BUILDING A PAYMENT GATEWAY

FOR

checkout.com

BY

Shahanah Begum Puttaroo

JUNE 2020

TABLE OF CONTENT

INTRODUCTION

This API-based application is a payment gateway that allows merchants to offer shoppers a method of payment.

REQUIREMENTS

The requirements are as follows:

1. A merchant should be able to process a payment through the payment gateway.
2. A merchant should be able to retrieve the details of a previously made payment.

GETTING STARTED

The API lives at the api/paymentgateway/ endpoint and responds to GET and POST.

The merchant account is authorized using an access token of type Bearer, obtained from Auth0. A token can be obtained by running the command in the obtainToken.txt file.

To use and test the API, Postman has been used.

The following command has been used to run the API:

```
sudo docker run -it --rm -p 8080:80 shahanah/paymentgateway:v1
```

The SQL Server connection string has been hardcoded in the file appsettings.json. The required values have to be provided. The EF Core migration features were used to create the tables. The commands are as follows:

```
dotnet ef migrations add AddPaymentToDB --context PaymentContext

dotnet ef database update --context PaymentContext

dotnet ef migrations add AddShopperToDB --context ShopperContext

dotnet ef database update --context ShopperContext

dotnet ef migrations add AddLogToDB --context LogContext

dotnet ef database update --LogContext
```

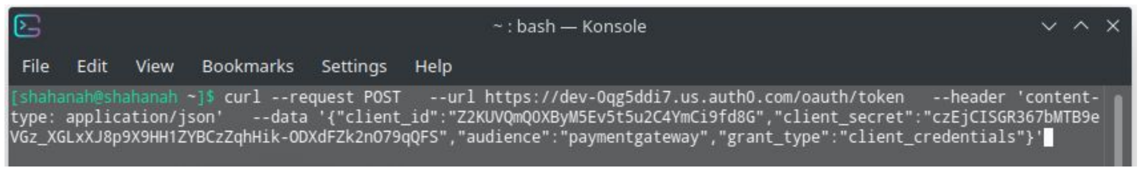
REQUESTS

The API is based on REST principles. Therefore, data resources are accessed via standard HTTP requests.

METHOD	ACTION
GET	Retrieve a payment
POST	Process payment

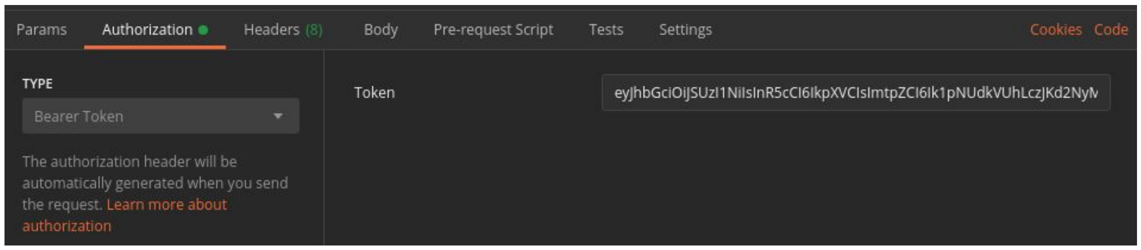
AUTHORIZATION

To be able to make requests to the API, a Bearer Token needs to be provided. This token is obtained from auth0 using the following command:



```
[shahanah@shahanah ~]$ curl --request POST   --url https://dev-0qg5ddi7.us.auth0.com/oauth/token   --header 'content-
type: application/json'   --data '{"client_id":"Z2KUVQmQ0XByM5Ev5t5u2C4YmCi9fd8G","client_secret":"czEjCISGR367bMTB9e
VGz_XGLxXJ8p9X9HH1ZYBCzZqhHik-ODXdFZk2nO79qQFS","audience":"paymentgateway","grant_type":"client_credentials"}'
```

The command is provided in the obtainToken.txt file in the project. This token is valid for 24 hours. It is specified in Postman as follows:



TYPE: Bearer Token

The authorization header will be automatically generated when you send the request. Learn more about authorization

Token: eyJhbGciOiJSUzI1NiIsInR5cCI6IkpXVCIsImtpZCI6Ik1pNUdkVUhLczJKd2NyM

GET

GET /api/paymentgateway/n, where n is the unique identifier of a payment.

The following properties of the payment are obtained.

PROPERTY	DESCRIPTION
Payment ID	Unique identifier of the payment
Card Number	Identifier of the card (masked)
Expiry Date	The expiry date of the card
Amount	Amount paid by the shopper
Currency	System of monetary units used
CVV	Card Verification Value
Status	Whether the payment is successful or unsuccessful

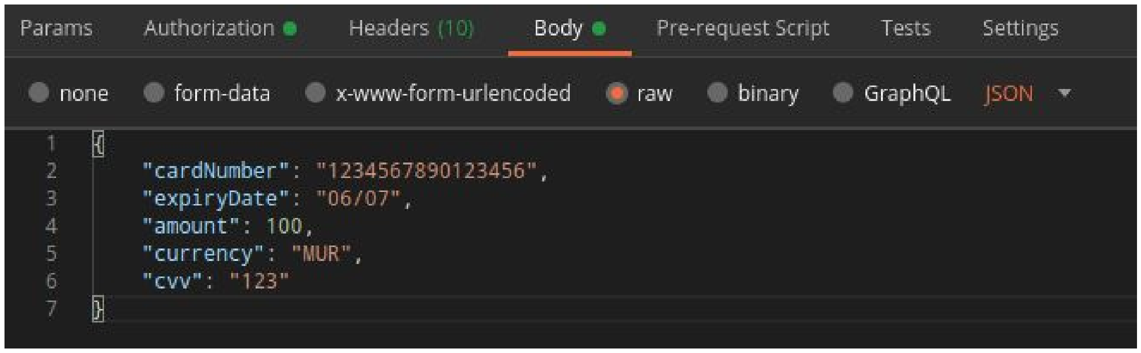
POST

POST /api/paymentgateway

The parameters that should be specified to make this request are as follows:

PARAMETER	DESCRIPTION	EXAMPLE
Card Number	Identifier of the card	“1234567890123456”
Expiry Date	The expiry date of the card	“06/07”
Amount	Amount paid by the shopper	100
Currency	System of monetary units used	“MUR”
CVV	Card Verification Value	“123”

The following shows how the parameters are entered in Postman.



```
{
    "cardNumber": "1234567890123456",
    "expiryDate": "06/07",
    "amount": 100,
    "currency": "MUR",
    "cvv": "123"
}
```

After the request is sent, it is checked whether it is valid or not. For this project, the bank component was simulated as follows:

1. It is checked whether the card number exists in the database.
2. It is checked whether the card is expired.
3. It is checked whether the shopper has enough money.

Appropriate status codes are returned so that integrating a real bank component would be easier.

After the request is sent, the following properties are obtained in the response:

PROPERTY	DESCRIPTION
Payment ID	Unique identifier of the payment
Status	Whether the payment is successful or unsuccessful

LOGGING

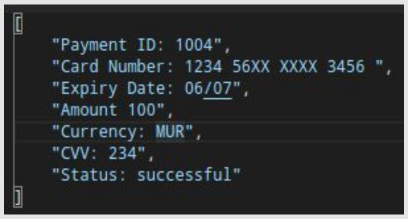
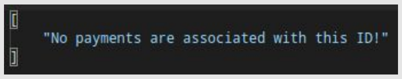
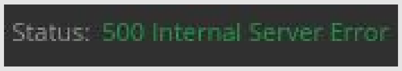
The date and details of each transaction are recorded in the database.

TESTING

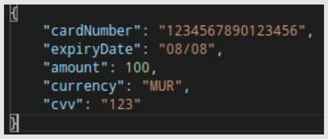
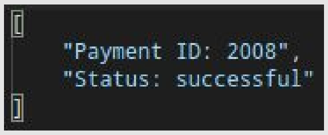
To verify whether the API works as expected, three types of testing are performed, namely

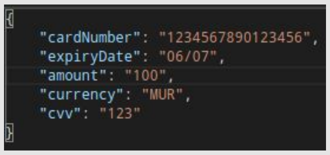
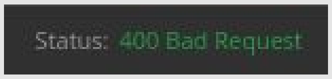
1. Unit testing
2. Integration testing
3. Load testing

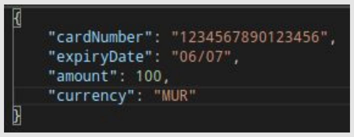
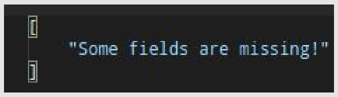
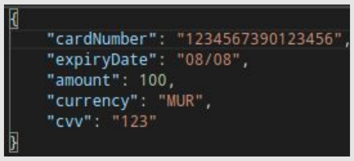
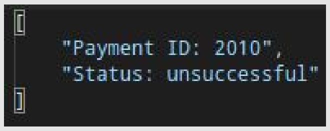
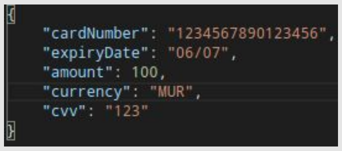
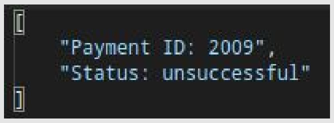
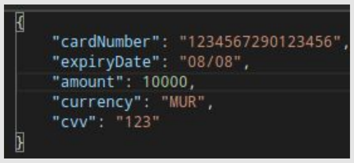
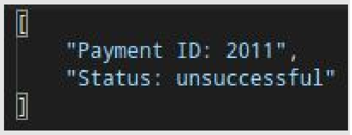
UNIT TESTING

REQUEST: GET /api/paymentgateway/n			
Test Case	**Test URL**	**Expected Result**	**Actual Result**
1. Obtain existing payment	/api/gateway/1004	Correctly obtain details	 ["Payment ID: 1004", "Card Number: 1234 56XX XXXX 3456 ", "Expiry Date: 06/07", "Amount 100", "Currency: MUR", "CVV: 234", "Status: successful"]
2. Obtain non-existing payment	/api/gateway/2	No details should be obtained	 ["No payments are associated with this ID!"]
3. No id is specified	/api/gateway	An error status code is returned	 Status: 500 Internal Server Error

REQUEST: POST /api/paymentgateway			
Test Case	**Test Data**	**Expected Result**	**Actual Result**

1. Enter correct details	 {
	"cardNumber": "1234567890123456",
	"expiryDate": "08/08",
	"amount": 100,
	"currency": "MUR",
	"cvv": "123"
	}
		Successful payment	 [
			"Payment ID: 2008",
			"Status: successful"
			]

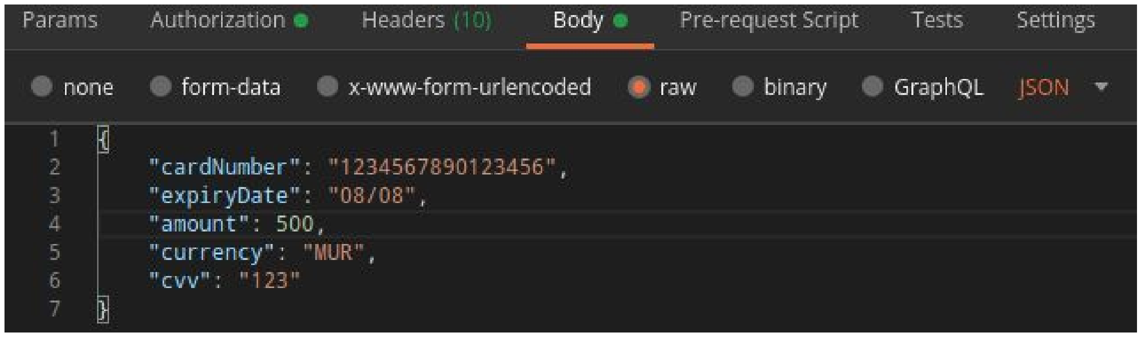
2. Enter details of the wrong datatype	 {
	"cardNumber": "1234567890123456",
	"expiryDate": "06/07",
	"amount": "100",
	"currency": "MUR",
	"cvv": "123"
	}
		Validation error	 Status: 400 Bad Request

3. A parameter is missing	 `{ "cardNumber": "1234567890123456", "expiryDate": "06/07", "amount": 100, "currency": "MUR" }`	Error message	 `[ "Some fields are missing!" ]`
4. The shopper does not exist	 `{ "cardNumber": "1234567390123456", "expiryDate": "08/08", "amount": 100, "currency": "MUR", "cvv": "123" }`	Unsuccessful transaction	 `[ "Payment ID: 2010", "Status: unsuccessful" ]`
5. Card has expired	 `{ "cardNumber": "1234567890123456", "expiryDate": "06/07", "amount": 100, "currency": "MUR", "cvv": "123" }`	Unsuccessful transaction	 `[ "Payment ID: 2009", "Status: unsuccessful" ]`
6. Insufficient funds	 `{ "cardNumber": "1234567290123456", "expiryDate": "08/08", "amount": 10000, "currency": "MUR", "cvv": "123" }`	Unsuccessful transaction	 `[ "Payment ID: 2011", "Status: unsuccessful" ]`

Note: For test cases 2 and 3, since the result is the same for all parameters, only one has been shown.

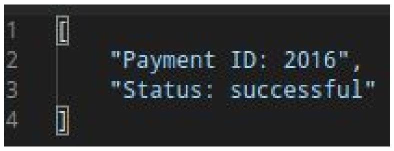
INTEGRATION TESTING

It is important to check whether both requests work as intended when integrated. To verify this, a new payment is requested as follows:



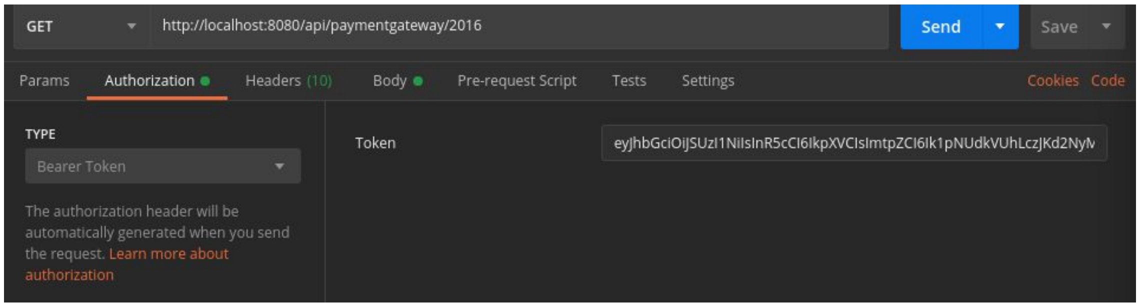
```
{
    "cardNumber": "1234567890123456",
    "expiryDate": "08/08",
    "amount": 500,
    "currency": "MUR",
    "cvv": "123"
}
```

The following response is obtained, showing the ID and status.



```
[
    "Payment ID: 2016",
    "Status: successful"
]
```

It is now checked whether the merchant can view the details of the payment using the GET request. The request is made as follows:



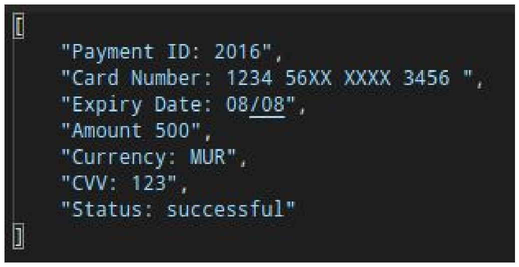
GET http://localhost:8080/api/paymentgateway/2016

TYPE: Bearer Token

The authorization header will be automatically generated when you send the request. Learn more about authorization

Token: eyJhbGciOiJSUzI1NiIsInR5cCI6IkpXVCIsImtpZCI6Ik1pNUdkVUhLczJKd2NyN

As shown in the following image, all the details are correctly obtained.

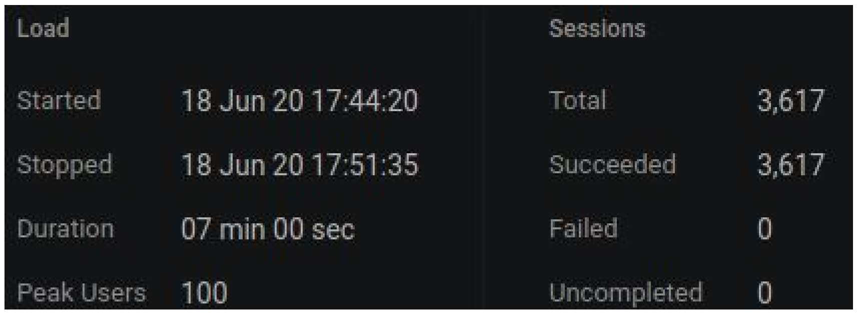


```
[
    "Payment ID: 2016",
    "Card Number: 1234 56XX XXXX 3456 ",
    "Expiry Date: 08/08",
    "Amount 500",
    "Currency: MUR",
    "CVV: 123",
    "Status: successful"
]
```

LOAD TESTING

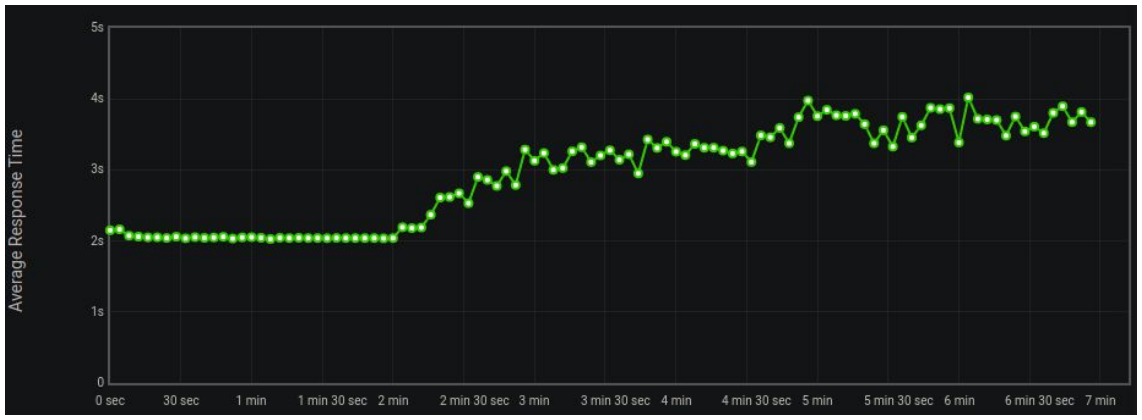
Load testing is a type of performance testing conducted to determine the behavior of the system under normal and peak conditions.

Using 100 virtual users for a duration of 7 minutes, the following result has been obtained.



Load		Sessions	
Started	18 Jun 20 17:44:20	Total	3,617
Stopped	18 Jun 20 17:51:35	Succeeded	3,617
Duration	07 min 00 sec	Failed	0
Peak Users	100	Uncompleted	0

The following graph shows the average response time during the load test.



As shown, the response times do not exceed four seconds.

IMPROVEMENTS

The response time of the API can be improved using some of the following methods:

1. Compression techniques
2. Asynchronous methods
3. Optimize web servers