Basic Test Cases (Success & Error)

Create Queue Test Case 1 (Success)

Title: Create Queue API - Successful request creates queue in database

Description: A successful request made to the server should create a new entry in the **companyQueue** table with the correct parameters.

Precondition: **companyQueue** table created with no existing **queue_id**.

Test Steps:

1. Start the Backend Server
2. Send a Create Queue request to the backend with a **valid company_id ,** and an **existing** and **valid queue_id**

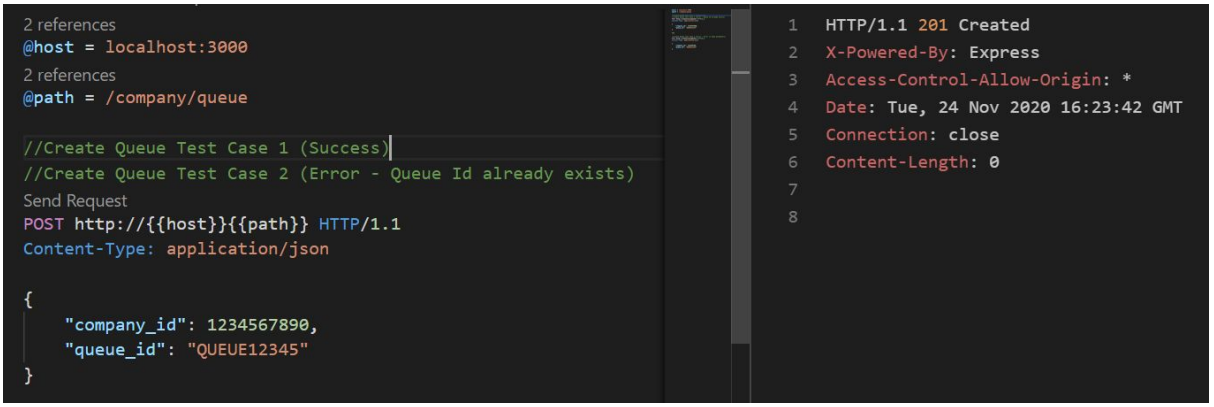
```
{
 "company_id": 1234567890,
 "queue_id": "QUEUE12345"
}
```

3. Receive a **201 Created** response
4. Go to elephantsql to inspect the **companyQueue** table

Expected Result: **Queue** '**QUEUE12345**' from **company 1234567890** is created in the **companyQueue** table

Actual Result: **Queue** '**QUEUE12345**' from **company 1234567890** is created in the **companyQueue** table (**PASS**)

Evidence:



Create Queue Test Case 2 (Error - Queue Id already exists)

Title: Create Queue API - Error in queue id request prevents create queue in database

Description: When an error request with duplicated **`queue id`** is made to the server, it should receive a 404 error response and not create a queue in the **`companyQueue`** table.

Precondition: **`companyQueue`** table created, **`queue_id QUEUE12345`** exists.

Test Steps:

1. Start the Backend Server
2. Send a Create Queue request to the backend with an **existing** and **valid** **`company_id`** and **`queue_id`**

```
{
 "company_id": 8912345678,
 "queue_id": "QUEUE12345"
}
```

3. Receive a **`422 Unprocessable Entity`** response with an error response body

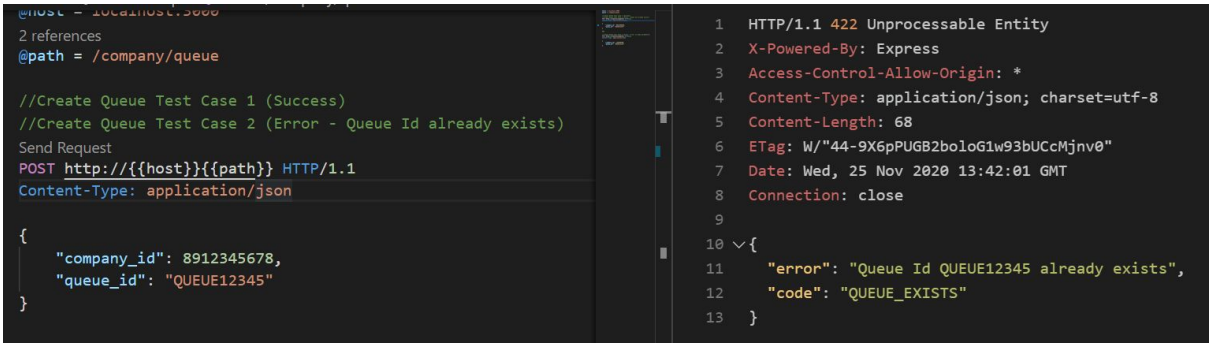
```
{
 "error": "Queue Id QUEUE12345 already exists",
 "code": "QUEUE_EXISTS"
}
```

4. Go to ElephantSQL to inspect the **`companyQueue`** table

Expected Result: No new row with the **`queue_id`** and **`company_id`** is created, the only row existing is what was created in the precondition.

Actual Result: No new row with the **`queue_id`** and **`company_id`** is created, the only row existing is what was created in the precondition (**PASS**)

Evidence:



Create Queue Test Case 3 (Error - Error in body parameters)

Title: Create Queue API - Invalid customer_id in request prevents create queue in database

Description: When an error request with an invalid **`customer id`** of less than 10 digits is made to the server, it should receive an error response and not create a queue in the **`companyQueue`** table.

Precondition: **`companyQueue`** table created with no existing **`queue_id`**.

Test Steps:

1. Start the Backend Server
2. Send a Create Queue request to the backend with an **invalid `company_id`** but a **valid `queue_id`**

```
{
 "company_id": 123456789,
 "queue_id": "QUEUE12345"
}
```

3. Receive a **`400 Bad request`** response with an error response body

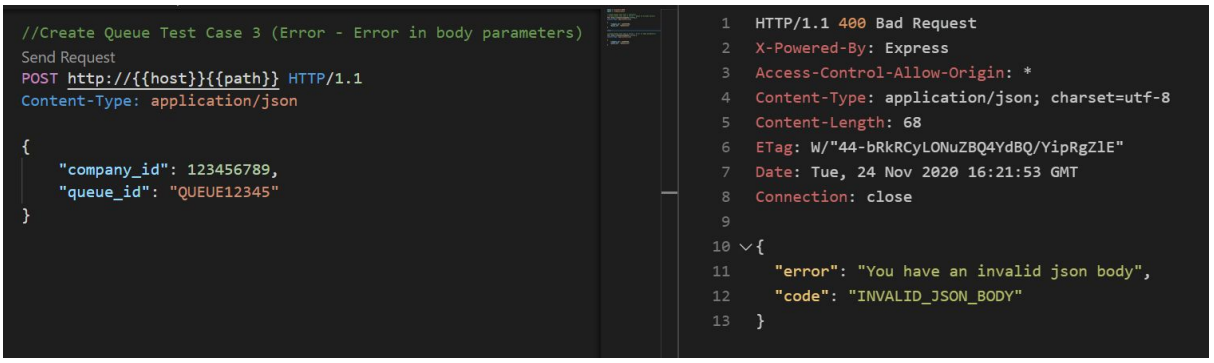
```
{
 "error": "You have an invalid json body",
 "code": "INVALID_JSON_BODY"
}
```

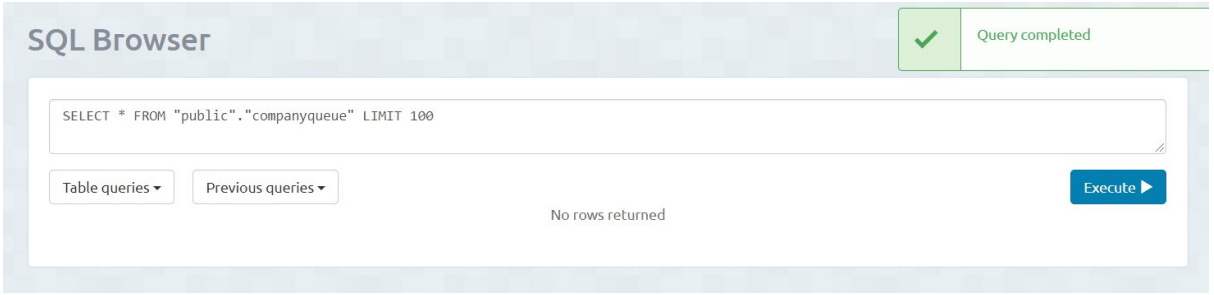
4. Go to elephantsql to inspect the **`companyQueue`** table

Expected Result: There is no data in companyQueue table

Actual Result: There is no data in companyQueue table (**PASS**)

Evidence:





Update Queue Test Case 1 (Success - Activate Queue)

Title: Update Queue API - Successful request updates queue **`status`** in database to '**`ACTIVATE`**'

Description: A successful request made to the server should update the **`status`** of the queue that already exists in the **`companyQueue`** table with the correct parameters.

Precondition: **`companyQueue`** table created with 1 row of data **`queue_id`** **`QUEUE12345`** with **`status`** '**`INACTIVE`**'.

Test Steps:

1. Start the Backend Server
2. Send an Update Queue request to the backend with a **valid `status`** '**`Activate`**' in the json body

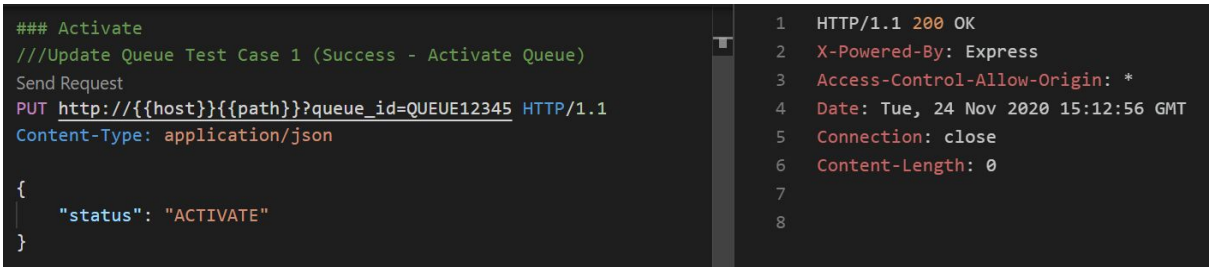
```
{
  "status": "ACTIVATE"
}
```

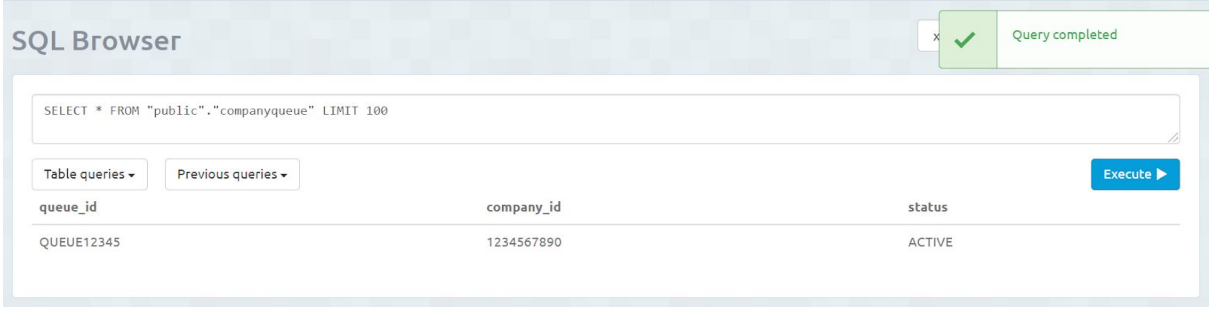
3. Receive a **`200 OK`** response
4. Go to elephantsql to inspect the **`companyQueue`** table

Expected Result: The **`status`** of the queue is changed to **'ACTIVE'**

Actual Result: The **`status`** of the queue is changed to **'ACTIVE'** (**PASS**)

Evidence:





Update Queue Test Case 2 (Success - Deactivate Queue)

Title: Update Queue API - Successful request updates queue **`status`** in database to '**`INACTIVE`**'

Description: A successful request made to the server should update the **`status`** of the queue that already exists in the **`companyQueue`** table with the correct parameters.

Precondition: **`companyQueue`** table created with 1 row of data **`queue_id`** **`QUEUE12345`** with **`status`** '**`ACTIVE`**'.

Test Steps:

1. Start the Backend Server
2. Send an Update Queue request to the backend with a **valid `status`** '**`DEACTIVATE`**' in the json body

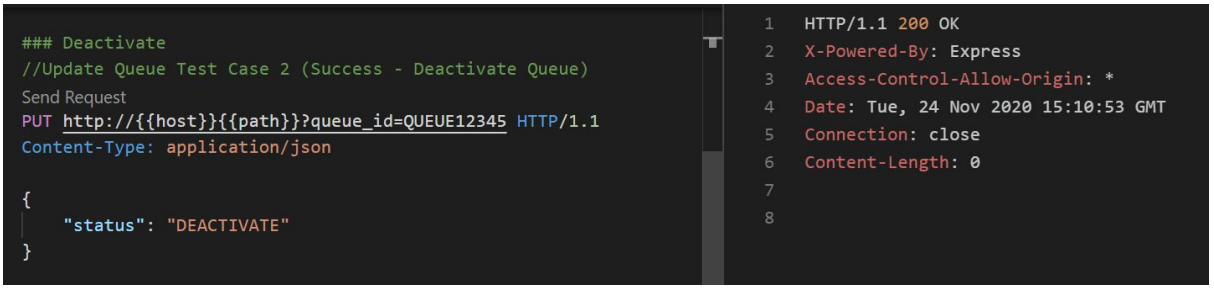
```
{
  "status": "DEACTIVATE"
}
```

3. Receive a **`200 OK`** response
4. Go to elephantsql to inspect the **`companyQueue`** table

Expected Result: The **`status`** of the queue is changed to **'INACTIVE'**

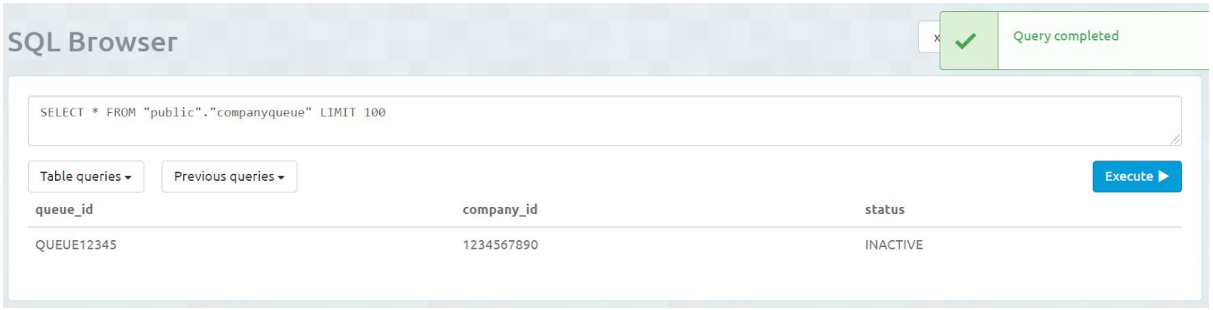
Actual Result: The **`status`** of the queue is changed to **'INACTIVE'** (**PASS**)

Evidence:



```
### Deactivate
//Update Queue Test Case 2 (Success - Deactivate Queue)
Send Request
PUT http://{{host}}{{path}}?queue_id=QUEUE12345 HTTP/1.1
Content-Type: application/json

{
    "status": "DEACTIVATE"
}
```

```
HTTP/1.1 200 OK
X-Powered-By: Express
Access-Control-Allow-Origin: *
Date: Tue, 24 Nov 2020 15:10:53 GMT
Connection: close
Content-Length: 0
```



Update Queue Test Case 3 (Error - Unknown Queue ID)

Title: Update Queue API - Unknown queue_id request prevents update queue status in database

Description: An error request made to the server should receive an error response and not update the queue **`status`** in the **`companyQueue`** table.

Precondition: **`companyQueue`** table created with 1 row of data **`queue_id`** **`QUEUE12345`** with **`status`** '**`INACTIVE`**'.

Test Steps:

1. Start the Backend Server
2. Send an Update Queue request to the backend with an **unknown `queue_id`** **`(queue_id=QUEUE12346)`** in the URL query and a **valid `status`** '**`DEACTIVATE`**' in the json body

```
PUT http://localhost:3000/company/queue?queue_id=QUEUE12346 HTTP/1.1

{
  "status": "DEACTIVATE"
}
```

3. Receive a **`404 Not Found`** response with an error response body

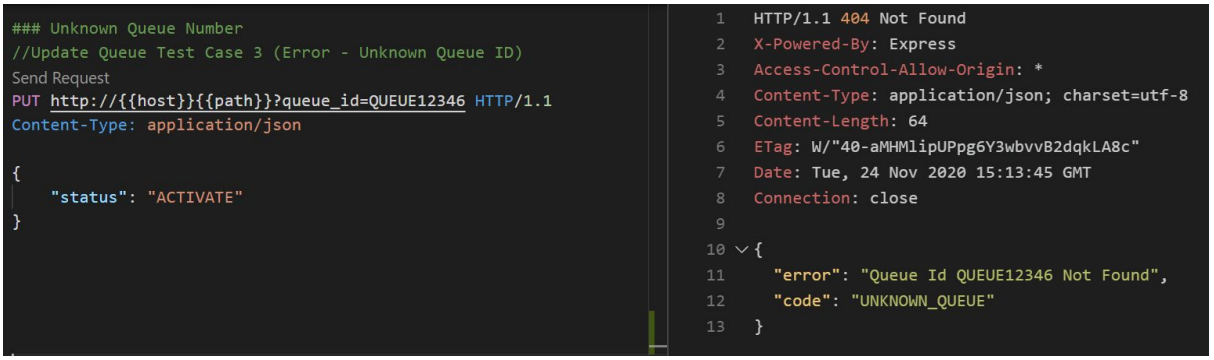
```
{
  "error": "Queue Id QUEUE12346 Not Found",
  "code": "UNKNOWN_QUEUE"
}
```

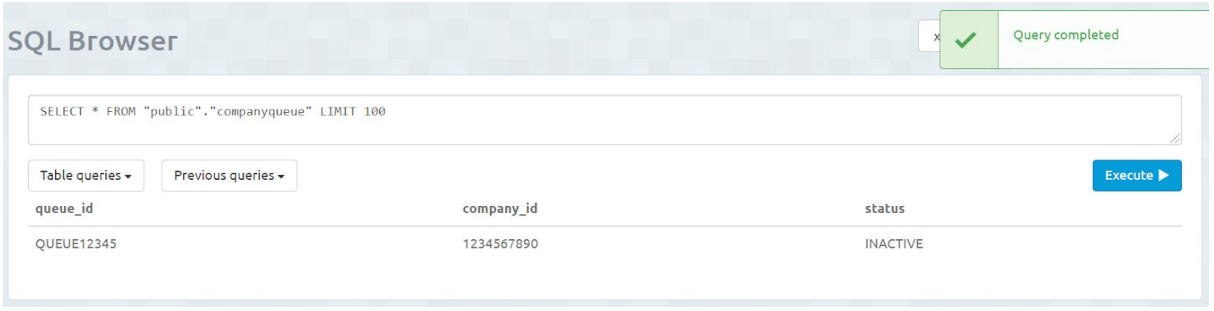
4. Go to elephantsql to inspect the **`companyQueue`** table

Expected Result: There is no change to the **`status`** in the companyTable, **`status`** remains '**`INACTIVE`**'

Actual Result: There is no change to the **`status`** in the companyTable, **`status`** remains '**`INACTIVE`**' (**PASS**)

Evidence:





Update Queue Test Case 4 (Error - Invalid Status)

Title: Update Queue API - Invalid status request prevents update queue status in database

Description: An error request made to the server should receive an error response and not update the queue **`status`** in the **`companyQueue`** table.

Precondition: **`companyQueue`** table created with 1 row of data **`queue_id`** **`QUEUE12345`** with **`status`** '**`INACTIVE`**'.

Test Steps:

1. Start the Backend Server
2. Send an Update Queue request to the backend with an **invalid `status`** '**`ACTI VATE`**' in the json body

```
{
  "status": "ACTI VATE"
}
```

3. Receive a **`400 Bad Request`** response with an error response body

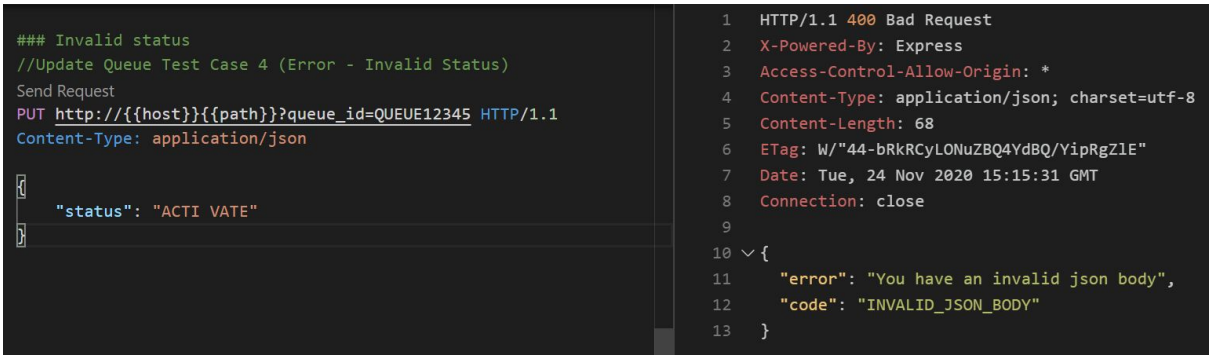
```
{
  "error": "You have an invalid json body",
  "code": "INVALID_JSON_BODY"
}
```

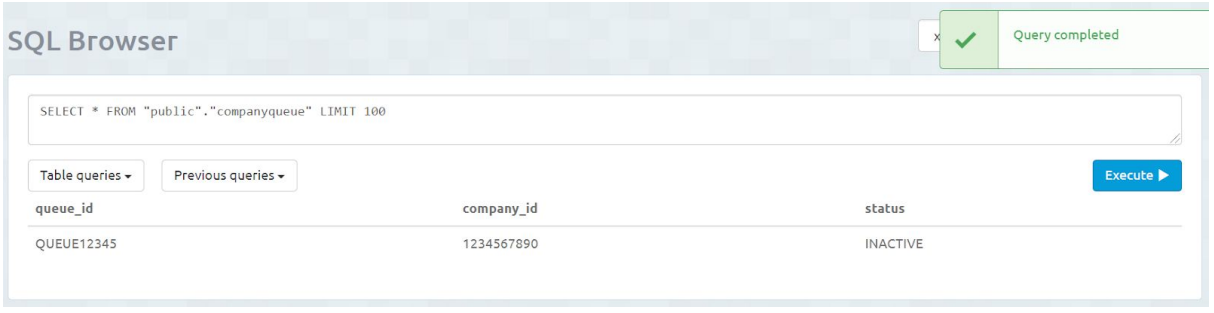
4. Go to elephantsql to inspect the **`companyQueue`** table

Expected Result: There is no change to the **`status`** in the companyTable, **`status`** remains '**`INACTIVE`**'

Actual Result: There is no change to the **`status`** in the companyTable, **`status`** remains '**`INACTIVE`**' (**PASS**)

Evidence:





Join Queue Test Case 1 (Success)

Title: Join Queue API - Successful request customer join queue in database

Description: A successful request made to the server should create a new entry in the **`customerQueue`** table with the correct parameters (**`customer_id = 1234567890, queue_id = QUEUE12345`**).

Precondition: **`companyQueue`** and **`customerQueue`** table created, 1 row with **`queue_id`** '**`QUEUE12345`**' and **`status`** '**`ACTIVE`**' in **`companyQueue`** table and there should be no rows existing in the **`customerQueue`**.

Test Steps:

1. Start the Backend Server
2. Send an Join Queue request to the backend with a **valid `customer_id`** and an **existing** and **valid `queue_id`** in the json body

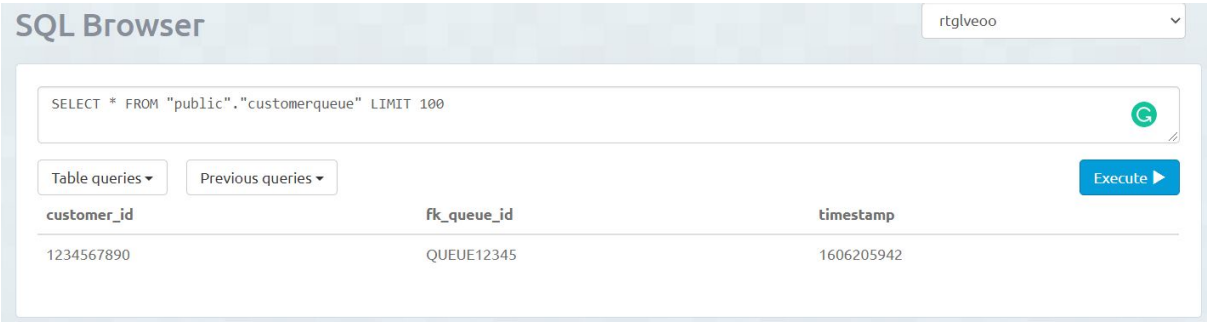
```
{
  "customer_id": 1234567890,
  "queue_id": "QUEUE12345"
}
```

3. Receive a **`201 Created`** response
4. Go to elephantsql to inspect the **`customerQueue`** table

Expected Result: A new row with the **`customer_id 1234567890`** and **`queue_id`** '**`QUEUE12345`**' is created

Actual Result: A new row with the **`customer_id 1234567890`** and **`queue_id`** '**`QUEUE12345`**' is created (**PASS**)

Evidence:



Join Queue Test Case 2 (Success - multiple people joining queue)

Title: Join Queue API - Successful request customer join queue in database

Description: A successful request made to the server should create a new entry in the **`customerQueue`** table with the correct parameters (**`customer_id = 9999999999, queue_id = QUEUE12345`**).

Precondition: **`companyQueue`** and **`customerQueue`** table created, 1 row with **`queue_id`** '**`QUEUE12345`**' and **`status`** '**`ACTIVE`**' in **`companyQueue`** table and there should be 1 existing **`customer_id 1234567890 in queue_id`** '**`QUEUE12345`**' in the **`customerQueue`**.

Test Steps:

1. Start the Backend Server
2. Send an Join Queue request to the backend with a **valid `customer_id`** and an **existing** and **valid `queue_id`** in the json body

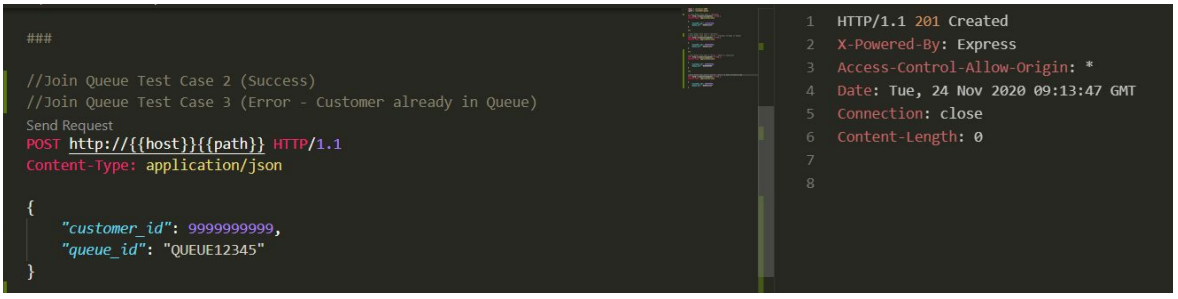
```
{
  "customer_id": 9999999999,
  "queue_id": "QUEUE12345"
}
```

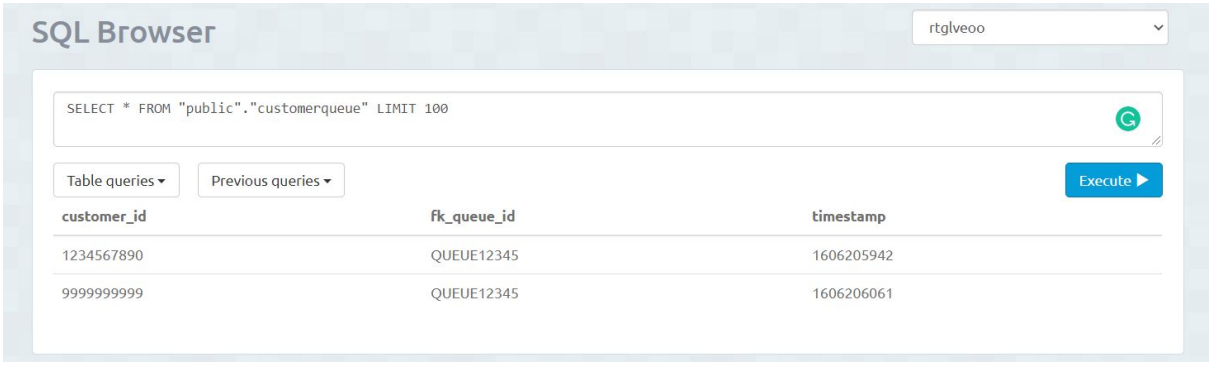
3. Receive a **`201 Created`** response
4. Go to elephantsql to inspect the **`customerQueue`** table

Expected Result: A new row with the **`customer_id 9999999999`** and **`queue_id`** '**`QUEUE12345`**' is created

Actual Result: A new row with the **`customer_id 9999999999`** and **`queue_id`** '**`QUEUE12345`**' is created (**PASS**)

Evidence:





Join Queue Test Case 3 (Error - Customer already in Queue)

Title: Join Queue API - Error request customer join queue in database

Description: An unsuccessful request with the customer joining the same queue is made to the server, an error message with the status code 422 will be generated.

Precondition: **`companyQueue`** and **`customerQueue`** table created, 1 row with **`queue_id`** '**`QUEUE12345`**' and **`status`** '**`ACTIVE`**' in **`companyQueue`** table and **`customer_id 1234567890`** and **`9999999999`** should already exist in the **`queue_id "QUEUE12345"`** in the **`customerQueue`**

Test Steps:

1. Start the Backend Server
2. Send an Join Queue request to the backend with an **existing `customer_id`** and an **existing** and **valid `queue_id`** in the json body

```
{
  "customer_id": 9999999999,
  "queue_id": "QUEUE12345"
}
```

3. Receive a **`422 Unprocessable Entity`** response with an error response body

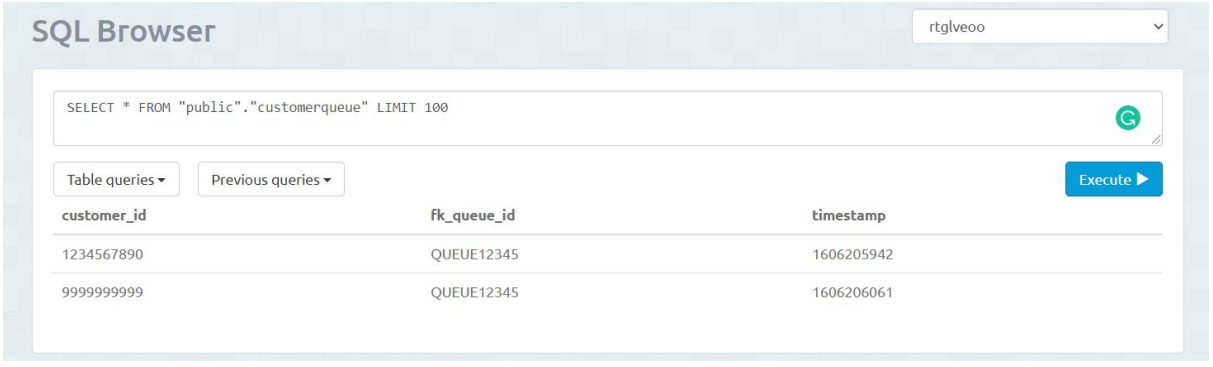
```
{
  "error": "Customer 9999999999 already in Queue QUEUE12345",
  "code": "ALREADY_IN_QUEUE"
}
```

4. Go to elephantsql to inspect the **`customerQueue`** table

Expected Result: Nothing is inserted into the **`customerQueue`** table

Actual Result: Nothing is inserted into the **`customerQueue`** table (**PASS**)

Evidence:



Join Queue Test Case 4 (Error - Queue is Inactive)

Title: Join Queue API - Error request customer join queue in database

Description: An unsuccessful request with an inactive queue is made to the server, an error message with the status 422 will be generated.

Precondition: **`companyQueue`** and **`customerQueue`** table created, 1 row with **`queue_id`** '**`QUEUE12345`**' and **`status`** '**`INACTIVE`**' in **`companyQueue`** table and no existing data in **`customerQueue`** table

Test Steps:

1. Start the Backend Server
2. Send an Join Queue request to the backend with a **valid `customer_id`** and an **existing** and **valid `queue_id`** which is **`inactive`** in the json body

```
{
  "customer_id": 9876543212,
  "queue_id": "QUEUE12345"
}
```

3. Receive a **`422 Unprocessable Entity`** response with an error response body

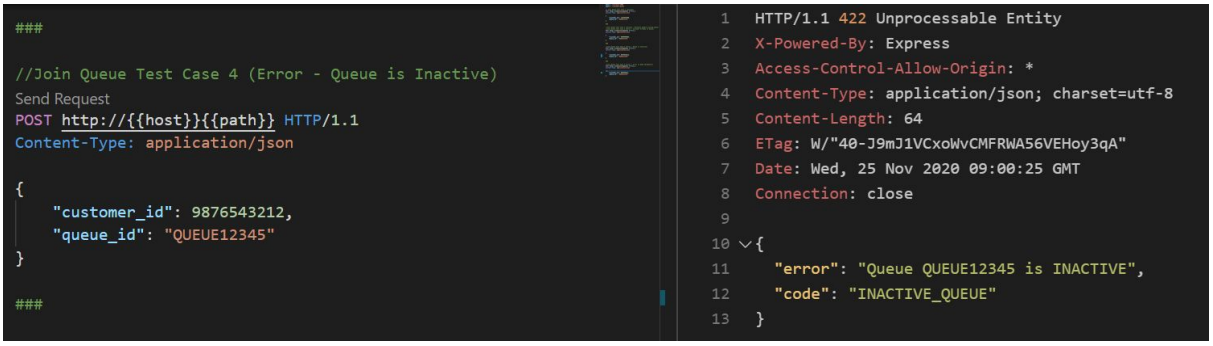
```
{
  "error": "Queue QUEUE12345 is INACTIVE",
  "code": "INACTIVE_QUEUE"
}
```

4. Go to elephantsql to inspect the **`customerQueue`** table

Expected Result: Nothing is inserted into the **`customerQueue`** table

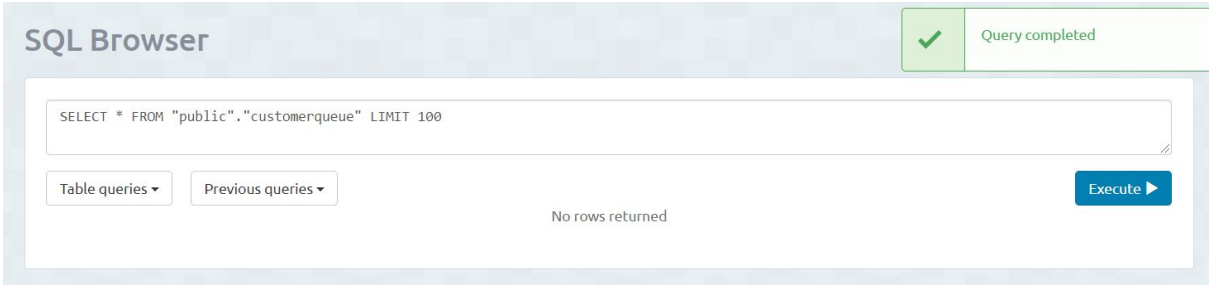
Actual Result: Nothing is inserted into the **`customerQueue`** table (**PASS**)

Evidence:



```
###

//Join Queue Test Case 4 (Error - Queue is Inactive)
Send Request
POST http://{{host}}{{path}} HTTP/1.1
Content-Type: application/json

{
    "customer_id": 9876543212,
    "queue_id": "QUEUE12345"
}

###
```

```
HTTP/1.1 422 Unprocessable Entity
X-Powered-By: Express
Access-Control-Allow-Origin: *
Content-Type: application/json; charset=utf-8
Content-Length: 64
ETag: W/"40-J9mJ1VCxoWvCMFRWA56VEHoy3qA"
Date: Wed, 25 Nov 2020 09:00:25 GMT
Connection: close

{
  "error": "Queue QUEUE12345 is INACTIVE",
  "code": "INACTIVE_QUEUE"
}
```



Join Queue Test Case 5 (Error - Error in Body Parameters)

Title: Join Queue API - Error request customer join queue in database

Description: An unsuccessful request with invalid **`customer_id`** is made to the server which generates an error message with the status code 400.

Precondition: **`companyQueue`** and **`customerQueue`** table created, 1 row with **`queue_id`** '**`QUEUE12345`**' and **`status`** '**`ACTIVE`**' in **`companyQueue`** table and **no** existing data in the **`customerQueue`** table

Test Steps:

1. Start the Backend Server
2. Send an Join Queue request to the backend with an **invalid `customer_id`** with less than 10 digits and an **existing** and **valid `queue_id`** in the json body

```
{
  "customer_id": 987654321,
  "queue_id": "QUEUE12345"
}
```

3. Receive a **`400 Bad Request`** response with an error response body

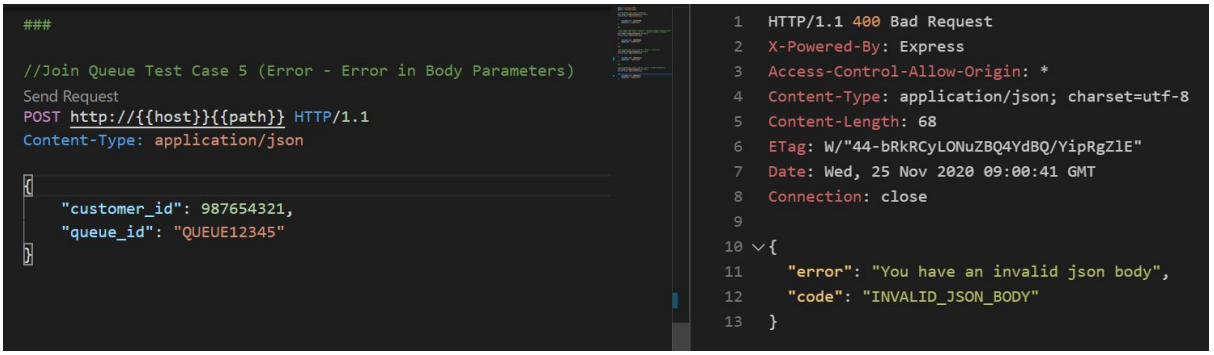
```
{
 "error": "You have an invalid json body",
 "code": "INVALID_JSON_BODY"
}
```

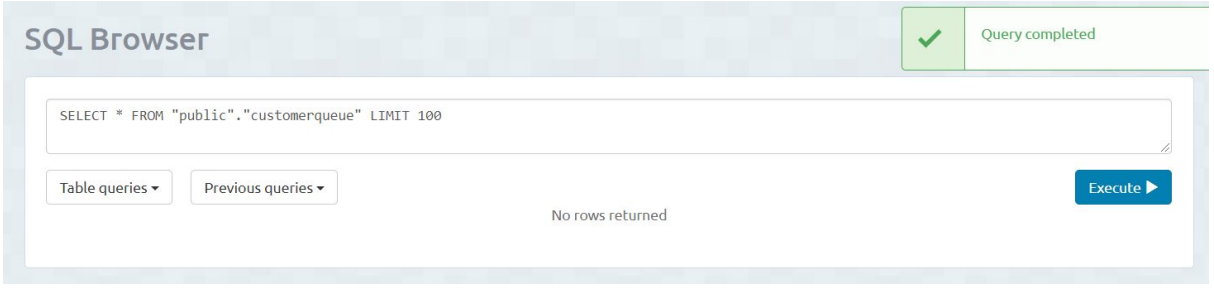
4. Go to elephantsql to inspect the **`customerQueue`** table

Expected Result: Nothing is inserted into the **`customerQueue`** table

Actual Result: Nothing is inserted into the **`customerQueue`** table (**PASS**)

Evidence:





Join Queue Test Case 6 (Error - Non-existence Queue Id)

Title: Join Queue API - Error request customer join queue in database

Description: An unsuccessful request with an non-existent queue is made to the server, generating an error message with the status code 400.

Precondition: **`companyQueue`** and **`customerQueue`** table created, 1 row with **`queue_id`** '**`QUEUE12345`**' and **`status`** '**`ACTIVE`**' in **`companyQueue`** table and no existing data in the **`customerQueue`** table

Test Steps:

1. Start the Backend Server
2. Send an Join Queue request to the backend with a **valid `customer_id`** but a **non-existent** and **valid `queue_id`** in the json body

```
{
  "customer_id": 999999999,
  "queue_id": "0000000000"
}
```

3. Receive a **`404 Not Found`** response with an error response body

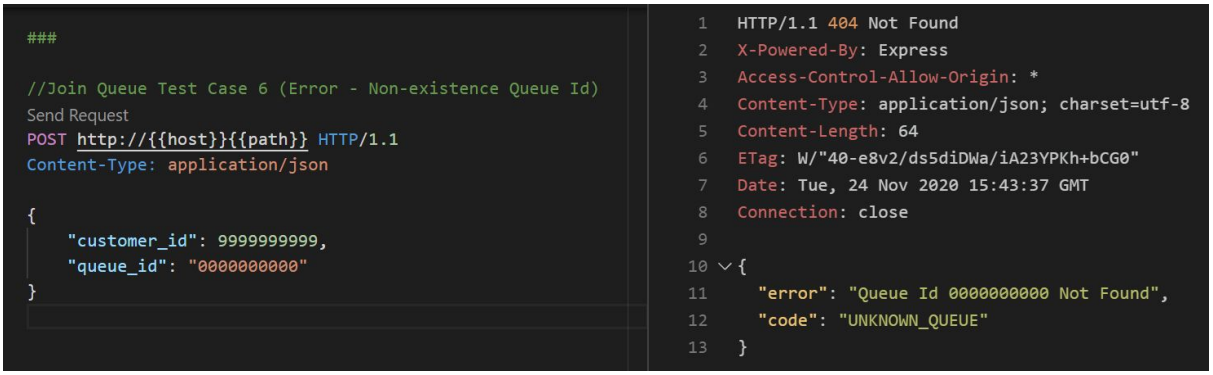
```
{
  "error": "Queue Id 0000000000 Not Found",
  "code": "UNKNOWN_QUEUE"
}
```

4. Go to elephantsql to inspect the **`customerQueue`** table

Expected Result: Nothing is inserted into the **`customerQueue`** table

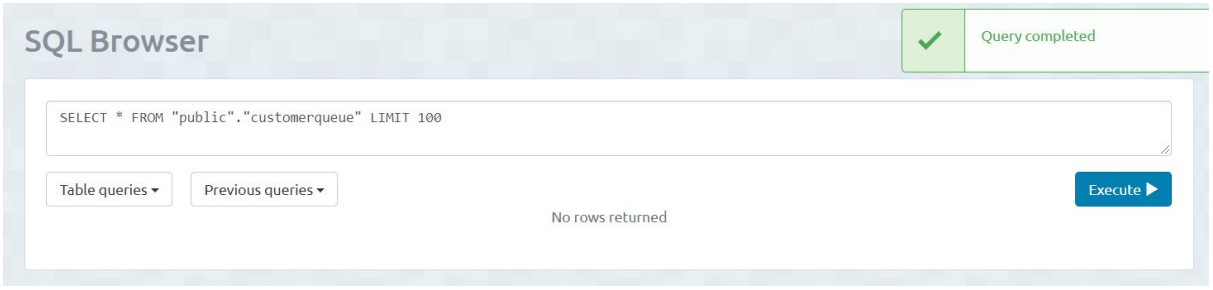
Actual Result: Nothing is inserted into the **`customerQueue`** table (**PASS**)

Evidence:



```
###

//Join Queue Test Case 6 (Error - Non-existence Queue Id)
Send Request
POST http://{{host}}{{path}} HTTP/1.1
Content-Type: application/json

{
    "customer_id": 9999999999,
    "queue_id": "0000000000"
}
```

```
HTTP/1.1 404 Not Found
X-Powered-By: Express
Access-Control-Allow-Origin: *
Content-Type: application/json; charset=utf-8
Content-Length: 64
ETag: W/"40-e8v2/ds5diDWa/iA23YPKh+bCG0"
Date: Tue, 24 Nov 2020 15:43:37 GMT
Connection: close

{
  "error": "Queue Id 0000000000 Not Found",
  "code": "UNKNOWN_QUEUE"
}
```



Server Available Test Case 1 (Success)

Title: Server Available API - Successful request returns the customer id of the next customer and updates the **`customerQueue`** table

Description: A successful request made to the server should check if the queue exists, return the first **`customer id`** of the queue, and update the **`customerQueue`** table.

Precondition: **`companyQueue`** and **`customerQueue`** table created, 1 row with **`queue_id`** '**`QUEUE12345`**' and and **`status`** '**`ACTIVE`**' in **`companyQueue`** table and 1 row of data with **`queue_id`** '**`QUEUE12345`**' and **`customer_id 1234567890`** in **`customerQueue`** table.

Test Steps:

1. Start the Backend Server
2. Send a Server Available request to the backend with a **valid `queue_id`**

```
{
 "queue_id": "QUEUE12345"
}
```

3. Receive a **`200 OK`** response with a response body

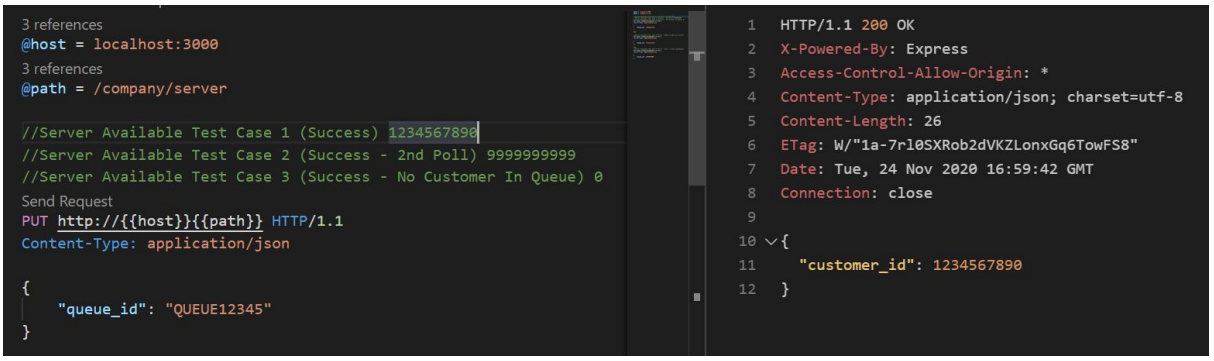
```
{
 "customer_id": 1234567890
}
```

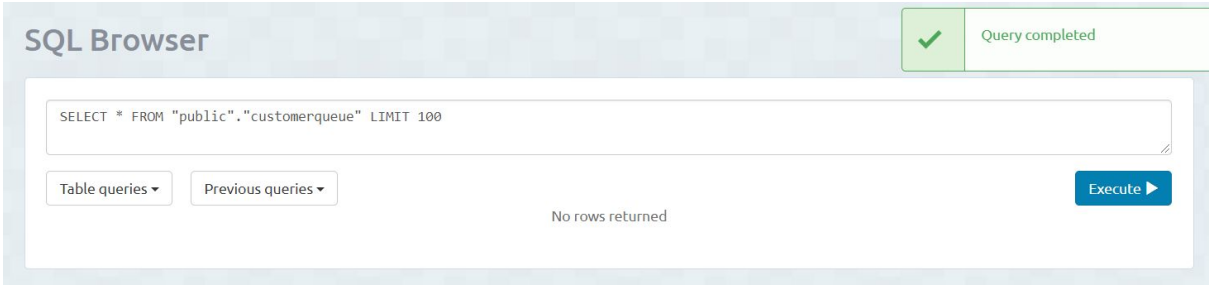
4. Go to elephantsql to inspect the **`customerQueue`** table

Expected Result: There is no data existing in the **`customerQueue`** table

Actual Result: There is no data existing in the **`customerQueue`** table (**PASS**)

Evidence:





Server Available Test Case 2 (Success - 2nd Poll)

Title: Server Available API - Successful request returns the customer id of the next customer and updates the **customerQueue** table

Description: A successful request made to the server should check if the queue exists, return the next **customer id** of the queue, and update the **customerQueue** table.

Precondition: **companyQueue** and **customerQueue** table created, 1 row with **queue_id** '**QUEUE12345**' and **status** '**ACTIVE**' in **companyQueue** table and 2 rows of data with the same **queue_id** '**QUEUE12345**' with different **customer_id 1234567890** and **customer_id 9999999999** in **customerQueue** table.

Test Steps:

1. Start the Backend Server
2. Send the Server Available request **twice** to the backend with a **valid queue_id**

```
{
 "queue_id": "QUEUE12345"
}
```

3. Receive a **200 OK** response with a response body

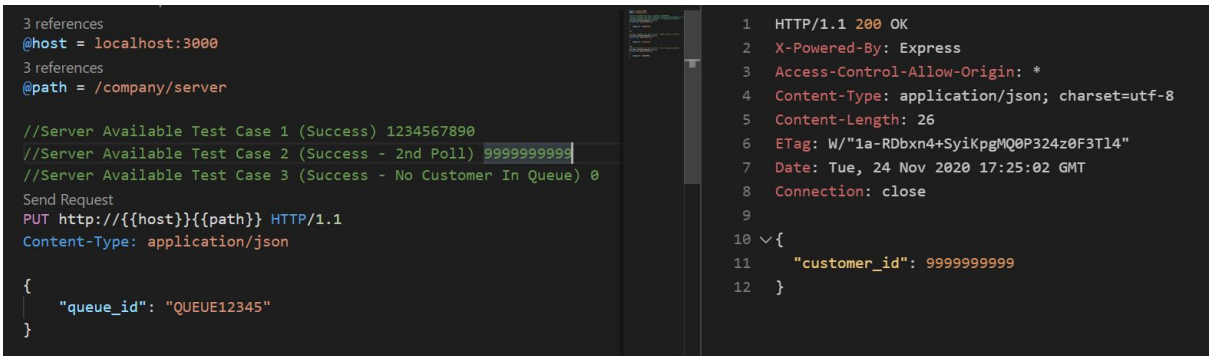
```
{
 "customer_id": 9999999999
}
```

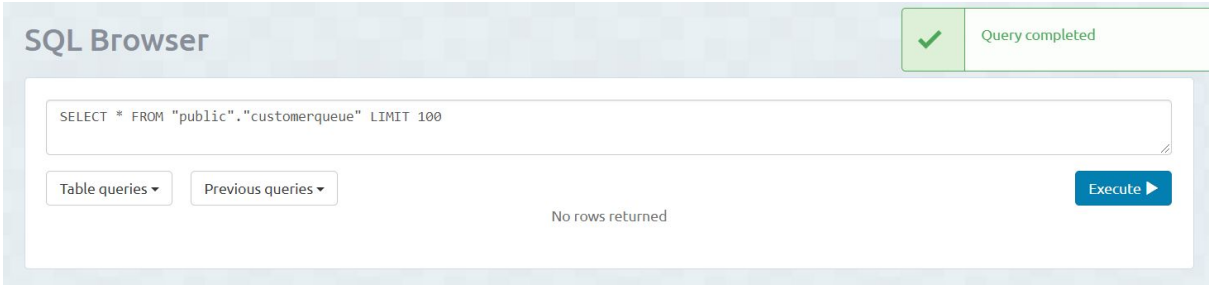
4. Go to elephantsql to inspect the **customerQueue** table

Expected Result: There is no data existing in the **customerQueue** table

Actual Result: There is no data existing in the **`customerQueue`** table (**PASS**)

Evidence:





Server Available Test Case 3 (Success - No Customer In Queue)

Title: Server Available API - Successful request returns 0 customer id

Description: A successful request made to the server should check if the queue exists, return **`customer id`** 0 of the queue when there is no customer in the queue.

Precondition: **`companyQueue`** and **`customerQueue`** table created, 1 row with **`queue_id`** '**`QUEUE12345`**' and **`status`** '**`ACTIVE`**' in **`companyQueue`** table and no data in **`customerQueue`** table.

Test Steps:

1. Start the Backend Server
2. Send the Server Available request to the backend with a **valid `queue_id`**

```
{
 "queue_id": "QUEUE12345"
}
```

3. Receive a **`200 OK`** response with a response body

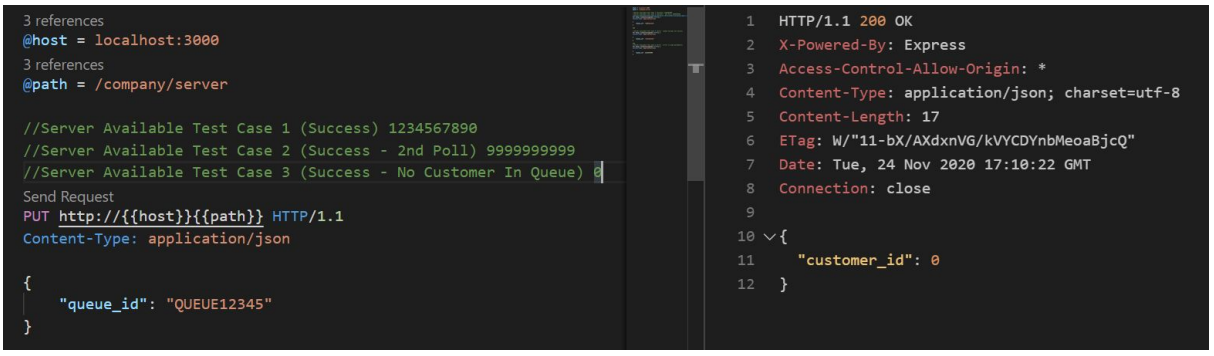
```
{
 "customer_id": 0
}
```

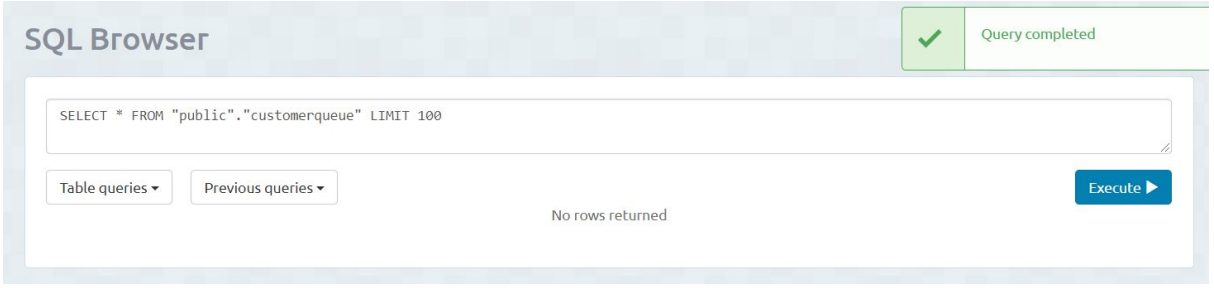
4. Go to elephantsql to inspect the **`customerQueue`** table

Expected Result: There is no data existing in the **`customerQueue`** table

Actual Result: There is no data existing in the **`customerQueue`** table (**PASS**)

Evidence:





Server Available Test Case 4 (Error - Queue Id does not exist)

Title: Server Available API - Non-existent queue id in request prevents selection of the customer id.

Description: An unsuccessful request made to the server should generate an error message with the status code 404 and not update the **`customerQueue`** table.

Precondition: **`companyQueue`** and **`customerQueue`** table created, 1 row with **`queue_id`** '**`QUEUE12345`**' and **`status`** '**`ACTIVE`**' in **`companyQueue`** table and no data in **`customerQueue`** table.

Test Steps:

1. Start the Backend Server
2. Send a Server Available request to the backend with a **non-existent** and **valid `queue_id`**

```
{
 "queue_id": "2222222222"
}
```

3. Receive a **`404 Not Found`** response with an error response body

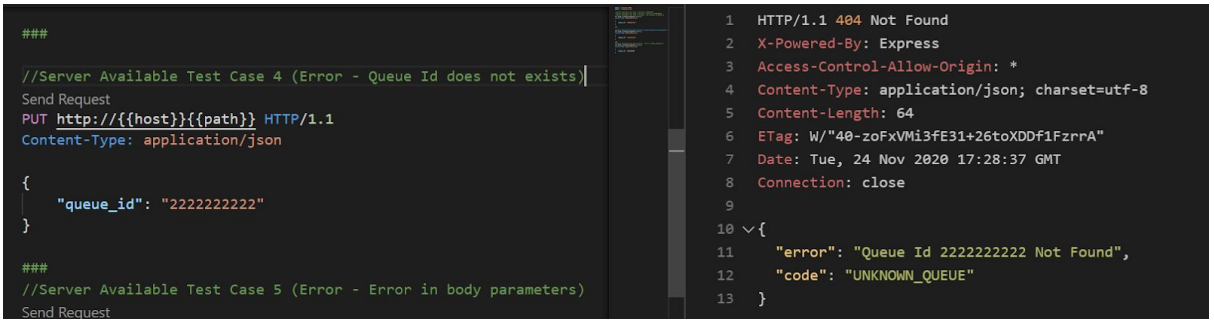
```
{
 "error": "Queue Id 2222222222 Not Found",
 "code": "UNKNOWN_QUEUE"
}
```

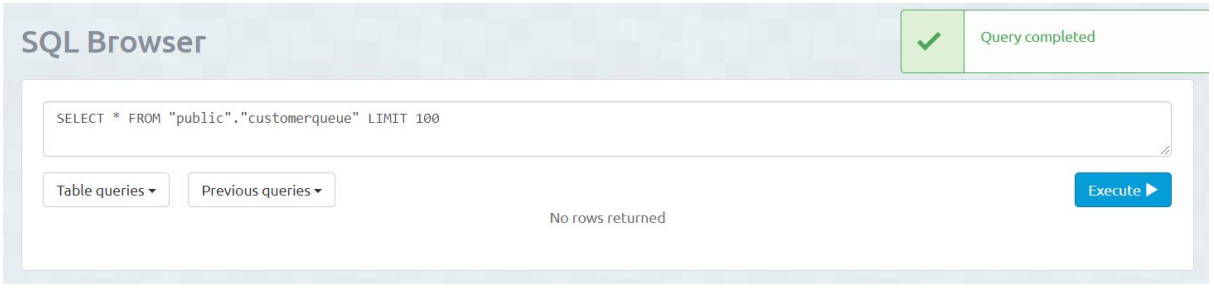
4. Go to ElephantSQL to inspect the **`customerQueue`** table

Expected Result: No update in the **`customerQueue`** table, there are still 0 rows

Actual Result: No update in the **`customerQueue`** table, there are still 0 rows (**PASS**)

Evidence:





Server Available Test Case 5 (Error - Error in body parameters)

Title: Server Available API - Invalid queue id in request prevents selection of the customer id.

Description: An unsuccessful request made to the server should generate an error message with the status code 400 and not update the **`customerQueue`** table.

Precondition: **`companyQueue`** and **`customerQueue`** table created, 1 row with **`queue_id`** '**`QUEUE12345`**' and **`status`** '**`ACTIVE`**' in **`companyQueue`** table and 1 rows of data with **`queue_id`** '**`QUEUE12345`**' and **`customer_id 1234567890`** in **`customerQueue`** table.

Test Steps:

1. Start the Backend Server
2. Send a Server Available request to the backend with an **invalid `queue_id`**

```
{
 "queue_id": 1234567890
}
```

3. Receive a **`400 Bad Request`** response with an error response body

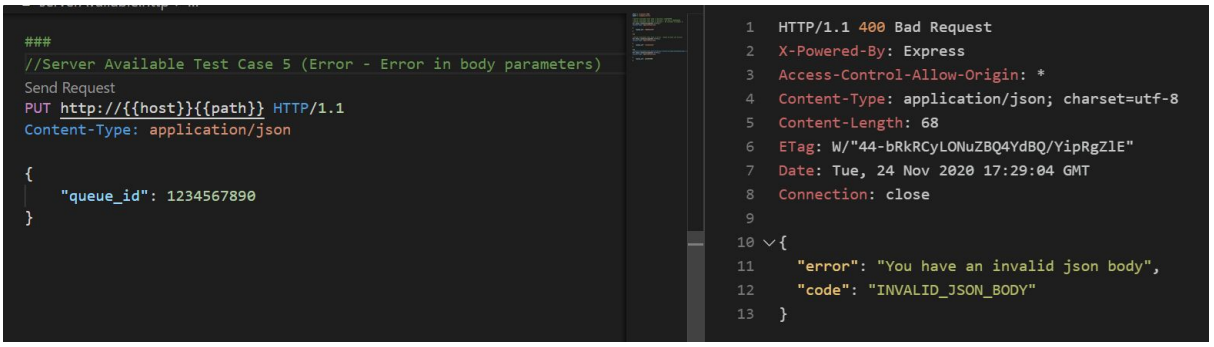
```
{
 "error": "You have an invalid json body",
 "code": "INVALID_JSON_BODY"
}
```

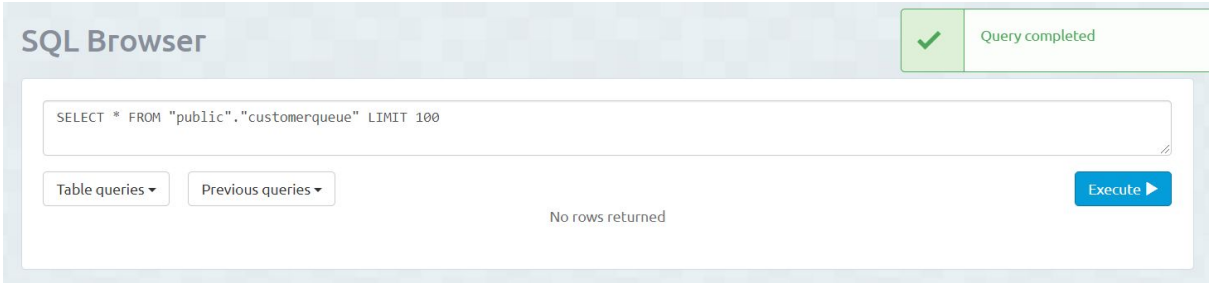
4. Go to ElephantSQL to inspect the **`customerQueue`** table

Expected Result: No update in the **`customerQueue`** table, there are still 0 rows

Actual Result: No update in the **`customerQueue`** table, there are still 0 rows (**PASS**)

Evidence:



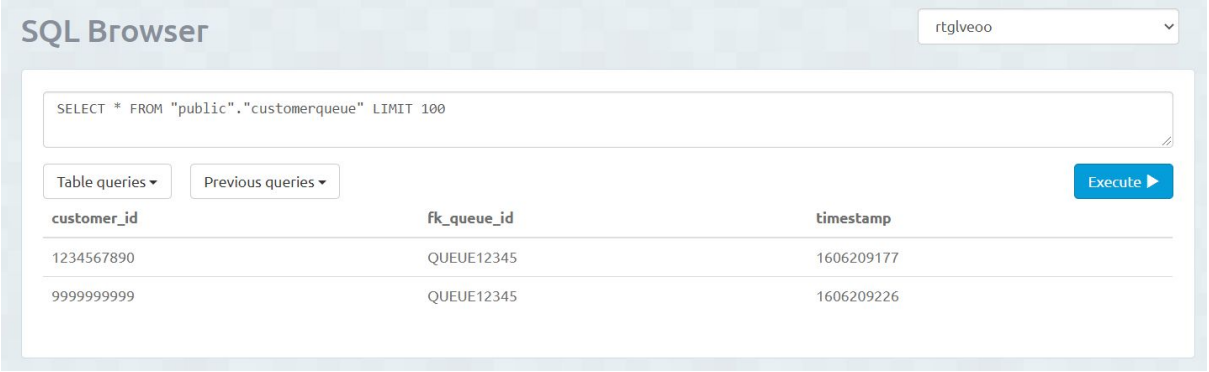


Check Queue Test Case 1 (Success)

Title: Check Queue API - Successful request customer check queue in database

Description: A successful request made to the server should generate the number of customers in a queue. When there are customers ahead in the queue, **`ahead`** will return the number of customers ahead.

Precondition: **`companyQueue`** and **`customerQueue`** table created, 1 row with **`queue_id`** '**`QUEUE12345`**' and **`status`** '**`ACTIVE`**' in **`companyQueue`** table and 2 rows of data with the same **`queue_id`** '**`QUEUE12345`**' with different **`customer_id 1234567890`** and **`customer_id 9999999999`** in **`customerQueue`** table.



Test Steps:

1. Start the Backend Server
2. Send a Check Queue request to the backend with a **valid `customer_id`** and an **existing** and **valid `queue_id`** in the url params.

```
GET
http://{{host}}{{path}}?queue_id=QUEUE12345&customer_id=9999999999
HTTP/1.1
```

3. Receive a **`200 OK`** response and a response body:

```
{
 "total": 2,
 "ahead": 1,
 "status": "ACTIVE"
}
```

Expected Result: Receive a **`200 OK`** response with the **`total`** number of customers (**2**), the number of customers **`ahead`** (**1**) and **`status`** (**"ACTIVE"**) in the response body.

Actual Result: Receive a **`200 OK`** response with the **`total`** number of customers (**`2`**), the number of customers **`ahead`** (**`1`**) and **`status`** (**`"ACTIVE"`**) in the response body. (**PASS**)

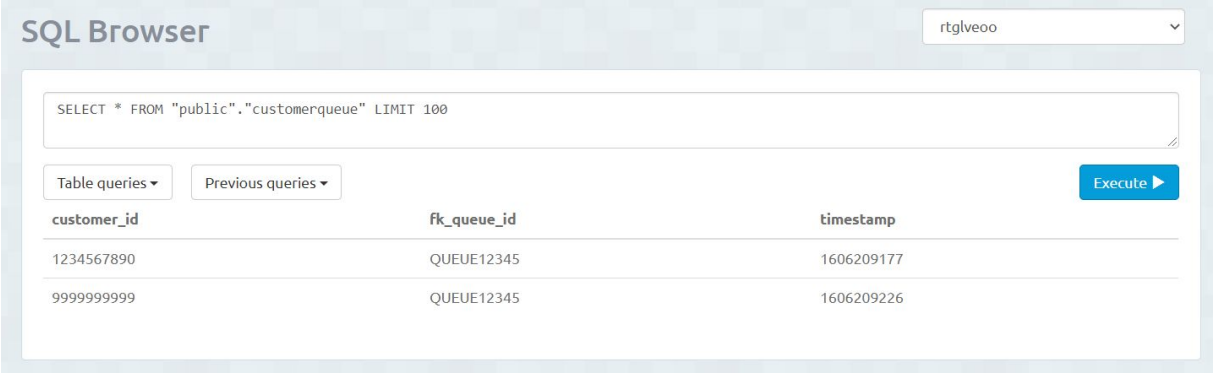
Evidence:

Check Queue Test Case 2 (Success - Next to be assigned to server)

Title: Check Queue API - Successful request customer check queue in database

Description: A successful request made to the server should generate the number of customers in a queue. When there is no customer ahead in the queue, it'll return **`ahead: 0`**.

Precondition: **`companyQueue`** and **`customerQueue`** table created, 1 row with **`queue_id`** '**`QUEUE12345`**' and **`status`** '**`ACTIVE`**' in **`companyQueue`** table and 2 rows of data with the same **`queue_id`** '**`QUEUE12345`**' with different **`customer_id 1234567890`** and **`customer_id 9999999999`** in **`customerQueue`** table.



Test Steps:

1. Start the Backend Server
2. Send a Check Queue request to the backend with a **valid `customer_id`** and an **existing** and **valid `queue_id`** in the url params.

```
GET
http://{{host}}{{path}}?queue_id=QUEUE12345&customer_id=1234567890
HTTP/1.1
```

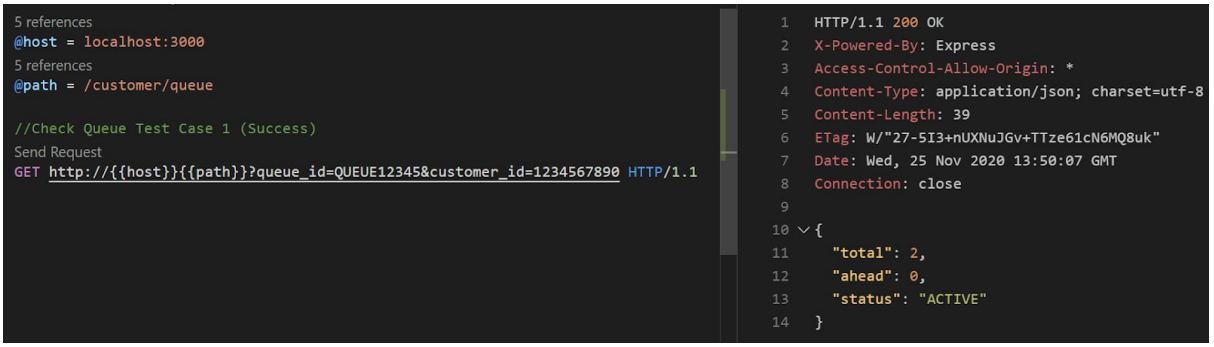
3. Receive a **`200 OK`** response and a response body:

```
{
 "total": 2,
 "ahead": 0,
 "status": "ACTIVE"
}
```

Expected Result: Receive a **`200 OK`** response with the **`total`** number of customers (**2**), the number of customers **`ahead`** (**0**) and **`status`** (**"ACTIVE"**) in the response body

Actual Result: Receive a **`200 OK`** response with the **`total`** number of customers (**`2`**), the number of customers **`ahead`** (**`0`**) and **`status`** (**`“ACTIVE”`**) in the response body (**PASS**)

Evidence:



Check Queue Test Case 3 (Success - Customer not in queue(Never Joined/Missed) /Customer Id not provided)

Title: Check Queue API - Successful request customer check queue in database

Description: A successful request made to the server should generate the number of customers in a queue. When there is no customer in queue or no **`customer id`** provided, it'll return **`ahead: -1`**.

Precondition: **`companyQueue`** and **`customerQueue`** table created, 1 row with **`queue_id`** '**`QUEUE12345`**' and **`status`** '**`ACTIVE`**' in **`companyQueue`** table and 2 rows of data with the same **`queue_id`** '**`QUEUE12345`**' with different **`customer_id 1234567890`** and **`customer_id 9999999999`** in **`customerQueue`** table.

Test Steps:

1. Start the Backend Server
2. Send a Check Queue request to the backend with only a **valid `queue_id`** and **no customer id** in the url params.

```
GET http://{{host}}{{path}}?queue_id=QUEUE12345 HTTP/1.1
```

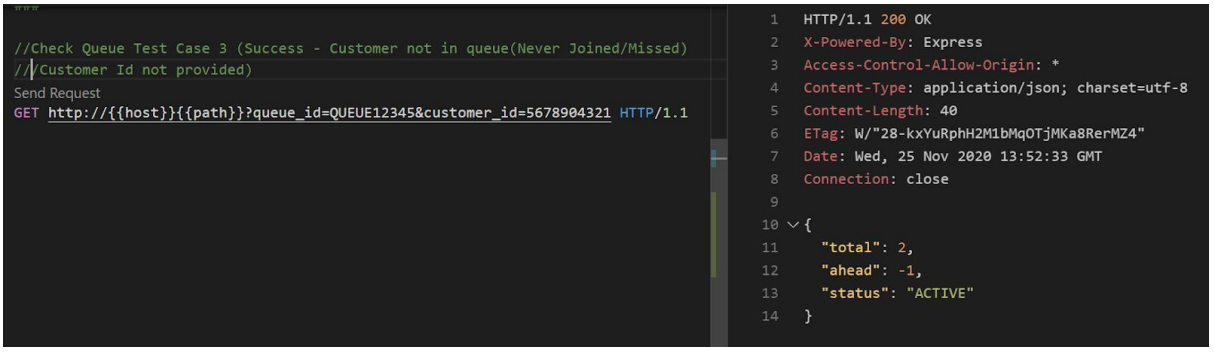
3. Receive a **`200 OK`** response and a response body:

```
{
 "total": 2,
 "ahead": -1,
 "status": "ACTIVE"
}
```

Expected Result: Receive a **`200 OK`** response with the **`total`** number of customers (**`2`**), the number of customers **`ahead`** (**`-1`**) and **`status`** (**"`ACTIVE`"**) in the response body

Actual Result: Receive a **`200 OK`** response with the **`total`** number of customers (**`2`**), the number of customers **`ahead`** (**`-1`**) and **`status`** (**“`ACTIVE`”**) in the response body (**PASS**)

Evidence:

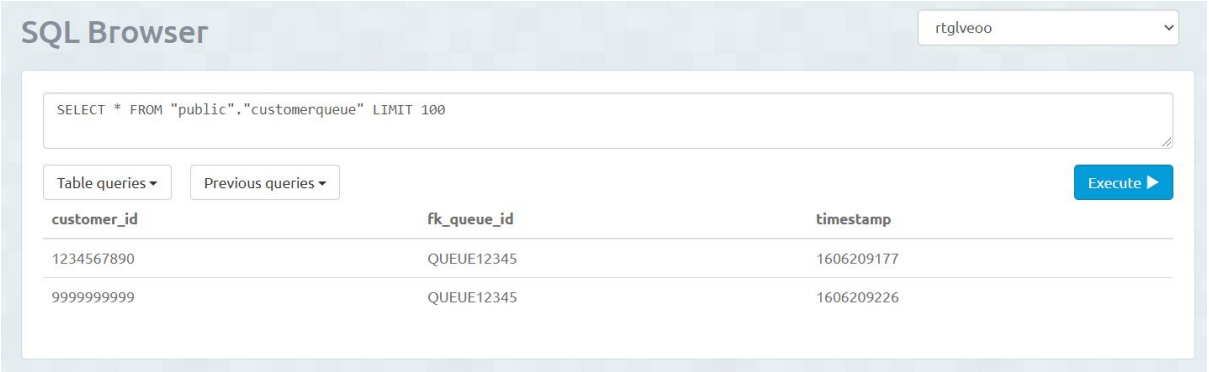


Check Queue Test Case 4 (Error - Non-existence Queue Id)

Title: Check Queue API - Unsuccessful request customer check queue in database

Description: An unsuccessful request made to the server should generate an error message with status code 404.

Precondition: **`companyQueue`** and **`customerQueue`** table created, 1 row with **`queue_id`** '**`QUEUE12345`**' and **`status`** '**`ACTIVE`**' in **`companyQueue`** table and 2 rows of data with the same **`queue_id`** '**`QUEUE12345`**' with different **`customer_id 1234567890`** and **`customer_id 9999999999`** in **`customerQueue`** table.



Test Steps:

1. Start the Backend Server
2. Send a Check Queue request to the backend with a **valid `customer_id`** and a **non-existent** and **valid `queue_id`** in the url params.

```
GET
http://{{host}}{{path}}?queue_id=1234567890&customer_id=1234567890
HTTP/1.1
```

3. Receive a **`404 Not Found`** response with an error response body

```
{
 "error": "Queue Id 1234567890 Not Found",
 "code": "UNKNOWN_QUEUE"
}
```

Expected Result: Receive a **`404 Not Found`** response with the error message and error code ("UNKNOWN_QUEUE") in the response body

Actual Result: Receive a **`404 Not Found`** response with the error message and error code (“UNKNOWN_QUEUE”) in the response body (**PASS**)

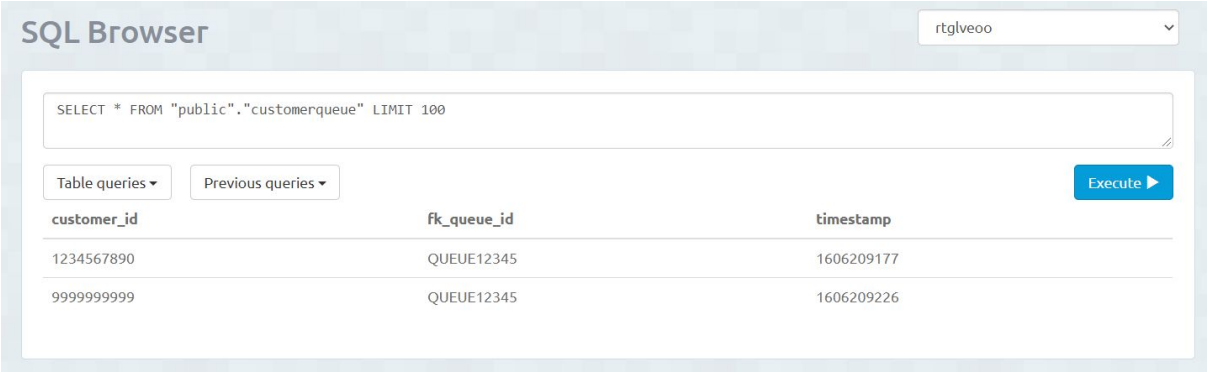
Evidence:

Check Queue Test Case 5 (Error - Error in query parameters)

Title: Check Queue API - Unsuccessful request customer check queue in database

Description: An unsuccessful request made to the server should generate an error message with status code 400.

Precondition: **`companyQueue`** and **`customerQueue`** table created, 1 row with **`queue_id`** '**`QUEUE12345`**' and **`status`** '**`ACTIVE`**' in **`companyQueue`** table and 2 rows of data with the same **`queue_id`** '**`QUEUE12345`**' with different **`customer_id 1234567890`** and **`customer_id 9999999999`** in **`customerQueue`** table.



Test Steps:

1. Start the Backend Server
2. Send a Check Queue request to the backend with an **invalid `customer_id`** with less than 10 digits and an **existing** and **valid `queue_id`** in the url params.

```
GET
http://{{host}}{{path}}?queue_id=QUEUE12345&customer_id=123456789
HTTP/1.1
```

3. Receive a **`400 Bad Request`** response with an error response body

```
{
 "error": "You have an invalid query",
 "code": "INVALID_QUERY_STRING"
}
```

Expected Result: Receive a **`400 Bad Request`** response with the error message and error code ("INVALID_QUERY_STRING") in the response body

Actual Result: Receive a **`400 Bad Request`** response with the error message and error code (“INVALID_QUERY_STRING”) in the response body (**PASS**)

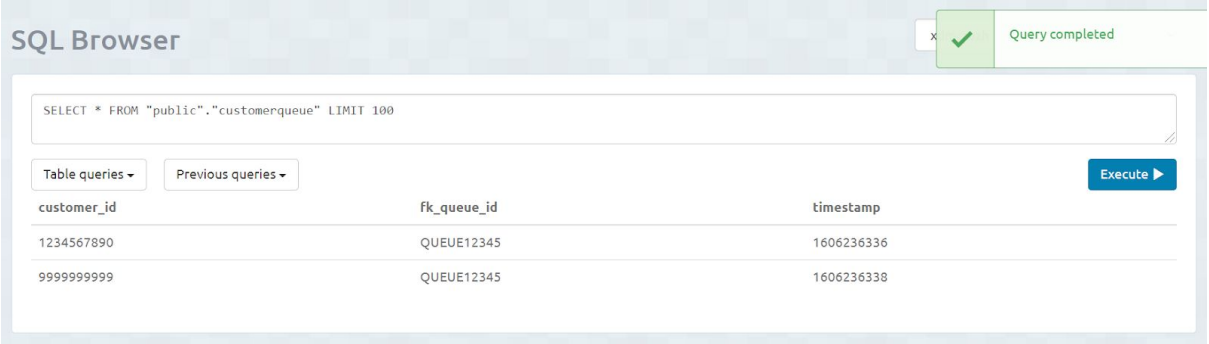
Evidence:

Arrival Rate Test Case 1 (Success)

Title: Arrival Rate API - Successful request to get the timestamp and number of customers from database

Description: A successful request made to the server should return a response body with timestamp and number of customers.

Precondition: **companyQueue** and **customerQueue** table created, 1 row with **queue_id** '**QUEUE12345**' and **status** '**ACTIVE**' in **companyQueue** table and 2 rows of data with the same **queue_id** '**QUEUE12345**' with different **customer_id 1234567890** and **customer_id 9999999999** in **customerQueue** table.



Test Steps:

1. Start the Backend Server
2. Send an Arrival Rate request to the backend with a **valid queue_id**, **from** (starting time) and **duration** in the URL query

```
GET
http://localhost:3000/company/arrival_rate?queue_id=queue12345
&from=2020-11-24T21%3A11%3A00%2B08%3A00&duration=1 HTTP/1.1
```

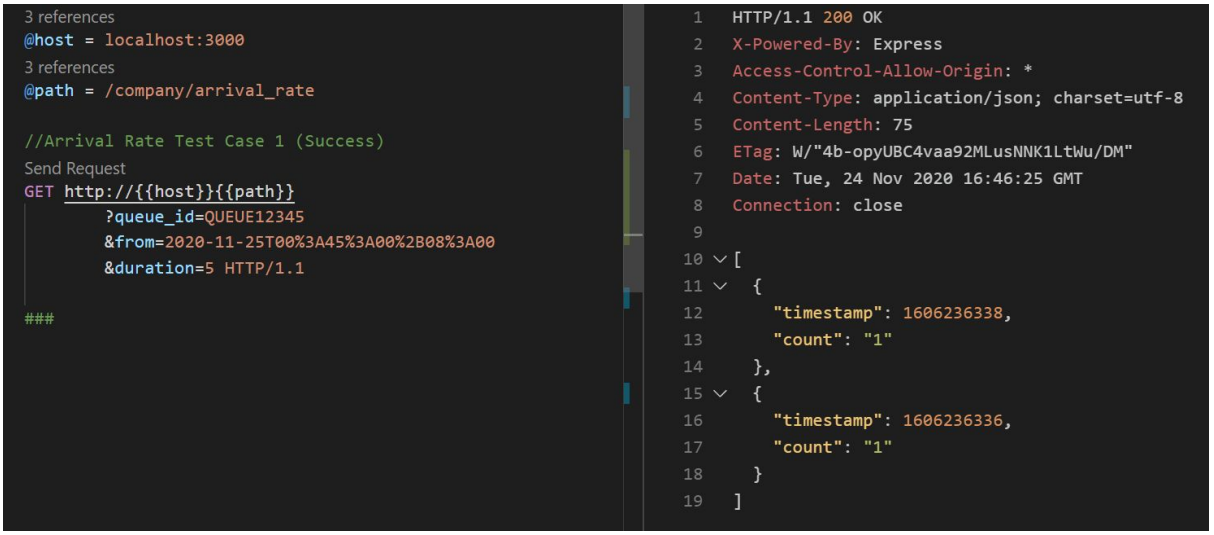
3. Receive a **200 OK** response with a response body

```
[
      {
            "timestamp": 1606236338,
            "count": "1"
      },
      {
            "timestamp": 1606236336,
            "count": "1"
      },
 ]
```

Expected Result: Receive a **`200 OK`** response with 2 objects in an array with **`timestamp`** and number of customers (**`count`**) in each object in the response body

Actual Result: Receive a **`200 OK`** response with 2 objects in an array with **`timestamp`** and number of customers (**`count`**) in each object in the response body (**PASS**)

Evidence:



```
3 references
@host = localhost:3000
3 references
@path = /company/arrival_rate

//Arrival Rate Test Case 1 (Success)
Send Request
GET http://{{host}}{{path}}
        ?queue_id=QUEUE12345
        &from=2020-11-25T00%3A45%3A00%2B08%3A00
        &duration=5 HTTP/1.1

###
```

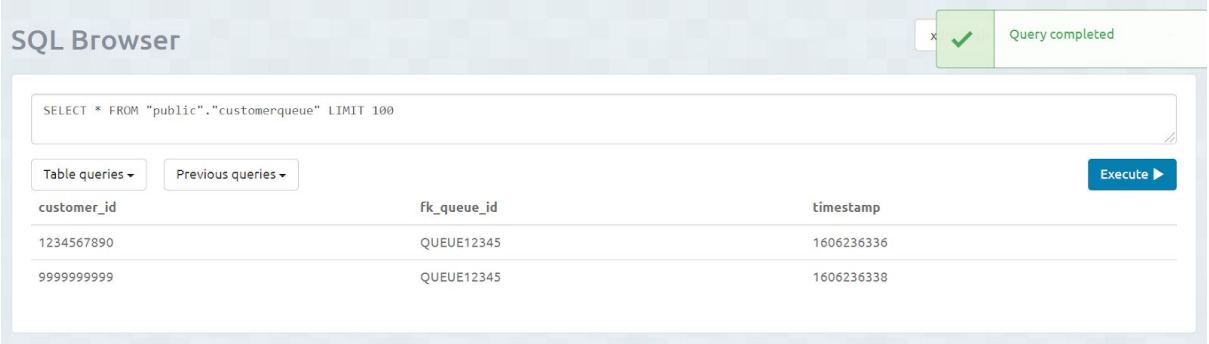
```
HTTP/1.1 200 OK
X-Powered-By: Express
Access-Control-Allow-Origin: *
Content-Type: application/json; charset=utf-8
Content-Length: 75
ETag: W/"4b-opyUBC4vaa92MLusNNK1LtWu/DM"
Date: Tue, 24 Nov 2020 16:46:25 GMT
Connection: close

[
  {
    "timestamp": 1606236338,
    "count": "1"
  },
  {
    "timestamp": 1606236336,
    "count": "1"
  }
]
```

Arrival Rate Test Case 2 (Error - Non-existence Queue Id)

Title: Arrival Rate API - Non-existence queue_id request prevents getting the timestamp and number of customers from database

Description: An unsuccessful request made to the server should generate an error message with the status code 404.

Precondition: **`companyQueue`** and **`customerQueue`** table created, 1 row with **`queue_id`** '**`QUEUE12345`**' and **`status`** '**`ACTIVE`**' in **`companyQueue`** table and 2 rows of data with the same **`queue_id`** '**`QUEUE12345`**' with different **`customer_id 1234567890`** and **`customer_id 9999999999`** in **`customerQueue`** table.



Test Steps:

1. Start the Backend Server
2. Send an Arrival Rate request to the backend with a **non-existent** and **valid `queue_id`**, a **valid `from`** (starting time) and a **valid `duration`** in the URL query

```
GET
http://localhost:3000/company/arrival_rate?queue_id=queue12346
&from=2020-11-24T21%3A11%3A00%2B08%3A00&duration=1 HTTP/1.1
```

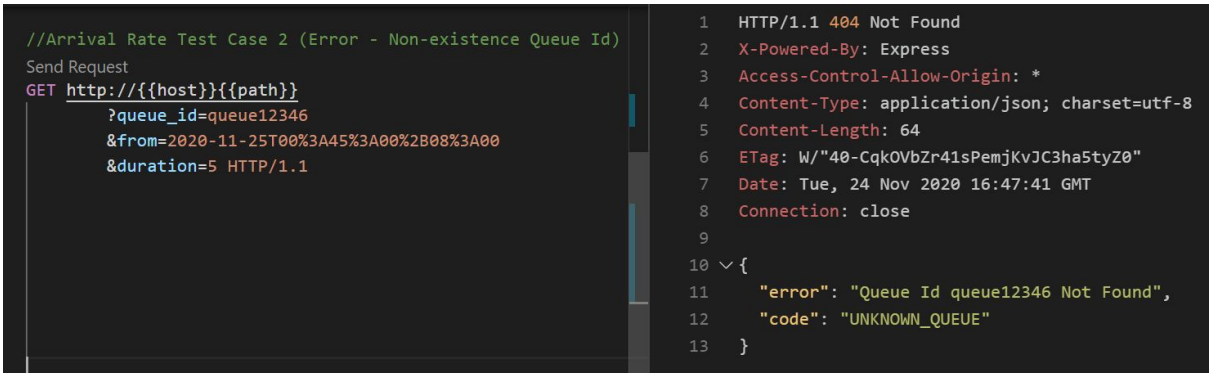
3. Receive a **`404 Not Found`** response with an error response body

```
{
  "error": "Queue Id queue12346 Not Found",
  "code": "UNKNOWN_QUEUE"
}
```

Expected Result: Receive a **`404 Not Found`** response with the error message and error code ("UNKNOWN_QUEUE") in the response body

Actual Result: Receive a **`404 Not Found`** response with the error message and error code (“UNKNOWN_QUEUE”) in the response body (**PASS**)

Evidence:

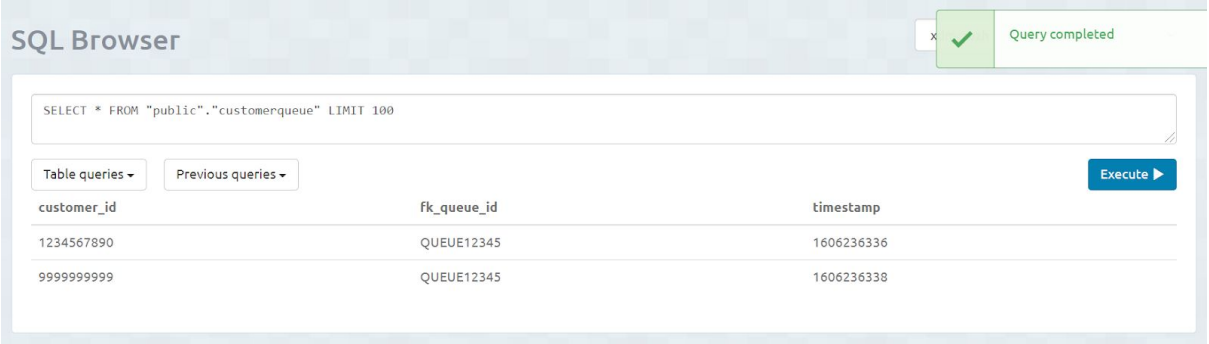


Arrival Rate Test Case 3 (Error - Error in query parameters)

Title: Arrival Rate API - Error in query parameters request prevents getting the timestamp and number of customers from database

Description: An unsuccessful request made to the server should generate an error message with the status code 400.

Precondition: **`companyQueue`** and **`customerQueue`** table created, 1 row with **`queue_id`** '**`QUEUE12345`**' and **`status`** '**`ACTIVE`**' in **`companyQueue`** table and 2 rows of data with the same **`queue_id`** '**`QUEUE12345`**' with different **`customer_id 1234567890`** and **`customer_id 9999999999`** in **`customerQueue`** table.



Test Steps:

1. Start the Backend Server
2. Send an Arrival Rate request to the backend with an **existing** and **valid** **`queue_id`**, an **invalid** **`from`** (starting time) and a **valid** **`duration`** in the URL query

```
GET
http://localhost:3000/company/arrival_rate?queue_id=queue12345
&from=2020-10-12T07:35:00Z&duration=5 HTTP/1.1
```

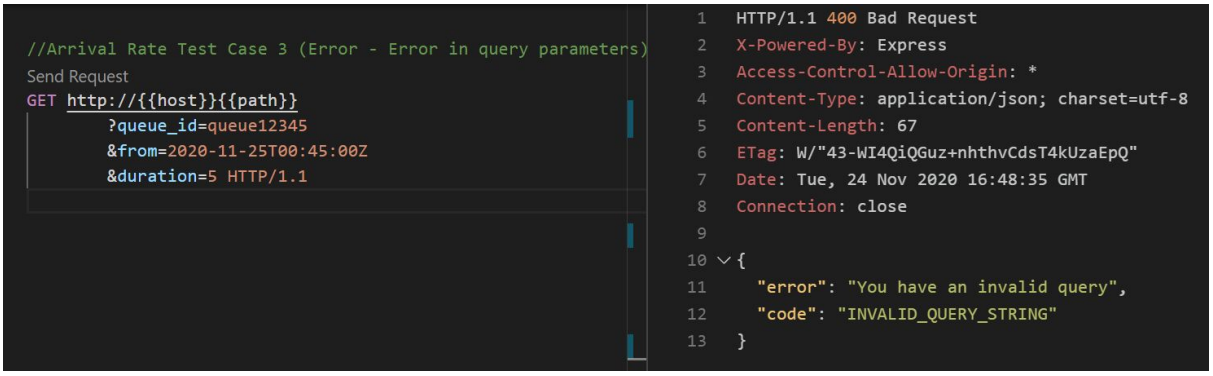
3. Receive a **`400 Bad Request`** response with an error response body

```
{
  "error": "You have an invalid query",
  "code": "INVALID_QUERY_STRING"
}
```

Expected Result: Receive a **`400 Bad Request`** response with the error message and error code ("INVALID_QUERY_STRING") in the response body

Actual Result: Receive a **`400 Bad Request`** response with the error message and error code (“INVALID_QUERY_STRING”) in the response body (**PASS**)

Evidence:



Reset Test Case 1 (Success)

Title: Reset API - Successful request reset all tables in database

Description: A successful request made to the server resets all that data in both **companyQueue** and **customerQueue** table

Precondition: **companyQueue** and **customerQueue** table created with at least 1 row of data exists in both **companyQueue** and **customerQueue** table

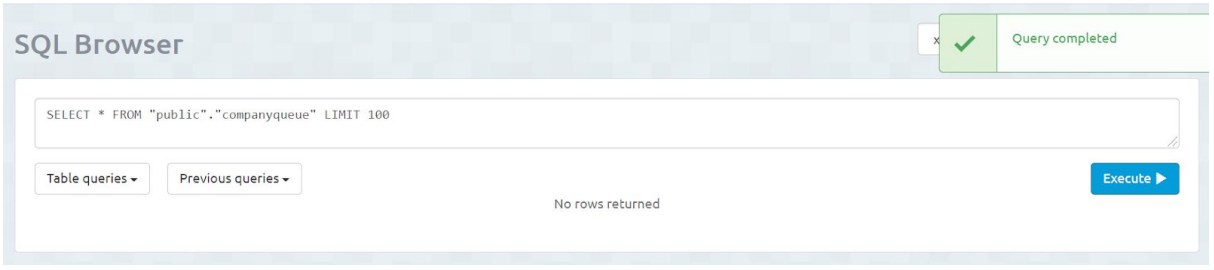
Test Steps:

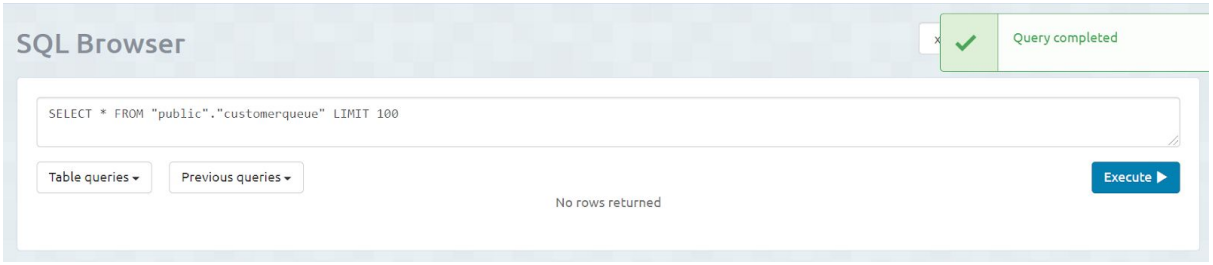
1. Start the Backend Server
2. Send an Reset request to the backend
3. Receive a **200 OK** response
4. Go to elephantsql to inspect **companyQueue** and **customerQueue** table

Expected Result: All the data in **companyQueue** and **customerQueue** table are deleted

Actual Result: All the data in **companyQueue** and **customerQueue** table are deleted (**PASS**)

Evidence:




SQL Browser
Query completed
SELECT * FROM "public"."customerqueue" LIMIT 100
Table queries
Previous queries
Execute
No rows returned

Scenario Test Cases

Company 1234567890 creates Queue ‘QUEUE12345’

Title: Create Queue API - Successful request creates **`Queue`** ‘**`QUEUE12345`**’ in database

Description: A successful request made to the server should create a new entry in the **`companyQueue`** table with the correct parameters.

Precondition: **`companyQueue`** table created with no existing **`queue_id`**.

Test Steps:

1. Start the Backend Server
2. Send a Create Queue request to the backend with a **valid `company_id`** and a **valid `queue_id`**

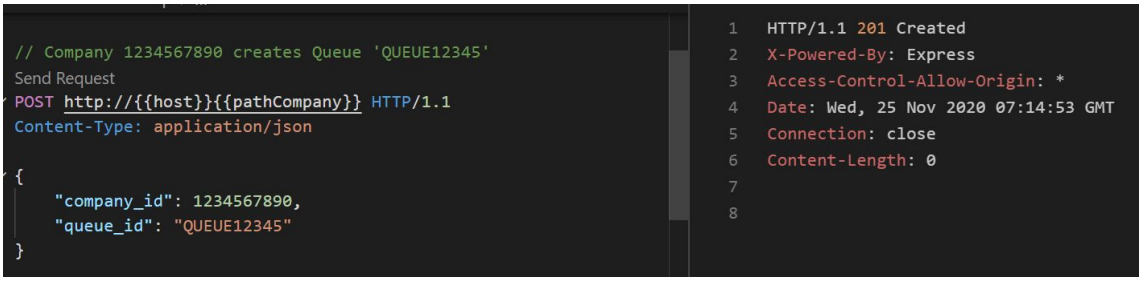
```
{
 "company_id": 1234567890,
 "queue_id": "QUEUE12345"
}
```

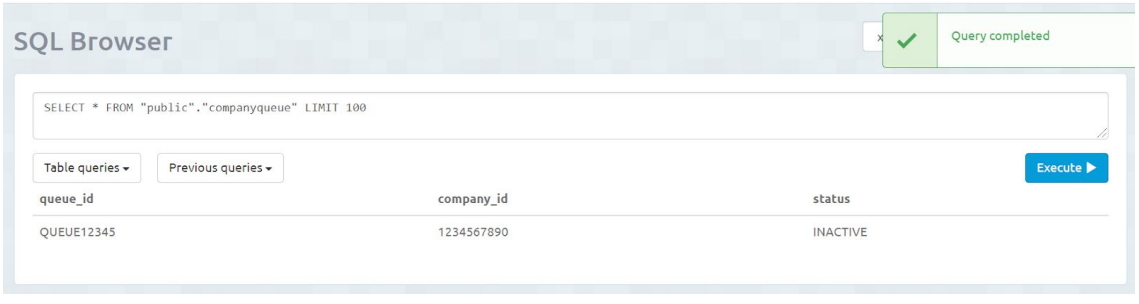
3. Receive a **`201 Created`** response
4. Go to elephantsql to inspect the **`companyQueue`** table

Expected Result: **`Queue`** ‘**`QUEUE12345`**’ is created in the **`companyQueue`** table

Actual Result: **`Queue`** ‘**`QUEUE12345`**’ is created in the **`companyQueue`** table (**PASS**)

Evidence:





Company 1234567890 activates Queue ‘QUEUE12345’

Title: Update Queue API - Successful request updates queue **`status`** in database to ‘**`ACTIVATE`**’

Description: A successful request made to the server should update the **`status`** of the queue that already exists in the **`companyQueue`** table with the correct parameters.

Precondition: **-**

Test Steps:

5. Send an Update Queue request to the backend with a valid **`status`** ‘**`Activate`**’ in the json body

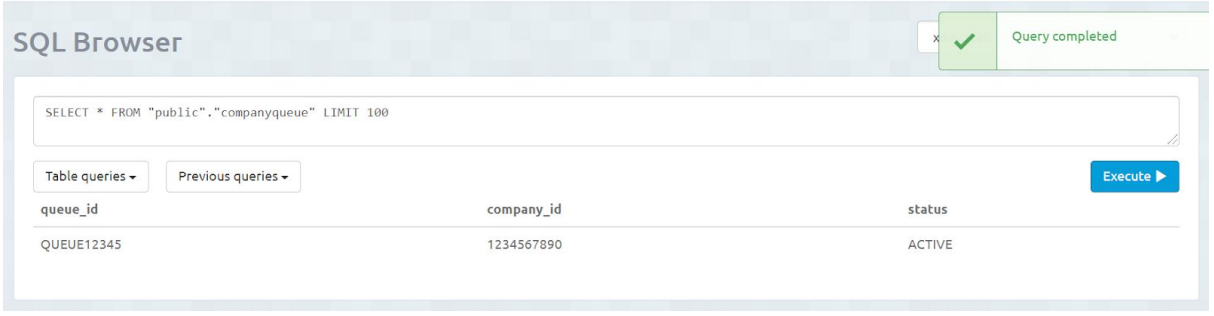
```
{
  "status": "ACTIVATE"
}
```

6. Receive a **`200 OK`** response
7. Go to elephantsql to inspect the **`companyQueue`** table

Expected Result: The **`status`** of the queue is changed to **‘ACTIVE’**

Actual Result: The **`status`** of the queue is changed to **‘ACTIVE’** (**PASS**)

Evidence:



Customer 3456789012 joins queue ‘QUEUE12345’

Title: Join Queue API - Successful request results in **`Customer 3456789012`** joining **`Queue ‘QUEUE12345’`**

Description: A successful request made to the server should create a new entry in the **`customerQueue`** table with the correct parameters (**`customer_id = 3456789012, queue_id = QUEUE12345`**).

Precondition: There should be no rows existing in the **`customerQueue`** table.

Test Steps:

8. Send an Join Queue request to the backend with a valid **`customer_id`**, and an existing and valid **`queue_id`** in the json body

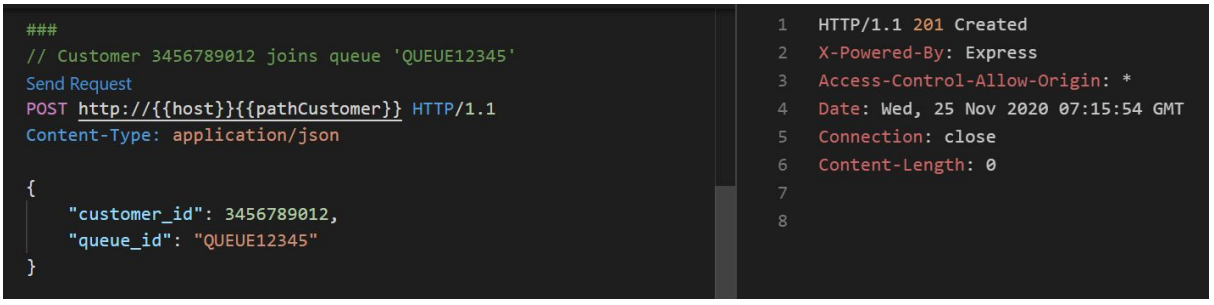
```
{
  "customer_id": 3456789012,
  "queue_id": "QUEUE12345"
}
```

9. Receive a **`201 Created`** response
10. Go to elephantsql to inspect the **`customerQueue`** table

Expected Result: **`Customer 3456789012`** joined the **`Queue ‘QUEUE12345’`**

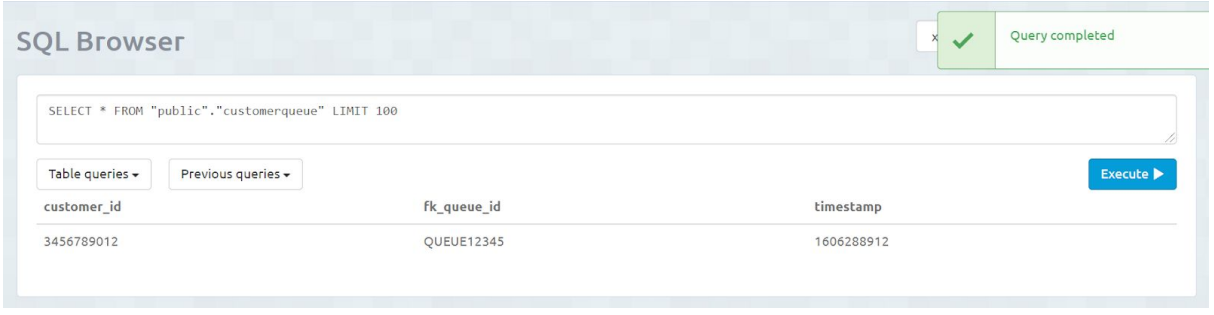
Actual Result: **`Customer 3456789012`** joined the **`Queue ‘QUEUE12345’`** (**PASS**)

Evidence:



```
###
// Customer 3456789012 joins queue 'QUEUE12345'
Send Request
POST http://{{host}}{{pathCustomer}} HTTP/1.1
Content-Type: application/json

{
    "customer_id": 3456789012,
    "queue_id": "QUEUE12345"
}
```

```
1  HTTP/1.1 201 Created
2  X-Powered-By: Express
3  Access-Control-Allow-Origin: *
4  Date: Wed, 25 Nov 2020 07:15:54 GMT
5  Connection: close
6  Content-Length: 0
7
8
```



Customer 9999999999 joins queue 'QUEUE12345'

Title: Join Queue API - Successful request results in **Customer 9999999999 joining queue 'QUEUE12345'**

Description: A successful request made to the server should create a new entry in the **customerQueue** table with the correct parameters (**customer_id = 9999999999, queue_id = QUEUE12345**).

Precondition: **Customer 9999999999** does not exist in **customerQueue** table.

Test Steps:

11. Send an Join Queue request to the backend with a **different valid customer_id** and the **same valid queue_id** in the json body

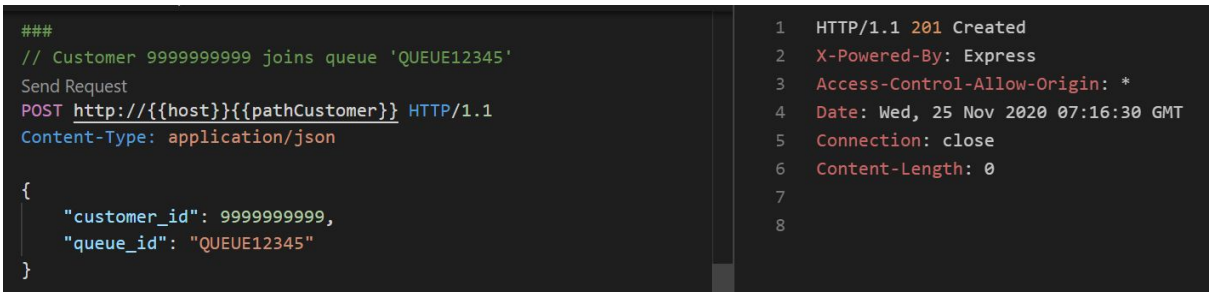
```
{
  "customer_id": 9999999999,
  "queue_id": "QUEUE12345"
}
```

12. Receive a **201 Created** response
13. Go to elephantsql to inspect the **customerQueue** table

Expected Result: **Customer 9999999999** joined the **Queue 'QUEUE12345',** there are now 2 existing customers in the **Queue 'QUEUE12345'**

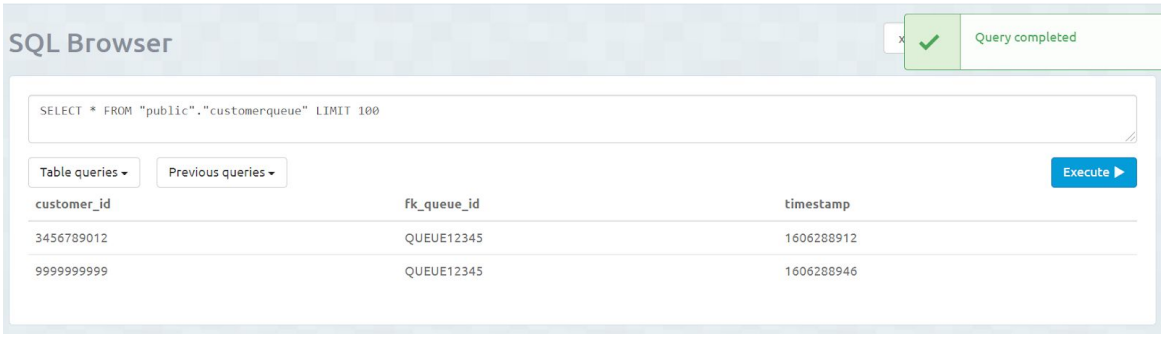
Actual Result: **Customer 9999999999** joined the **Queue 'QUEUE12345',** there are now 2 existing customers in the **Queue 'QUEUE12345'** (**PASS**)

Evidence:



```
###
// Customer 9999999999 joins queue 'QUEUE12345'
Send Request
POST http://{{host}}{{pathCustomer}} HTTP/1.1
Content-Type: application/json

{
    "customer_id": 9999999999,
    "queue_id": "QUEUE12345"
}
```

```
1  HTTP/1.1 201 Created
2  X-Powered-By: Express
3  Access-Control-Allow-Origin: *
4  Date: Wed, 25 Nov 2020 07:16:30 GMT
5  Connection: close
6  Content-Length: 0
7
8
```



Company 1234567890 assign next customer to server when server is available

Title: Server Available API - Successful request returns the customer id of the next customer and updates the **`customerQueue`** table

Description: A successful request made to the server should check if the queue exists, return the next **`customer id`** of the queue, and update the **`customerQueue`** table.

Precondition: -

Test Steps:

14. Send the Server Available request to the backend with a **valid `queue_id`**

```
{
 "queue_id": "QUEUE12345"
}
```

15. Receive a **`200 OK`** response with a response body

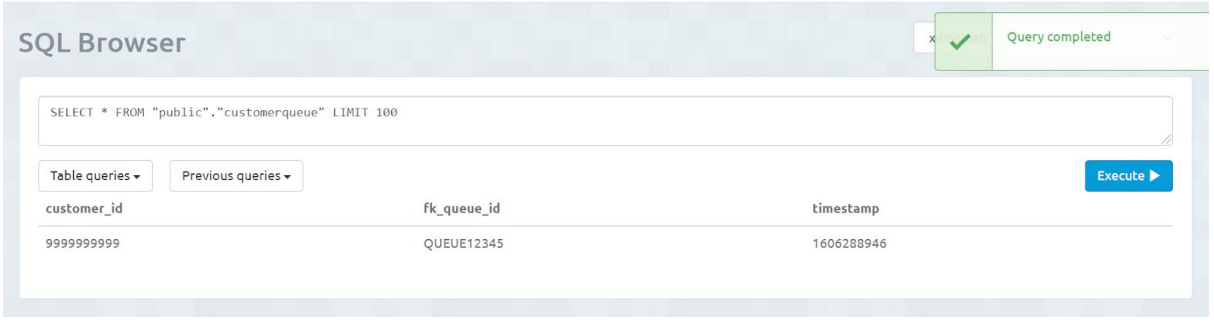
```
{
 "customer_id": 3456789012
}
```

16. Go to elephantsql to inspect the **`customerQueue`** table

Expected Result: There is only **Customer 9999999999** left in the **Queue 'QUEUE12345'** in the customerQueue table

Actual Result: There is only **Customer 9999999999** left in the **Queue ‘QUEUE12345’** in the customerQueue table (**PASS**)

Evidence:



Customer 9999999999 checks Queue ‘QUEUE12345’

Title: Check Queue API - Successful request customer check queue in database

Description: A successful request made to the server should generate the number of customers in a queue. When there are customers ahead in the queue, **ahead** will return the number of customers ahead.

Precondition: **-**

Test Steps:

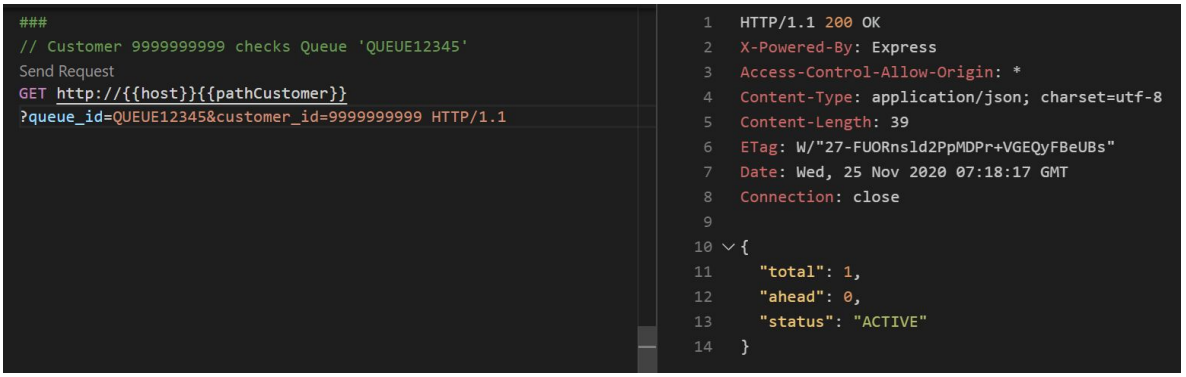
17. Send a Check Queue request to the backend with a **valid** and **existing** **`customer_id`** and **`queue_id`** in the url params.

```
GET
http://{{host}}{{path}}?queue_id=QUEUE12345&customer_id=9999999999
HTTP/1.1
```

Expected Result: Receive a **`200 OK`** response with the **`total`** number of customers (**`1`**), the number of customers **`ahead`** (**`0`**) and **`status`** (**`“ACTIVE”`**) in the response body.

Actual Result: Receive a **`200 OK`** response with the **`total`** number of customers (**`1`**), the number of customers **`ahead`** (**`0`**) and **`status`** (**`“ACTIVE”`**) in the response body. (**PASS**)

Evidence:



Company 1234567890 checks for the number of arrivals within a certain period of time

Title: Arrival Rate API - Successful request to get the timestamp and number of customers from database

Description: A successful request made to the server should return a response body with timestamp and number of customers.

Precondition: -

Test Steps:

18. Start the Backend Server
19. Send an Arrival Rate request to the backend with **`queue_id`**, **`from`** (starting time) and **`duration`** in the URL query

```
GET
http://localhost:3000/company/arrival_rate?queue_id=queue12345
&from=2020-11-24T21%3A11%3A00%2B08%3A00&duration=1 HTTP/1.1
```

20. Receive a **`200 OK`** response with a response body

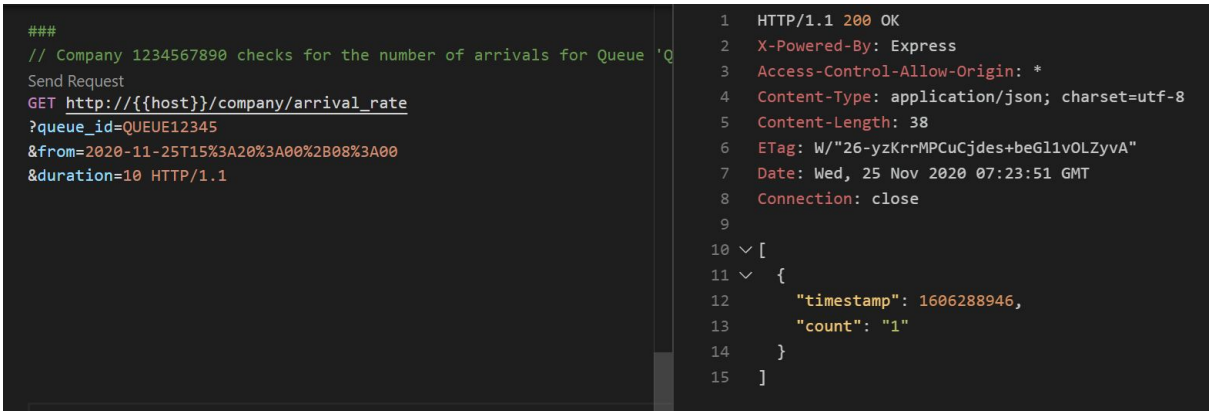
```
[
      {
            “timestamp”: 1606288946,
            “count”: “1”
```

```
        }
    ]
```

Expected Result: Receive a **`200 OK`** response with a **`timestamp`** and number of customers (**`count`**) in the response body

Actual Result: Receive a **`200 OK`** response with a **`timestamp`** and number of customers (**`count`**) in the response body (**PASS**)

Evidence:



Company 1234567890 deactivates Queue 'QUEUE12345'

Title: Update Queue API - Successful request updates queue **`status`** in database to '**`INACTIVE`**'

Description: A successful request made to the server should update the **`status`** of the queue that already exists in the **`companyQueue`** table with the correct parameters.

Precondition: **-**

Test Steps:

21. Send an Update Queue request to the backend with a valid **`status`** '**`DEACTIVATE`**' in the json body

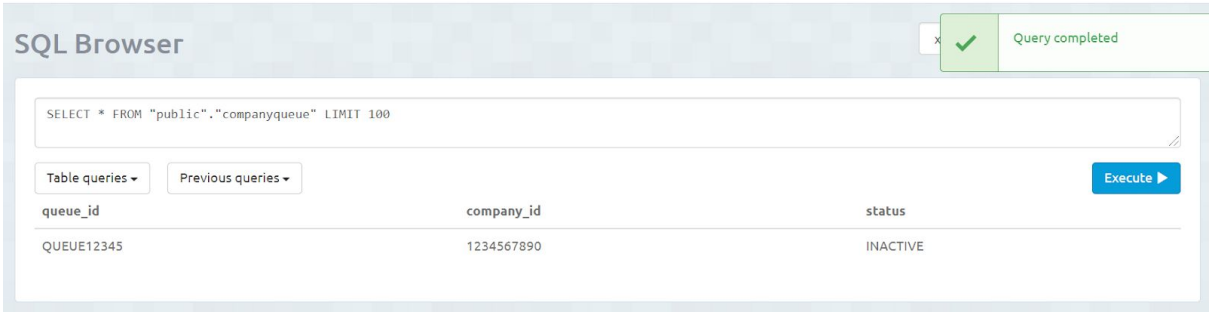
```
{
  "status": "DEACTIVATE"
}
```

22. Receive a **`200 OK`** response
23. Go to elephantsql to inspect the **`companyQueue`** table

Expected Result: The **`status`** of the queue is changed to **'INACTIVE'**

Actual Result: The **`status`** of the queue is changed to **'INACTIVE'** (**PASS**)

Evidence:



Reset System

Title: Reset API - Successful request resets all tables in database

Description: A successful request made to the server resets all that data in both **`companyQueue`** and **`customerQueue`** table

Precondition: -

Test Steps:

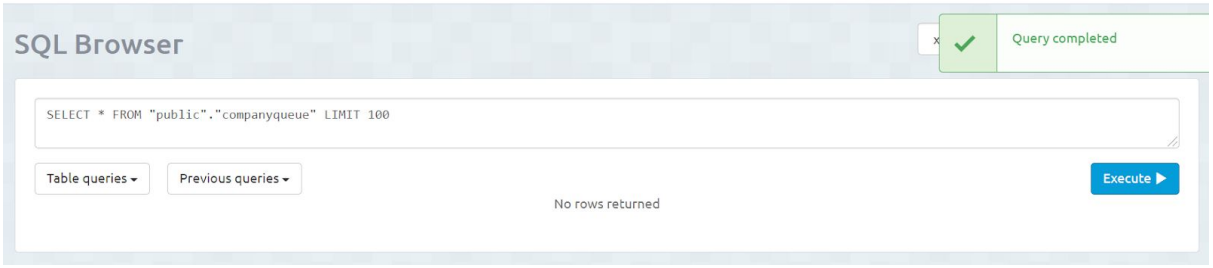
24. Start the Backend Server
25. Send an Reset request to the backend
26. Receive a **`200 OK`** response
27. Go to elephantsql to inspect **`companyQueue`** and **`customerQueue`** table

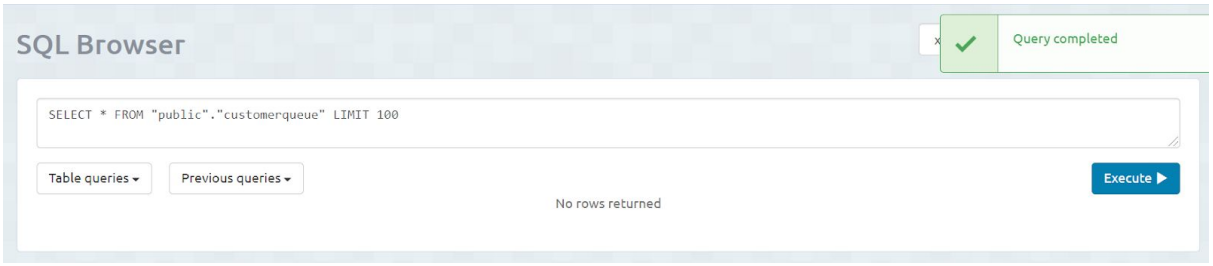
Expected Result: All the data in **`companyQueue`** and **`customerQueue`** table are deleted

Actual Result: All the data in **`companyQueue`** and **`customerQueue`** table are deleted (**PASS**)

Evidence:


###
// Reset System
Send Request
POST http://{{host}}/reset HTTP/1.1
1 HTTP/1.1 200 OK
2 X-Powered-By: Express
3 Access-Control-Allow-Origin: *
4 Content-Length: 0
5 ETag: W/"0-2jmj715rSw0yVb/vlWAYkK/YBwk"
6 Date: Wed, 25 Nov 2020 07:26:04 GMT
7 Connection: close
8


SQL Browser
Query completed
SELECT * FROM "public"."companyqueue" LIMIT 100
Table queries
Previous queries
Execute
No rows returned


SQL Browser
Query completed
SELECT * FROM "public"."customerqueue" LIMIT 100
Table queries
Previous queries
Execute
No rows returned

SCRUM meetings

SCRUM Meeting 1 (09/11/2020): Week 4

Present:
Lynette Jean Tay
Tsai Pei Yu
Tan Jia Xin Vanessa

1. **What have you done?**

Lynette Jean Tay: Clone repository

Tsai Pei Yu: Clone repository

Tan Jia Xin Vanessa: Clone repository

2. **How is the progress?**

Lynette Jean Tay: Good, no issues

Tsai Pei Yu: Good, no issues

Tan Jia Xin Vanessa: Good, no issues

3. **What are you going to do next?**

Lynette Jean Tay: Create and design database, create product backlog in github repository, reset and Create Queue API

Tsai Pei Yu: Create and design database, create product backlog in github repository, Update Queue API

Tan Jia Xin Vanessa: Create and design database, create product backlog in github repository, Join Queue API

SCRUM Meeting 2 (16/11/2020): Week 5

Present:
Lynette Jean Tay
Tsai Pei Yu
Tan Jia Xin Vanessa

1. **What have you done?**

Lynette Jean Tay: Create and design database, create product backlog in github repository, reset and Create Queue API
Tsai Pei Yu: Create and design database, create product backlog in github repository, Update Queue API
Tan Jia Xin Vanessa: Create and design database, create product backlog in github repository, Join Queue API

2. **How is the progress?**

Lynette Jean Tay: Good, no issues
Tsai Pei Yu: Good, no issues
Tan Jia Xin Vanessa: Good, no issues

3. **What are you going to do next?**

Lynette Jean Tay: Server Available API
Tsai Pei Yu: Arrival Rate API and Pool
Tan Jia Xin Vanessa: Check Queue API

SCRUM Meeting 3 (23/11/2020): Week 6

Present:
Lynette Jean Tay
Tsai Pei Yu
Tan Jia Xin Vanessa

1. **What have you done?**

Lynette Jean Tay: Server Available API

Tsai Pei Yu: Arrival Rate API and Pool

Tan Jia Xin Vanessa: Check Queue API

2. **How is the progress?**

Lynette Jean Tay: Good, no issues

Tsai Pei Yu: Good, no issues

Tan Jia Xin Vanessa: Good, no issues

3. **What are you going to do next?**

Lynette Jean Tay: Test case and documentation

Tsai Pei Yu: Test case and documentation

Tan Jia Xin Vanessa: Test case and documentation