

# **Sri Lanka Institute of Information Technology**



**Assignment 03 : Implement microservice backend for the minimal viable product (MVP)**

**IT4020 - Modern Topics in IT**

**Year 4, Semester 1**

**(2023) – Weekend Batch**

## Group Details

Group ID - MTIT-011

Project Name - Implement microservice backend for the minimal viable product (MVP)

**Our business domain is Dialog Axiata company**

Batch - 2023 -Y4S1

Registration Number	Student Name
IT20147396	Peiris B M G
IT20178154	Dilshan P A D S D
IT20081416	Ahamed M M Z
IT20122782	Amani M P N

# Student Contribution

REGISTRATION NUMBER	STUDENT CONTRIBUTION
IT20147396	<ul style="list-style-type: none"><li>• Shop Manager can ADD communication items.</li><li>• Shop Manager can VIEW communication items.</li><li>• Shop Manager can UPDATE communication items.</li><li>• Shop Manager can DELETE communication items.</li></ul>
IT20178154	<ul style="list-style-type: none"><li>• Supplier can ADD supplier menu.</li><li>• Supplier can VIEW the supplier menu.</li><li>• Supplier can UPDATE supplier menu.</li><li>• Supplier can DELETE supplier menu.</li></ul>
IT20122782	<ul style="list-style-type: none"><li>• Storekeeper can ADD storekeeper menu.</li><li>• Storekeeper can VIEW the storekeeper menu.</li><li>• Storekeeper can UPDATE storekeeper menu.</li><li>• Storekeeper can DELETE storekeeper menu</li></ul>
IT20081416	<ul style="list-style-type: none"><li>• Delivery person can ADD delivery items.</li><li>• Delivery person can VIEW delivery items.</li><li>• Delivery person can UPDATE delivery items .</li><li>• Delivery person can DELETE delivery items .</li></ul>

# The Scenario

## Our business domain is Dialog Axiata company

Dialog Axiata is a Sri Lankan telecommunications company that provides mobile, fixed-line, and broadband services to customers in Sri Lanka. The company has embraced microservices and cloud computing as a way to deliver services more efficiently and effectively. Dialog Axiata has implemented a number of microservices, including Manager API, Storekeeper API, Supplier API, and Delivery Person API. Each microservice has a specific role and set of responsibilities:

**Storekeeper API microservice:** This microservice provides a set of RESTful endpoints to manage inventory-related resources, such as stock levels, product information, and order fulfillment. The Storekeeper API microservice helps to optimize inventory management processes, reduce costs, and improve customer satisfaction.

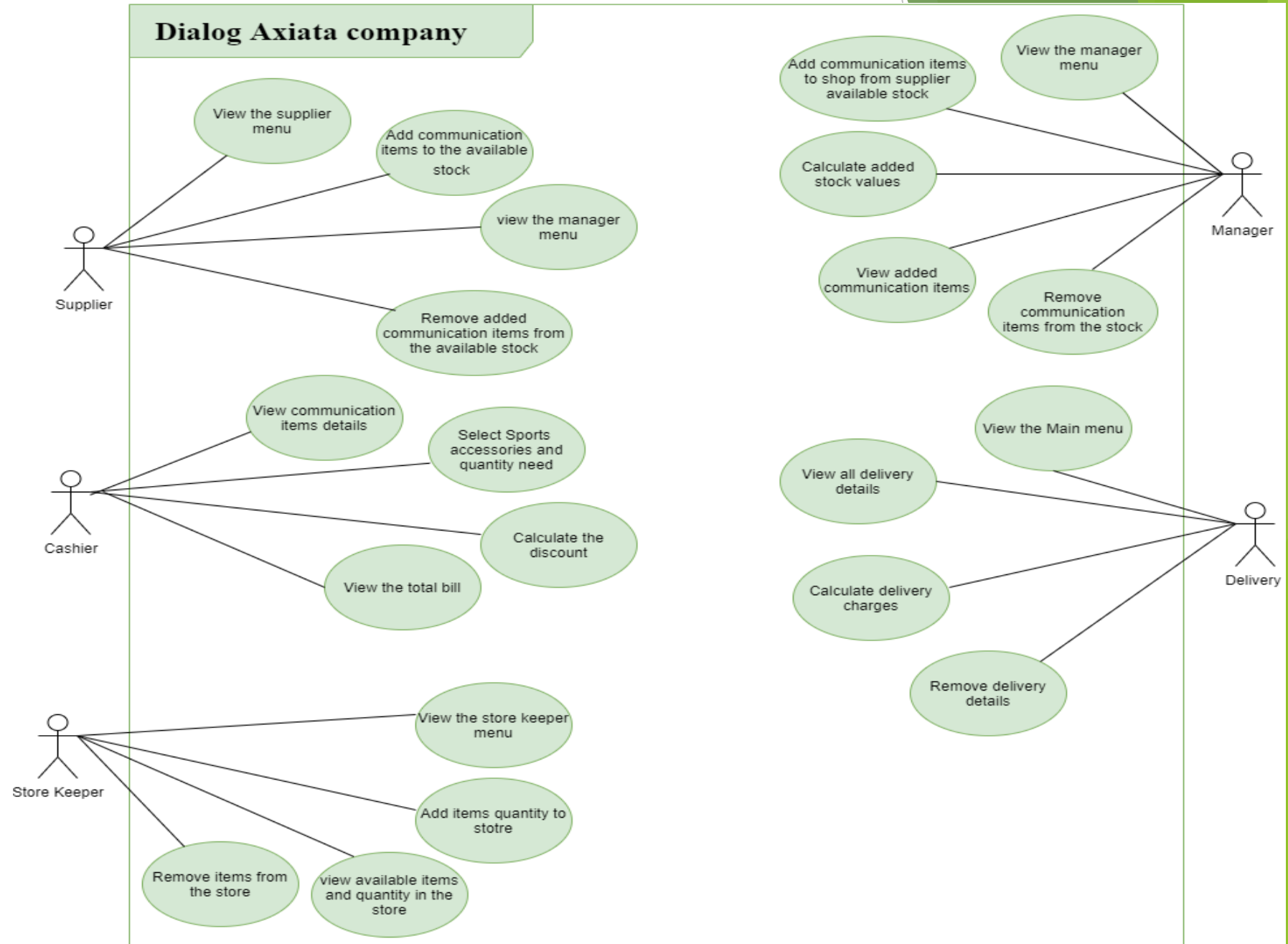
**Supplier API microservice:** This microservice provides a set of RESTful endpoints to manage supplier-related resources, such as supplier information, purchase orders, and payment processing. The Supplier API microservice helps to streamline supplier management processes and improve the efficiency of the procurement process.

**Manager API microservice:** This microservice provides a set of RESTful endpoints to manage employee-related resources, such as employee information, work schedules, and performance reviews. The Manager API microservice helps managers to efficiently manage their teams and improve employee productivity.

**Delivery Person API microservice:** This microservice provides a set of RESTful endpoints to manage delivery-related resources, such as delivery orders, delivery schedules, and delivery routes. The Delivery Person API microservice helps to optimize delivery management processes, improve delivery performance, and enhance the overall customer experience.

To implement these microservices, Dialog Axiata has also adopted cloud computing technologies, such as containerization, Kubernetes, and cloud-based databases. These technologies enable Dialog Axiata to deploy and manage microservices more efficiently and cost-effectively, while also providing scalability, reliability, and high availability.

# System Diagram



# Define and elaborate the microservices

## Sports Delivery service

### **1. Define the scope and functionality of the Delivery Person API microservice:**

Identify the delivery-related resources that the microservice will manage. Define the CRUD operations that will be supported by the microservice for each resource. Specify the input and output parameters for each operation.

### **2. Design the architecture of the Delivery Person API microservice:**

Choose a programming language and framework to build the microservice. Define the data models for the delivery-related resources that will be managed by the microservice. Decide on the data storage mechanism for the microservice (e.g., relational database, NoSQL database, in-memory cache).

### **3. Implement the Delivery Person API microservice:**

Write the code for the RESTful endpoints that will handle the requests from clients. Implement the business logic for each endpoint. Integrate the microservice with any necessary external services (e.g., delivery management systems, authentication and authorization services).

### **4. Test and deploy the Delivery Person API microservice:**

Write test cases for each endpoint to ensure that the microservice functions as expected. Deploy the microservice to a production environment. Monitor the performance of the microservice and make any necessary improvements.

### **5. Document the Delivery Person API microservice:**

Create API documentation that describes the endpoints, input/output parameters, and error codes. Provide examples of how to use the API. Publish the documentation to a centralized location for developers to access.

## Sports Supplier service

### **1. Define the scope and functionality of the Supplier API microservice:**

Identify the supplier-related resources that the microservice will manage. Define the CRUD operations that will be supported by the microservice for each resource. Specify the input and output parameters for each operation.

### **2. Design the architecture of the Supplier API microservice:**

Choose a programming language and framework to build the microservice. Define the data models for the supplier-related resources that will be managed by the microservice. Decide on the data storage mechanism for the microservice (e.g., relational database, NoSQL database, in-memory cache).

### **3. Implement the Supplier API microservice:**

Write the code for the RESTful endpoints that will handle the requests from clients. Implement the business logic for each endpoint. Integrate the microservice with any necessary external services (e.g., supplier databases, authentication and authorization services).

### **4. Test and deploy the Supplier API microservice:**

Write test cases for each endpoint to ensure that the microservice functions as expected. Deploy the microservice to a production environment. Monitor the performance of the microservice and make any necessary improvements.

### **5. Document the Supplier API microservice:**

Create API documentation that describes the endpoints, input/output parameters, and error codes. Provide examples of how to use the API. Publish the documentation to a centralized location for developers to access.

## Sports Manager service

### 1. Define the scope and functionality of the Manager API microservice:

Identify the resources that the microservice will manage. Define the CRUD operations that will be supported by the microservice for each resource.

Specify the input and output parameters for each operation.

### 2. Design the architecture of the Manager API microservice:

Choose a programming language and framework to build the microservice. Define the data models for the resources that will be managed by the microservice. Decide on the data storage mechanism for the microservice (e.g., relational database, NoSQL database, in-memory cache).

### 3. Implement the Manager API microservice:

Write the code for the RESTful endpoints that will handle the requests from clients. Implement the business logic for each endpoint. Integrate the microservice with any necessary external services.

### 4. Test and deploy the Manager API microservice:

Write test cases for each endpoint to ensure that the microservice functions as expected. Deploy the microservice to a production environment.

Monitor the performance of the microservice and make any necessary improvements.

### 5. Document the Manager API microservice:

Create API documentation that describes the endpoints, input/output parameters, and error codes. Provide examples of how to use the API.

Publish the documentation to a centralized location for developers to access.



## Sports Storekeeper service

### 1. Define the scope and functionality of the Storekeeper API microservice:

Identify the inventory-related resources that the microservice will manage. Define the CRUD operations that will be supported by the microservice for each resource. Specify the input and output parameters for each operation.

### 2. Design the architecture of the Storekeeper API microservice:

Choose a programming language and framework to build the microservice. Define the data models for the inventory-related resources that will be managed by the microservice. Decide on the data storage mechanism for the microservice (e.g., relational database, NoSQL database, in-memory cache).

### 3. Implement the Storekeeper API microservice:

Write the code for the RESTful endpoints that will handle the requests from clients. Implement the business logic for each endpoint. Integrate the microservice with any necessary external services (e.g., inventory management systems, authentication and authorization services).

### 4. Test and deploy the Storekeeper API microservice:

Write test cases for each endpoint to ensure that the microservice functions as expected. Deploy the microservice to a production environment. Monitor the performance of the microservice and make any necessary improvements.

### 5. Document the Storekeeper API microservice:

Create API documentation that describes the endpoints, input/output parameters, and error codes. Provide examples of how to use the API. Publish the documentation to a centralized location for developers to access.

## Explain the usage of API gateway

- Centralized API management - An API Gateway serves as a centralized point for managing all the APIs in a system. It can handle requests from multiple clients and route them to the appropriate backend services.
- Security - An API Gateway can provide an additional layer of security to APIs by enforcing authentication and authorization rules. It can also protect against common attacks like DDoS, SQL injection, and cross-site scripting.
- Traffic management- An API Gateway can distribute incoming requests across multiple instances of a backend service to ensure optimal performance and availability. It can also perform load balancing, caching, and rate limiting to prevent service overload.
- Monitoring and analytics-An API Gateway can collect data on the usage of APIs and provide insights into performance, errors, and usage patterns. This information can be used to optimize the API design and improve the user experience.
- Protocol translation-An API Gateway can translate between different protocols and data formats to enable communication between different systems. For example, it can translate RESTful API requests into SOAP requests or vice versa.
- Simplified client development- An API Gateway can provide a unified interface for multiple backend services, making it easier for clients to interact with them. It can also hide the complexity of the underlying services by providing a simpler API.

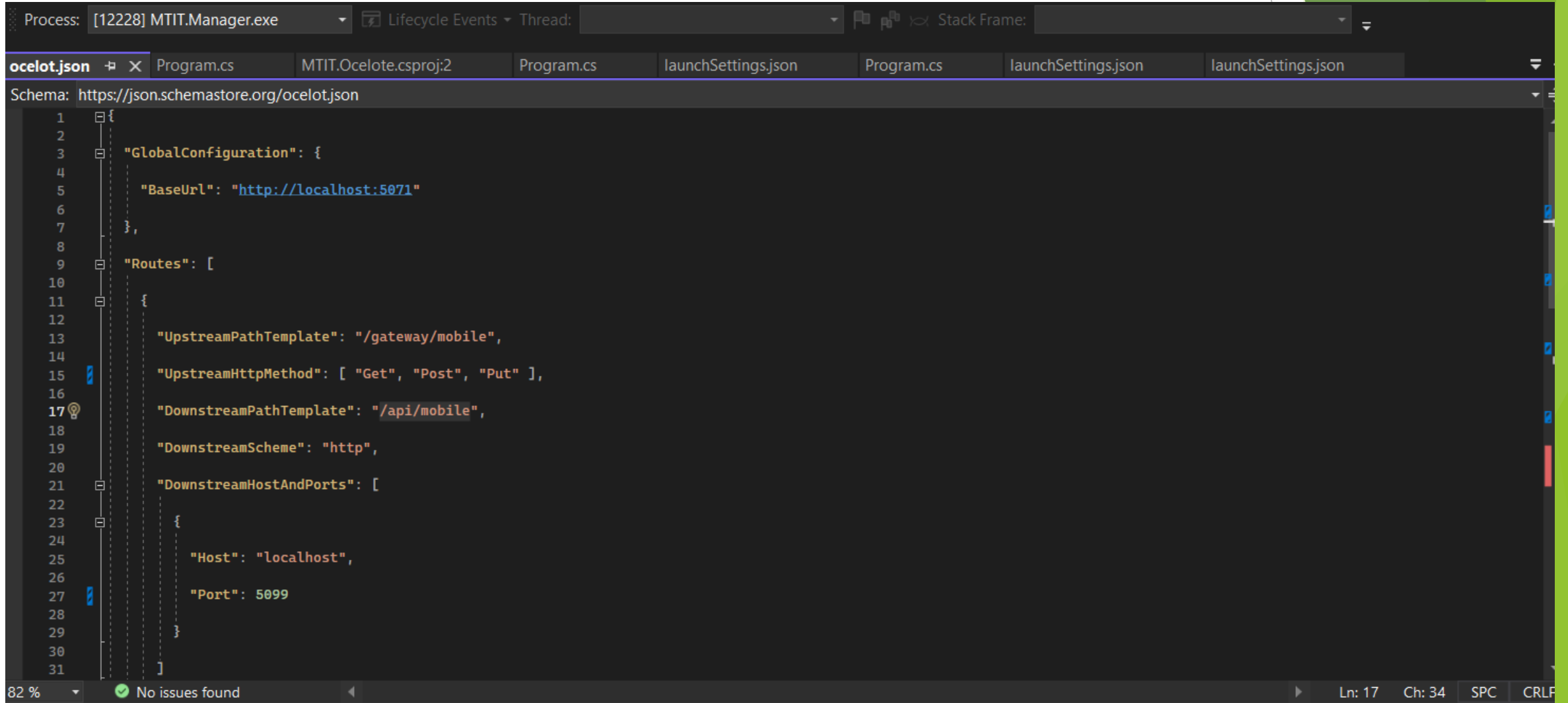
The background features abstract, overlapping green geometric shapes, primarily triangles and polygons, in various shades of green, creating a modern and dynamic visual effect.

**IT20178154**  
**Dilshan P A D S D**  
**Supplier Service**

## Supplier Service

Elaborate how you can avoid having multiple ports with the gateway you implemented and Solution should have proper folder structure with minimum two microservices and an Ocelot API Gateway.

### Ocelot.json in Supplier Service



```
Process: [12228] MTIT.Manager.exe Lifecycle Events Thread: Stack Frame:
ocelot.json Program.cs MTIT.Ocelote.csproj:2 Program.cs launchSettings.json Program.cs launchSettings.json launchSettings.json
Schema: https://json.schemastore.org/ocelot.json
1 {
2
3   "GlobalConfiguration": {
4     "BaseUrl": "http://localhost:5071"
5   },
6
7   "Routes": [
8     {
9       "UpstreamPathTemplate": "/gateway/mobile",
10      "UpstreamHttpMethod": [ "Get", "Post", "Put" ],
11      "DownstreamPathTemplate": "/api/mobile",
12      "DownstreamScheme": "http",
13      "DownstreamHostAndPorts": [
14        {
15          "Host": "localhost",
16          "Port": 5099
17        }
18      ]
19    }
20  ]
21 }
22
23
24
25
26
27
28
29
30
31
```

82 % No issues found Ln: 17 Ch: 34 SPC CRLF

## Ocelot.json in Supplier Service

Process: [12228] MTIT.Manager.exe    Lifecycle Events    Thread:    Stack Frame:

ocelot.json    Program.cs    MTIT.Ocelote.csproj:2    Program.cs    launchSettings.json    Program.cs    launchSettings.json    launchSettings.json

Schema: <https://json.schemastore.org/ocelot.json>

```
33  },
34
35  {
36
37    "UpstreamPathTemplate": "/gateway/mobile/{Id}",
38
39    "UpstreamHttpMethod": [ "Get", "Delete" ],
40
41    "DownstreamPathTemplate": "/api/mobile/{Id}",
42
43    "DownstreamScheme": "http",
44
45    "DownstreamHostAndPorts": [
46
47      {
48
49        "Host": "localhost",
50
51        "Port": 5099
52      }
53    ]
54  }
55 ]
56
57 }
58
59 ]
60
61 }
62 }
```

# Screen captures OF Supplier API

The screenshot shows a web browser window displaying the Swagger UI for the MTIT.Supplier.Mobile v1 API. The browser's address bar shows the URL `localhost:5099/swagger/index.html`. The Swagger UI header includes the Swagger logo and the text "Supported by SMARTBEAR". A dropdown menu labeled "Select a definition" is set to "MTIT.Supplier.Mobile v1".

The main content area displays the API definition for "MTIT.Supplier.Mobile" with version "1.0" and "OAS3" specification. The URL `http://localhost:5099/swagger/v1/swagger.json` is provided. The API is organized into two sections: "Mobile" and "WeatherForecast".

**Mobile**

- GET** `/api/Mobile`
- POST** `/api/Mobile`
- PUT** `/api/Mobile`
- GET** `/api/Mobile/{id}`
- DELETE** `/api/Mobile/{id}`

**WeatherForecast**

- GET** `/WeatherForecast`

Below the "WeatherForecast" section, the word "Schemas" is partially visible.

# GET Method

Swagger UI

localhost:5099/swagger/index.html

GET /api/Mobile

Parameters

No parameters

Execute Clear

Responses

Curl

```
curl -X 'GET' \
'http://localhost:5099/api/Mobile' \
-H 'accept: */*'
```

Request URL

```
http://localhost:5099/api/Mobile
```

Server response

Code Details

200

Response body

```
[
  {
    "id": 1,
    "brand": "Apple",
    "country": "USA",
    "price": 200000
  },
  {
    "id": 2,
    "brand": "Huawei",
    "country": "China",
    "price": 22000
  },
  {
    "id": 3,
    "brand": "Samsung",
    "country": "Korea",
    "price": 100000
  },
  {
    "id": 4,
    "brand": "Oppo",
    "country": "India",
    "price": 23000
  },
  {
    "id": 5,
    "brand": "Nokia",
    "country": "Finland",
    "price": 15000
  }
]
```

Download

# POST Method

**POST** /api/Mobile

Parameters

No parameters

Cancel

Reset

Request body

application/json

```
{
  "id": 6,
  "brand": "Sony",
  "country": "England",
  "price": 23000
}
```

Responses

Curl

```
curl -X 'POST' \
  'http://localhost:5099/api/Mobile' \
  -H 'accept: */*' \
  -H 'Content-Type: application/json' \
  -d '{
    "id": 6,
    "brand": "Sony",
    "country": "England",
    "price": 23000
  }'
```

Request URL

http://localhost:5099/api/Mobile

Server response

Code	Details
200	<div><div>Response body</div><div><pre>{   "id": 6,   "brand": "Sony",   "country": "England",   "price": 23000 }</pre></div><div><div>Download</div></div></div> <div><div>Response headers</div><div><pre>content-type: application/json; charset=utf-8 date: Thu, 13 Apr 2023 06:25:13 GMT server: Kestrel transfer-encoding: chunked</pre></div></div>

Responses

Code	Description	Links
200	Success	No links



# PUT Method

PUT /api/Mobile

## Parameters

Cancel

Reset

No parameters

Request body

application/json

```
{
  "id": 1,
  "brand": "Apple 13",
  "country": "string",
  "price": 0
}
```

## Responses

Curl

```
curl -X 'PUT' \
  'http://localhost:5099/api/Mobile' \
  -H 'accept: */*' \
  -H 'Content-type: application/json' \
  -d '{
    "id": 1,
    "brand": "Apple 13",
    "country": "string",
    "price": 0
  }'
```

Request URL

http://localhost:5099/api/Mobile

Server response

Code	Details
------	---------

200	<div>Response body</div> <pre>{   "id": 1,   "brand": "Apple 13",   "country": "string",   "price": 0 }</pre> <div>Download</div>
-----	---

Response headers

```
content-type: application/json; charset=utf-8
date: Thu, 13 Apr 2023 06:29:20 GMT
server: Kestrel
transfer-encoding: chunked
```

Responses

Code	Description
------	-------------

200	Success
-----	---------

Links

No links

# SEARCH

GET

/api/Mobile/{id}

Parameters

Cancel

Name	Description
<b>id</b> * required integer(\$int32) (path)	<input type="text" value="4"/>

Execute

Clear

Responses

Curl

```
curl -X 'GET' \  
'http://localhost:5099/api/Mobile/4' \  
-H 'accept: */*'
```

Request URL

```
http://localhost:5099/api/Mobile/4
```

Server response

Code	Details
200	<div><div>Response body</div><div><pre>{   "id": 4,   "brand": "Oppo",   "country": "India",   "price": 23000 }</pre></div><div><div></div><div>Download</div></div></div> <div><div>Response headers</div><div><pre>content-type: application/json; charset=utf-8 date: Thu, 13 Apr 2023 06:34:35 GMT server: Kestrel transfer-encoding: chunked</pre></div></div>

Responses

Code	Description	Links
200	Success	No links

# DELETE Method

DELETE /api/Mobile/{id}

## Parameters

Cancel

Name	Description
------	-------------

id \* required

integer(\$int32)

(path)

6

Execute

Clear

## Responses

Curl

```
curl -X 'DELETE' \
'http://localhost:5099/api/Mobile/6' \
-H 'accept: */*'
```

Request URL

http://localhost:5099/api/Mobile/6

Server response

Code	Details
------	---------

200

Response body

mobile with ID:6 got deleted successfully.

Response headers

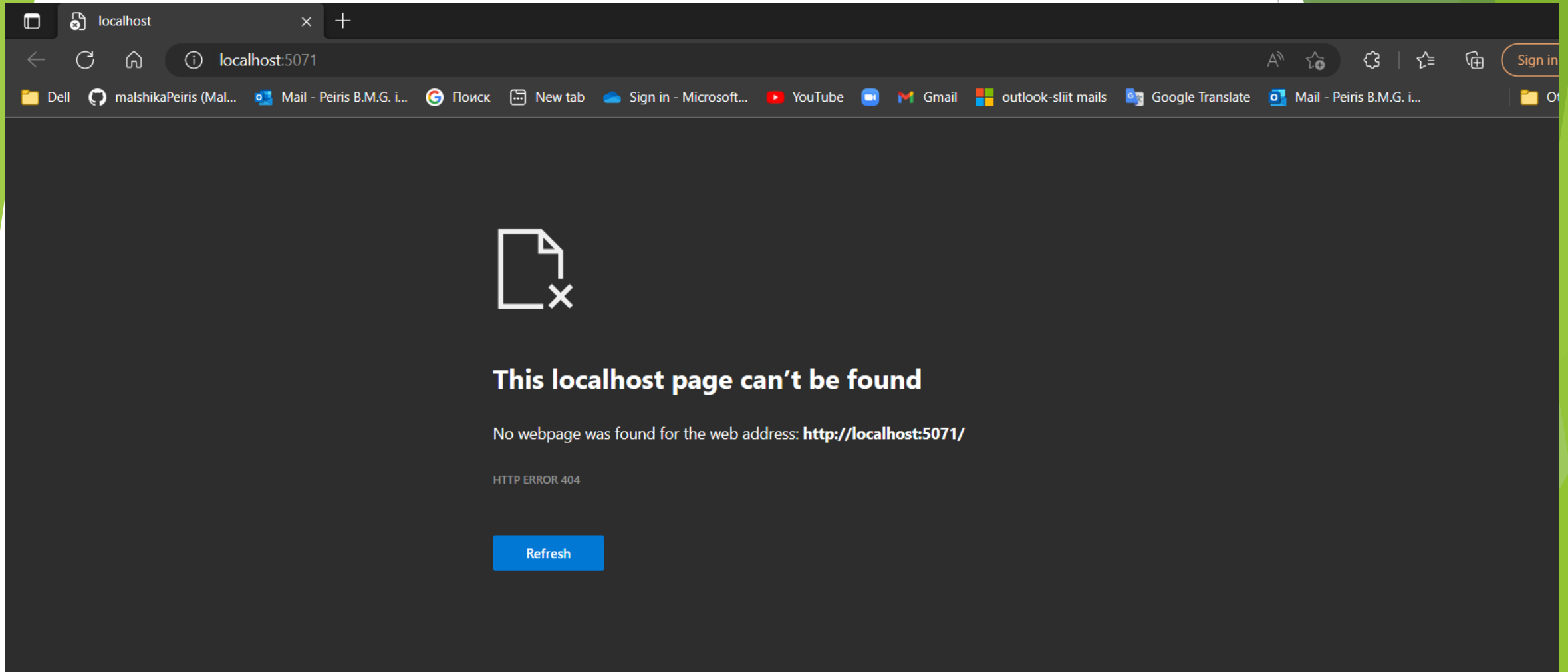
```
content-type: text/plain; charset=utf-8
date: Thu, 13 Apr 2023 06:37:03 GMT
server: Kestrel
transfer-encoding: chunked
```



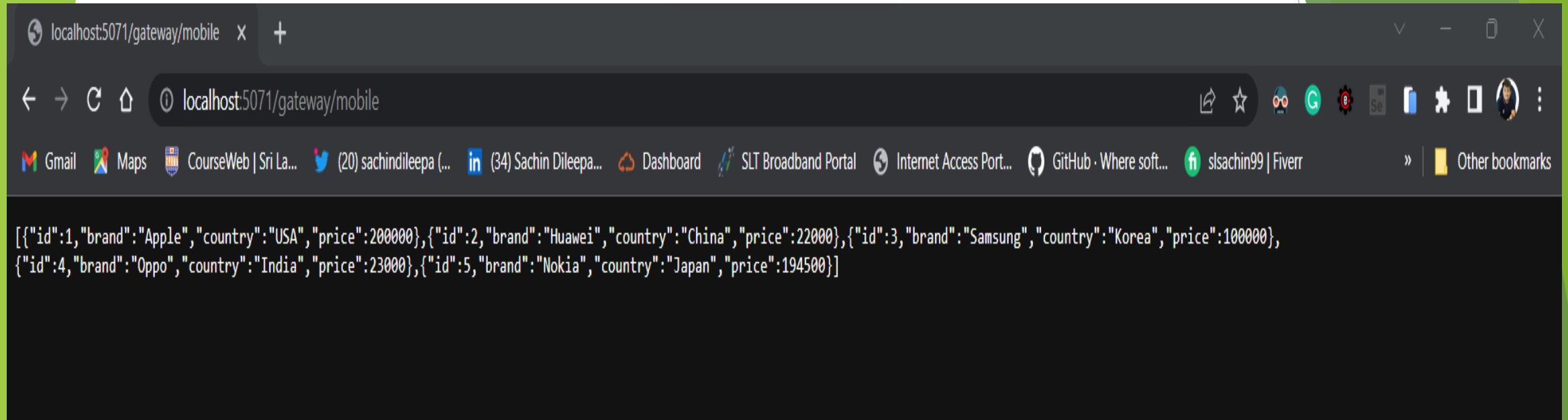
Download

Gateway project and the API project will both run and open up in two separate browsers.

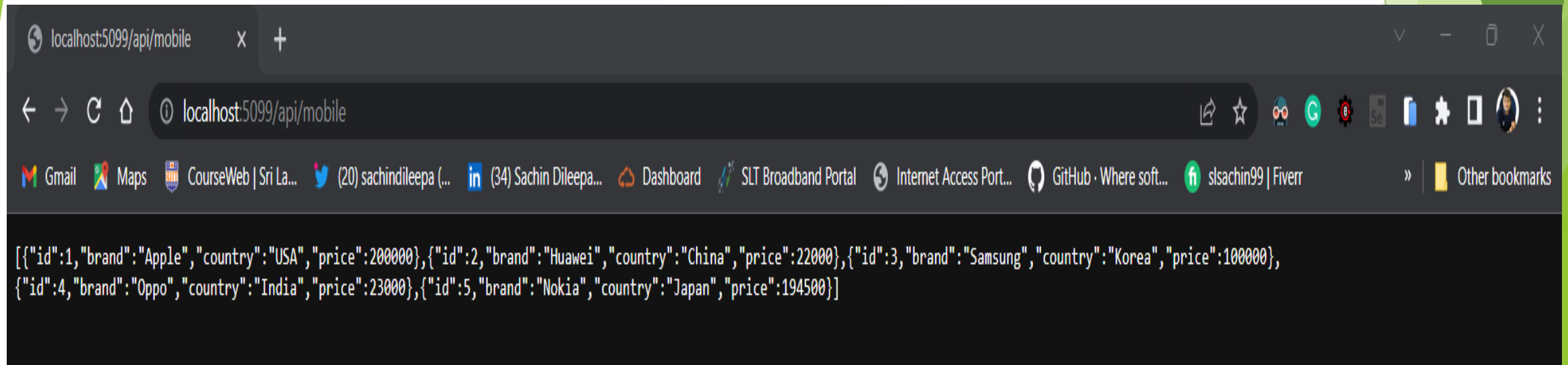
Gateway:



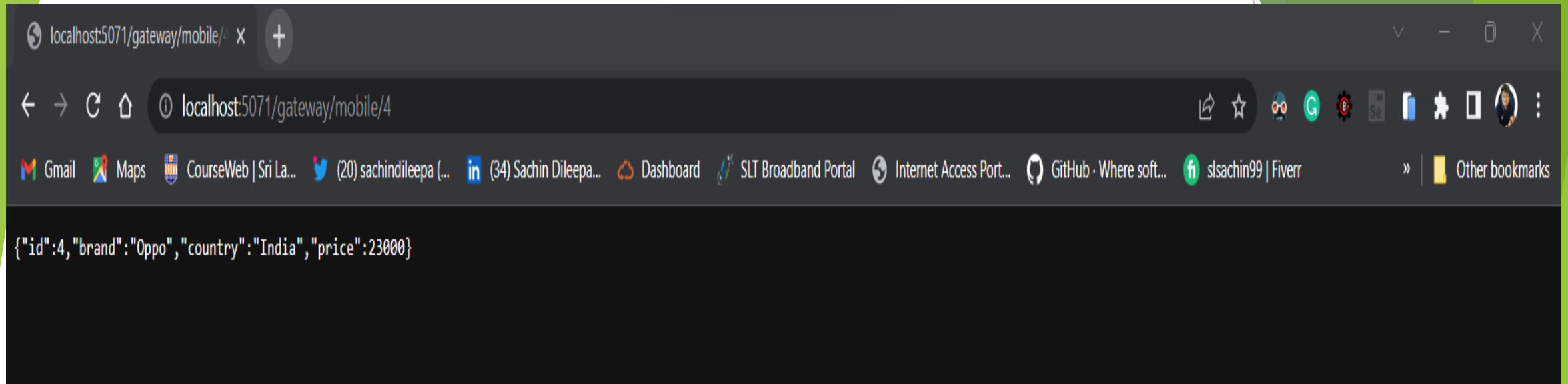
Test the upstream URL in browser.



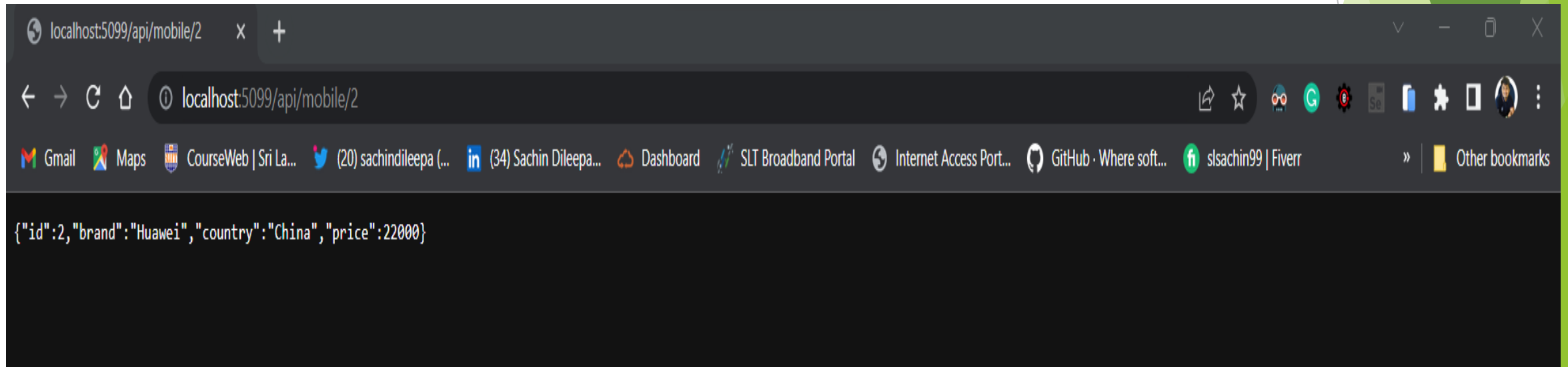
Test the downstream URL in browser.



Test the upstream URL in browser.



Test the downstream URL in browser.



**IT20147396**

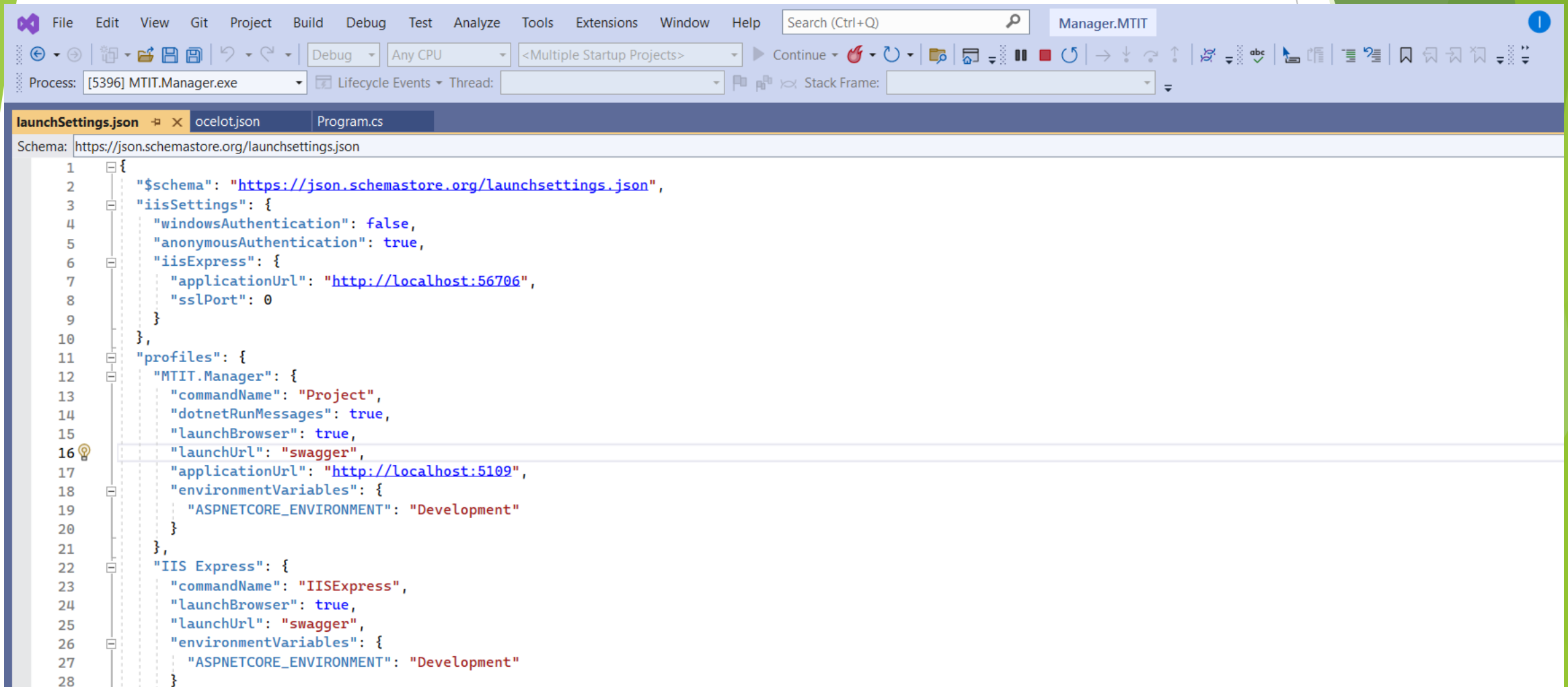
**Peiris B M G**

**Manager Service**

# Manager Service

Elaborate how you can avoid having multiple ports with the gateway you implemented and Solution should have proper folder structure with minimum two microservices and an Ocelot API Gateway.

## launchsettings.json in Manager service



```
1 {
2   "$schema": "https://json.schemastore.org/launchsettings.json",
3   "iisSettings": {
4     "windowsAuthentication": false,
5     "anonymousAuthentication": true,
6     "iisExpress": {
7       "applicationUrl": "http://localhost:56706",
8       "sslPort": 0
9     }
10  },
11  "profiles": {
12    "MTIT.Manager": {
13      "commandName": "Project",
14      "dotnetRunMessages": true,
15      "launchBrowser": true,
16      "launchUrl": "swagger",
17      "applicationUrl": "http://localhost:5109",
18      "environmentVariables": {
19        "ASPNETCORE_ENVIRONMENT": "Development"
20      }
21    },
22    "IIS Express": {
23      "commandName": "IISExpress",
24      "launchBrowser": true,
25      "launchUrl": "swagger",
26      "environmentVariables": {
27        "ASPNETCORE_ENVIRONMENT": "Development"
28      }
29    }
30  }
31 }
```



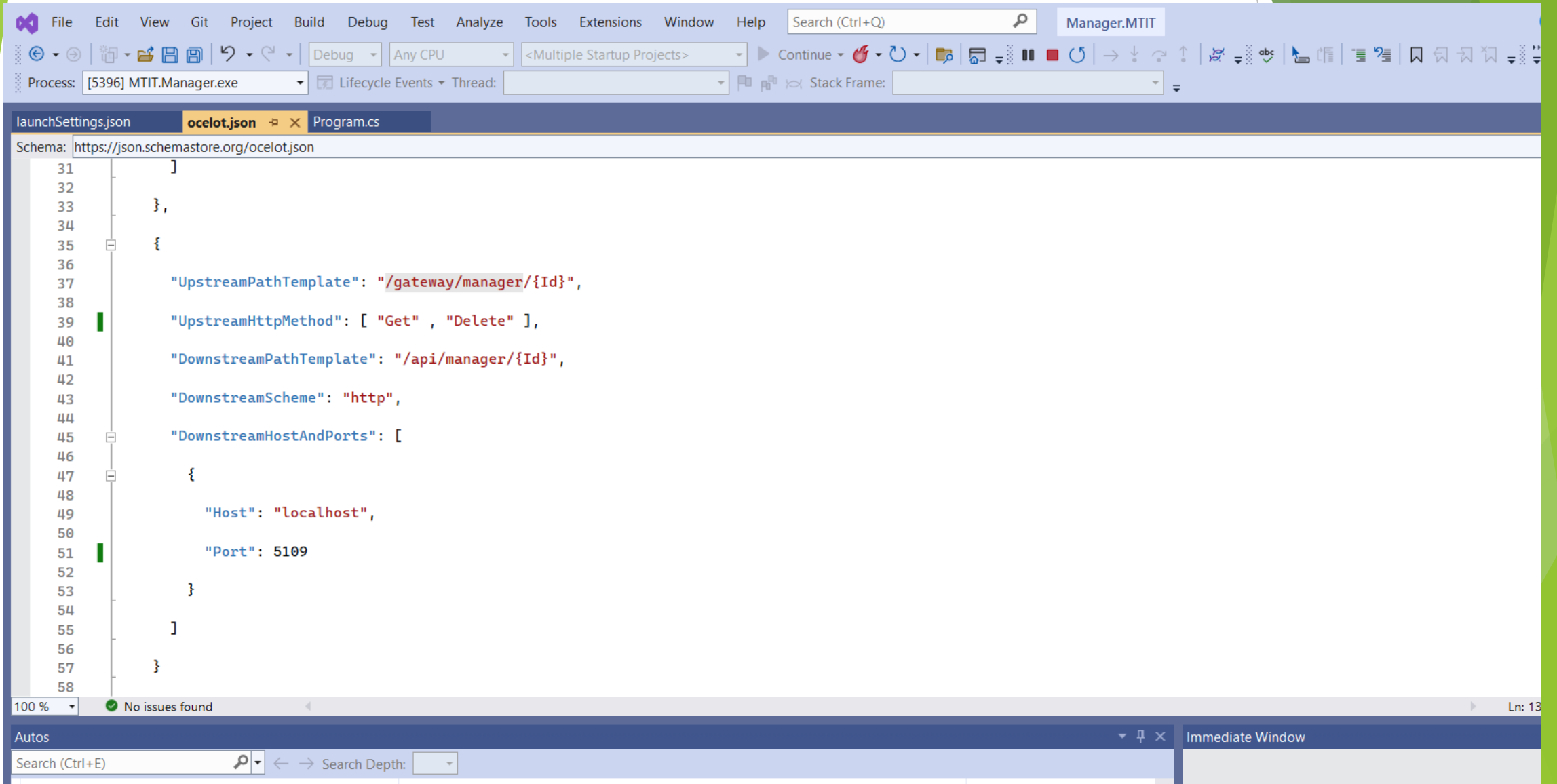
# Ocelot.json in Manager service

The screenshot shows the Visual Studio IDE with the **Manager.MTIT** project selected. The **ocelot.json** file is open in the editor, displaying its JSON configuration. The file is located at `https://json.schemastore.org/ocelot.json`. The configuration includes a **GlobalConfiguration** section with a **BaseUrl** of `http://localhost:5071` and a **Routes** section. The first route in the **Routes** array has an **UpstreamPathTemplate** of `/gateway/manager`, an **UpstreamHttpMethod** of `["Get", "Post", "Put"]`, a **DownstreamPathTemplate** of `/api/manager`, a **DownstreamScheme** of `http`, and a **DownstreamHostAndPorts** array containing a single object with **Host** `localhost` and **Port** `5109`.

```
1  {
2
3  "GlobalConfiguration": {
4      "BaseUrl": "http://localhost:5071"
5  },
6
7  "Routes": [
8      {
9          "UpstreamPathTemplate": "/gateway/manager",
10         "UpstreamHttpMethod": [ "Get" , "Post" , "Put" ],
11         "DownstreamPathTemplate": "/api/manager",
12         "DownstreamScheme": "http",
13         "DownstreamHostAndPorts": [
14             {
15                 "Host": "localhost",
16                 "Port": 5109
17             }
18         ]
19     }
20 ]
21 }
```

The bottom of the IDE shows the **Autos** window with a search bar and the **Immediate Window**.

## Ocelot.json in Manager service

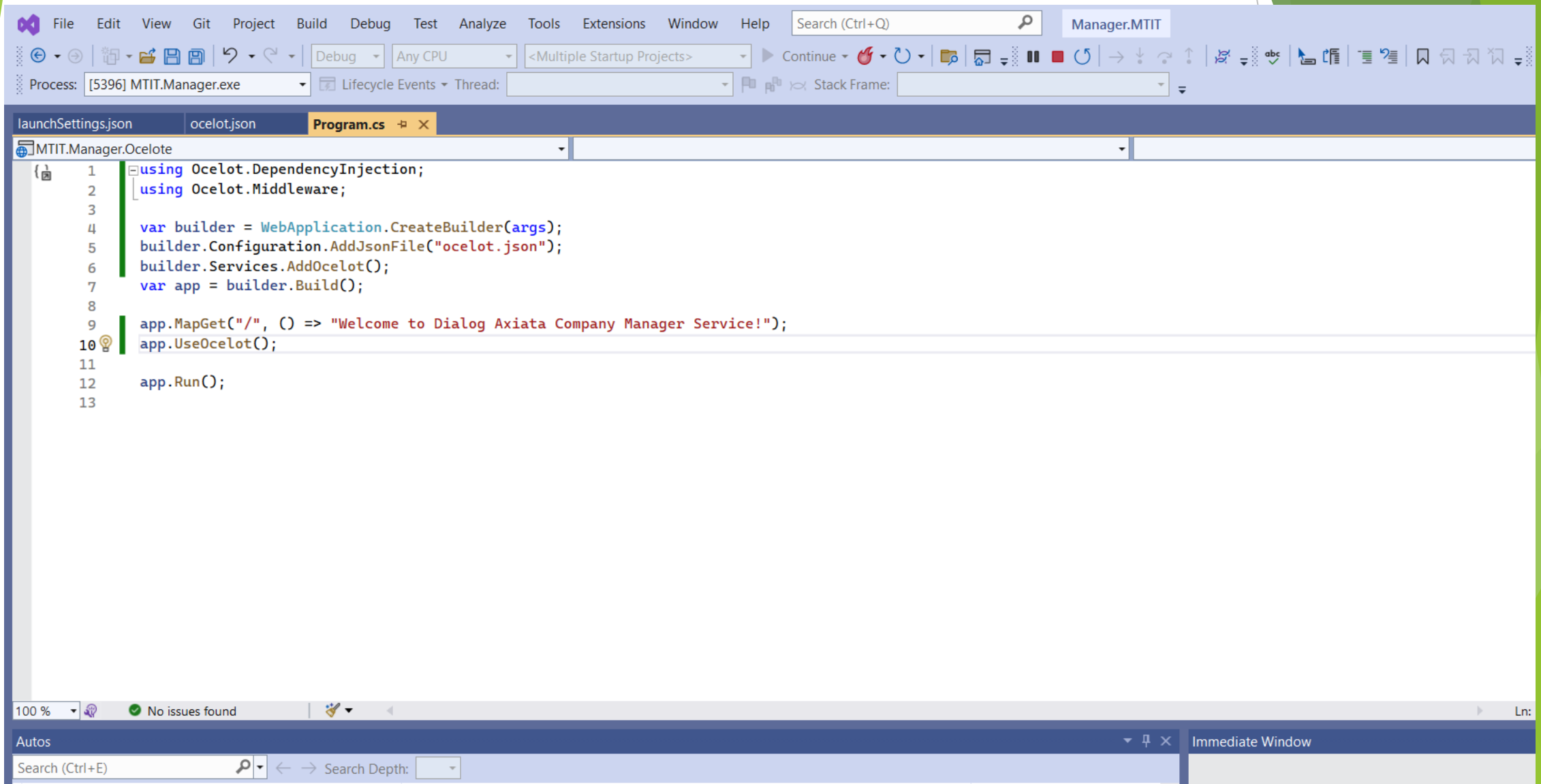


The screenshot displays the Visual Studio Code editor with the `ocelot.json` file open. The file is a JSON configuration for an API gateway, defining upstream and downstream paths, methods, and host/port information. The configuration is as follows:

```
31    ]
32
33    },
34
35    {
36
37      "UpstreamPathTemplate": "/gateway/manager/{Id}",
38
39      "UpstreamHttpMethod": [ "Get" , "Delete" ],
40
41      "DownstreamPathTemplate": "/api/manager/{Id}",
42
43      "DownstreamScheme": "http",
44
45      "DownstreamHostAndPorts": [
46
47        {
48
49          "Host": "localhost",
50
51          "Port": 5109
52
53        }
54
55      ]
56
57    }
58
```

The editor shows the file `ocelot.json` is selected, and the schema is `https://json.schemastore.org/ocelot.json`. The status bar at the bottom indicates "No issues found" and "Ln: 13".

## Program.cs in Manager service



The screenshot displays the Visual Studio IDE with the **Program.cs** file open in the **MTIT.Manager.Ocelote** project. The code is as follows:

```
1 using Ocelot.DependencyInjection;
2 using Ocelot.Middleware;
3
4 var builder = WebApplication.CreateBuilder(args);
5 builder.Configuration.AddJsonFile("ocelot.json");
6 builder.Services.AddOcelot();
7 var app = builder.Build();
8
9 app.MapGet("/", () => "Welcome to Dialog Axiata Company Manager Service!");
10 app.UseOcelot();
11
12 app.Run();
13
```

The interface includes a menu bar (File, Edit, View, Git, Project, Build, Debug, Test, Analyze, Tools, Extensions, Window, Help), a toolbar with search and execution icons, and a status bar at the bottom showing "100 %", "No issues found", and a search bar.

# Screen captures OF Manager API

The screenshot shows the Swagger UI for the MTIT.Manager v1.0 API. The browser address bar indicates the URL is `localhost:5109/swagger/index.html`. The Swagger logo and version information (1.0 OAS3) are displayed at the top left. A dropdown menu at the top right allows selecting the API definition, currently set to "MTIT.Manager v1".

The API endpoints are listed under two main sections: "Manager" and "WeatherForecast". Each endpoint is represented by a colored bar indicating the HTTP method and the endpoint path. The "Manager" section includes five endpoints: GET, POST, PUT, GET, and DELETE. The "WeatherForecast" section includes one endpoint: GET.

Method	Endpoint
GET	/api/Manager
POST	/api/Manager
PUT	/api/Manager
GET	/api/Manager/{id}
DELETE	/api/Manager/{id}
GET	/WeatherForecast

Below the endpoints, there is a section for "Schemas" which is partially visible at the bottom of the screenshot.

# GET Method

## Manager

GET /api/Manager

### Parameters

Cancel

No parameters

Execute

Clear

### Responses

#### Response body

```
[
  {
    "id": 1,
    "itemName": "Dialog Broadband",
    "itemQuantity": "15",
    "itemPrice": 200000
  },
  {
    "id": 2,
    "itemName": "Dialog TV",
    "itemQuantity": "40",
    "itemPrice": 22000
  },
  {
    "id": 3,
    "itemName": "Dialog Phone",
    "itemQuantity": "78",
    "itemPrice": 21000
  },
  {
    "id": 4,
    "itemName": "Dialog Satellite ",
    "itemQuantity": "32",
    "itemPrice": 23000
  },
  {
    "id": 5,
    "itemName": "Dialog Internet",
    "itemQuantity": "10",
    "itemPrice": 15000
  }
]
```



Download

# POST Method

**POST** /api/Manager

Parameters

No parameters

Cancel

Reset

Request body


application/json

```
{
  "id": 6,
  "itemName": "SIM",
  "itemQuantity": "30",
  "itemPrice": 700
}
```

Request URL

http://localhost:5109/api/Manager

Server response

Code	Details
200	<div><div>Response body</div><div><pre>{   "id": 6,   "itemName": "SIM",   "itemQuantity": "30",   "itemPrice": 700 }</pre><div> Download</div></div></div> <div><div>Response headers</div><div><pre>content-type: application/json; charset=utf-8 date: Wed, 12 Apr 2023 14:30:25 GMT server: Kestrel transfer-encoding: chunked</pre></div></div>

# PUT Method

**PUT** /api/Manager ^

Parameters

Cancel

Reset

No parameters

Request body


application/json v

```
{
  "id": 6,
  "itemName": "SIM",
  "itemQuantity": "44",
  "itemPrice": 890
}
```

Request URL

http://localhost:5109/api/Manager

Server response

Code	Details
200	<div>Response body<div><pre>{   "id": 6,   "itemName": "SIM",   "itemQuantity": "44",   "itemPrice": 890 }</pre><div> <div>Download</div></div></div></div>

# SEARCH

GET

/api/Manager/{id}

^

Parameters

Cancel

Name	Description
id * required	
integer(\$int32)	6
(path)	

Execute

Clear

Curl

```
curl -X 'GET' \
'http://localhost:5109/api/Manager/6' \
-H 'accept: */*'
```

Request URL

```
http://localhost:5109/api/Manager/6
```

Server response

Code	Details
200	<div><div>Response body</div><div><pre>{   "id": 6,   "itemName": "SIM",   "itemQuantity": "44",   "itemPrice": 890 }</pre></div><div><div></div><div>Download</div></div></div>



# DELETE Method

**DELETE** /api/Manager/{id} ^

Parameters Cancel

Name	Description
<b>id</b> * required	
integer(\$int32)	<input type="text" value="6"/>
(path)	

Execute

Clear

Curl

```
curl -X 'DELETE' \  
  'http://localhost:5109/api/Manager/6' \  
  -H 'accept: */*'
```

Request URL

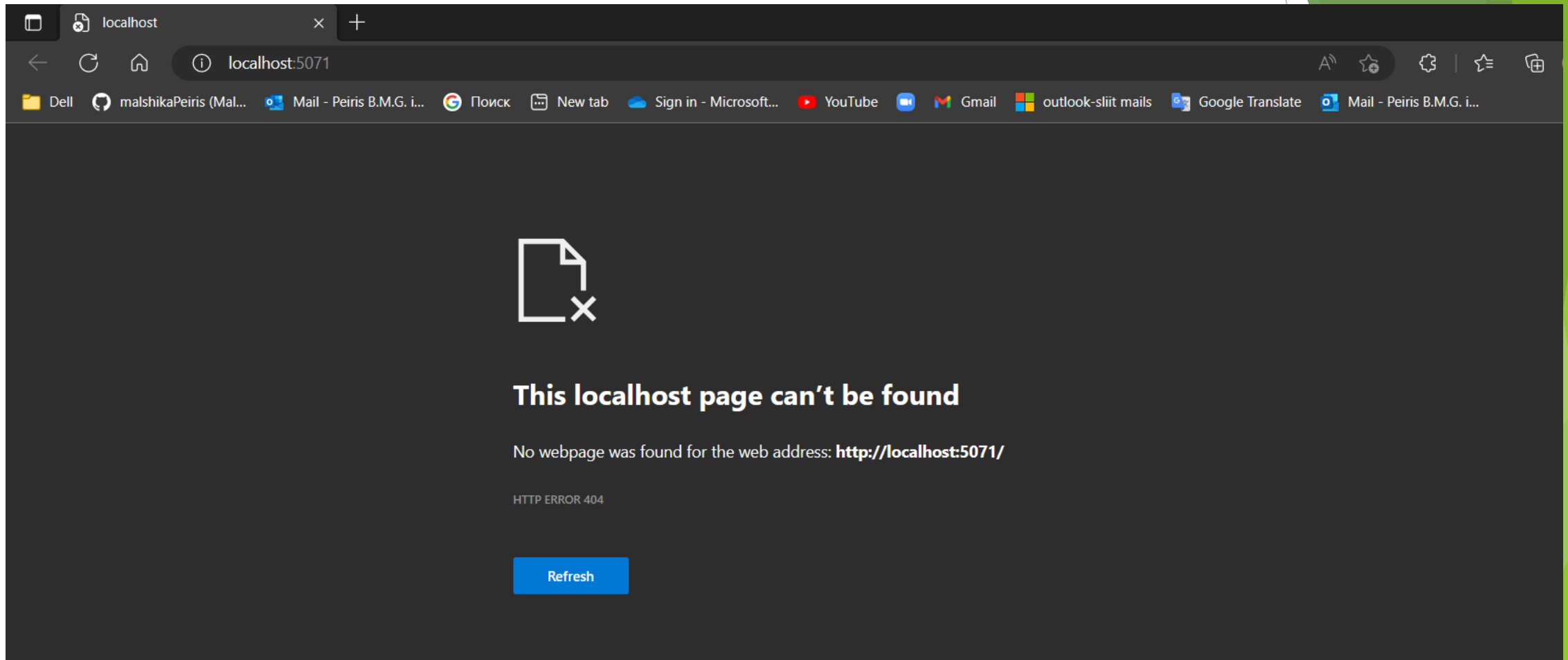
```
http://localhost:5109/api/Manager/6
```

Server response

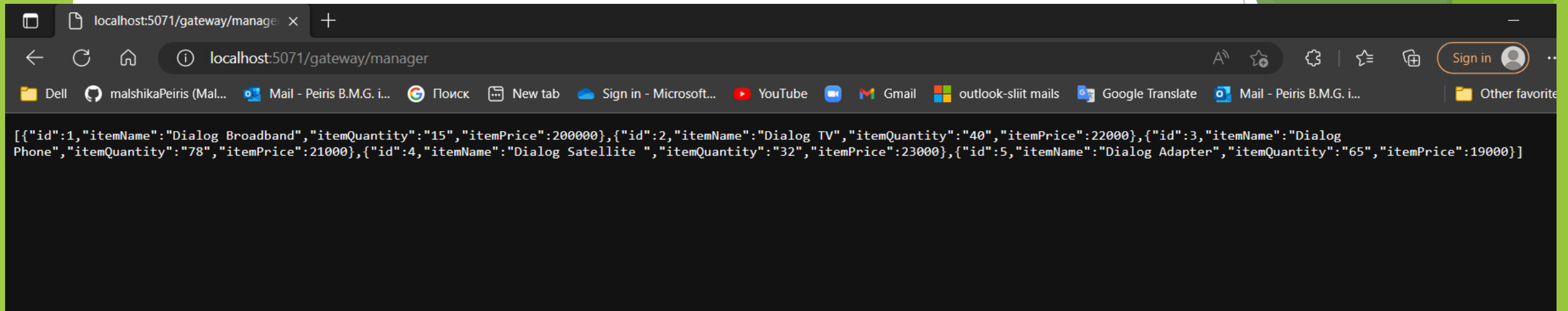
Code	Details
200	<div>Response body</div> <div>manager with ID:6 got deleted successfully.</div> <div><span>Download</span></div>

Gateway project and the API project will both run and open up in two separate browsers.

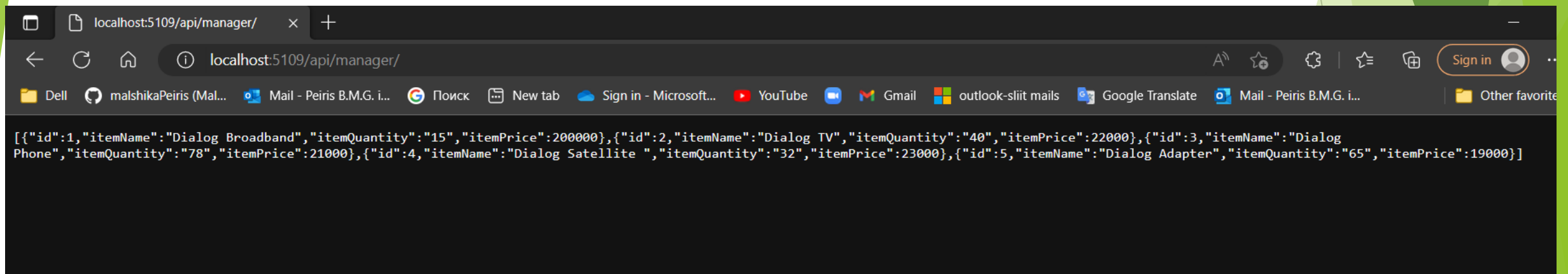
**Gateway:**



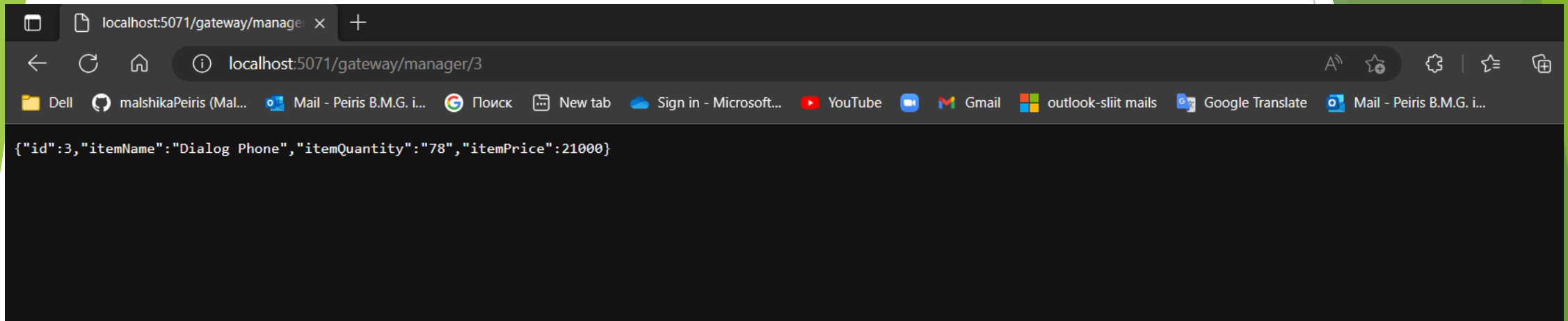
Test the upstream URL in browser.



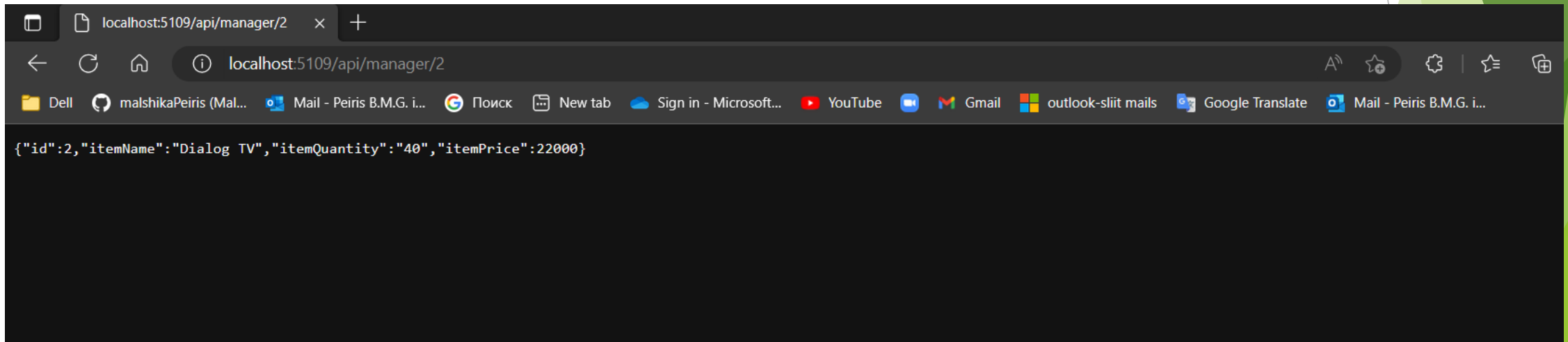
Test the downstream URL in browser.



Test the upstream URL in browser.



Test the downstream URL in browser.



**IT20122782**

**Amani M P N**

**Store Keeper Service**

## StoreKeeper Service

Elaborate how you can avoid having multiple ports with the gateway you implemented and Solution should have proper folder structure with minimum two microservices and an Ocelot API Gateway.

**launchsettings.json in StoreKeeper service**

The screenshot displays the Visual Studio IDE with the `launchSettings.json` file open for the `MTIT.Storekeeper` service. The file is configured with the following settings:

```
1 {
2   "$schema": "https://json.schemastore.org/launchsettings.json",
3   "iisSettings": {
4     "windowsAuthentication": false,
5     "anonymousAuthentication": true,
6     "iisExpress": {
7       "applicationUrl": "http://localhost:48409",
8       "sslPort": 0
9     }
10  },
11  "profiles": {
12    "MTIT.Storekeeper": {
13      "commandName": "Project",
14      "dotnetRunMessages": true,
15      "launchBrowser": true,
16      "launchUrl": "swagger",
17      "applicationUrl": "http://localhost:5177",
18      "environmentVariables": {
19        "ASPNETCORE_ENVIRONMENT": "Development"
20      }
21    },
22    "IIS Express": {
23      "commandName": "IISExpress",
24      "launchBrowser": true,
25      "launchUrl": "swagger",
26      "environmentVariables": {
27        "ASPNETCORE_ENVIRONMENT": "Development"
28      }
29    }
30  }
31 }
```

The Solution Explorer on the right shows the project structure for the `MTIT.Supplier` solution, which includes the `Gateway` and `Microservices` folders. The `Microservices` folder contains the `MTIT.Storekeeper` project, which is the current context of the `launchSettings.json` file.

## Ocelot.json in StoreKeeper service

The screenshot displays the Visual Studio IDE with the `ocelot.json` file open in the `MTIT.Ocelote` project. The file is a JSON configuration for Ocelot, defining two API routes. The first route maps the upstream path `/gateway/storekeeper` to the downstream path `/api/storekeeper`. The second route maps the upstream path `/gateway/storekeeper/{Id}` to the downstream path `/api/storekeeper/{Id}`. Both routes use the HTTP scheme and are hosted on `localhost:5177`.

```
132  "UpstreamPathTemplate": "/gateway/storekeeper",
133
134  "UpstreamHttpMethod": [ "Get", "Post", "Put" ],
135
136  "DownstreamPathTemplate": "/api/storekeeper",
137
138  "DownstreamScheme": "http",
139
140  "DownstreamHostAndPorts": [
141
142    {
143      "Host": "localhost",
144      "Port": 5177
145    }
146  ],
147
148  },
149
150  {
151    "UpstreamPathTemplate": "/gateway/storekeeper/{Id}",
152    "UpstreamHttpMethod": [ "Get", "Delete" ],
153    "DownstreamPathTemplate": "/api/storekeeper/{Id}",
154    "DownstreamScheme": "http",
155    "DownstreamHostAndPorts": [
156      {
157        "Host": "localhost",
158        "Port": 5177
159      }
160    ]
161  }
162
163  ],
164
165  }
```

The Solution Explorer on the right shows the project structure for `MTIT.Supplier` (5 of 5 projects). The `Gateway` folder contains `MTIT.Ocelote`, which has `Connected Services`, `Dependencies`, `Properties`, `appsettings.json`, `ocelot.json`, and `Program.cs`. The `Microservices` folder contains `MTIT.Manager`, `MTIT.Storekeeper`, `MTIT.Supplier.Mobile`, and `MTIT.Transport`.

# Screen captures OF Storekeeper API

The screenshot shows a web browser window displaying the Swagger UI for the MTIT.Storekeeper v1 API. The browser's address bar shows the URL `localhost:5003/swagger/index.html`. The Swagger UI header includes the Swagger logo, the text "Supported by SMARTBEAR", and a dropdown menu labeled "Select a definition" with "MTIT.Storekeeper v1" selected. Below the header, the API title "MTIT.Storekeeper" is displayed with a "1.0" version tag and an "OAS3" specification tag. The URL `http://localhost:5003/swagger/v1/swagger.json` is also shown. The main content area lists the API endpoints under two sections: "Storekeeper" and "WeatherForecast". The "Storekeeper" section contains five endpoints: a GET endpoint for `/api/Storekeeper`, a POST endpoint for `/api/Storekeeper`, a GET endpoint for `/api/Storekeeper/{id}`, a PUT endpoint for `/api/Storekeeper/{id}`, and a DELETE endpoint for `/api/Storekeeper/{id}`. The "WeatherForecast" section contains one endpoint: a GET endpoint for `/WeatherForecast`. Each endpoint is represented by a colored bar with the HTTP method on the left and the endpoint path on the right. A vertical scrollbar is visible on the right side of the page.

Swagger UI  
Supported by SMARTBEAR

Select a definition MTIT.Storekeeper v1

**MTIT.Storekeeper** 1.0 OAS3  
<http://localhost:5003/swagger/v1/swagger.json>

**Storekeeper**

- GET `/api/Storekeeper`
- POST `/api/Storekeeper`
- GET `/api/Storekeeper/{id}`
- PUT `/api/Storekeeper/{id}`
- DELETE `/api/Storekeeper/{id}`

**WeatherForecast**

- GET `/WeatherForecast`



# GET Method

Swagger UI

localhost:5003/swagger/index.html

## Responses

Curl

```
curl -X 'GET' \
  'http://localhost:5003/api/Storekeeper' \
  -H 'accept: */*'
```

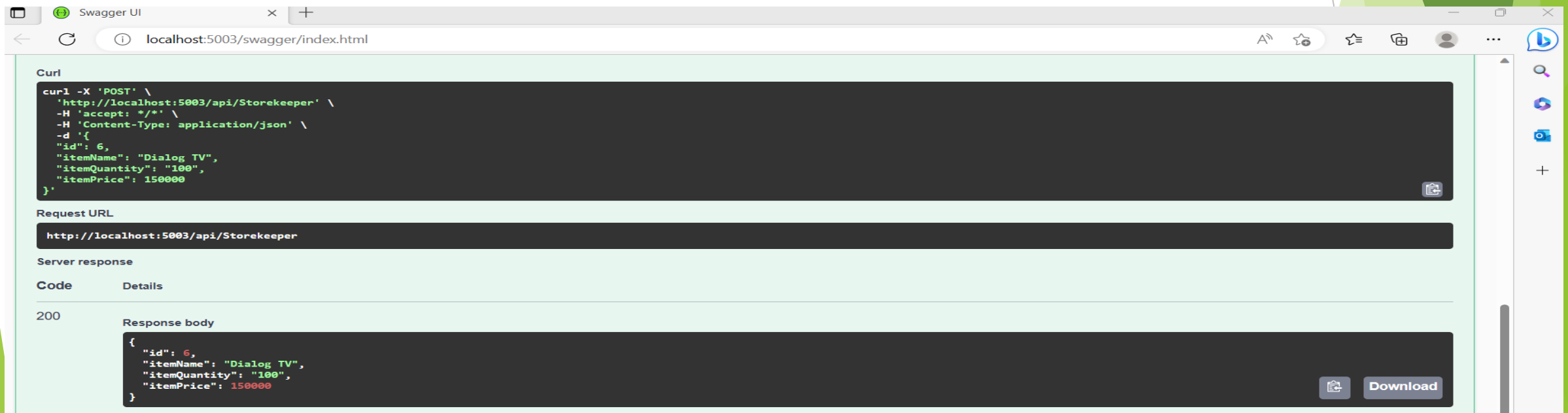
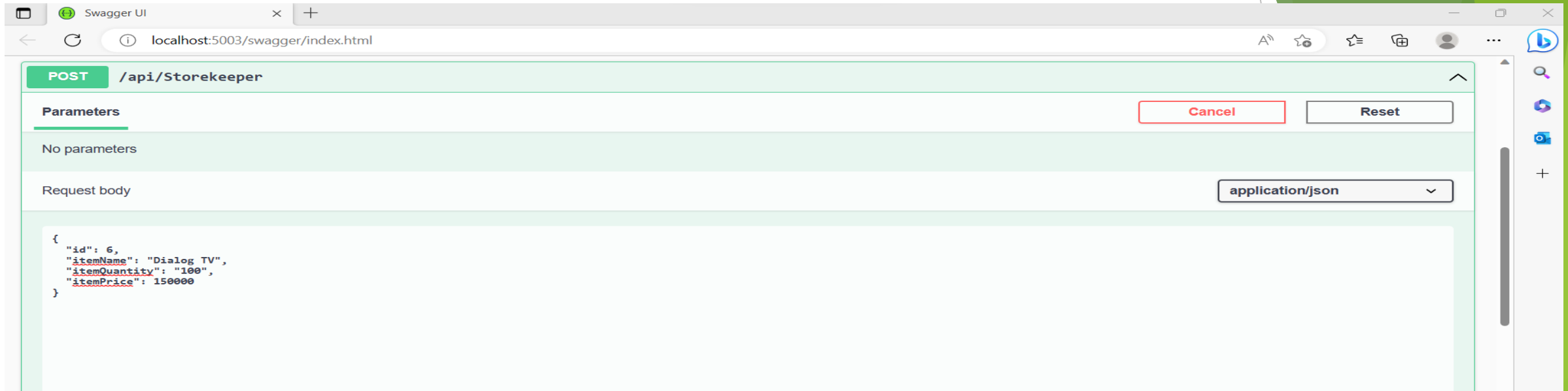
Request URL

```
http://localhost:5003/api/Storekeeper
```

Server response

Code	Details
200	<p>Response body</p> <pre>{   "id": 1,   "itemName": "Dialog Adapter",   "itemQuantity": "25",   "itemPrice": 25000 }, {   "id": 2,   "itemName": "Dialog Phone",   "itemQuantity": "40",   "itemPrice": 20000 }, {   "id": 3,   "itemName": "Dialog Satellite",   "itemQuantity": "30",   "itemPrice": 24000 }, {   "id": 4,   "itemName": "Dialog Broadband ",   "itemQuantity": "100",   "itemPrice": 18000 }, { </pre>

# POST Method



# PUT Method

Swagger UI

localhost:5003/swagger/index.html

**PUT** /api/Storekeeper/{id}

Parameters

Cancel Reset

Name	Description
<b>id</b> * required string (path)	<input type="text" value="2"/>

Request body

application/json

```
{  "id": 2,  "itemName": "Dialog Phone",  "itemQuantity": "500",  "itemPrice": 200000}
```

Swagger UI

localhost:5003/swagger/index.html

**Responses**

Curl

```
curl -X 'PUT' \  'http://localhost:5003/api/Storekeeper/2' \  -H 'accept: */*' \  -H 'Content-Type: application/json' \  -d '{  "id": 2,  "itemName": "Dialog Phone",  "itemQuantity": "500",  "itemPrice": 200000  }'
```

Request URL

http://localhost:5003/api/Storekeeper/2

Server response

Code	Details
200	<p>Response body</p> <pre>{  "id": 2,  "itemName": "Dialog Phone",  "itemQuantity": "500",  "itemPrice": 200000}</pre> <p>Download</p>

# SEARCH

Swagger UI

localhost:5003/swagger/index.html

GET /api/Storekeeper/{id}

Parameters

Cancel

Name	Description
id * required	
integer(\$int32)	6
(path)	

Swagger UI

localhost:5003/swagger/index.html

Curl

```
curl -X 'GET' \
'http://localhost:5003/api/Storekeeper/6' \
-H 'accept: */*'
```

Request URL

```
http://localhost:5003/api/Storekeeper/6
```

Server response

Code	Details
200	<p>Response body</p> <pre>{   "id": 6,   "itemName": "Dialog TV",   "itemQuantity": "100",   "itemPrice": 150000 }</pre> <p>Download</p>

# DELETE Method

Swagger UI

localhost:5003/swagger/index.html

**DELETE** /api/Storekeeper/{id}

Parameters

Cancel

Name	Description
id * required	
integer(\$int32)	
(path)	

6

Swagger UI

localhost:5003/swagger/index.html

Responses

Curl

```
curl -X 'DELETE' \
  'http://localhost:5003/api/Storekeeper/6' \
  -H 'accept: */*'
```

Request URL

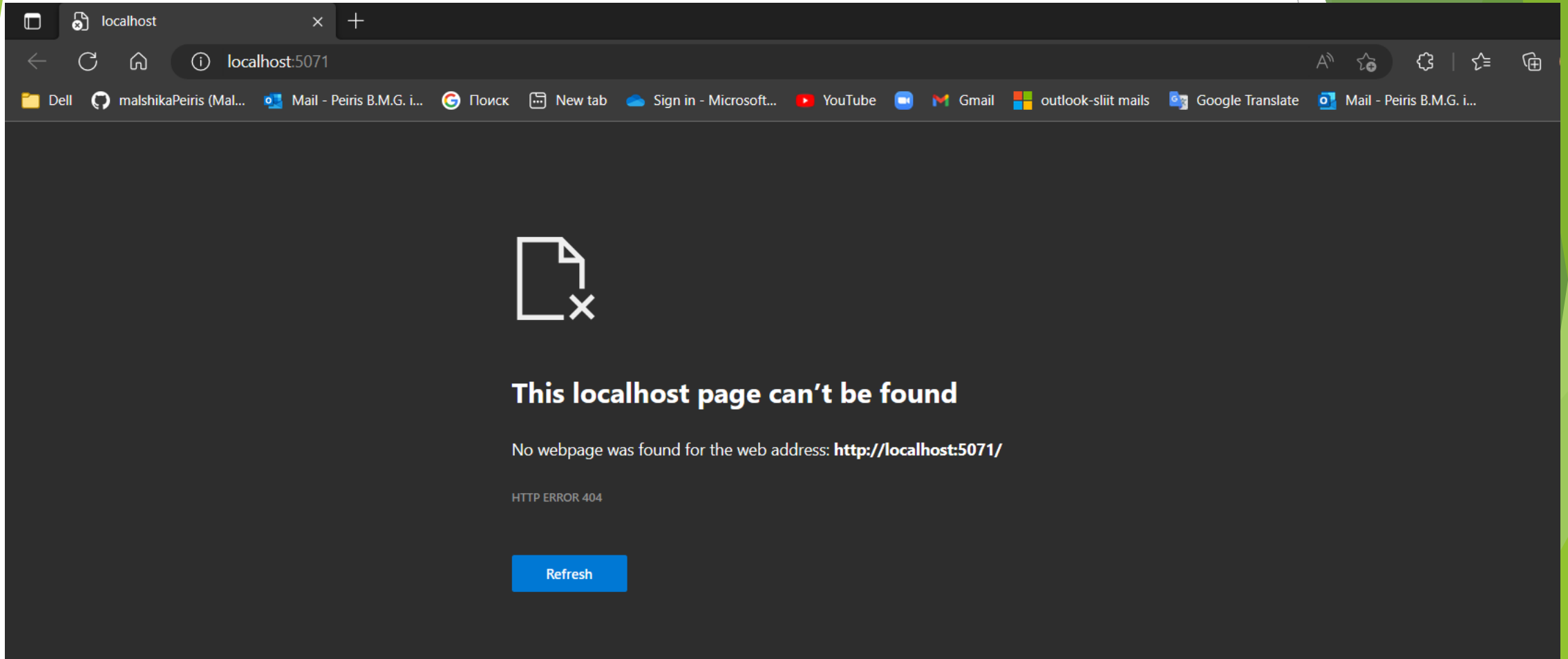
```
http://localhost:5003/api/Storekeeper/6
```

Server response

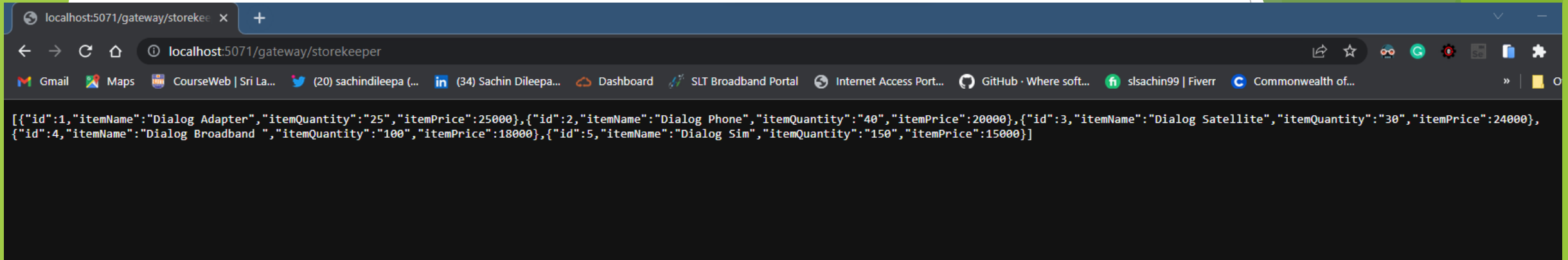
Code	Details
200	<p>Response body</p> <pre>storekeeper with ID:6 got deleted successfully.</pre> <p>Download</p>

Gateway project and the API project will both run and open up in two separate browsers.

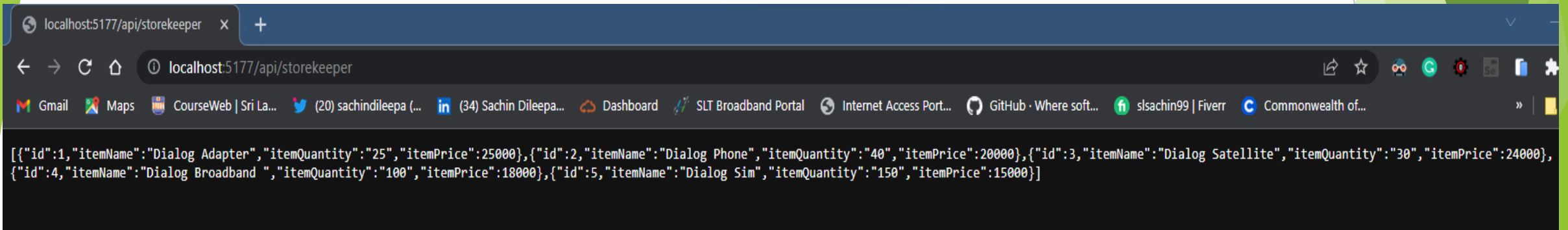
Gateway:



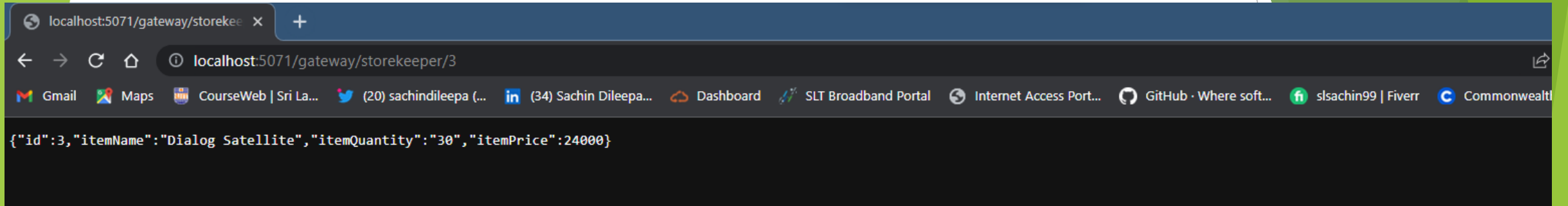
Test the upstream URL in browser.



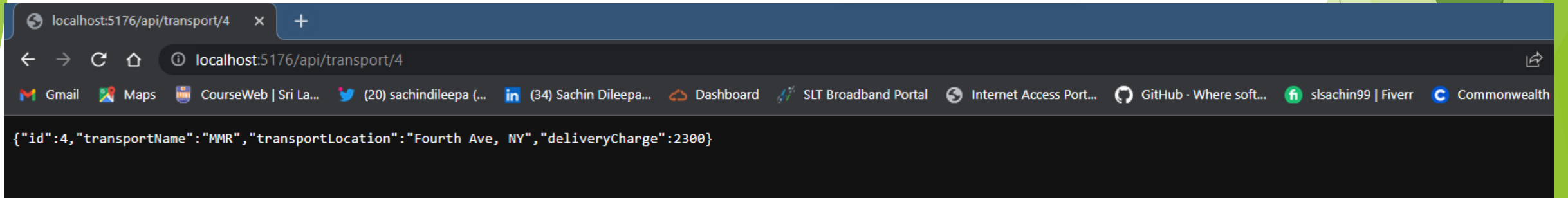
Test the downstream URL in browser.



Test the upstream URL in browser.



Test the downstream URL in browser.





**IT20081416**

**Ahamed M.M.Z**

**Transport Service**

## Transport Service

Elaborate how you can avoid having multiple ports with the gateway you implemented and Solution should have proper folder structure with minimum two microservices and an Ocelot API Gateway.

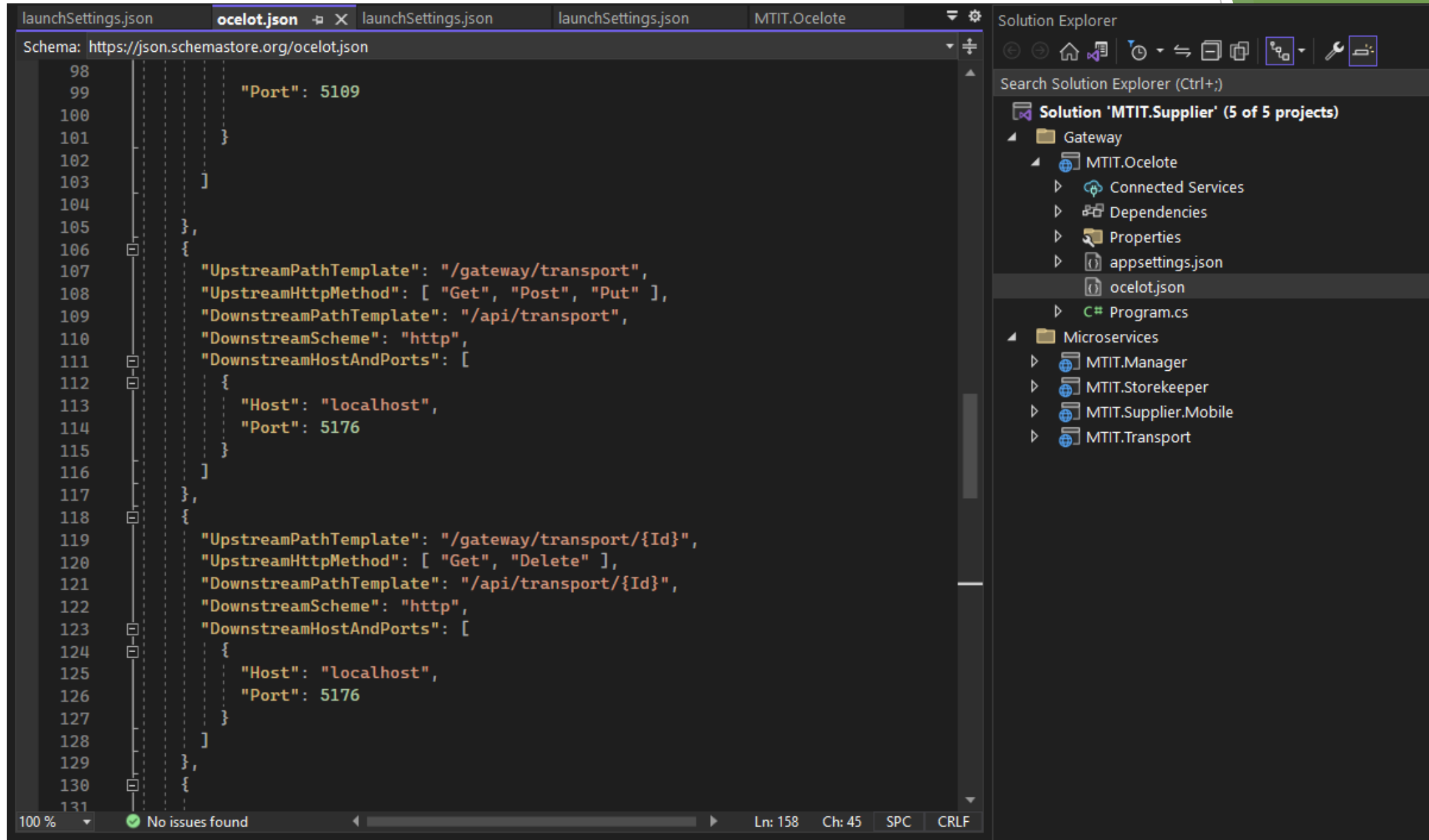
### launchsettings.json in Transport service

The screenshot displays the Visual Studio IDE with the `launchSettings.json` file open for the `MTIT.Transport` service. The file is configured with the following JSON:

```
{
  "$schema": "https://json.schemastore.org/launchsettings.json",
  "iisSettings": {
    "windowsAuthentication": false,
    "anonymousAuthentication": true,
    "iisExpress": {
      "applicationUrl": "http://localhost:40573",
      "sslPort": 0
    }
  },
  "profiles": {
    "MTIT.Transport": {
      "commandName": "Project",
      "dotnetRunMessages": true,
      "launchBrowser": true,
      "launchUrl": "swagger",
      "applicationUrl": "http://localhost:5176",
      "environmentVariables": {
        "ASPNETCORE_ENVIRONMENT": "Development"
      }
    },
    "IIS Express": {
      "commandName": "IISExpress",
      "launchBrowser": true,
      "launchUrl": "swagger",
      "environmentVariables": {
        "ASPNETCORE_ENVIRONMENT": "Development"
      }
    }
  }
}
```

The Solution Explorer on the right shows the project structure for the `MTIT.Supplier` solution, which includes the `Gateway` and `Microservices` folders. The `Microservices` folder contains the `MTIT.Transport` service, which is the current context of the `launchSettings.json` file.

## Ocelot.json in Transport service



The screenshot displays the Visual Studio IDE with the `ocelot.json` file open in the editor. The file is part of the `MTIT.Ocelote` project within the `MTIT.Supplier` solution. The JSON content defines two routes for the transport service.

```
Schema: https://json.schemastore.org/ocelot.json
98
99     "Port": 5109
100
101   }
102
103   ],
104
105   },
106   {
107     "UpstreamPathTemplate": "/gateway/transport",
108     "UpstreamHttpMethod": [ "Get", "Post", "Put" ],
109     "DownstreamPathTemplate": "/api/transport",
110     "DownstreamScheme": "http",
111     "DownstreamHostAndPorts": [
112       {
113         "Host": "localhost",
114         "Port": 5176
115       }
116     ]
117   },
118   {
119     "UpstreamPathTemplate": "/gateway/transport/{Id}",
120     "UpstreamHttpMethod": [ "Get", "Delete" ],
121     "DownstreamPathTemplate": "/api/transport/{Id}",
122     "DownstreamScheme": "http",
123     "DownstreamHostAndPorts": [
124       {
125         "Host": "localhost",
126         "Port": 5176
127       }
128     ]
129   },
130   {
131
```

The Solution Explorer on the right shows the project structure:

- Solution 'MTIT.Supplier' (5 of 5 projects)
  - Gateway
    - MTIT.Ocelote
      - Connected Services
      - Dependencies
      - Properties
      - appsettings.json
      - ocelot.json
      - C# Program.cs
  - Microservices
    - MTIT.Manager
    - MTIT.Storekeeper
    - MTIT.Supplier.Mobile
    - MTIT.Transport

The status bar at the bottom indicates 100% zoom, no issues found, and the current position is Ln: 158, Ch: 45.

# Screen captures OF Transpoart API

The screenshot shows the Swagger UI interface for the MTIT.Assignment3.Transport API. The browser address bar indicates the URL is localhost:5176/swagger/index.html. The Swagger logo and version 1.0 OAS3 are visible. The API title is MTIT.Assignment3.Transport. The URL for the swagger.json file is http://localhost:5176/swagger/v1/swagger.json. The API is categorized under 'Transport'. Five endpoints are listed: GET /api/Transport, POST /api/Transport, PUT /api/Transport, GET /api/Transport/{id}, and DELETE /api/Transport/{id}. A 'Schemas' section is partially visible at the bottom, showing a 'Transport' schema.

Swagger UI

localhost:5176/swagger/index.html

Swagger  
Supported by SMARTBEAR

Select a definition MTIT.Assignment3.Transport v1

## MTIT.Assignment3.Transport <sup>1.0</sup> OAS3

<http://localhost:5176/swagger/v1/swagger.json>

### Transport

- GET /api/Transport
- POST /api/Transport
- PUT /api/Transport
- GET /api/Transport/{id}
- DELETE /api/Transport/{id}

### Schemas

Transport >

# GET Method

The screenshot displays the Swagger UI interface for a REST API. The browser address bar shows the URL `localhost:5176/swagger/index.html`. The main interface is titled `GET /api/Transport`.

**Parameters**

No parameters

**Execute** **Clear**

**Responses**

**Curl**

```
curl -X 'GET' \
'http://localhost:5176/api/Transport' \
-H 'accept: */*'
```

**Request URL**

```
http://localhost:5176/api/Transport
```

**Server response**

**Code** **Details**

200

**Response body**

```
[
  {
    "id": 1,
    "transportName": "LLC",
    "transportLocation": "Main St, California",
    "deliveryCharge": 2000
  },
  {
    "id": 2,
    "transportName": "JBC",
    "transportLocation": "Second Ave, LA",
    "deliveryCharge": 2200
  },
  {
    "id": 3,
    "transportName": "HCB",
    "transportLocation": "Third St, Vegas",
    "deliveryCharge": 2100
  },
  {
    "id": 4,
    "transportName": "MMR",
    "transportLocation": "Fourth Ave, NY",
    "deliveryCharge": 2300
  }
]
```

# POST Method

Swagger UI

localhost:5176/swagger/index.html

POST /api/Transport

Parameters

No parameters

Request body

application/json

```
{  "id": 6,  "transportName": "New Transport",  "transportLocation": "Sixth lane, NJ",  "deliveryCharge": 5600}
```

Execute

Clear

Responses

Curl

# PUT Method

Swagger UI

localhost:5176/swagger/index.html

PUT /api/Transport

Parameters

No parameters

Request body

application/json

```
{
  "id": 1,
  "transportName": "MSC",
  "transportLocation": "New lane, LA",
  "deliveryCharge": 620
}
```

Execute

Clear

Responses

# Search

GET

/api/Transport/{id}

Parameters

Cancel

Name	Description
<b>id</b> * required	
integer(\$int32)	2
(path)	

Execute

Clear

Responses

Curl

```
curl -X 'GET' \  
  'http://localhost:5176/api/Transport/2' \  
  -H 'accept: */*'
```

Request URL

```
http://localhost:5176/api/Transport/2
```

Server response

Code	Details
200	<div><div>Response body</div><div><pre>{   "id": 2,   "transportName": "JBC",   "transportLocation": "Second Ave, LA",   "deliveryCharge": 2200 }</pre></div><div><div></div><div>Download</div></div></div>



# DELETE Method

Swagger UI

localhost:5176/swagger/index.html

**DELETE** /api/Transport/{id}

Parameters

Cancel

Name	Description
id * required	
integer(\$int32)	1
(path)	

ExecuteClear

Swagger UI

localhost:5176/swagger/index.html

Responses

Curl

```
curl -X 'DELETE' \
  'http://localhost:5176/api/Transport/1' \
  -H 'accept: */*'

```

Request URL

```
http://localhost:5176/api/Transport/1

```

Server response

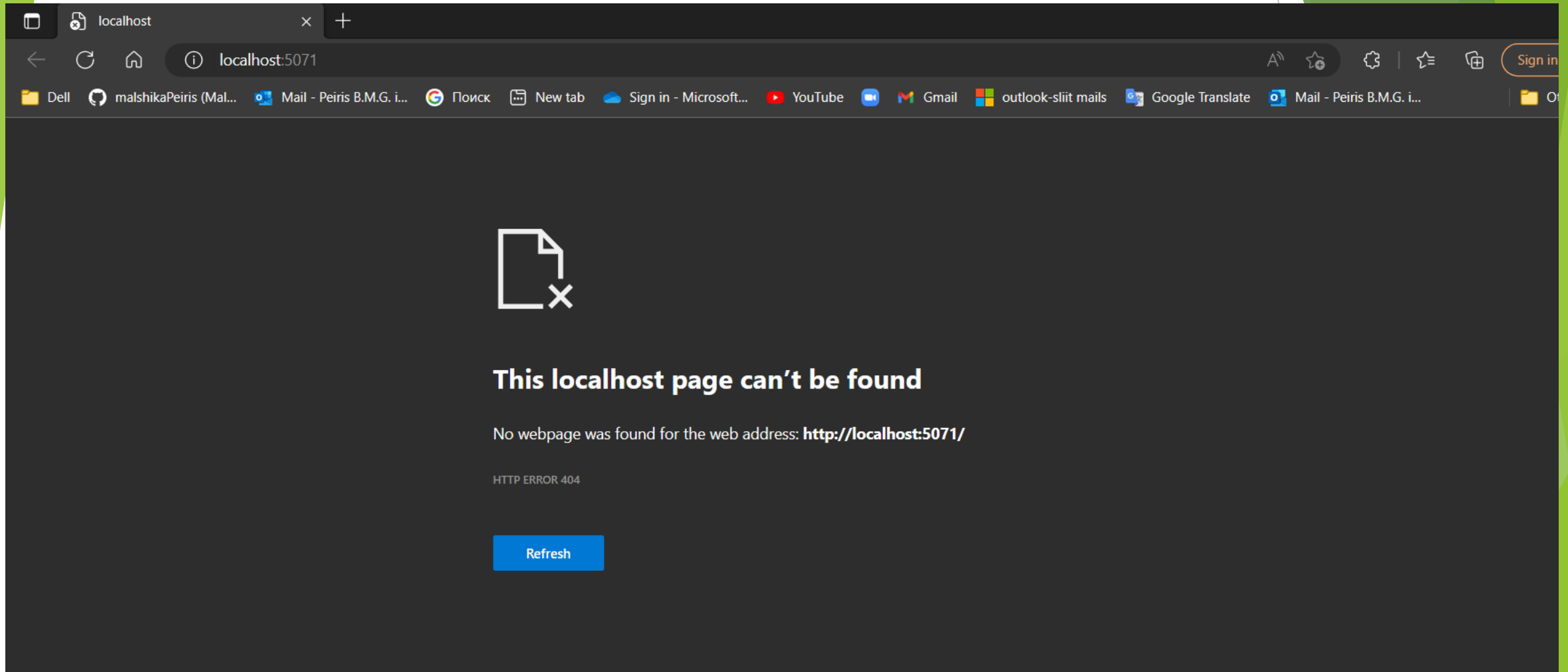
Code	Details
200	<div>Response body</div> <div>Transport ID: 1 got deleted successfully.</div> <div>Response headers</div> <div>Content-type: text/plain; charset=utf-8 date: Sat, 15 Apr 2023 09:48:43 GMT server: Kestrel transfer-encoding: chunked</div>

Responses

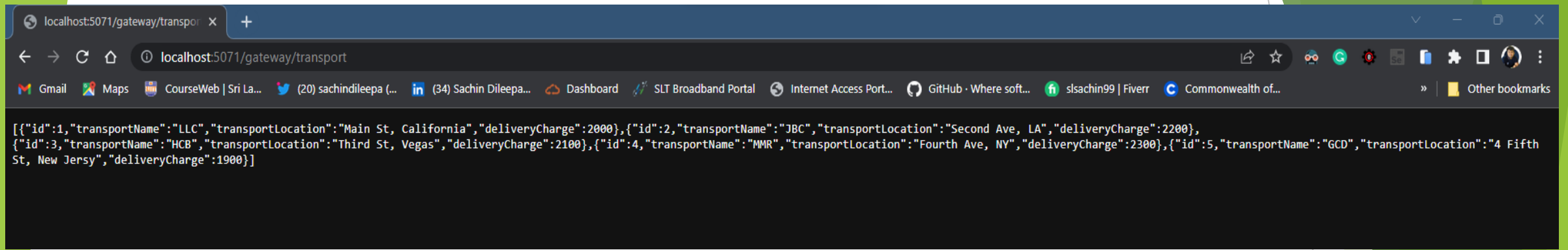
Code	Description	Links
200	Success	No links

Gateway project and the API project will both run and open up in two separate browsers.

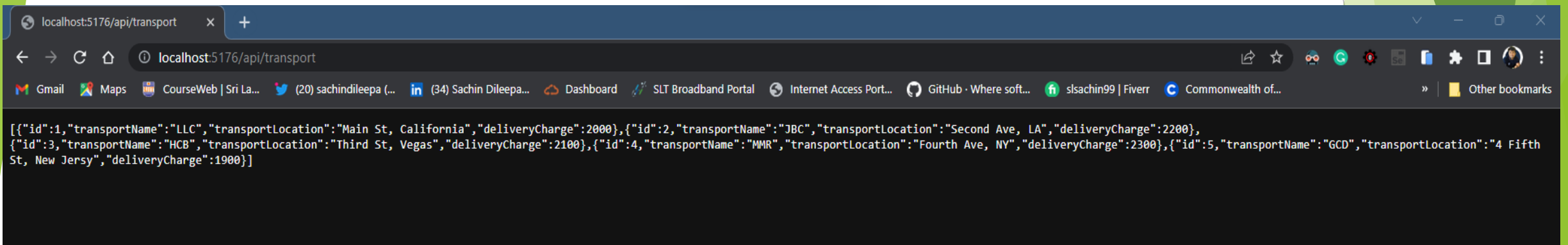
Gateway:



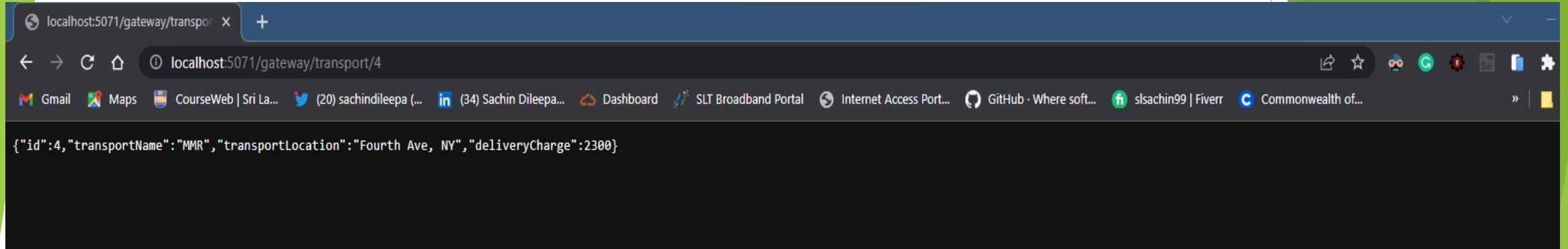
Test the upstream URL in browser.



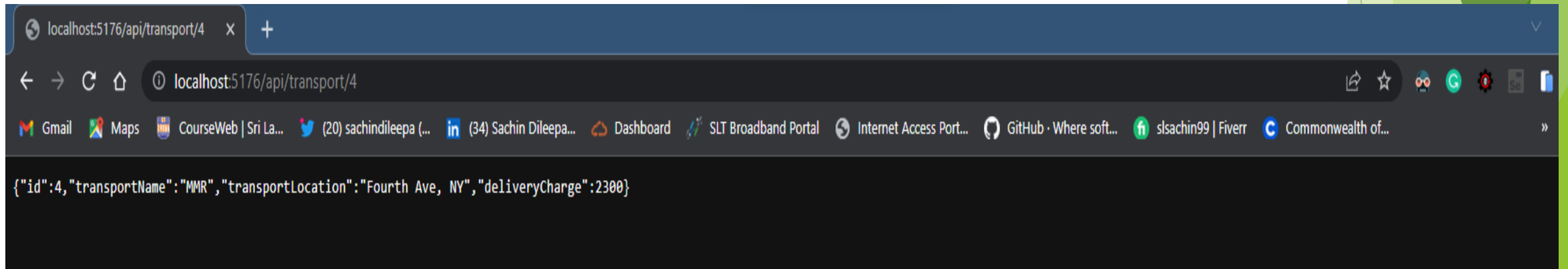
Test the downstream URL in browser.



Test the upstream URL in browser.



Test the downstream URL in browser.



## REFERENCES

ASP.NET Core Web API CRUD With Entity Framework - Full Course = <https://youtu.be/3NWT9k-6xGg>

ASP.NET Core Web API + Entity Framework Core : Database First - EP01= <https://youtu.be/CLVJVA9cTuU>

ASP.NET Core Web API + Entity Framework Core : REST API Methods - EP03 = <https://youtu.be/uPJ2-6YPRjM>

ASP.NET CORE Crud operation using Entity framework using swagger= <https://youtu.be/RsaUVrg9aNo>