering Service database is connected and running.
* The login process has already been done with the details of the default user ‘Alice’.
* All the required mocks are active, filled with the relevant test data and ready for action.
* All the services that are simulated by the mocks (basket, payment and catalog) are down.
* RabbitMQ is connected and ready for action, and all of the service queues are empty.
* Postman is connected and ready for action.

Traceability: 1.3.1, 5.4

Test Type:

* Functional
* API
* Security

|  | **Steps** | **Expected Result** | **Test Result** |
| --- | --- | --- | --- |
| 1 | Send from the basket mock to the Ordering queue the following message: [Input#1](#_76f4h77dmzn3) | * The basket queue received the following message: [Output#1](#_glbdazqfclpu) * New Order entity is created within the OrderingDb inside the ordering table, with OrderStatusID = 1. |  |
| 2 | Send the following API request from Postman:  [request#1](#_bgjg04abhzcf). | * 200 HTTP status code should be returned. * The OrderStatusID in the ordering table is updated to 6. |  |

## **Verify that the user can cancel his order when the order status is ‘awaitingvalidation’.**

Test Case Purpose: Verify that the canceling order functionality of the microservice is working on order status number 2.

Testing Environment:

* Ordering Service software
* Ordering Service database
* Basket mock
* Catalog mock
* RabbitMQ
* Postman

Preconditions:

* The Ordering Service is up and running.
* The Ordering Service database is connected and running.
* The login process has already been done with the details of the default user ‘Alice’.
* All the required mocks are active, filled with the relevant test data and ready for action.
* All the services that are simulated by the mocks (basket, payment and catalog) are down.
* RabbitMQ is connected and ready for action, and all of the service queues are empty.
* Postman is connected and ready for action.

Traceability: 1.3.2, 5.4

Test Type:

* Functional
* API
* Security

|  | **Steps** | **Expected Result** | **Test Result** |
| --- | --- | --- | --- |
| 1 | Send from the basket mock to the Ordering queue the following message: [Input#1](#_76f4h77dmzn3) | * The basket queue received the following message: [Output#1](#_glbdazqfclpu) * New Order entity is created within the OrderingDb inside the ordering table, with OrderStatusID = 1. |  |
| 2 | Verify that the catalog queue received from the ordering service the following message: [Output#2](#_fy07el923stq). | * The catalog queue received the following message: [Output#2](#_fy07el923stq). * The OrderStatusID in the ordering table is updated to 2. |  |
| 3 | Send the following API request from Postman:  [request#1](#_bgjg04abhzcf). | * 200 HTTP status code should be returned. * The OrderStatusID in the ordering table is updated to 6. |  |

# 

# Verify that the user can cancel his order when the order status is ‘stockconfirmd’.

Test Case Purpose: Verify that the canceling order functionality of the microservice is working on order status number 3.

Testing Environment:

* Ordering Service software
* Ordering Service database
* Basket mock
* Catalog mock
* RabbitMQ
* Postman

Preconditions:

* The Ordering Service is up and running.
* The Ordering Service database is connected and running.
* The login process has already been done with the details of the default user ‘Alice’.
* All the required mocks are active, filled with the relevant test data and ready for action.
* All the services that are simulated by the mocks (basket, payment and catalog) are down.
* RabbitMQ is connected and ready for action, and all of the service queues are empty.
* Postman is connected and ready for action.

Traceability: 1.3.3, 5.4

Test Type:

* Functional
* API
* Security

|  | **Steps** | **Expected Result** | **Test Result** |
| --- | --- | --- | --- |
| 1 | Send from the basket mock to the Ordering queue the following message: [Input#1](https://docs.google.com/document/d/1P06K0LAtOx-pWbx5jL-hiIX0isExkIIy0pEHRR4sGMY/edit#heading=h.76f4h77dmzn3) | * The basket queue received the following message: [Output#1](https://docs.google.com/document/d/1P06K0LAtOx-pWbx5jL-hiIX0isExkIIy0pEHRR4sGMY/edit#heading=h.glbdazqfclpu) * New Order entity is created within the OrderingDb inside the ordering table, with OrderStatusID = 1. |  |
| 2 | Verify that the catalog queue received from the ordering service the following message: [Output#2](https://docs.google.com/document/d/1P06K0LAtOx-pWbx5jL-hiIX0isExkIIy0pEHRR4sGMY/edit#heading=h.fy07el923stq). | * The catalog queue received the following message: [Output#2](https://docs.google.com/document/d/1P06K0LAtOx-pWbx5jL-hiIX0isExkIIy0pEHRR4sGMY/edit#heading=h.fy07el923stq). * The OrderStatusID is updated to 2. |  |
| 3 | Send from the catalog mock to the Ordering queue the following message: [Input#2](https://docs.google.com/document/d/1P06K0LAtOx-pWbx5jL-hiIX0isExkIIy0pEHRR4sGMY/edit#heading=h.2knja88zi6ps). | The OrderStatusID is updated to 3. |  |
| 4 | Send the following API request from Postman:  [request#1](https://docs.google.com/document/d/1P06K0LAtOx-pWbx5jL-hiIX0isExkIIy0pEHRR4sGMY/edit#heading=h.bgjg04abhzcf). | * 200 HTTP status code should be returned. * The OrderStatusID is updated to 6. |  |

# 

## **Verify that the user can not cancel his order when the order status is ‘paid’.**

Test Case Purpose: Verify that the canceling order functionality of the microservice is working on order status number 4.

Testing Environment:

* Ordering Service software
* Ordering Service database
* Basket mock
* Catalog mock
* Payment mock
* RabbitMQ
* Postman

Preconditions:

* The Ordering Service is up and running.
* The Ordering Service database is connected and running.
* The login process has already been done with the details of the default user ‘Alice’.
* All the required mocks are active, filled with the relevant test data and ready for action.
* All the services that are simulated by the mocks (basket, payment and catalog) are down.
* RabbitMQ is connected and ready for action, and all of the service queues are empty.
* Postman is connected and ready for action.

Traceability: 1.3.4

Test Type:

* Functional
* API

|  | **Steps** | **Expected Result** | **Test Result** |
| --- | --- | --- | --- |
| 1 | Send from the basket mock to the Ordering queue the following message: [Input#1](#_76f4h77dmzn3) | * The basket queue received the following message: [Output#1](#_glbdazqfclpu) * New Order entity is created within the OrderingDb inside the ordering table, with OrderStatusID = 1. |  |
| 2 | Verify that the catalog queue received from the ordering service the following message: [Output#2](#_fy07el923stq). | * The catalog queue received the following message: [Output#2](#_fy07el923stq). * The OrderStatusID in the ordering table is updated to 2. |  |
| 3 | Send from the catalog mock to the Ordering queue the following message: [Input#2](#_2knja88zi6ps). | The OrderStatusID in the ordering table is updated to 3. |  |
| 4 | Verify that the payment queue received from the ordering service the following message: [Output#3](#_i2goziy2oll). | The payment queue received the following message: [Output#3](#_i2goziy2oll). |  |
| 5 | Send from the payment mock to the Ordering queue the following message: [Input#3](#_orjqb39ik4oy). | The OrderStatusID in the ordering table is updated to 4. |  |
| 6 | Send the following API request from Postman:  [request#1](#_bgjg04abhzcf). | * 400 HTTP status code should be returned. * The OrderStatusID in the ordering table remains 4. |  |

# 

## **Verify that the user can not cancel his order when the order status is ‘shipped’.**

Test Case Purpose: Verify that the canceling order functionality of the microservice is working on order status number 5.

Testing Environment:

* Ordering Service software
* Ordering Service database
* Basket mock
* Catalog mock
* Payment mock
* RabbitMQ
* Postman

Preconditions:

* The Ordering Service is up and running.
* The Ordering Service database is connected and running.
* The login process has already been done with the details of the default user ‘Alice’.
* All the required mocks are active, filled with the relevant test data and ready for action.
* All the services that are simulated by the mocks (basket, payment and catalog) are down.
* RabbitMQ is connected and ready for action, and all of the service queues are empty.
* Postman is connected and ready for action.

Traceability: 1.3.5

Test Type:

* Functional
* API

|  | **Steps** | **Expected Result** | **Test Result** |
| --- | --- | --- | --- |
| 1 | Send from the basket mock to the Ordering queue the following message: [Input#1](#_76f4h77dmzn3) | * The basket queue received the following message: [Output#1](#_glbdazqfclpu) * New Order entity is created within the OrderingDb inside the ordering table, with OrderStatusID = 1. |  |
| 2 | Verify that the catalog queue received from the ordering service the following message: [Output#2](#_fy07el923stq). | * The catalog queue received the following message: [Output#2](#_fy07el923stq). * The OrderStatusID in the ordering table is updated to 2. |  |
| 3 | Send from the catalog mock to the Ordering queue the following message: [Input#2](#_2knja88zi6ps). | The OrderStatusID in the ordering table is updated to 3. |  |
| 4 | Verify that the payment queue received from the ordering service the following message: [Output#3](#_i2goziy2oll). | The payment queue received the following message: [Output#3](#_i2goziy2oll). |  |
| 5 | Send from the payment mock to the Ordering queue the following message: [Input#3](#_orjqb39ik4oy). | The OrderStatusID in the ordering table is updated to 4. |  |
| 6 | Send the following API request from Postman:  [request#2](#_miqkoomeg1rc). | * 200 HTTP status code should be returned. * The OrderStatusID in the ordering table is updating to 5 . |  |
| 7 | Send the following API request from Postman:  [request#1](#_bgjg04abhzcf). | * 400 HTTP status code should be returned. * The OrderStatusID in the ordering table remains 4. |  |

# 

# **Verify that the service allows updating order status to ‘shipped’ from ‘paid’ status.**

Test Case Purpose: Verify that the updating order functionality of the microservice is working (updating the status) on order status number 4.

Testing Environment:

* Ordering Service software
* Ordering Service database
* Basket mock
* Catalog mock
* Payment mock
* RabbitMQ
* Postman

Preconditions:

* The Ordering Service is up and running.
* The Ordering Service database is connected and running.
* The login process has already been done with the details of the default user ‘Alice’.
* All the required mocks are active, filled with the relevant test data and ready for action.
* All the services that are simulated by the mocks (basket, payment and catalog) are down.
* RabbitMQ is connected and ready for action, and all of the service queues are empty.
* Postman is connected and ready for action.

Traceability: 1.4.1, 5.5

Test Type:

* Functional
* Security

|  | **Steps** | **Expected Result** | **Test Result** |
| --- | --- | --- | --- |
| 1 | Send from the basket mock to the Ordering queue the following message: [Input#1](#_76f4h77dmzn3) | * The basket queue received the following message: [Output#1](#_glbdazqfclpu) * New Order entity is created within the OrderingDb inside the ordering table, with OrderStatusID = 1. |  |
| 2 | Verify that the catalog queue received from the ordering service the following message: [Output#2](#_fy07el923stq). | * The catalog queue received the following message: [Output#2](#_fy07el923stq). * The OrderStatusID in the ordering table is updated to 2. |  |
| 3 | Send from the catalog mock to the Ordering queue the following message: [Input#2](#_2knja88zi6ps). | The OrderStatusID in the ordering table is updated to 3. |  |
| 4 | Verify that the payment queue received from the ordering service the following message: [Output#3](#_i2goziy2oll). | The payment queue received the following message: [Output#3](#_i2goziy2oll). |  |
| 5 | Send from the payment mock to the Ordering queue the following message: [Input#3](#_orjqb39ik4oy). | The OrderStatusID in the ordering table is updated to 4. |  |
| 6 | Send the following API request from Postman:  [request#](#_miqkoomeg1rc)2. | * 200 HTTP status code should be returned. * The OrderStatusID in the ordering table updated to 5. |  |

# 

# **Verify that the service does not allow updating the order status to ‘shipped’ from ‘submitted’ status.**

Test Case Purpose: Verify that the updating order functionality of the microservice is working (not updating the status) on order status number 1.

Testing Environment:

* Ordering Service software
* Ordering Service database
* Basket mock
* RabbitMQ
* Postman

Preconditions:

* The Ordering Service is up and running.
* The Ordering Service database is connected and running.
* The login process has already been done with the details of the default user ‘Alice’.
* All the required mocks are active, filled with the relevant test data and ready for action.
* All the services that are simulated by the mocks (basket, payment and catalog) are down.
* RabbitMQ is connected and ready for action, and all of the service queues are empty.
* Postman is connected and ready for action.

Traceability: 1.4.2

Test Type:

* Functional
* API

|  | **Steps** | **Expected Result** | **Test Result** |
| --- | --- | --- | --- |
| 1 | Send from the basket mock to the Ordering queue the following message: [Input#1](#_76f4h77dmzn3) | * The basket queue received the following message: [Output#1](#_glbdazqfclpu) * New Order entity is created within the OrderingDb inside the ordering table, with OrderStatusID = 1. |  |
| 2 | Send the following API request from Postman:  [request#2.](#_miqkoomeg1rc) | * 400 HTTP status code should be returned. * The OrderStatusID in the ordering table remains 1. |  |

# 

# **Verify that the service does not allow updating the order status to ‘shipped’ from ‘awatingvalidation’ status.**

Test Case Purpose: Verify that the updating order functionality of the microservice is working (not updating the status) on order status number 2.

Testing Environment:

* Ordering Service software
* Ordering Service database
* Basket mock
* Catalog mock
* RabbitMQ
* Postman

Preconditions:

* The Ordering Service is up and running.
* The Ordering Service database is connected and running.
* The login process has already been done with the details of the default user ‘Alice’.
* All the required mocks are active, filled with the relevant test data and ready for action.
* All the services that are simulated by the mocks (basket, payment and catalog) are down.
* RabbitMQ is connected and ready for action, and all of the service queues are empty.
* Postman is connected and ready for action.

Traceability: 1.4.3

Test Type:

* Functional
* API

|  | **Steps** | **Expected Result** | **Test Result** |
| --- | --- | --- | --- |
| 1 | Send from the basket mock to the Ordering queue the following message: [Input#1](#_76f4h77dmzn3) | * The basket queue received the following message: [Output#1](#_glbdazqfclpu) * New Order entity is created within the OrderingDb inside the ordering table, with OrderStatusID = 1. |  |
| 2 | Verify that the catalog queue received from the ordering service the following message: [Output#2](#_fy07el923stq). | * The catalog queue received the following message: [Output#2](#_fy07el923stq). * The OrderStatusID in the ordering table is updated to 2. |  |
| 3 | Send the following API request from Postman:  [request#](#_miqkoomeg1rc)2. | * 400 HTTP status code should be returned. * The OrderStatusID in the ordering table remains 2. |  |

# **Verify that the service does not allow updating the order status to ‘shipped’ from ‘stockconfirmd’ status.**

Test Case Purpose: Verify that the updating order functionality of the microservice is working (not updating the status) on order status number 3.

Testing Environment:

* Ordering Service software
* Ordering Service database
* Basket mock
* Catalog mock
* RabbitMQ
* Postman

Preconditions:

* The Ordering Service is up and running.
* The Ordering Service database is connected and running.
* The login process has already been done with the details of the default user ‘Alice’.
* All the required mocks are active, filled with the relevant test data and ready for action.
* All the services that are simulated by the mocks (basket, payment and catalog) are down.
* RabbitMQ is connected and ready for action, and all of the service queues are empty.
* Postman is connected and ready for action.

Traceability: 1.4.4

Test Type:

* Functional
* API

|  | **Steps** | **Expected Result** | **Test Result** |
| --- | --- | --- | --- |
| 1 | Send from the basket mock to the Ordering queue the following message: [Input#1](#_76f4h77dmzn3) | * The basket queue received the following message: [Output#1](#_glbdazqfclpu) * New Order entity is created within the OrderingDb inside the ordering table, with OrderStatusID = 1. |  |
| 2 | Verify that the catalog queue received from the ordering service the following message: [Output#2](#_fy07el923stq). | * The catalog queue received the following message: [Output#2](#_fy07el923stq). * The OrderStatusID in the ordering table in the ordering table is updated to 2. |  |
| 3 | Send from the catalog mock to the Ordering queue the following message: [Input#2](#_2knja88zi6ps). | The OrderStatusID in the ordering table is updated to 3. |  |
| 4 | Send the following API request from Postman:  [request#](#_miqkoomeg1rc)2. | * 400 HTTP status code should be returned. * The OrderStatusID in the ordering table remains 3. |  |

# 

# **Verify that the service does not allow updating the order status to ‘shipped’ from ‘canceled’ status.**

Test Case Purpose: Verify that the updating order functionality of the microservice is working (not updating the status) on order status number 6.

Testing Environment:

* Ordering Service software
* Ordering Service database
* Basket mock
* RabbitMQ
* Postman

Preconditions:

* The Ordering Service is up and running.
* The Ordering Service database is connected and running.
* The login process has already been done with the details of the default user ‘Alice’.
* All the required mocks are active, filled with the relevant test data and ready for action.
* All the services that are simulated by the mocks (basket, payment and catalog) are down.
* RabbitMQ is connected and ready for action, and all of the service queues are empty.
* Postman is connected and ready for action.

Traceability: 1.4.5

Test Type:

* Functional
* API

|  | **Steps** | **Expected Result** | **Test Result** |
| --- | --- | --- | --- |
| 1 | Send from the basket mock to the Ordering queue the following message: [Input#1](#_76f4h77dmzn3) | * The basket queue received the following message: [Output#1](#_glbdazqfclpu) * New Order entity is created within the OrderingDb inside the ordering table, with OrderStatusID = 1. |  |
| 2 | Send the following API request from Postman:  [request#1](#_bgjg04abhzcf). | * 200 HTTP status code should be returned. * The OrderStatusID in the ordering table is updated to 6. |  |
| 3 | Send the following API request from Postman:  [request#](#_miqkoomeg1rc)2. | * 400 HTTP status code should be returned. * The OrderStatusID in the ordering table remains 6. |  |

# **Verify that the ‘submitted’ order status appears at the correct service state.**

Test Case Purpose: Verify each order status is updated in the orders DB, according to the current order state.

Testing Environment:

* Ordering Service software
* Ordering Service database
* Basket mock
* RabbitMQ

Preconditions:

* The Ordering Service is up and running.
* The Ordering Service database is connected and running.
* The login process has already been done with the details of the default user ‘Alice’.
* All the required mocks are active, filled with the relevant test data and ready for action.
* All the services that are simulated by the mocks (basket, payment and catalog) are down.
* RabbitMQ is connected and ready for action, and all of the service queues are empty.

Traceability: 2.1

Test Type:

* Functional

|  | **Steps** | **Expected Result** | **Test Result** |
| --- | --- | --- | --- |
| 1 | Send from the basket mock to the Ordering queue the following message: [Input#1](#_76f4h77dmzn3) | The basket queue received the following message: [Output#1](#_glbdazqfclpu) |  |
| 2 | Verify in the ordering table within the OrderingDb that the OrderStatusID has been created and set to 1. | The OrderStatusID in the ordering table has been created and set to 1. |  |

# **Verify that the ‘awatingvalidation’ order status appears at the correct service state.**

Test Case Purpose: Verify each order status is updated in the orders DB, according to the current order state.

Testing Environment:

* Ordering Service software
* Ordering Service database
* Basket mock
* Catalog mock
* RabbitMQ

Preconditions:

* The Ordering Service is up and running.
* The Ordering Service database is connected and running.
* The login process has already been done with the details of the default user ‘Alice’.
* All the required mocks are active, filled with the relevant test data and ready for action.
* All the services that are simulated by the mocks (basket, payment and catalog) are down.
* RabbitMQ is connected and ready for action, and all of the service queues are empty.

Traceability: 2.2

Test Type:

* Functional

|  | **Steps** | **Expected Result** | **Test Result** |
| --- | --- | --- | --- |
| 1 | Send from the basket mock to the Ordering queue the following message: [Input#1](#_76f4h77dmzn3) | * The basket queue received the following message: [Output#1](#_glbdazqfclpu) * New Order entity is created within the OrderingDb inside the ordering table, with OrderStatusID = 1. |  |
| 2 | Verify that the catalog queue received from the ordering service the following message: [Output#2](#_fy07el923stq). | The catalog queue received the following message: [Output#2](#_fy07el923stq). |  |
| 3 | Verify in the ordering table within the OrderingDb that the OrderStatusID has been updated to 2. | The OrderStatusID in the ordering table has been updated to 2. |  |

# Verify that the ‘stockconfirmd’ order status appears at the correct service state.

Test Case Purpose: Verify each order status is updated in the orders DB, according to the current order state.

Testing Environment:

* Ordering Service software
* Ordering Service database
* Basket mock
* Catalog mock
* RabbitMQ
* Postman

Preconditions:

* The Ordering Service is up and running.
* The Ordering Service database is connected and running.
* The login process has already been done with the details of the default user ‘Alice’.
* All the required mocks are active, filled with the relevant test data and ready for action.
* All the services that are simulated by the mocks (basket, payment and catalog) are down.
* RabbitMQ is connected and ready for action, and all of the service queues are empty.
* Postman is connected and ready for action.

Traceability: 2.3, 4.1

Test Type:

* Functional

|  | **Steps** | **Expected Result** | **Test Result** |
| --- | --- | --- | --- |
| 1 | Send from the basket mock to the Ordering queue the following message: [Input#1](#_76f4h77dmzn3) | * The basket queue received the following message: [Output#1](#_glbdazqfclpu) * New Order entity is created within the OrderingDb inside the ordering table, with OrderStatusID = 1. |  |
| 2 | Verify that the catalog queue received from the ordering service the following message: [Output#2](#_fy07el923stq). | * The catalog queue received the following message: [Output#2](#_fy07el923stq). * The OrderStatusID in the ordering table is updated to 2. |  |
| 3 | Send from the catalog mock to the Ordering queue the following message: [Input#2](#_2knja88zi6ps). | The OrderStatusID in the ordering table is updated to 3. |  |

# Verify that the ‘paid’ order status appears at the correct service state.

Test Case Purpose: Verify each order status is updated in the orders DB, according to the current order state.

Testing Environment:

* Ordering Service software
* Ordering Service database
* Basket mock
* Catalog mock
* Payment mock
* RabbitMQ
* Postman

Preconditions:

* The Ordering Service is up and running.
* The Ordering Service database is connected and running.
* The login process has already been done with the details of the default user ‘Alice’.
* All the required mocks are active, filled with the relevant test data and ready for action.
* All the services that are simulated by the mocks (basket, payment and catalog) are down.
* RabbitMQ is connected and ready for action, and all of the service queues are empty.
* Postman is connected and ready for action.

Traceability: 2.4

Test Type:

* Functional

|  | **Steps** | **Expected Result** | **Test Result** |
| --- | --- | --- | --- |
| 1 | Send from the basket mock to the Ordering queue the following message: [Input#1](#_76f4h77dmzn3) | * The basket queue received the following message: [Output#1](#_glbdazqfclpu) * New Order entity is created within the OrderingDb inside the ordering table, with OrderStatusID = 1. |  |
| 2 | Verify that the catalog queue received from the ordering service the following message: [Output#2](#_fy07el923stq). | * The catalog queue received the following message: [Output#2](#_fy07el923stq). * The OrderStatusID in the ordering table is updated to 2. |  |
| 3 | Send from the catalog mock to the Ordering queue the following message: [Input#2](#_2knja88zi6ps). | The OrderStatusID in the ordering table is updated to 3. |  |
| 4 | Verify that the payment queue received from the ordering service the following message: [Output#3](#_i2goziy2oll). | The payment queue received the following message: [Output#3](#_i2goziy2oll). |  |
| 5 | Send from the payment mock to the Ordering queue the following message: [Input#3](#_orjqb39ik4oy). | The OrderStatusID in the ordering table is updated to 4. |  |

# 

# **Verify that the ‘shipped’ order status appears at the correct service state.**

Test Case Purpose: Verify each order status is updated in the orders DB, according to the current order state.

Testing Environment:

* Ordering Service software
* Ordering Service database
* Basket mock
* Catalog mock
* Payment mock
* RabbitMQ
* Postman

Preconditions:

* The Ordering Service is up and running.
* The Ordering Service database is connected and running.
* The login process has already been done with the details of the default user ‘Alice’.
* All the required mocks are active, filled with the relevant test data and ready for action.
* All the services that are simulated by the mocks (basket, payment and catalog) are down.
* RabbitMQ is connected and ready for action, and all of the service queues are empty.
* Postman is connected and ready for action.

Traceability: 2.5

Test Type:

* Functional
* API

|  | **Steps** | **Expected Result** | **Test Result** |
| --- | --- | --- | --- |
| 1 | Send from the basket mock to the Ordering queue the following message: [Input#1](#_76f4h77dmzn3) | * The basket queue received the following message: [Output#1](#_glbdazqfclpu) * New Order entity is created within the OrderingDb inside the ordering table, with OrderStatusID = 1. |  |
| 2 | Verify that the catalog queue received from the ordering service the following message: [Output#2](#_fy07el923stq). | * The catalog queue received the following message: [Output#2](#_fy07el923stq). * The OrderStatusID in the ordering table is updated to 2. |  |
| 3 | Send from the catalog mock to the Ordering queue the following message: [Input#2](#_2knja88zi6ps). | The OrderStatusID in the ordering table is updated to 3. |  |
| 4 | Verify that the payment queue received from the ordering service the following message: [Output#3](#_i2goziy2oll). | The payment queue received the following message: [Output#3](#_i2goziy2oll). |  |
| 5 | Send from the payment mock to the Ordering queue the following message: [Input#3](#_orjqb39ik4oy). | The OrderStatusID in the ordering table is updated to 4. |  |
| 6 | Send the following API request from Postman:  [request#](#_miqkoomeg1rc)2. | * 200 HTTP status code should be returned. |  |
| 7 | Verify in the ordering table within the OrderingDb that the OrderStatusID has been updated to 5. | The OrderStatusID in the ordering table updated to 5. |  |

# 

# **Verify that the ‘canceled’ order status appears at the correct service state.**

Test Case Purpose: Verify each order status is updated in the orders DB, according to the current order state.

Testing Environment:

* Ordering Service software
* Ordering Service database
* Basket mock
* Catalog mock
* RabbitMQ
* Postman

Preconditions:

* The Ordering Service is up and running.
* The Ordering Service database is connected and running.
* The login process has already been done with the details of the default user ‘Alice’.
* All the required mocks are active, filled with the relevant test data and ready for action.
* All the services that are simulated by the mocks (basket, payment and catalog) are down.
* RabbitMQ is connected and ready for action, and all of the service queues are empty.
* Postman is connected and ready for action.

Traceability: 2.6

Test Type:

* Functional
* API

|  | **Steps** | **Expected Result** | **Test Result** |
| --- | --- | --- | --- |
| 1 | Send from the basket mock to the Ordering queue the following message: [Input#1](#_76f4h77dmzn3) | * The basket queue received the following message: [Output#1](#_glbdazqfclpu) * New Order entity is created within the OrderingDb inside the ordering table, with OrderStatusID = 1. |  |
| 2 | Verify that the catalog queue received from the ordering service the following message: [Output#2](#_fy07el923stq). | * The catalog queue received the following message: [Output#2](#_fy07el923stq). * The OrderStatusID in the ordering table is updated to 2. |  |
| 3 | Send from the catalog mock to the Ordering queue the following message: [Input#2](#_2knja88zi6ps). | The OrderStatusID in the ordering table is updated to 3. |  |
| 4 | Send the following API request from Postman:  [request#1](#_bgjg04abhzcf). | 200 HTTP status code should be returned. |  |
| 5 | Verify in the ordering table within the OrderingDb that the OrderStatusID has been updated to 6. | The OrderStatusID in the ordering table updated to 6. |  |

# 

## **Verify that the order process continues whenever the payment process has succeeded.**

Test Case Purpose: Verify that the integration with the payment service and its queue are working.

Testing Environment:

* Ordering Service software
* Ordering Service database
* Basket mock
* Catalog mock
* Payment mock
* RabbitMQ

Preconditions:

* Ordering Service software is up and running.
* The Ordering Service database is connected and running.
* The login process has already been done with the details of the default user ‘Alice’.
* All the required mocks are active, filled with the relevant test data and ready for action.
* All the services that are simulated by the mocks (basket, payment and catalog) are down.
* RabbitMQ is connected and ready for action, and all of the service queues are empty.

Traceability: 3.1

Test Type:

* Functional

|  | **Steps** | **Expected Result** | **Test Result** |
| --- | --- | --- | --- |
| 1 | Send from the basket mock to the Ordering queue the following message: [Input#1](#_76f4h77dmzn3) | * The basket queue received the following message: [Output#1](#_glbdazqfclpu) * New order entity is created within the ordering table, with OrderStatusID = 1. |  |
| 2 | Verify that the catalog queue received from the ordering service the following message: [Output#2](#_fy07el923stq). | * The catalog queue received the following message: [Output#2](#_fy07el923stq). * The OrderStatusID in the ordering table is updated to 2. |  |
| 3 | Send from the catalog mock to the Ordering queue the following message: [Input#2](#_2knja88zi6ps). | The OrderStatusID in the ordering table is updated to 3. |  |
| 4 | Verify that the payment queue received from the ordering service the following message: [Output#3](#_i2goziy2oll). | The payment queue received the following message: [Output#3](#_i2goziy2oll). |  |
| 5 | Send from the payment mock to the Ordering queue the following message: [Input#3](#_orjqb39ik4oy). | The OrderStatusID is updated to 4. |  |

## **Verify that the order is canceled whenever the payment process has failed.**

Test Case Purpose: Verify that the integration with the payment service and its queue are working.

Testing Environment:

* Ordering Service software
* Ordering Service database
* Basket mock
* Catalog mock
* Payment mock
* RabbitMQ

Preconditions:

* Ordering Service software is up and running.
* The Ordering Service database is connected and running.
* The login process has already been done with the details of the default user ‘Alice’.
* All the required mocks are active, filled with the relevant test data and ready for action.
* All the services that are simulated by the mocks (basket, payment and catalog) are down.
* RabbitMQ is connected and ready for action, and all of the service queues are empty.

Traceability: 3.2

Test Type:

* Functional

|  | **Steps** | **Expected Result** | **Test Result** |
| --- | --- | --- | --- |
| 1 | Send from the basket mock to the Ordering queue the following message: [Input#1](#_76f4h77dmzn3) | * The basket queue received the following message: [Output#1](#_glbdazqfclpu) * New order entity is created within the ordering table, with OrderStatusID = 1. |  |
| 2 | Verify that the catalog queue received from the ordering service the following message: [Output#2](#_fy07el923stq). | * The catalog queue received the following message: [Output#2](#_fy07el923stq). * The OrderStatusID in the ordering table is updated to 2. |  |
| 3 | Send from the catalog mock to the Ordering queue the following message: [Input#2](#_2knja88zi6ps). | The OrderStatusID in the ordering table is updated to 3. |  |
| 4 | Verify that the payment queue received from the ordering service the following message: [Output#3](#_i2goziy2oll). | The payment queue received the following message: [Output#3](#_i2goziy2oll). |  |
| 5 | Send from the payment mock to the Ordering queue the following message: [InvalidInput#3](#_5r3tr7y22hlf). | The OrderStatusID is updated to 6. |  |

# 

## **Validate that the service will cancel the ordering process when the service is on ‘confirmstock’ status and the user does not initiate any action for 1 hour.**

Test Case Purpose: Verify that the integration with the payment service and its queue are working.

Testing Environment:

* Ordering Service software
* Ordering Service database
* Basket mock
* Catalog mock
* RabbitMQ

Preconditions:

* Ordering Service software is up and running.
* The Ordering Service database is connected and running.
* The login process has already been done with the details of the default user ‘Alice’.
* All the required mocks are active, filled with the relevant test data and ready for action.
* All the services that are simulated by the mocks (basket, payment and catalog) are down.
* Payment service is down.
* RabbitMQ is connected and ready for action, and all of the service queues are empty.

Traceability: 3.3

Test Type:

* Functional
* Security

|  | **Steps** | **Expected Result** | **Test Result** |
| --- | --- | --- | --- |
| 1 | Send from the basket mock to the Ordering queue the following message: [Input#1](#_76f4h77dmzn3) | * The basket queue received the following message: [Output#1](#_glbdazqfclpu) * New order entity is created within the ordering table, with OrderStatusID = 1. |  |
| 2 | Verify that the catalog queue received from the ordering service the following message: [Output#2](#_fy07el923stq). | * The catalog queue received the following message: [Output#2](#_fy07el923stq). * The OrderStatusID in the ordering table is updated to 2. |  |
| 3 | Send from the catalog mock to the Ordering queue the following message: [Input#2](#_2knja88zi6ps). | The OrderStatusID in the ordering table is updated to 3. |  |
| 4 | Verify that the payment queue received from the ordering service the following message: [Output#3](#_i2goziy2oll). | The payment queue received the following message: [Output#3](#_i2goziy2oll). |  |
| 5 | Wait 10 seconds, and verify that the message is still waiting in the payment queue. | The [Output#3](#_i2goziy2oll) message is still waiting in the payment queue. |  |
| 6 | Wait 1 hour, within a range of 1 hour, the OrderStatusID should be updated to 6. | Within a range of 1 hour, the OrderStatusID is updated to 6. |  |

# Verify that the order is canceled whenever a rejection message has been received from the Catalog service.

Test Case Purpose: Verify that the integration with the Catalog service and its queue are working.

Testing Environment:

* Ordering Service software
* Ordering Service database
* Basket mock
* Catalog mock
* RabbitMQ

Preconditions:

* Ordering Service software is up and running.
* The Ordering Service database is connected and running.
* The login process has already been done with the details of the default user ‘Alice’.
* All the required mocks are active, filled with the relevant test data and ready for action.
* All the services that are simulated by the mocks (basket, payment and catalog) are down.
* RabbitMQ is connected and ready for action, and all of the service queues are empty.

Traceability: 4.2

Test Type:

* Functional

|  | **Steps** | **Expected Result** | **Test Result** |
| --- | --- | --- | --- |
| 1 | Send from the basket mock to the Ordering queue the following message: [Input#1](#_76f4h77dmzn3) | * The basket queue received the following message: [Output#1](#_glbdazqfclpu) * New order entity is created within the ordering table, with OrderStatusID = 1. |  |
| 2 | Verify that the catalog queue received from the ordering service the following message: [Output#2](#_fy07el923stq). | * The catalog queue received the following message: [Output#2](#_fy07el923stq). * The OrderStatusID in the ordering table is updated to 2. |  |
| 3 | Send from the catalog mock to the Ordering queue the following message: [InvalidInput#2](#_hqi5or5uvwh6). | The OrderStatusID in the ordering table is updated to 6. |  |

# 

# Verify that a signed-in user is only exposed to his own orders.

Test Case Purpose: Verify that the user cannot be exposed to other users orders.

Testing Environment:

* Ordering Service software
* Ordering Service database
* Basket mock
* Catalog mock
* Payment mock
* RabbitMQ
* Postman

Preconditions:

* Ordering Service software is up and running.
* The Ordering Service database is connected and running.
* The login process has already been done with the details of the default user ‘Alice’.
* All the required mocks are active, filled with the relevant test data and ready for action.
* The ordering table in the DB for the signed-in user is empty.
* All the services that are simulated by the mocks (basket, payment and catalog) are down.
* RabbitMQ is connected and ready for action, and all of the service queues are empty.
* Postman is connected and ready for action.

Traceability: 5.1

Test Type:

* Security
* API

|  | **Steps** | **Expected Result** | **Test Result** |
| --- | --- | --- | --- |
| 1 | Send from the basket mock to the Ordering queue the following message: [Input#1](#_76f4h77dmzn3) | * The basket queue received the following message: [Output#1](#_glbdazqfclpu) * New order entity is created within the ordering table, with OrderStatusID = 1. |  |
| 2 | Verify that the catalog queue received from the ordering service the following message: [Output#2](#_fy07el923stq). | * The catalog queue received the following message: [Output#2](#_fy07el923stq). * The OrderStatusID in the ordering table is updated to 2. |  |
| 3 | Send from the catalog mock to the Ordering queue the following message: [Input#2](#_2knja88zi6ps). | The OrderStatusID in the ordering table is updated to 3. |  |
| 4 | Verify that the payment queue received from the ordering service the following message: [Output#3](#_i2goziy2oll). | The payment queue received the following message: [Output#3](#_i2goziy2oll). |  |
| 5 | Send from the payment mock to the Ordering queue the following message: [Input#3](#_orjqb39ik4oy). | The OrderStatusID is updated to 4. |  |
| 6 | Send the following API request from Postman:  [Request#3](#_agdhv3r96ne). | HTTP response with the last order we create should return: [Response#1](#_g7vaogb3nrxb) |  |

# 

# Verify that a signed-in user is only exposed to his own card types.

Test Case Purpose: Verify that the user cannot be exposed to other users card types.

Testing Environment:

* Ordering Service software
* Postman

Preconditions:

* Ordering Service software is up and running.
* The Ordering Service is ready for action, and the login process has already been done as with default user ‘Alice’ details.
* Postman is connected and ready for action.

Traceability: 5.2

Test Type:

* Security
* API

|  | **Steps** | **Expected Result** | **Test Result** |
| --- | --- | --- | --- |
| 1 | Send the following API request from Postman:  [Request#5](#_yd3lnnmee2s8). | The following HTTP response should return:  [Response#2](#_im549qc0j4ji) |  |

# 

# Verify that a signed-in user receives only his own orders when he tries to fetch an order by id.

Test Case Purpose: Verify that the user cannot be exposed to other users orders (by order number).

Testing Environment:

* Ordering Service software
* Ordering Service database
* Basket mock
* Catalog mock
* Payment mock
* Postman

Preconditions:

* Ordering Service software is up and running.
* The Ordering Service database is connected and running.
* The login process has already been done with the details of the default user ‘Alice’.
* All the required mocks are active, filled with the relevant test data and ready for action.
* All the services that are simulated by the mocks (basket, payment and catalog) are down.
* RabbitMQ is connected and ready for action, and all of the service queues are empty.
* Postman is connected and ready for action.

Traceability: 5.3

Test Type:

* Security
* API

|  | **Steps** | **Expected Result** | **Test Result** |
| --- | --- | --- | --- |
| 1 | Send from the basket mock to the Ordering queue the following message: [Input#1](https://docs.google.com/document/d/15H6qLAmQQ7J45IEDh7S4ot8jYbGgOssYHwk2HsqmpK0/edit#heading=h.76f4h77dmzn3) | * The basket queue received the following message: [Output#1](https://docs.google.com/document/d/15H6qLAmQQ7J45IEDh7S4ot8jYbGgOssYHwk2HsqmpK0/edit#heading=h.glbdazqfclpu) * New order entity is created within the ordering table, with OrderStatusID = 1. |  |
| 2 | Verify that the catalog queue received from the ordering service the following message: [Output#2](https://docs.google.com/document/d/15H6qLAmQQ7J45IEDh7S4ot8jYbGgOssYHwk2HsqmpK0/edit#heading=h.fy07el923stq). | * The catalog queue received the following message: [Output#2](https://docs.google.com/document/d/15H6qLAmQQ7J45IEDh7S4ot8jYbGgOssYHwk2HsqmpK0/edit#heading=h.fy07el923stq). * The OrderStatusID in the ordering table is updated to 2. |  |
| 3 | Send from the catalog mock to the Ordering queue the following message: [Input#2](https://docs.google.com/document/d/15H6qLAmQQ7J45IEDh7S4ot8jYbGgOssYHwk2HsqmpK0/edit#heading=h.2knja88zi6ps). | The OrderStatusID in the ordering table is updated to 3. |  |
| 4 | Verify that the payment queue received from the ordering service the following message: [Output#3](https://docs.google.com/document/d/15H6qLAmQQ7J45IEDh7S4ot8jYbGgOssYHwk2HsqmpK0/edit#heading=h.i2goziy2oll). | The payment queue received the following message: [Output#3](https://docs.google.com/document/d/15H6qLAmQQ7J45IEDh7S4ot8jYbGgOssYHwk2HsqmpK0/edit#heading=h.i2goziy2oll). |  |
| 5 | Send from the payment mock to the Ordering queue the following message: [Input#3](https://docs.google.com/document/d/15H6qLAmQQ7J45IEDh7S4ot8jYbGgOssYHwk2HsqmpK0/edit#heading=h.orjqb39ik4oy). | The OrderStatusID is updated to 4. |  |
| 6 | Send the following API request from Postman:  [Request#6](#_uu43x2dizxb6). | The following HTTP response should return:  [Response#3](#_im549qc0j4ji) |  |

# 

# Verify that an unauthorized request to get all orders is denied.

Test Case Purpose: Verify that the microservice API is designed to deny unauthorized requests.

Testing Environment:

* Ordering Service software
* Postman

Preconditions:

* Ordering Service software is up and running.
* The login process has already been done with the details of the default user ‘Alice’.
* Postman is connected and ready for action.

Traceability: 5.6

Test Type:

* Security
* API

|  | **Steps** | **Expected Result** | **Test Result** |
| --- | --- | --- | --- |
| 1 | Send the following API request from Postman:  [request#4](#_rowsiciuqumm). | 401 HTTP status code should return. |  |

# 

# Verify that an unauthorized request to get an order by id is denied.

Test Case Purpose: Verify that the microservice API is designed to deny unauthorized requests.

Testing Environment:

* Ordering Service software
* Postman

Preconditions:

* Ordering Service software is up and running.
* The login process has already been done with the details of the default user ‘Alice’.
* Postman is connected and ready for action.

Traceability: 5.7

Test Type:

* Security
* API

|  | **Steps** | **Expected Result** | **Test Result** |
| --- | --- | --- | --- |
| 1 | Send the following API request from Postman:  [request#7](#_13i85z73kio5). | 401 HTTP status code should return. |  |

# 

# Verify that an unauthorized request to get all card types is denied.

Test Case Purpose: Verify that the microservice API is designed to deny unauthorized requests.

Testing Environment:

* Ordering Service software
* Postman

Preconditions:

* Ordering Service software is up and running.
* The login process has already been done with the details of the default user ‘Alice’.
* Postman is connected and ready for action.

Traceability: 5.8

Test Type:

* Security
* API

|  | **Steps** | **Expected Result** | **Test Result** |
| --- | --- | --- | --- |
| 1 | Send the following API request from Postman:  [request#8](#_3xgklbm9pi83). | 401 HTTP status code should return. |  |

# 

# Verify that an unauthorized request to ship order is denied.

Test Case Purpose: Verify that the microservice API is designed to deny unauthorized requests.

Testing Environment:

* Ordering Service software
* Postman

Preconditions:

* The Ordering Service is ready for action.
* The login process has already been done with the details of the default user ‘Alice’.
* Postman is connected and ready for action.

Traceability: 5.9

Test Type:

* Security
* API

|  | **Steps** | **Expected Result** | **Test Result** |
| --- | --- | --- | --- |
| 1 | Send the following API request from Postman:  [request#9](#_bcypzsv4lmsn). | 401 HTTP status code should return. |  |

# 

# Verify that an unauthorized request to cancel order is denied.

Test Case Purpose: Verify that the microservice API is designed to deny unauthorized requests.

Testing Environment:

* Ordering Service software
* Postman

Preconditions:

* The Ordering Service is ready for action.
* The login process has already been done with the details of the default user ‘Alice’.
* Postman is connected and ready for action.

Traceability: 5.10

Test Type:

* Security
* API

|  | **Steps** | **Expected Result** | **Test Result** |
| --- | --- | --- | --- |
| 1 | Send the following API request from Postman:  [request#10](#_fhe6m9faz5xd). | 401 HTTP status code should return. |  |

# 

# Verify that the service is able to perform 300 orders while a DDOS attack is being simulated on it.

Test Case Purpose: Verify that continuously sending an API request to get all the orders will not interrupt the service, while it tries to perform the ordering process multiple times sequentially.

Testing Environment:

* Ordering Service software
* Ordering Service database
* Basket mock
* Catalog mock
* Payment mock
* RabbitMQ

Preconditions:

* Ordering Service software is up and running.
* The Ordering Service database is connected and running.
* The login process has already been done with the details of the default user ‘Alice’.
* All the required mocks are active, filled with the relevant test data and ready for action.
* All the services that are simulated by the mocks (basket, payment and catalog) are down.
* RabbitMQ is connected and ready for action, and all of the service queues are empty.

Traceability: 5.11

Test Type:

* Functional
* API
* Lodas
* Scalability
* Reliability

|  | **Steps** | **Expected Result** |
| --- | --- | --- |
| 1 | Start infinity requests loop to send the following requests:  [Request#3](#_agdhv3r96ne). | For each iteration, the following response is received:  [Response#1](#_g7vaogb3nrxb) |
| 2 | Send from the basket mock to the Ordering queue the following message: [Input#1](#_76f4h77dmzn3) | * The basket queue received the following message: [Output#1](#_glbdazqfclpu) * New order entity is created within the ordering table, with OrderStatusID = 1. |
| 3 | Verify that the catalog queue received from the ordering service the following message: [Output#2](#_fy07el923stq). | * The catalog queue received the following message: [Output#2](#_fy07el923stq). * The OrderStatusID in the ordering table is updated to 2. |
| 4 | Send from the catalog mock to the Ordering queue the following message: [Input#2](#_2knja88zi6ps). | The OrderStatusID in the ordering table is updated to 3. |
| 5 | Verify that the payment queue received from the ordering service the following message: [Output#3](#_i2goziy2oll). | The payment queue received the following message: [Output#3](#_i2goziy2oll). |
| 6 | Send from the payment mock to the Ordering queue the following message: [Input#3](#_orjqb39ik4oy). | The OrderStatusID is updated to 4. |
| 7 | Send the following API request from Postman:  [Request#2.](#_miqkoomeg1rc) | The OrderStatusID is updated to 5. |
| 8 | Repeat steps 2-6, for 300 times. | For each iteration, each OrderStatusID for each order that is being created, is updated to 5. |
| 9 | Stop the infinity requests loop from step 1. | No more requests are sent to the service. |
| 10 | Verify that there are 300 new orders in the ordering table. | There are 300 new orders in the ordering table. |

# 

# 

# Verify that the service can consume 150 messages that are waiting in the queue in a maximum time of one hour.

Test Case Purpose: Verify the service can handle a large amount of messages in a given predefined time.

Testing Environment:

* Ordering Service software
* Basket mock
* RabbitMQ

Preconditions:

* Ordering Service is down (both api and background task services).
* The login process has already been done with the details of the default user ‘Alice’.
* All the required mocks are active, filled with the relevant test data and ready for action.
* All the services that are simulated by the mocks (basket, payment and catalog) are down.
* RabbitMQ is connected and ready for action, and all of the service queues are empty.

Traceability: 6.1

Test Type:

* Functional
* Lodas
* Scalability
* Reliability

|  | **Steps** | **Expected Result** | **Test Result** |
| --- | --- | --- | --- |
| 1 | Send from the basket mock to the Ordering queue the following message: [Input#1](#_76f4h77dmzn3) | The ordering queue received the following message: [Input#1](#_76f4h77dmzn3). |  |
| 2 | repeat step 1 for 150 times. | There are 150 messages waiting in the ordering queue. |  |
| 3 | Start the ordering service. | The ordering service starts to consume the messages from its queue. |  |
| 4 | Wait for 1 hour. | The ordering queue is empty. |  |

# 

# Verify that the service is reading the first message in the queue in a maximum time of 3 seconds.

Test Case Purpose: Verify the service does not consume the first message from its queue after more than 3 seconds.

Testing Environment:

* Ordering Service software
* Basket mock
* RabbitMQ

Preconditions:

* Ordering Service software is up (both api and background task services).
* The login process has already been done with the details of the default user ‘Alice’.
* All the required mocks are active, filled with the relevant test data and ready for action.
* All the services that are simulated by the mocks (basket, payment and catalog) are down.
* RabbitMQ is connected and ready for action, and all of the service queues are empty.

Traceability: 6.2

Test Type:

* Scalability
* Reliability

|  | **Steps** | **Expected Result** | **Test Result** |
| --- | --- | --- | --- |
| 1 | Send from the basket mock to the Ordering queue the following message: [Input#1](#_76f4h77dmzn3) | The ordering queue received the following message: [Input#1](#_76f4h77dmzn3). |  |
| 2 | Verify that the ordering queue is empty after less than 3 seconds. | The ordering queue cleared from messages after less than 3 seconds. |  |

# 

# 

# Verify that the service is able to recover the data whenever the service crashes between the ‘submitted’ and ‘awaitingvalidation’ states.

Test Case Purpose: Verify that the service crashes, it has the ability to recover the ordering process data, from the exact point when the crash occurred.

Testing Environment:

* Ordering Service software
* Ordering Service database
* Basket mock
* Catalog mock
* RabbitMQ

Preconditions:

* The Ordering Service software is up (both api and background task services).
* The Ordering Service database is connected and running.
* The login process has already been done with the details of the default user ‘Alice’.
* All the required mocks are active, filled with the relevant test data and ready for action.
* All the services that are simulated by the mocks (basket, payment and catalog) are down.
* The ordering.signalrhub service is down.
* RabbitMQ is connected and ready for action, and all of the service queues are empty.

Traceability: 7.2

Test Type:

* Reliability

|  | **Steps** | **Expected Result** | **Test Result** |
| --- | --- | --- | --- |
| 1 | Sends from the basket mock to the Ordering queue the following message: [Input#1](#_76f4h77dmzn3) (send 5 messages). | The basket queue received 5 following message 5 times: [Output#1](#_glbdazqfclpu). |  |
| 2 | Verify that 5 new order entities have been created within the ordering table, with OrderStatusID of 1. | 5 new order entities are created within the ordering table, with OrderStatusID of 1. |  |
| 3 | Immediately (in a maximum time of 2 second) stop the ordering service (both api and background task services). | The ordering service is down. |  |
| 4 | Verify that there are still 5 messages of I[nput#1](#_76f4h77dmzn3) in the order queue or in the basket queue or distributed in both queues. | There are still 5 messages of [Input#1](#_76f4h77dmzn3) in the order queue or in the basket queue or distributed in both queues. |  |
| 5 | Start the ordering service (both api and background task services). | The ordering service is up. |  |
| 6 | Verify that the catalog queue received from the ordering service the following message: [Output#2](#_fy07el923stq). | * The catalog queue received the following message: [Output#2](#_fy07el923stq). * The OrderStatusID in the ordering table is updated to 2. |  |
| 7 | Verify that 5 order entities have been updated within the ordering table to OrderStatusID 2. | 5 order entities have been updated within the ordering table to OrderStatusID of 1 or 2. |  |

# 

# Verify that the service is able to recover the data whenever the service crashes between the ‘awaitingvalidation’ and ‘stockconfirmed’ states.

Test Case Purpose: Verify that the service crashes, it has the ability to recover the ordering process data, from the exact point when the crash occurred.

Testing Environment:

* Ordering Service software
* Ordering Service database
* Basket mock
* Catalog mock
* RabbitMQ

Preconditions:

* The Ordering Service software is up (both api and background task services).
* The Ordering Service database is connected and running.
* The login process has already been done with the details of the default user ‘Alice’.
* All the required mocks are active, filled with the relevant test data and ready for action.
* All the services that are simulated by the mocks (basket, payment and catalog) are down.
* The ordering.signalrhub service is down.
* RabbitMQ is connected and ready for action, and all of the service queues are empty.

Traceability: 7.3

Test Type:

* Reliability

|  | **Steps** | **Expected Result** | **Test Result** |
| --- | --- | --- | --- |
| 1 | Sends from the basket mock to the Ordering queue the following message: [Input#1](#_76f4h77dmzn3) (send 5 messages). | The basket queue received 5 following message 5 times: [Output#1](#_glbdazqfclpu) |  |
| 2 | Verify that 5 new order entities have been created within the ordering table, with OrderStatusID of 1. | 5 new order entities are created within the ordering table, with OrderStatusID of 1. |  |
| 3 | Verify that the catalog queue received from the ordering service the following message: [Output#2](#_fy07el923stq). | * The catalog queue received the following message: [Output#2](#_fy07el923stq). * 5 order entities have been updated within the ordering table to OrderStatusID of 2. |  |
| 4 | Immediately (in a maximum time of 2 second) stop the ordering service (both api and background task services). | The ordering service is down. |  |
| 5 | Send from the catalog mock to the Ordering queue the following message: [Input#2](#_2knja88zi6ps) (send 5 messages). | There are 5 messages of[Input#2](https://docs.google.com/document/d/15H6qLAmQQ7J45IEDh7S4ot8jYbGgOssYHwk2HsqmpK0/edit#heading=h.2knja88zi6ps) waiting in the ordering queue |  |
| 6 | Start the ordering service (both api and background task services). | The ordering service is up. |  |
| 7 | Immediately (in a maximum time of 2 second) stop the ordering service (both api and background task services). | The ordering service is down. |  |
| 8 | Verify that there are still 5 messages of [Input#2](#_2knja88zi6ps) in the order queue or in the basket queue or distributed in both queues. | There are still 5 messages of [Input#2](#_76f4h77dmzn3) in the order queue or in the basket queue or distributed in both queues. |  |
| 9 | Start the ordering service (both api and background task services). | The ordering service is up. |  |
| 10 | Verify that 5 order entities have been updated within the ordering table to OrderStatusID 3. | 5 order entities have been updated within the ordering table to OrderStatusID of 2 or 3. |  |

# 

# Verify that the service is able to recover the data whenever the service crashes between the ‘stockconfirmed’ and ‘paid’ states.

Test Case Purpose: Verify that the service crashes, it has the ability to recover the ordering process data, from the exact point when the crash occurred.

Testing Environment:

* Ordering Service software
* Ordering Service database
* Basket mock
* Catalog mock
* RabbitMQ

Preconditions:

* The Ordering Service software is down(both api and background task services).
* The Ordering Service database is connected and running.
* The login process has already been done with the details of the default user ‘Alice’.
* All the required mocks are active, filled with the relevant test data and ready for action.
* All the services that are simulated by the mocks (basket, payment and catalog) are down.
* The ordering.signalrhub service is down.
* RabbitMQ is connected and ready for action, and all of the service queues are empty.

Traceability: 7.3

Test Type:

* Reliability

|  | **Steps** | **Expected Result** | **Test Result** |
| --- | --- | --- | --- |
| 1 | Sends from the basket mock to the Ordering queue the following message: [Input#1](#_76f4h77dmzn3) (send 5 messages). | The basket queue received 5 following message 5 times: [Output#1](#_glbdazqfclpu) |  |
| 2 | Verify that 5 new order entities have been created within the ordering table, with OrderStatusID of 1. | 5 new order entities are created within the ordering table, with OrderStatusID of 1. |  |
| 3 | Verify that the catalog queue received from the ordering service the following message: [Output#2](#_fy07el923stq). | * The catalog queue received the following message: [Output#2](#_fy07el923stq). * 5 order entities have been updated within the ordering table to OrderStatusID of 2. |  |
| 4 | Verify that the payment queue received from the ordering service the following message: [Output#3](#_i2goziy2oll). | * The catalog queue received the following message:[Output#3](#_i2goziy2oll). * 5 order entities have been updated within the ordering table to OrderStatusID of 3. |  |
| 5 | Immediately (in a maximum time of 2 second) stop the ordering service (both api and background task services). | The ordering service is down. |  |
| 6 | Send from the payment mock to the Ordering queue the following message: [Input#3](#_orjqb39ik4oy) (send 5 messages). | There are 5 messages of[Input#3](#_orjqb39ik4oy) waiting in the ordering queue |  |
| 7 | Start the ordering service (both api and background task services). | The ordering service is up. |  |
| 8 | Immediately (in a maximum time of 2 second) stop the ordering service (both api and background task services). | The ordering service is down. |  |
| 9 | Verify that there are still 5 messages of [Input#3](#_orjqb39ik4oy) in the order queue or in the basket queue or distributed in both queues. | There are still 5 messages of [Input#3](#_orjqb39ik4oy) in the order queue or in the basket queue or distributed in both queues. |  |
| 10 | Start the ordering service (both api and background task services). | The ordering service is up. |  |
| 11 | Verify that 5 order entities have been updated within the ordering table to OrderStatusID 4. | 5 order entities have been updated within the ordering table to OrderStatusID of 3 or 4. |  |

# Verify that the Ordering Service is logging whenever the order status is ‘submitted’.

Test Case Purpose: Verify the service integration with external logging systems.

Testing Environment:

* Ordering Service software
* Basket mock
* RabbitMQ
* Seq service software

Preconditions:

* Ordering Service software is up and running.
* Seq service is connected to the ordering service, running, and ready for action.
* The login process has already been done with the details of the default user ‘Alice’.
* All the required mocks are active, filled with the relevant test data and ready for action.
* All the services that are simulated by the mocks (basket, payment and catalog) are down.
* RabbitMQ is connected and ready for action, and all of the service queues are empty.

Traceability: 8.1

Test Type:

* Reporting and Analytics
* Integration

|  | **Steps** | **Expected Result** | **Test Result** |
| --- | --- | --- | --- |
| 1 | Send from the basket mock to the Ordering queue the following message: [Input#1](https://docs.google.com/document/d/13vJ2tp8nXGxaUgwYOUn_hCu5beQGxmPzcLUslSgFjls/edit#heading=h.3vac5uf) | The ordering queue received the following message: [Input#1](https://docs.google.com/document/d/13vJ2tp8nXGxaUgwYOUn_hCu5beQGxmPzcLUslSgFjls/edit#heading=h.3vac5uf). |  |
| 2 | Verify inside the Seq logs, that the following log message appears from the current date and time that the message was sent from the basket mock to the ordering queue:  [Message#1](https://docs.google.com/document/d/13vJ2tp8nXGxaUgwYOUn_hCu5beQGxmPzcLUslSgFjls/edit#heading=h.2koq656) | The message: [Message#1](https://docs.google.com/document/d/13vJ2tp8nXGxaUgwYOUn_hCu5beQGxmPzcLUslSgFjls/edit#heading=h.2koq656)  appears in the Seq logs. |  |

# 

# Verify that the Ordering Service is logging whenever the order status is ‘awaitingvalidation’.

Test Case Purpose: Verify the service integration with external logging systems.

Testing Environment:

* Ordering Service software
* Basket mock
* RabbitMQ
* Seq service software

Preconditions:

* Ordering Service software is up and running.
* Seq service is connected to the ordering service, running, and ready for action.
* The login process has already been done with the details of the default user ‘Alice’.
* All the required mocks are active, filled with the relevant test data and ready for action.
* All the services that are simulated by the mocks (basket, payment and catalog) are down.
* RabbitMQ is connected and ready for action, and all of the service queues are empty.

Traceability: 8.2

Test Type:

* Reporting and Analytics
* Integration

|  | **Steps** | **Expected Result** | **Test Result** |
| --- | --- | --- | --- |
| 1 | Send from the basket mock to the Ordering queue the following message: [Input#1](#_76f4h77dmzn3) | The ordering queue received the following message: [Input#1](#_76f4h77dmzn3). |  |
| 2 | Verify that the catalog queue received from the ordering service the following message: [Output#2](#_fy07el923stq). | * The catalog queue received the following message: [Output#2](#_fy07el923stq). * The OrderStatusID in the ordering table is updated to 2. |  |
| 3 | Verify inside the Seq logs, that the following log message appears from the current date and time that the message was sent from the basket mock to the ordering queue:  [Message#2](#_ouu8p2ugg2x) | The message: [Message#2](#_ouu8p2ugg2x)  appears in the Seq logs. |  |

# Verify that the Ordering Service is logging whenever the order status is ‘stockconfirmed’.

Test Case Purpose: Verify the service integration with external logging systems.

Testing Environment:

* Ordering Service software
* Ordering Service database
* Basket mock
* Catalog mock
* RabbitMQ
* Seq service software

Preconditions:

* Ordering Service software is up and running.
* The Ordering Service database is connected and running.
* Seq service is connected to the ordering service, running, and ready for action.
* The login process has already been done with the details of the default user ‘Alice’.
* All the required mocks are active, filled with the relevant test data and ready for action.
* All the services that are simulated by the mocks (basket, payment and catalog) are down.
* RabbitMQ is connected and ready for action, and all of the service queues are empty.

Traceability: 8.3

Test Type:

* Reporting and Analytics
* Integration

|  | **Steps** | **Expected Result** | **Test Result** |
| --- | --- | --- | --- |
| 1 | Send from the basket mock to the Ordering queue the following message: [Input#1](#_76f4h77dmzn3) | The ordering queue received the following message: [Input#1](#_76f4h77dmzn3). |  |
| 2 | Verify that the catalog queue received from the ordering service the following message: [Output#2](#_fy07el923stq). | * The catalog queue received the following message: [Output#2](#_fy07el923stq). * The OrderStatusID in the ordering table is updated to 2. |  |
| 3 | Send from the catalog mock to the Ordering queue the following message: [Input#2](#_2knja88zi6ps). | The OrderStatusID in the ordering table is updated to 3. |  |
| 4 | Verify inside the Seq logs, that the following log message appears from the current date and time that the message was sent from the basket mock to the ordering queue:  [Message#3](#_o1wzqf930lic) | The message: [Message#3](#_o1wzqf930lic)  appears in the Seq logs. |  |

# 

# Verify that the Ordering Service is logging whenever the order status is ‘paid’.

Test Case Purpose: Verify the service integration with external logging systems.

Testing Environment:

* Ordering Service software
* Ordering Service database
* Basket mock
* Catalog mock
* Payment mock
* RabbitMQ
* Seq service software

Preconditions:

* Ordering Service software is up and running.
* The Ordering Service database is connected and running.
* Seq service is connected to the ordering service, running, and ready for action.
* The login process has already been done with the details of the default user ‘Alice’.
* All the required mocks are active, filled with the relevant test data and ready for action.
* All the services that are simulated by the mocks (basket, payment and catalog) are down.
* RabbitMQ is connected and ready for action, and all of the service queues are empty.

Traceability: 8.4

Test Type:

* Reporting and Analytics
* Integration

|  | **Steps** | **Expected Result** | **Test Result** |
| --- | --- | --- | --- |
| 1 | Send from the basket mock to the Ordering queue the following message: [Input#1](#_76f4h77dmzn3) | The ordering queue received the following message: [Input#1](#_76f4h77dmzn3). |  |
| 2 | Verify that the catalog queue received from the ordering service the following message: [Output#2](#_fy07el923stq). | * The catalog queue received the following message: [Output#2](#_fy07el923stq). * The OrderStatusID in the ordering table is updated to 2. |  |
| 3 | Send from the catalog mock to the Ordering queue the following message: [Input#2](#_2knja88zi6ps). | The OrderStatusID in the ordering table is updated to 3. |  |
| 4 | Verify that the payment queue received from the ordering service the following message: [Output#3](#_i2goziy2oll). | The payment queue received the following message: [Output#3](#_i2goziy2oll). |  |
| 5 | Send from the payment mock to the Ordering queue the following message: [Input#3](#_orjqb39ik4oy). | The OrderStatusID is updated to 4. |  |
| 6 | Verify inside the Seq logs, that the following log message appears from the current date and time that the message was sent from the basket mock to the ordering queue:  [Message#4](#_d5n9o9p1g8r9) | The message: [Message#4](#_d5n9o9p1g8r9)  appears in the Seq logs. |  |

# 

# Verify that the Ordering Service is logging whenever the order status is ‘shipped’.

Test Case Purpose: Verify the service integration with external logging systems.

Testing Environment:

* Ordering Service software
* Ordering Service database
* Basket mock
* Catalog mock
* Payment mock
* RabbitMQ
* Postman
* Seq service software

Preconditions:

* Ordering Service software is up and running.
* The Ordering Service database is connected and running.
* Seq service is connected to the ordering service, running, and ready for action.
* The login process has already been done with the details of the default user ‘Alice’.
* All the required mocks are active, filled with the relevant test data and ready for action.
* All the services that are simulated by the mocks (basket, payment and catalog) are down.
* RabbitMQ is connected and ready for action, and all of the service queues are empty.
* Postman is connected and ready for action.

Traceability: 8.5

Test Type:

* Reporting and Analytics
* Integration

|  | **Steps** | **Expected Result** | **Test Result** |
| --- | --- | --- | --- |
| 1 | Send from the basket mock to the Ordering queue the following message: [Input#1](#_76f4h77dmzn3) | The ordering queue received the following message: [Input#1](#_76f4h77dmzn3). |  |
| 2 | Verify that the catalog queue received from the ordering service the following message: [Output#2](#_fy07el923stq). | * The catalog queue received the following message: [Output#2](#_fy07el923stq). * The OrderStatusID in the ordering table is updated to 2. |  |
| 3 | Send from the catalog mock to the Ordering queue the following message: [Input#2](#_2knja88zi6ps). | The OrderStatusID in the ordering table is updated to 3. |  |
| 4 | Verify that the payment queue received from the ordering service the following message: [Output#3](#_i2goziy2oll). | The payment queue received the following message: [Output#3](#_i2goziy2oll). |  |
| 5 | Send from the payment mock to the Ordering queue the following message: [Input#3](#_orjqb39ik4oy). | The OrderStatusID is updated to 4. |  |
| 6 | Send the following API request from Postman:  [request#](#_miqkoomeg1rc)2. | * 200 HTTP status code should be returned. * The OrderStatusID in the ordering table updated to 5. |  |
| 7 | Verify inside the Seq logs, that the following log message appears from the current date and time that the message was sent from the basket mock to the ordering queue:  [Message#5](#_mdi7oc5tqw20). | The message: [Message#5](#_mdi7oc5tqw20)  appears in the Seq logs. |  |

# 

# Verify that the Ordering Service is logging whenever the order status is ‘canceled’.

Test Case Purpose: Verify the service integration with external logging systems.

Testing Environment:

* Ordering Service software
* Ordering Service database
* Basket mock
* Catalog mock
* Payment mock
* RabbitMQ
* Postman
* Seq service software

Preconditions:

* Ordering Service software is up and running.
* The Ordering Service database is connected and running.
* Seq service is connected to the ordering service, running, and ready for action.
* The login process has already been done with the details of the default user ‘Alice’.
* All the required mocks are active, filled with the relevant test data and ready for action.
* All the services that are simulated by the mocks (basket, payment and catalog) are down.
* RabbitMQ is connected and ready for action, and all of the service queues are empty.
* Postman is connected and ready for action.

Traceability: 8.6

Test Type:

* Reporting and Analytics
* Integration

|  | **Steps** | **Expected Result** | **Test Result** |
| --- | --- | --- | --- |
| 1 | Send from the basket mock to the Ordering queue the following message: [Input#1](#_76f4h77dmzn3) | The ordering queue received the following message: [Input#1](#_76f4h77dmzn3). |  |
| 2 | Send the following API request from Postman:  [request#1](https://docs.google.com/document/d/1P06K0LAtOx-pWbx5jL-hiIX0isExkIIy0pEHRR4sGMY/edit#heading=h.bgjg04abhzcf). | * 200 HTTP status code should be returned. * The OrderStatusID is updated to 6. |  |
| 2 | Verify inside the Seq logs, that the following log message appears from the current date and time that the message was sent from the basket mock to the ordering queue:  [Message#6](#_ggu6meyx6dex). | The message: [Message#6](#_ggu6meyx6dex)  appears in the Seq logs. |  |

# 

# Verify that the Ordering Service is logging whenever the app crashes.

Test Case Purpose: Verify the service integration with external logging systems.

Testing Environment:

* Ordering Service software
* Seq service software

Preconditions:

* Ordering Service software is up and running.
* Seq service is connected to the ordering service, running, and ready for action.

Traceability: 8.7

Test Type:

* Reporting and Analytics
* Integration

|  | **Steps** | **Expected Result** | **Test Result** |
| --- | --- | --- | --- |
| 1 | Stop the ordering service. | The ordering service is down. |  |
| 2 | Verify inside the Seq logs, that the following log message appears from the current date and time that the message was sent from the basket mock to the ordering queue:  [Message#7](#_edtkyoqxt2h7). | The message: [Message#7](#_edtkyoqxt2h7)  appears in the Seq logs. |  |

# 

# Messages

## Inputs & Outputs

### Input#1

Path: Basket -> Ordering

Routing Key: UserCheckoutAcceptedIntegrationEvent

Details: The basket tells the Ordering Service to ‘take’ the order items from the basket.

Data:

{

"UserId": "5b2eb009-f2b4-4406-a2a5-2949721f7872",

"UserName": "alice",

"OrderNumber": 0,

"City": "Redmond",

"Street": "15703 NE 61st Ct",

"State": "WA",

"Country": "U.S.",

"ZipCode": "98052",

"CardNumber": "4012888888881881",

"CardHolderName": "Alice Smith",

"CardExpiration": "2024-12-31T22:00:00Z",

"CardSecurityNumber": "123",

"CardTypeId": 1,

"Buyer": null,

"RequestId": "a5890489-db5d-4e3e-81e9-56dae3ac9dde",

"Basket": {

"BuyerId": "5b2eb009-f2b4-4406-a2a5-2949721f7872",

"Items": [

{

"Id": "ec13598b-9a25-4624-b0a0-e9069be548d2",

"ProductId": 1,

"ProductName": ".NET Bot Black Hoodie",

"UnitPrice": 19.5,

"OldUnitPrice": 0,

"Quantity": 1,

"PictureUrl": "http://host.docker.internal:5202/c/api/v1/catalog/items/1/pic/"

},

{

"Id": "43b0d9d0-802b-4987-b9a1-b648b094f5d3",

"ProductId": 6,

"ProductName": ".NET Blue Hoodie",

"UnitPrice": 12,

"OldUnitPrice": 0,

"Quantity": 1,

"PictureUrl": "http://host.docker.internal:5202/c/api/v1/catalog/items/6/pic/"

},

{

"Id": "1c82cfd8-099b-4ea2-854f-7ee287684a08",

"ProductId": 2,

"ProductName": ".NET Black \u0026 White Mug",

"UnitPrice": 8.5,

"OldUnitPrice": 0,

"Quantity": 1,

"PictureUrl": "http://host.docker.internal:5202/c/api/v1/catalog/items/2/pic/"

}

]

},

"Id": "c008b3e0-e95a-44ec-a1a0-0916931c33d0",

"CreationDate": "2023-03-06T19:18:10.7326158Z"

}

### Output#1

Path: Ordering -> Basket

Routing Key: OrderStartedIntegrationEvent

Details: Confirmation messages sent from the Ordering Service to the basket queue, to start the ordering process.

Data:

{

"UserId":"b9e5dcdd-dae2-4b1c-a991-f74aae042814",

"Id":"a2fbc6c0-07cf-4536-a1f5-fc9ede9fec07",

"CreationDate":"2023-03-05T13:43:13.8898923Z"

}

### Input#2

Path: Catalog -> Ordering

Routing Key: OrderStockConfirmedIntegrationEvent

Details: The catalog service sends to the orders queue the stock validation confirmation message.

Data:

{

"OrderId": <current test case order id>,

"Id": "b84dc7a5-1d0e-429e-a800-d3024d9c724f",

"CreationDate": "2023-03-05T15:33:18.1376971Z"

}

### Output#2

Path: Ordering -> Catalog

Routing Key: OrderStatusChangedToAwaitingValidationIntegrationEvent

Details: The Ordering Service tells the catalog service that it is waiting for a catalog validation message.

Data:

{

"OrderId": <current test case order id>,

"OrderStatus":"awaitingvalidation",

"BuyerName": "alice",

"Id":"cca155c0-4480-4c93-a763-910e54218040",

"CreationDate":"2023-03-05T17:07:35.6306122Z"

}

### 

### Input#3

Path: Payment -> Ordering

Routing Key: OrderPaymentSucceededIntegrationEvent

Details: The payment service sends to the orders queue the payment validation confirmation message.

Data:

{

"OrderId": <current test case order id>,

"Id": "b84dc7a5-1d0e-429e-a800-d3024d9c724f",

"CreationDate": "2023-03-05T15:33:18.1376971Z"

}

### Output#3

Path: Ordering -> Payment

Routing Key: OrderStatusChangedToStockConfirmedIntegrationEvent

Details: The Ordering Service tells the payment service that it is waiting for a payment validation message.

Data:

{

"OrderId": <current test case order id>,

"OrderStatus":"stockconfirmed",

"BuyerName": "alice",

"Id":"cca155c0-4480-4c93-a763-910e54218040",

"CreationDate":"2023-03-05T17:07:35.6306122Z"

}

## Invalid Inputs

### InvalidInput#1

Basket -> Ordering

Routing Key: UserCheckoutAcceptedIntegrationEvent

Details: The basket tells the Ordering Service to ‘take’ the checkout items, with an exceeding quantity of a single item.

Data:

{

"UserId": "5b2eb009-f2b4-4406-a2a5-2949721f7872",

"UserName": "alice",

"OrderNumber": 0,

"City": "Redmond",

"Street": "15703 NE 61st Ct",

"State": "WA",

"Country": "U.S.",

"ZipCode": "98052",

"CardNumber": "4012888888881881",

"CardHolderName": "Alice Smith",

"CardExpiration": "2024-12-31T22:00:00Z",

"CardSecurityNumber": "123",

"CardTypeId": 1,

"Buyer": null,

"RequestId": "a5890489-db5d-4e3e-81e9-56dae3ac8dde",

"Basket": {

"BuyerId": "5b2eb009-f2b4-4406-a2a5-2949721f7872",

"Items": [

{

"Id": "ec13598b-9a25-4624-b0a0-e9069be548d2",

"ProductId": 1,

"ProductName": ".NET Bot Black Hoodie",

"UnitPrice": 19.5,

"OldUnitPrice": 0,

"Quantity": 100000000,

"PictureUrl": "http://host.docker.internal:5202/c/api/v1/catalog/items/1/pic/"

},

{

"Id": "43b0d9d0-802b-4987-b9a1-b648b094f5d3",

"ProductId": 6,

"ProductName": ".NET Blue Hoodie",

"UnitPrice": 12,

"OldUnitPrice": 0,

"Quantity": 1,

"PictureUrl": "http://host.docker.internal:5202/c/api/v1/catalog/items/6/pic/"

},

{

"Id": "1c82cfd8-099b-4ea2-854f-7ee287684a08",

"ProductId": 2,

"ProductName": ".NET Black \u0026 White Mug",

"UnitPrice": 8.5,

"OldUnitPrice": 0,

"Quantity": 1,

"PictureUrl": "http://host.docker.internal:5202/c/api/v1/catalog/items/2/pic/"

}

]

},

"Id": "c008b3e0-e95a-44ec-a1a0-0916931c33d0",

"CreationDate": "2023-03-06T19:18:10.7326158Z"

}

### InvalidInput#2

Catalog -> Ordering

Routing Key: OrderStockRejectedIntegrationEvent

Reason: The catalog service sends to the orders service that the stock validation process has failed.

Data:

{

"OrderId": <current test case order id>,

"OrderStockItems": [

{

"ProductId": 1,

"HasStock": false

}

],

"Id": "99c3f974-c6ed-41a4-8e01-5cb00f9e6335",

"CreationDate": "2023-03-05T15:51:11.5458796Z"

}

### InvalidInput#3

Payment-> Ordering

Routing Key: OrderPaymentFailedIntegrationEvent

Reason: The payment service sends to the orders service that the payment process has failed.

Data:

{

"OrderId": <current test case order id>,

"OrderStatus": "stockconfirmed",

"BuyerName": "alice",

"Id": "cca155c0-4480-4c93-a763-910e54218040",

"CreationDate": "2023-03-05T17:07:35.6306122Z"

}

## API Requests

### Request#1 - Cancel the order

HTTP Method: PUT

Request URL: <http://host.docker.internal:5102/api/v1/Orders/cancel>

Authorization: <Retrieve token from <http://host.docker.internal:5102/swagger/index.html>>

x-requestid: <Generate from here you're UUID: <https://www.uuidgenerator.net>>

Request Body:

{

"orderNumber": <current test case order number>

}

### Request#2 - Ship the order

HTTP Method: PUT

Request URL: <http://host.docker.internal:5102/ordering-api/api/v1/Orders/ship>

Authorization: <Retrieve token from <http://host.docker.internal:5102/swagger/index.html>>

x-requestid: <Generate from here you're UUID: <https://www.uuidgenerator.net>>

Request Body:

{

"orderNumber": <current test case order number>

}

### Request#3 - Get all orders

HTTP Method: GET

Request URL: <http://host.docker.internal:5102/ordering-api/api/v1/Orders>

Authorization: <Retrieve token from <http://host.docker.internal:5102/swagger/index.html>>

x-requestid: <Generate from here you're UUID: <https://www.uuidgenerator.net>>

### Request#4 - Unauthorized request to get all orders

HTTP Method: GET

Request URL: <http://host.docker.internal:5102/ordering-api/api/v1/Orders>

Authorization: 123alice

x-requestid: <Generate from here you're UUID: <https://www.uuidgenerator.net>>

### Request#5 - Get all card types

HTTP Method: GET

Request URL: <http://host.docker.internal:5102/ordering-api/api/v1/Orders/cardtypes>

Authorization: <Retrieve token from <http://host.docker.internal:5102/swagger/index.html>>

x-requestid: <Generate from here you're UUID: <https://www.uuidgenerator.net>>

### Request#6 - Get order by id

HTTP Method: GET

Request URL: [http://host.docker.internal:5102/ordering-api/api/v1/Orders/](http://host.docker.internal:5102/ordering-api/api/v1/Orders/1)<current test case order id>

Authorization: <Retrieve token from <http://host.docker.internal:5102/swagger/index.html>>

x-requestid: <Generate from here you're UUID: <https://www.uuidgenerator.net>>

### Request#7 - Unauthorized request to get order by id

HTTP Method: GET

Request URL: [http://host.docker.internal:5102/ordering-api/api/v1/Orders/](http://host.docker.internal:5102/ordering-api/api/v1/Orders/1)<current test case order id>

Authorization: 123alice

x-requestid: <Generate from here you're UUID: <https://www.uuidgenerator.net>>

### Request#8 - Unauthorized request to get all card types

HTTP Method: GET

Request URL: <http://host.docker.internal:5102/ordering-api/api/v1/Orders/cardtypes>

Authorization: 123alice

x-requestid: <Generate from here you're UUID: <https://www.uuidgenerator.net>>

### Request#9 - Unauthorized request to ship order

HTTP Method: PUT

Request URL: <http://host.docker.internal:5102/ordering-api/api/v1/Orders/ship>

Authorization: 123alice

x-requestid: <Generate from here you're UUID: <https://www.uuidgenerator.net>>

Request Body:

{

"orderNumber": <current test case order number>

}

### Request#10 - Unauthorized request to cancel order

HTTP Method: PUT

Request URL: <http://host.docker.internal:5102/api/v1/Orders/cancel>

Authorization: 123alice

x-requestid: <Generate from here you're UUID: <https://www.uuidgenerator.net>>

Request Body:

{

"orderNumber": <current test case order number>

}

### 

### 

## API Responses

### Response#1 - Get all orders

Response Status Code: 200

Response body:

[

{

"ordernumber": 1,

"date": "2023-03-12T16:33:41.7108509",

"status": "paid",

"total": 90

}

]

### Responset#2 - Get all card types

Response Status Code: 200

Response body:

[

{

"id": 1,

"name": "Amex"

},

{

"id": 2,

"name": "Visa"

},

{

"id": 3,

"name": "MasterCard"

}

]

### Responset#3 - Get order by id

Response Status Code: 200

Response body example:

{

"ordernumber": <current test case order id>,

"date": "2023-03-06T17:45:31.6700604",

"status": "paid",

"description": null,

"street": "15703 NE 61st Ct",

"city": "Redmond",

"zipcode": "98052",

"country": "U.S.",

"orderitems": [

{

"productname": ".NET Blue Hoodie",

"units": 1,

"unitprice": 12,

"pictureurl": "http://host.docker.internal:5202/c/api/v1/catalog/items/6/pic/"

},

{

"productname": ".NET Black & White Mug",

"units": 1,

"unitprice": 8.5,

"pictureurl": "http://host.docker.internal:5202/c/api/v1/catalog/items/2/pic/"

},

{

"productname": ".NET Bot Black Hoodie",

"units": 1,

"unitprice": 19.5,

"pictureurl": "http://host.docker.internal:5202/c/api/v1/catalog/items/1/pic/"

}

],

"total": 40

}

## 

## Logs Messages

### Message#1 - Status Changed to Submitted

----- CreateOrderCommand suceeded - RequestId: <current test case request id>

### Message#2 - Status Changed to ‘awaitvalidation’

----- Command SetAwaitingValidationOrderStatusCommand handled - response: true

### Message#3 - Status Changed to ‘stockcomfirmed’

----- Command SetStockConfirmedOrderStatusCommand handled - response: true

### 

### Message#4 - Status Changed to ‘paid’

----- Command SetPaidOrderStatusCommand handled - response: true

### Message#5 - Status Changed to ‘shipped’

----- Command IdentifiedCommand<ShipOrderCommand,Boolean> handled - response: true

### Message#6 - Status Changed to ‘canceled’

----- Command IdentifiedCommand<CancelOrderCommand,Boolean> handled - response: true

### Message#7 - Service Crashes

GetHealthReport threw an exception when trying to get report from http://ordering-api/hc configured with name Ordering HTTP Check.