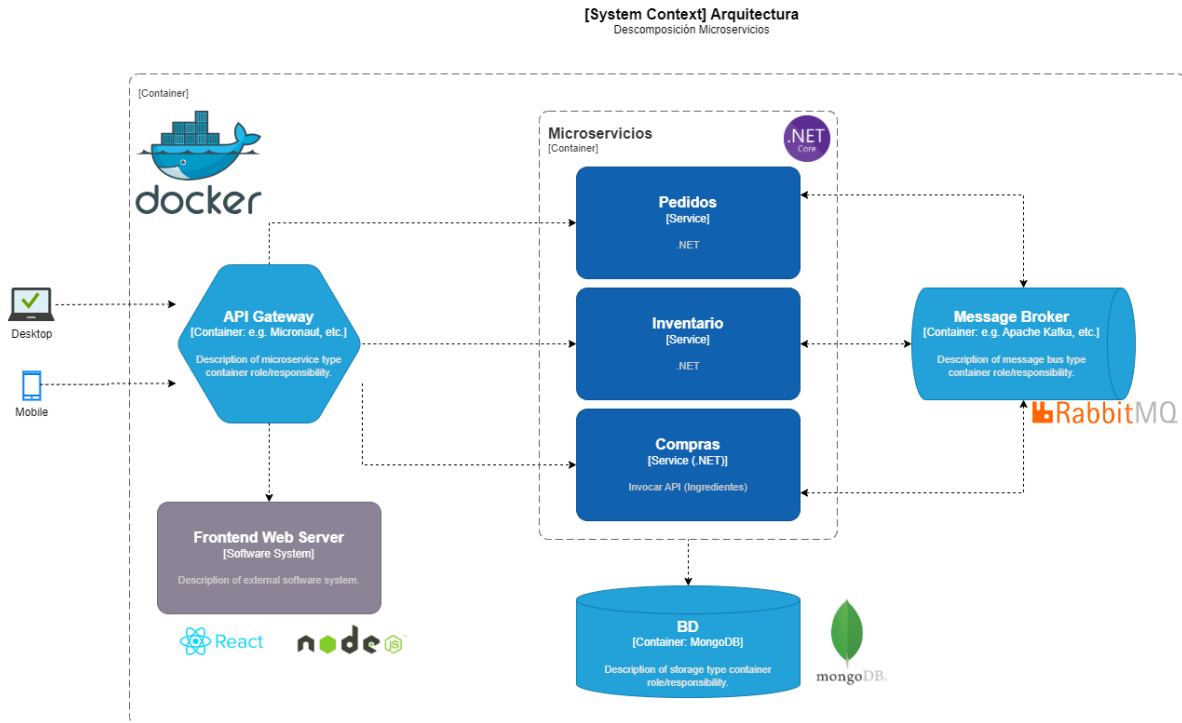


ARQUITECTURA DE SOFTWARE (2S-2023)  
PROYECTO CORTE 2 (MICROSERVICIOS)  
EDISSON CABRERA ERASO

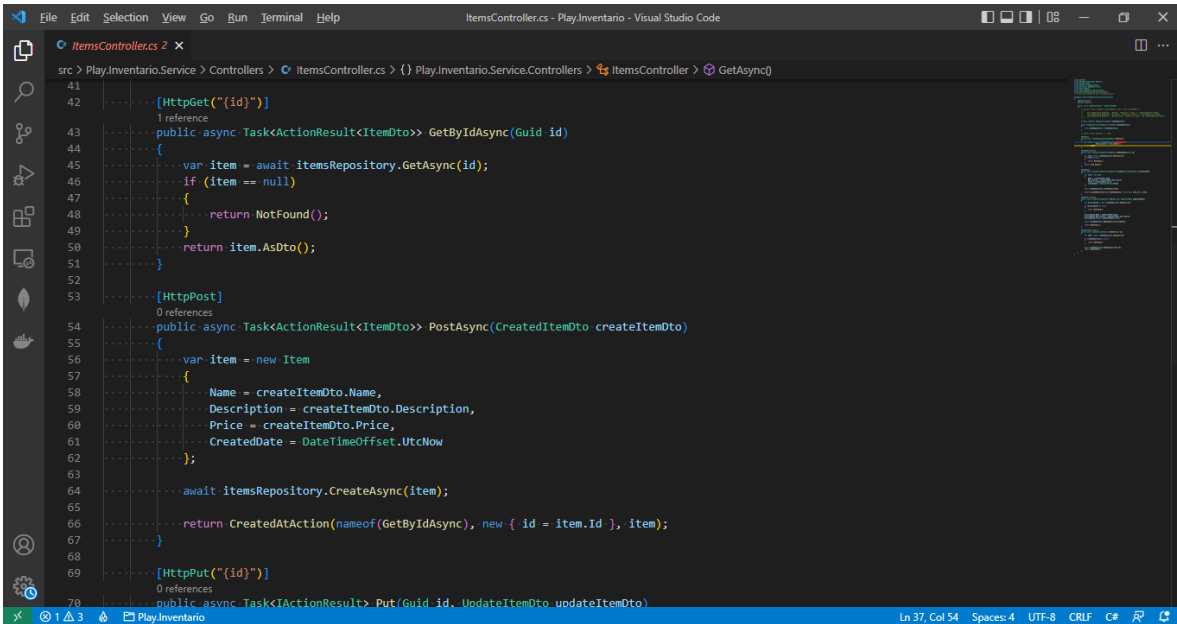
## Tabla de contenido

Arquitectura Propuesta .....	2
API's .....	2
MongoDB.....	3
Docker Compose (yml) .....	3
Iniciar Docker Mongo .....	4
Comunicación asíncrona entre microservicios.....	5
Message Broker (RabbitMQ + MassTransit) .....	5
Enviar mensaje mediante RabbitMQ .....	9
RabbitMQ Console .....	9
Enviar y recibir mensajes entre microservicios a través de RabbitMQ .....	10
Frontend (React).....	10
Hosted (Node.js) Web Server .....	11
CORS (Cross-Origin Resource Sharing) .....	11
Para la comunicación entre el frontend y los microservicios .....	11
Consumir API plaza de mercado desde API propia .....	12
Interconexión entre microservicios.....	13
Recetas .....	13
Deploy App NETLIFY .....	15
GitHub Repositorio .....	16

## Arquitectura Propuesta



## API's

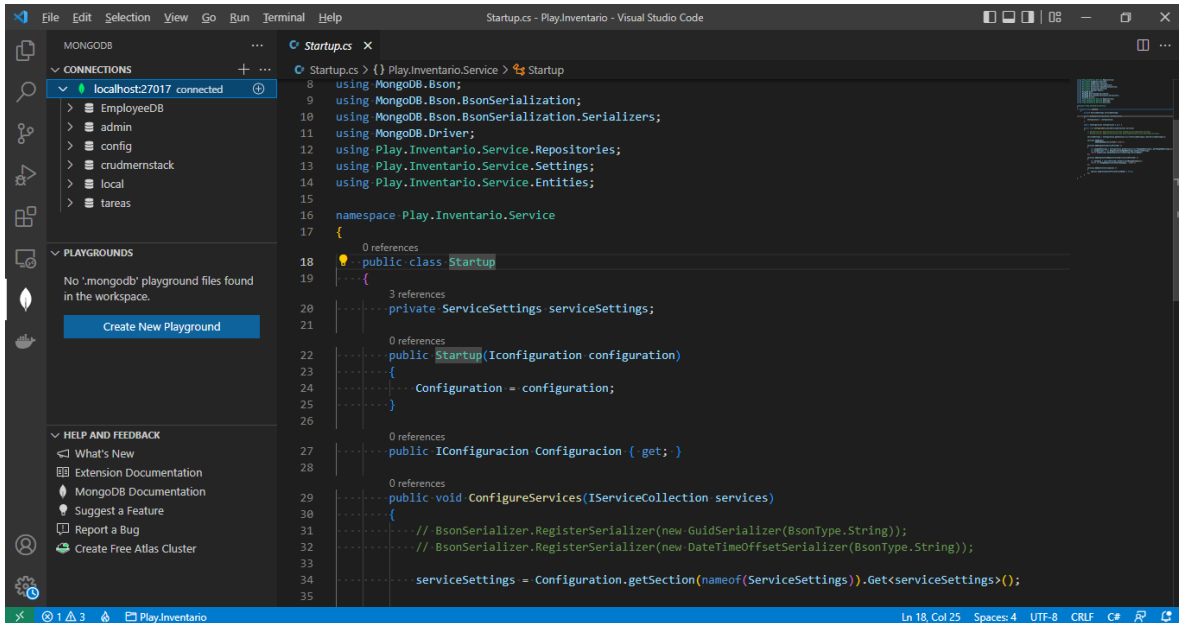


The screenshot shows the Visual Studio Code editor with the file `ItemsController.cs` open. The code is written in C# and implements an API for managing items. The code is as follows:

```

1  File Edit Selection View Go Run Terminal Help
2  ItemsController.cs - Play.Inventario - Visual Studio Code
3
4  src > Play.Inventario.Service > Controllers > ItemsController.cs > {} Play.Inventario.Service.Controllers > ItemsController > GetAsync()
5
6  41
7  42 [HttpGet("{id}")]
8  43     1 reference
9  44     public async Task<ActionResult<ItemDto>> GetByIdAsync(Guid id)
10 45     {
11 46         var item = await itemsRepository.GetAsync(id);
12 47         if (item == null)
13 48         {
14 49             return NotFound();
15 50         }
16 51         return item.AsDto();
17 52     }
18 53
19 54 [HttpPost]
20 55     0 references
21 56     public async Task<ActionResult<ItemDto>> PostAsync(CreatedItemDto createItemDto)
22 57     {
23 58         var item = new Item
24 59         {
25 60             Name = createItemDto.Name,
26 61             Description = createItemDto.Description,
27 62             Price = createItemDto.Price,
28 63             CreatedDate = DateTimeOffset.UtcNow
29 64         };
30 65         await itemsRepository.CreateAsync(item);
31 66         return CreatedAtAction(nameof(GetByIdAsync), new { id = item.Id }, item);
32 67     }
33 68
34 69 [HttpPut("{id}")]
35 70     0 references
36 71     public async Task<ActionResult> Put(Guid id, UpdateItemDto updateItemDto)
37
38  Ln 37, Col 54  Spaces: 4  UTF-8  CRLF  C#
  
```

## MongoDB

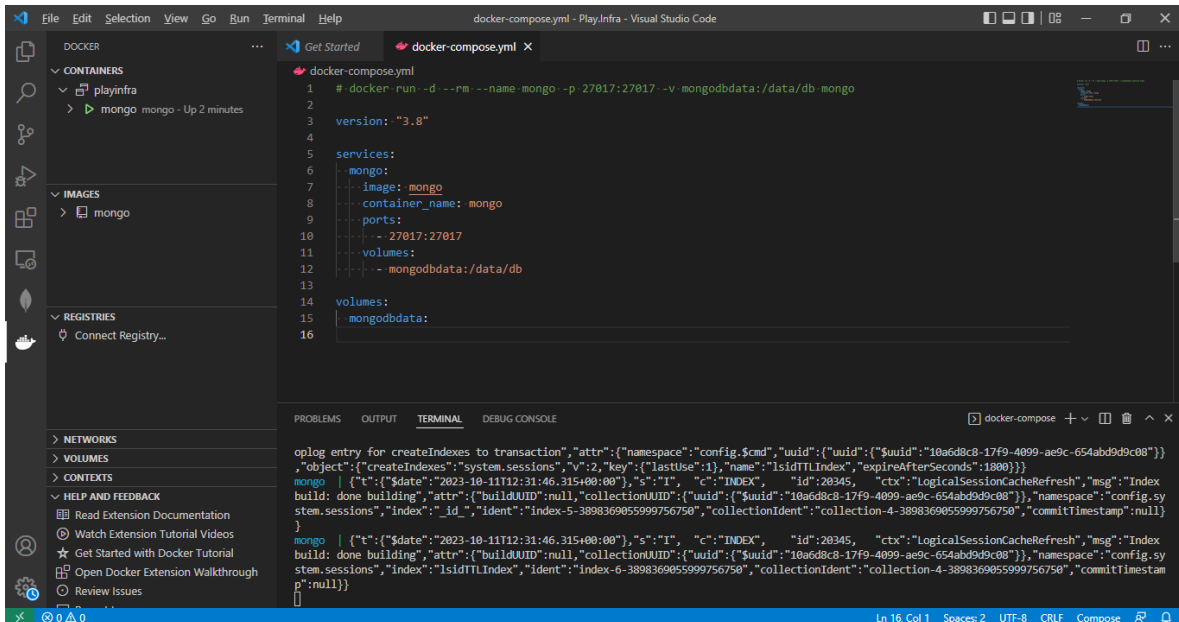


The screenshot shows the Visual Studio Code interface. On the left, the 'MongoDB' sidebar is open, showing a connection to 'localhost:27017' with a database named 'EmployeeDB'. The main editor displays the 'Startup.cs' file for the 'Play.Inventario' project. The code includes MongoDB drivers and a 'Startup' class that configures services and registers serializers.

```

Startup.cs
Startup.cs (Play.Inventario.Service) > Startup
8  using MongoDB.Bson;
9  using MongoDB.Bson.BsonSerialization;
10 using MongoDB.Bson.BsonSerialization.Serializers;
11 using MongoDB.Driver;
12 using Play.Inventario.Service.Repositories;
13 using Play.Inventario.Service.Settings;
14 using Play.Inventario.Service.Entities;
15
16 namespace Play.Inventario.Service
17 {
18     0 references
19     public class Startup
20     {
21         3 references
22         private ServiceSettings serviceSettings;
23
24         0 references
25         public Startup(IConfiguration configuration)
26         {
27             Configuration = configuration;
28         }
29
30         0 references
31         public IConfiguration Configuration { get; }
32
33         0 references
34         public void ConfigureServices(IServiceCollection services)
35         {
36             // BsonSerializer.RegisterSerializer(new GuidSerializer(BsonType.String));
37             // BsonSerializer.RegisterSerializer(new DateTimeOffsetSerializer(BsonType.String));
38             serviceSettings = Configuration.GetSection(nameof(ServiceSettings)).Get<ServiceSettings>();
39         }
40     }
  
```

## Docker Compose (yaml)



The screenshot shows the Visual Studio Code interface with a Docker Compose file named 'docker-compose.yml'. The file defines a 'mongo' service using the 'mongo' image, with a container name, ports, and a volume for the database data. The terminal at the bottom shows the output of the 'docker-compose up' command, indicating that the service is running successfully.

```

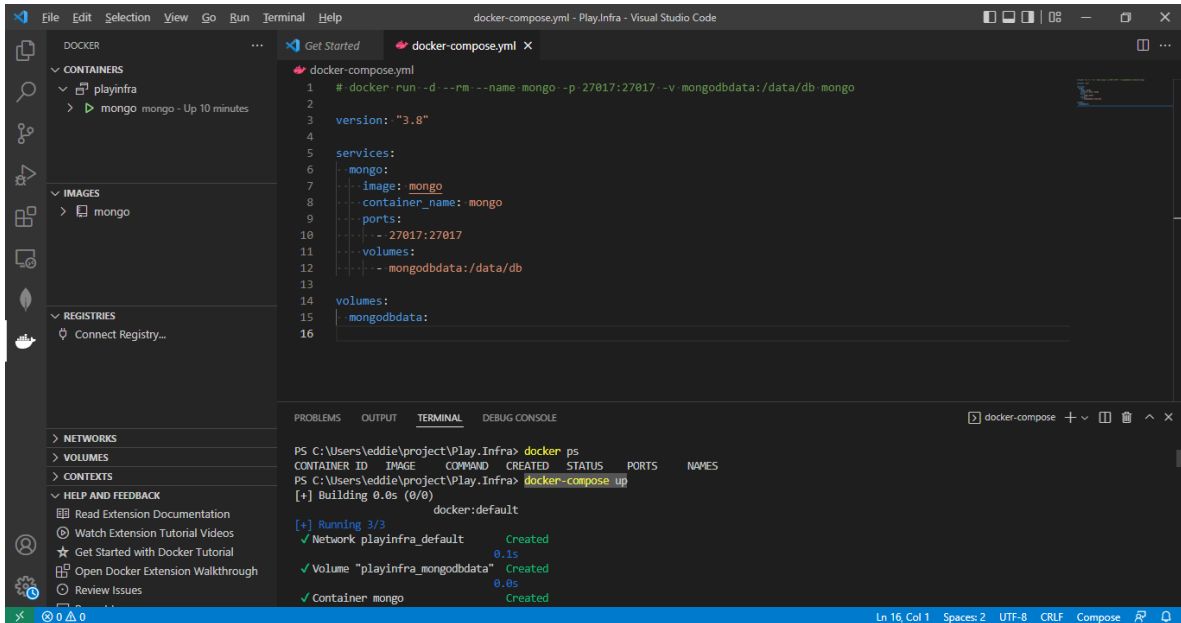
docker-compose.yml
1 # docker run -d --rm --name mongo -p 27017:27017 -v mongodbddata:/data/db mongo
2
3 version: "3.8"
4
5 services:
6   mongo:
7     image: mongo
8     container_name: mongo
9     ports:
10      - 27017:27017
11     volumes:
12      - mongodbddata:/data/db
13
14 volumes:
15   mongodbddata:
16
  
```

Terminal Output:

```

oplog entry for createIndexes to transaction, "attr": {"namespace": "config.$cmd", "uid": {"uid": {"$uid": "10a6d8c8-17f9-4099-ae9c-654abd9d9c08"}}}, "object": {"createIndexes": "system.sessions", "v": 2, "key": {"lastUse": 1}, "name": "lsidTTLIndex", "expireAfterSeconds": 1800}}
mongo | {"t":{"date":"2023-10-11T12:31:46.315400Z"},"s":"I", "c":"INDEX", "id":20345, "ctx":"LogicalSessionCacheRefresh","msg":"Index build: done building", "attr":{"buildUUID":null,"collectionUUID":{"uid":{"$uid":"10a6d8c8-17f9-4099-ae9c-654abd9d9c08"}}, "namespace":"config.system.sessions", "index":{"id":"_id","ident":"Index-5-389836985999756750","collectionIdent":"collection-4-389836985999756750","commitTimestamp":null}}
mongo | {"t":{"date":"2023-10-11T12:31:46.315400Z"},"s":"I", "c":"INDEX", "id":20345, "ctx":"LogicalSessionCacheRefresh","msg":"Index build: done building", "attr":{"buildUUID":null,"collectionUUID":{"uid":{"$uid":"10a6d8c8-17f9-4099-ae9c-654abd9d9c08"}}, "namespace":"config.system.sessions", "index":{"lsidTTLIndex", "ident":"Index-6-389836985999756750","collectionIdent":"collection-4-389836985999756750","commitTimestamp":null}}
  
```

## Iniciar Docker Mongo

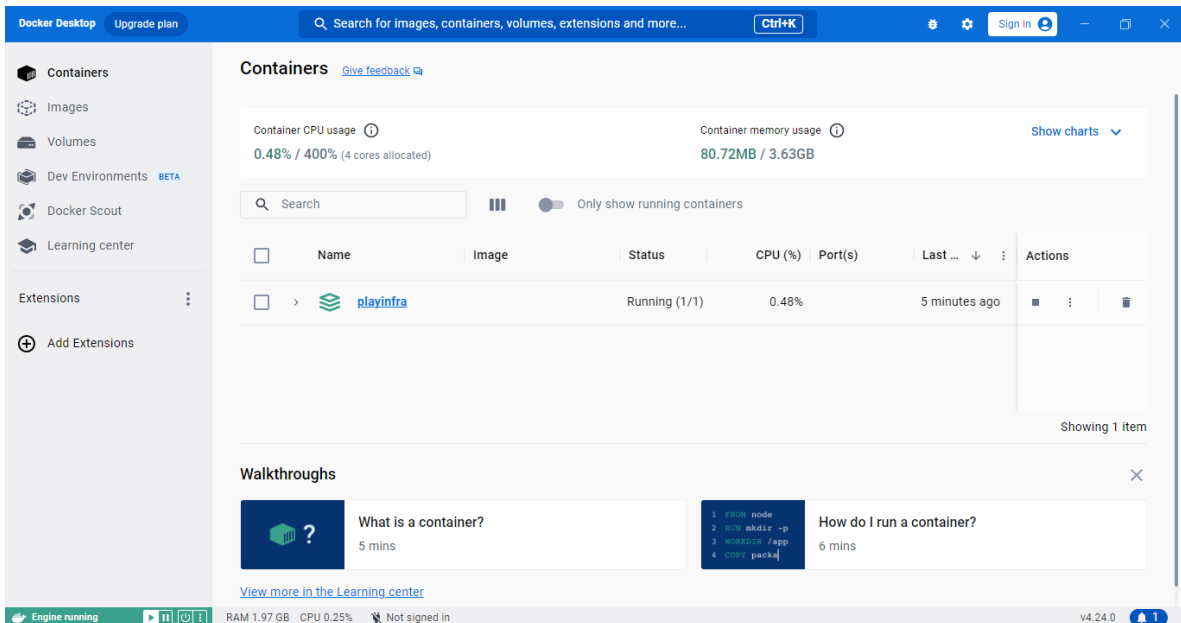


The screenshot shows the Visual Studio Code editor with the `docker-compose.yml` file open. The file contains the following configuration:

```
1 # docker: run -d --rm --name mongo -p 27017:27017 --v mongodbddata:/data/db mongo
2
3 version: "3.8"
4
5 services:
6   mongo:
7     image: mongo
8     container_name: mongo
9     ports:
10      - 27017:27017
11     volumes:
12      - mongodbddata:/data/db
13
14 volumes:
15   mongodbddata:
```

The terminal at the bottom shows the output of the `docker-compose up` command:

```
PS C:\Users\eddie\project\Play.Infra> docker ps
CONTAINER ID   IMAGE      COMMAND                  CREATED        STATUS        PORTS        NAMES
PS C:\Users\eddie\project\Play.Infra> docker-compose up
[+] Building 0.0s (0/0)
docker:default
[+] Running 3/3
  ✓ Network playinfra_default      Created
  ✓ Volume "playinfra_mongodbddata" Created
  ✓ Container mongo                Created
```



The screenshot shows the Docker Desktop interface. The left sidebar contains navigation options: Containers, Images, Volumes, Dev Environments (BETA), Docker Scout, and Learning center. The main area displays the 'Containers' tab with a search bar and a toggle for 'Only show running containers'.

Below the search bar, there is a table of containers:

Name	Image	Status	CPU (%)	Port(s)	Last ...	Actions
> playinfra		Running (1/1)	0.48%		5 minutes ago	Stop, Restart, Delete

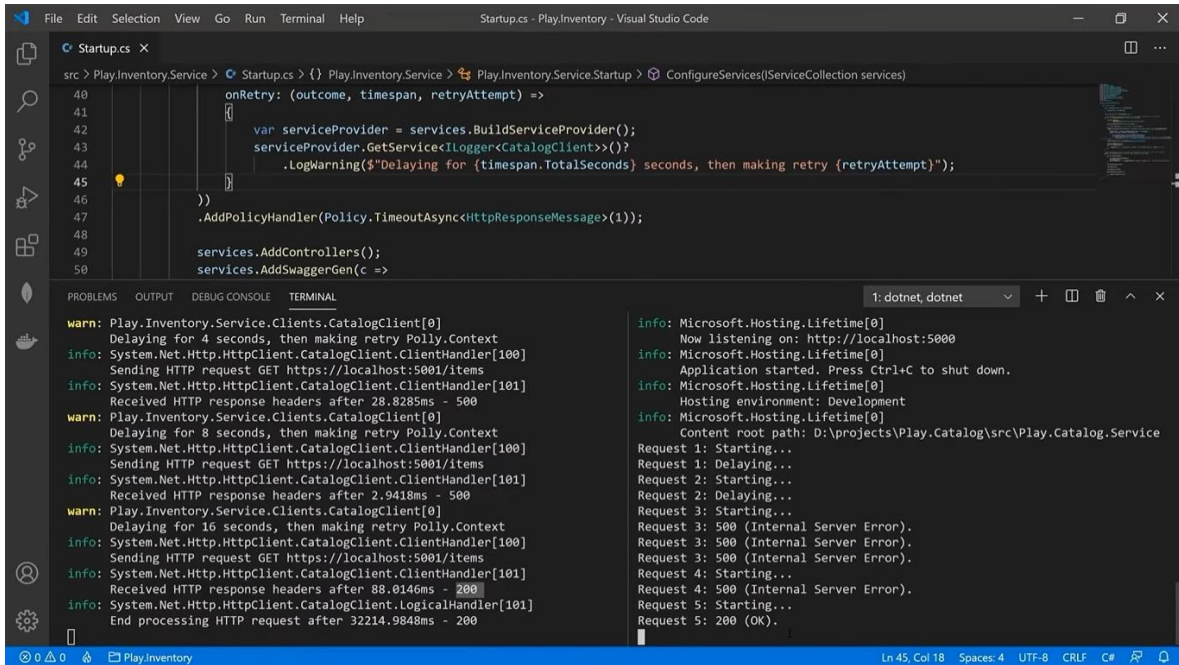
Below the table, it says 'Showing 1 item'.

At the bottom, there is a 'Walkthroughs' section with two cards:

- What is a container?** (5 mins)
- How do I run a container?** (6 mins)

The bottom status bar shows 'Engine running', 'RAM 1.97 GB', 'CPU 0.25%', and 'Not signed in'.

## Comunicación asíncrona entre microservicios



The screenshot shows the Visual Studio Code editor with the `Startup.cs` file open. The code defines a `ConfigureServices` method for `Play.Inventory.Service`. It includes a `retryPolicy` for `HttpClient` calls to `https://localhost:5001/items`. The terminal output shows the application starting, listening on `http://localhost:5000`, and making several HTTP requests. The first request is delayed for 4 seconds, and the second is delayed for 8 seconds. The third request is delayed for 16 seconds and results in a 500 (Internal Server Error). The fourth request is delayed for 16 seconds and results in a 500 (Internal Server Error). The fifth request is delayed for 16 seconds and results in a 200 (OK).

```

40     retryPolicy: (outcome, timespan, retryAttempt) =>
41     {
42         var serviceProvider = services.BuildServiceProvider();
43         serviceProvider.GetService<ILogger<CatalogClient>>().?
44         .LogWarning($"Delaying for {timespan.TotalSeconds} seconds, then making retry {retryAttempt}");
45     }
46 }
47
48 .AddPolicyHandler(Policy.TimeoutAsync<HttpResponseMessage>(1));
49
50 services.AddControllers();
51 services.AddSwaggerGen(c =>

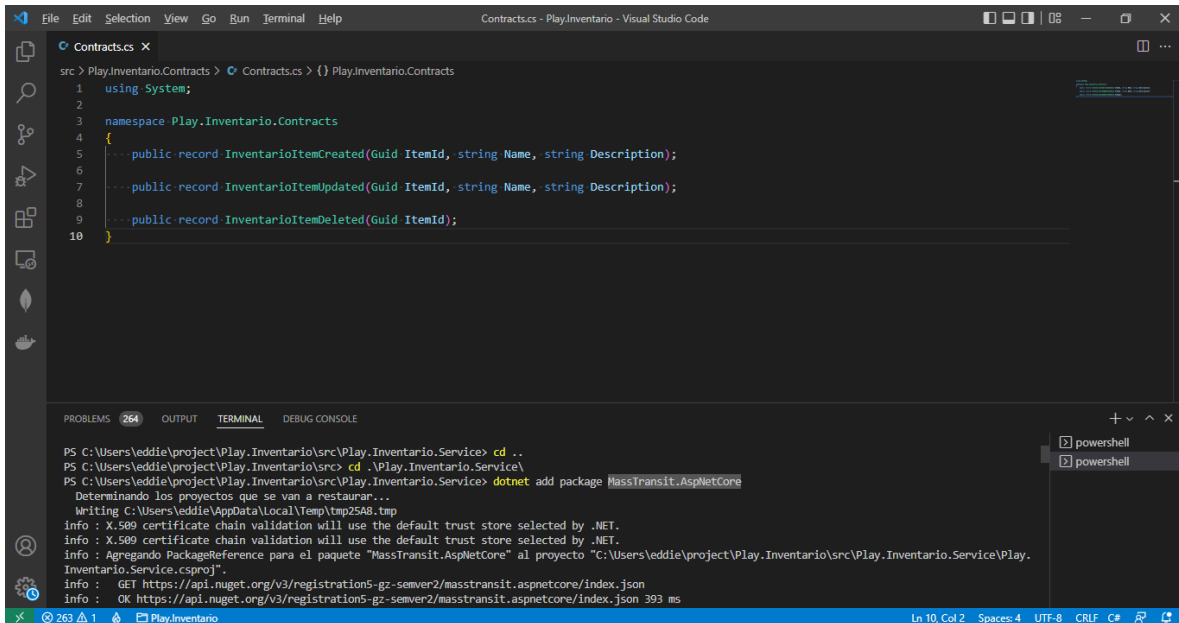
```

```

warn: Play.Inventory.Service.Clients.CatalogClient[0]
      Delaying for 4 seconds, then making retry Polly.Context
info: System.Net.Http.HttpClient.CatalogClient.ClientHandler[100]
      Sending HTTP request GET https://localhost:5001/items
info: System.Net.Http.HttpClient.CatalogClient.ClientHandler[101]
      Received HTTP response headers after 28.8285ms - 500
warn: Play.Inventory.Service.Clients.CatalogClient[0]
      Delaying for 8 seconds, then making retry Polly.Context
info: System.Net.Http.HttpClient.CatalogClient.ClientHandler[100]
      Sending HTTP request GET https://localhost:5001/items
info: System.Net.Http.HttpClient.CatalogClient.ClientHandler[101]
      Received HTTP response headers after 2.9418ms - 500
warn: Play.Inventory.Service.Clients.CatalogClient[0]
      Delaying for 16 seconds, then making retry Polly.Context
info: System.Net.Http.HttpClient.CatalogClient.ClientHandler[100]
      Sending HTTP request GET https://localhost:5001/items
info: System.Net.Http.HttpClient.CatalogClient.ClientHandler[101]
      Received HTTP response headers after 88.0146ms - 200
info: System.Net.Http.HttpClient.CatalogClient.LogicalHandler[101]
      End processing HTTP request after 32214.9848ms - 200
info: Microsoft.Hosting.Lifetime[0]
      Now listening on: http://localhost:5000
info: Microsoft.Hosting.Lifetime[0]
      Application started. Press Ctrl+C to shut down.
info: Microsoft.Hosting.Lifetime[0]
      Hosting environment: Development
info: Microsoft.Hosting.Lifetime[0]
      Content root path: D:\projects\Play.Catalog\src\Play.Catalog.Service
Request 1: Starting...
Request 1: Delaying...
Request 2: Starting...
Request 2: Delaying...
Request 3: Starting...
Request 3: 500 (Internal Server Error).
Request 3: 500 (Internal Server Error).
Request 3: 500 (Internal Server Error).
Request 4: Starting...
Request 4: 500 (Internal Server Error).
Request 5: Starting...
Request 5: 200 (OK).

```

## Message Broker (RabbitMQ + MassTransit)



The screenshot shows the Visual Studio Code editor with the `Contracts.cs` file open. The code defines a `namespace Play.Inventario.Contracts` with three `public record` types: `InventarioItemCreated`, `InventarioItemUpdated`, and `InventarioItemDeleted`. The terminal output shows the application starting, listening on `http://localhost:5000`, and making several HTTP requests. The first request is delayed for 4 seconds, and the second is delayed for 8 seconds. The third request is delayed for 16 seconds and results in a 500 (Internal Server Error). The fourth request is delayed for 16 seconds and results in a 500 (Internal Server Error). The fifth request is delayed for 16 seconds and results in a 200 (OK).

```

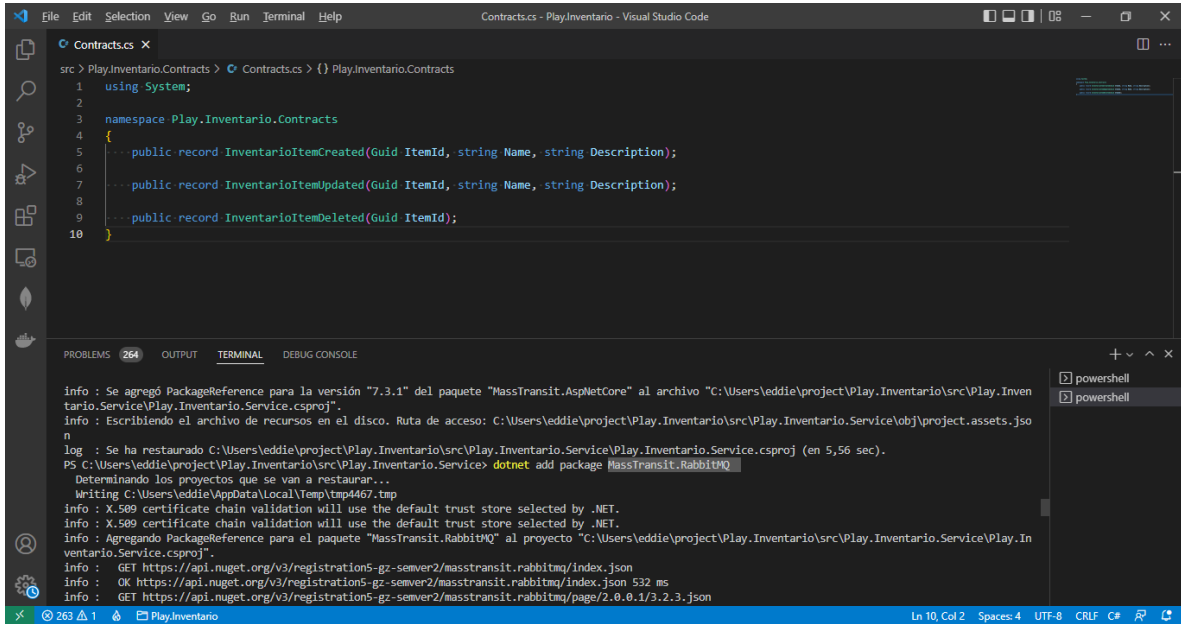
1 using System;
2
3 namespace Play.Inventario.Contracts
4 {
5     public record InventarioItemCreated(Guid ItemId, string Name, string Description);
6
7     public record InventarioItemUpdated(Guid ItemId, string Name, string Description);
8
9     public record InventarioItemDeleted(Guid ItemId);
10 }

```

```

PS C:\Users\eddie\project\Play.Inventario\src\Play.Inventory.Service> cd ..
PS C:\Users\eddie\project\Play.Inventario\src> cd \Play.Inventory.Service\
PS C:\Users\eddie\project\Play.Inventory\src\Play.Inventory.Service> dotnet add package MassTransit.AspNetCore
Determinando los proyectos que se van a restaurar...
Writing C:\Users\eddie\AppData\Local\Temp\tmp25A8.tmp
info : X.509 certificate chain validation will use the default trust store selected by .NET.
info : X.509 certificate chain validation will use the default trust store selected by .NET.
info : Agregando PackageReference para el paquete "MassTransit.AspNetCore" al proyecto "C:\Users\eddie\project\Play.Inventory\src\Play.Inventory.Service\Play.
      Inventario.Service.csproj".
info : GET https://api.nuget.org/v3/registrations-gz-sonmer2/masstransit.aspnetcore/index.json
info : OK https://api.nuget.org/v3/registrations-gz-sonmer2/masstransit.aspnetcore/index.json 393 ms

```

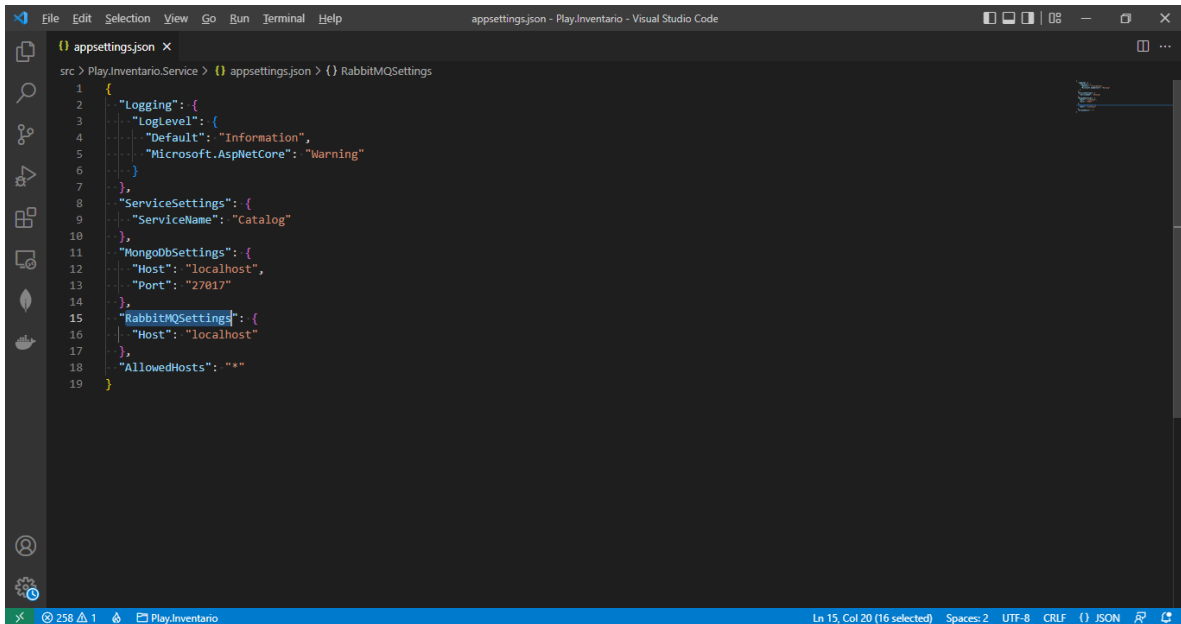


Visual Studio Code interface showing the `Contracts.cs` file in the `Play.Inventario` project. The file contains the following code:

```
1 using System;
2
3 namespace Play.Inventario.Contracts
4 {
5     public record InventarioItemCreated(Guid ItemId, string Name, string Description);
6
7     public record InventarioItemUpdated(Guid ItemId, string Name, string Description);
8
9     public record InventarioItemDeleted(Guid ItemId);
10 }
```

The bottom panel shows the `TERMINAL` output:

```
info : Se agregó PackageReference para la versión "7.3.1" del paquete "MassTransit.AspNetCore" al archivo "C:\Users\eddie\project\Play.Inventario\src\Play.Inventario\Service\Play.Inventario.Service.csproj".
info : Escribiendo el archivo de recursos en el disco. Ruta de acceso: C:\Users\eddie\project\Play.Inventario\src\Play.Inventario.Service\obj\project.assets.json
log : Se ha restaurado C:\Users\eddie\project\Play.Inventario\src\Play.Inventario.Service\Play.Inventario.Service.csproj (en 5,56 sec).
PS C:\Users\eddie\project\Play.Inventario\src\Play.Inventario.Service> dotnet add package MassTransit.RabbitMQ
Determinando los proyectos que se van a restaurar...
Writing C:\Users\eddie\AppData\Local\Temp\tmp4467.tmp
info : X.509 certificate chain validation will use the default trust store selected by .NET.
info : X.509 certificate chain validation will use the default trust store selected by .NET.
info : Agregando PackageReference para el paquete "MassTransit.RabbitMQ" al proyecto "C:\Users\eddie\project\Play.Inventario\src\Play.Inventario.Service\Play.Inventario.Service.csproj".
info : GET https://api.nuget.org/v3/registration5-gz-semver2/masstransit.rabbitmq/index.json
info : OK https://api.nuget.org/v3/registration5-gz-semver2/masstransit.rabbitmq/index.json 532 ms
info : GET https://api.nuget.org/v3/registration5-gz-semver2/masstransit.rabbitmq/page/2.0.0.1/3.2.3.json
```



Visual Studio Code interface showing the `appsettings.json` file in the `Play.Inventario` project. The file contains the following JSON configuration:

```
1 {
2   "Logging": {
3     "LogLevel": {
4       "Default": "Information",
5       "Microsoft.AspNetCore": "Warning"
6     }
7   },
8   "ServiceSettings": {
9     "ServiceName": "Catalog"
10  },
11  "MongoDbSettings": {
12    "Host": "localhost",
13    "Port": "27017"
14  },
15  "RabbitMQSettings": {
16    "Host": "localhost"
17  },
18  "AllowedHosts": "*"
19 }
```

docker-compose.yml - Play.Infra - Visual Studio Code

```

1  image: mongo
2  container_name: mongo
3  ports:
4    - 27017:27017
5  volumes:
6    - mongoddata:/data/db
7
8  rabbitmq:
9    image: rabbitmq:management
10   container_name: rabbitmq
11   ports:
12     - 5672:5672
13     - 15672:15672
14   volumes:
15     - rabbitmqdata:/var/lib/rabbitmq
16   hostname: rabbitmq
17
18 volumes:
19   mongoddata:
20   rabbitmqdata:
  
```

PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE

p":null}}  
 Gracefully stopping... (press Ctrl+C again to force)  
 Aborting on container exit...  
 [+] Stopping 1/1  
 ✓ Container mongo Stopped 2.8s  
 canceled  
 PS C:\Users\eddie\project\Play.Infra> docker-compose up -d  
 [+] Building 0.0s (0/0)  
 [+] Running 1/1  
 ✓ Container mongo Started 0.0s  
 PS C:\Users\eddie\project\Play.Infra>

Ln 22, Col 23 Spaces: 2 UTF-8 CRLF Compose

docker-compose.yml - Play.Infra - Visual Studio Code

```

14  rabbitmq:
15    image: rabbitmq:management
16    container_name: rabbitmq
17    ports:
18      - 5672:5672
19      - 15672:15672
20    volumes:
21      - rabbitmqdata:/var/lib/rabbitmq
22    hostname: rabbitmq
23
24 volumes:
25   mongoddata:
26   rabbitmqdata:
27
  
```

PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE

✓ Container mongo Started 0.0s  
 PS C:\Users\eddie\project\Play.Infra> docker-compose up -d  
 [+] Running 13/1  
 ✓ rabbitmq 12 layers [██████████] 0B/0B Pulled 43.0s  
 [+] Building 0.0s (0/0)  
 [+] Running 3/3  
 ✓ Volume "playinfra\_rabbitmqdata" Created 0.0s  
 ✓ Container rabbitmq Started 1.3s  
 ✓ Container mongo Started 0.0s  
 PS C:\Users\eddie\project\Play.Infra> docker ps

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS
541f27020ea7	rabbitmq:management	"docker-entrypoint.s..."	18 seconds ago	Up 14 seconds	4369/tcp, 5671/tcp, 0.0.0.0:5672->5672/tcp, 15671/tcp, 15691-15692/tcp, 25672/tcp, 0.0.0.0:15672->15672/tcp
04f1dea64185	mongo	"docker-entrypoint.s..."	13 hours ago	Up 14 seconds	0.0.0.0:27017->27017/tcp

PS C:\Users\eddie\project\Play.Infra>

Ln 25, Col 1 Spaces: 2 UTF-8 CRLF Compose

Docker Desktop

Upgrade plan

Search for images, containers, volumes, extensions and more...

Ctrl+K

Sign in

Containers

Images

Volumes

Dev Environments BETA

Docker Scout

Learning center

Extensions

Add Extensions

Containers

Give feedback

Container CPU usage ⓘ

108.65% / 400% (4 cores allocated)

Container memory usage ⓘ

263.38MB / 3.63GB

Show charts ▾

Search

Only show running containers

<input type="checkbox"/>	Name	Image	Status	CPU (%)	Port(s)	Last started	Actions
<input checked="" type="checkbox"/>	playinfra		Running (2/2)	108.65%		53 seconds ago	<div></div> <div></div> <div></div>
<input type="checkbox"/>	mongo	04f1dea64185 <a href="#">mongo</a>	Running	0.64%	<a href="#">27017:27017</a>	53 seconds ago	<div></div> <div></div> <div></div>
<input type="checkbox"/>	rabbitmq	541f27028ea7 <a href="#">rabbitmq.management</a>	Running	108.01%	<a href="#">15672:15672</a> <a href="#">Show all ports (2)</a>	53 seconds ago	<div></div> <div></div> <div></div>

Showing 3 items

Walkthroughs

What is a container?

5 mins

How do I run a container?

6 mins

[View more in the Learning center](#)

Engine running

RAM 2.66 GB

CPU 55.61%

Not signed in

v4.24.0

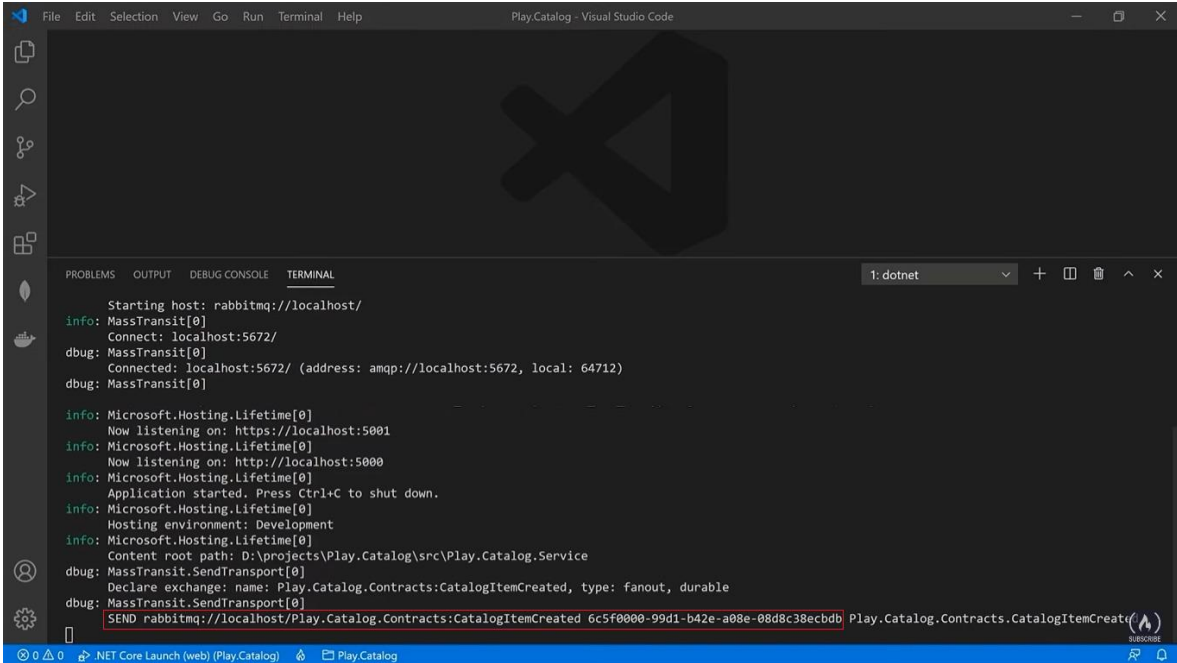
1

The screenshot shows the Visual Studio IDE with the "Play.Catalog - Visual Studio Code" window active. The interface includes a menu bar at the top with options like File, Edit, Selection, View, Go, Run, Terminal, and Help. On the left side, there are icons for Explorer, Search, Source Control, Run and Debug, Extensions, and Testing. The main editor area displays the code for "Program.cs". Below the code editor, the "TERMINAL" tab is selected, showing the output of the application. The terminal output indicates that the application built successfully and started without errors, listening on https://localhost:5001 and http://localhost:5000. A red box highlights the connection string in the terminal output.

```
Building...  
dbug: MassTransit.Registration.BusDepot[0]  
      Starting bus instances: IBus  
dbug: MassTransit[0]  
      Starting host: rabbitmq://localhost/  
info: MassTransit[0]  
      Connect: localhost:5672/  
dbug: MassTransit[0]  
      Connected: localhost:5672/ (address: amqp://localhost:5672, local: 64712)  
dbug: MassTransit[0]  
      Endpoint Ready: rabbitmq://localhost/JULIODESKTOP_PlayCatalogService_bus_ptxooyr34g4nh7izbcc8dwhyi?temporary=true  
info: Microsoft.Hosting.Lifetime[0]  
      Now listening on: https://localhost:5001  
info: Microsoft.Hosting.Lifetime[0]  
      Now listening on: http://localhost:5000  
info: Microsoft.Hosting.Lifetime[0]  
      Application started. Press Ctrl+C to shut down.  
info: Microsoft.Hosting.Lifetime[0]  
      Hosting environment: Development  
info: Microsoft.Hosting.Lifetime[0]  
      Content root path: D:\projects\Play.Catalog\src\Play.Catalog.Service
```



## Enviar mensaje mediante RabbitMQ



```

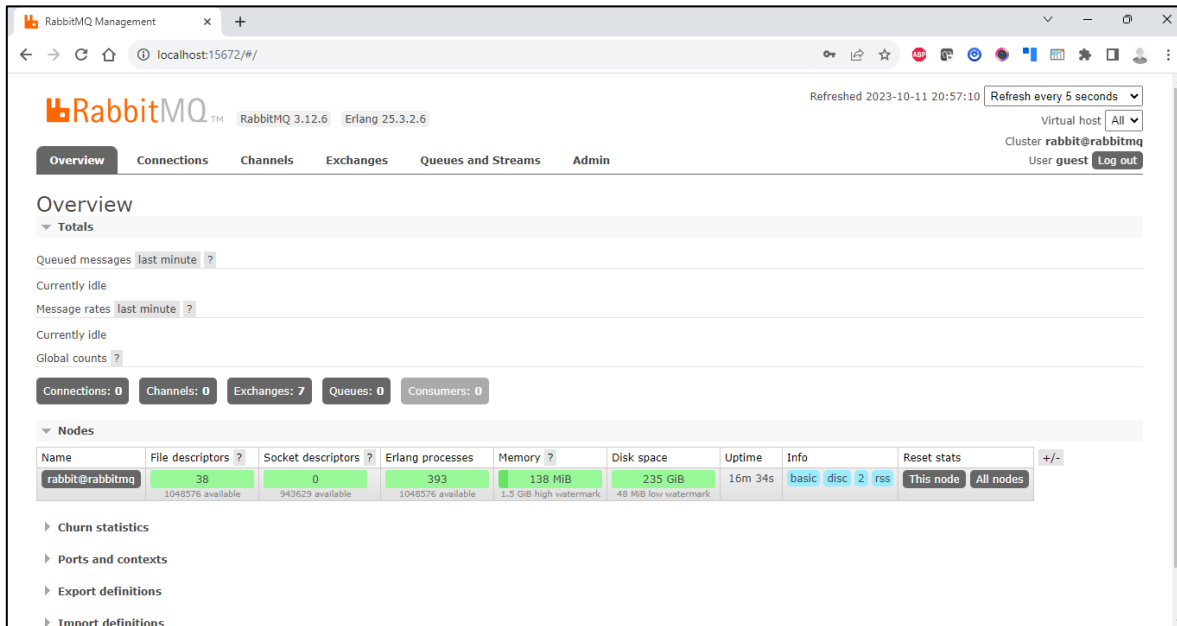
File Edit Selection View Go Run Terminal Help Play.Catalog - Visual Studio Code

Starting host: rabbitmq://localhost/
info: MassTransit[0]
Connect: localhost:5672/
debug: MassTransit[0]
Connected: localhost:5672/ (address: amqp://localhost:5672, local: 64712)
debug: MassTransit[0]

info: Microsoft.Hosting.Lifetime[0]
Now listening on: https://localhost:5001
info: Microsoft.Hosting.Lifetime[0]
Now listening on: http://localhost:5000
info: Microsoft.Hosting.Lifetime[0]
Application started. Press Ctrl+C to shut down.
info: Microsoft.Hosting.Lifetime[0]
Hosting environment: Development
info: Microsoft.Hosting.Lifetime[0]
Content root path: D:\projects\Play.Catalog\src\Play.Catalog.Service
debug: MassTransit.SendTransport[0]
Declare exchange: name: Play.Catalog.Contracts:CatalogItemCreated, type: fanout, durable
debug: MassTransit.SendTransport[0]
SEND rabbitmq://localhost/Play.Catalog.Contracts:CatalogItemCreated 6c5f0000-99d1-b42e-a08e-08d8c38ecbdb Play.Catalog.Contracts.CatalogItemCreat

```

## RabbitMQ Console



RabbitMQ Management x +

localhost:15672/#/

Refreshed 2023-10-11 20:57:10 Refresh every 5 seconds

Virtual host All

Cluster rabbit@rabbitmq

User guest Log out

**Overview** Connections Channels Exchanges Queues and Streams Admin

**Overview**

Totals

Queued messages last minute ?

Currently idle

Message rates last minute ?

Currently idle

Global counts ?

Connections: 0 Channels: 0 Exchanges: 7 Queues: 0 Consumers: 0

Nodes

Name	File descriptors ?	Socket descriptors ?	Erlang processes	Memory ?	Disk space	Uptime	Info	Reset stats
rabbit@rabbitmq	38 1048576 available	0 943629 available	393 1048576 available	138 MiB 1.5 GiB high watermark	235 GiB 48 MiB low watermark	16m 34s	basic disc 2 rss	This node All nodes

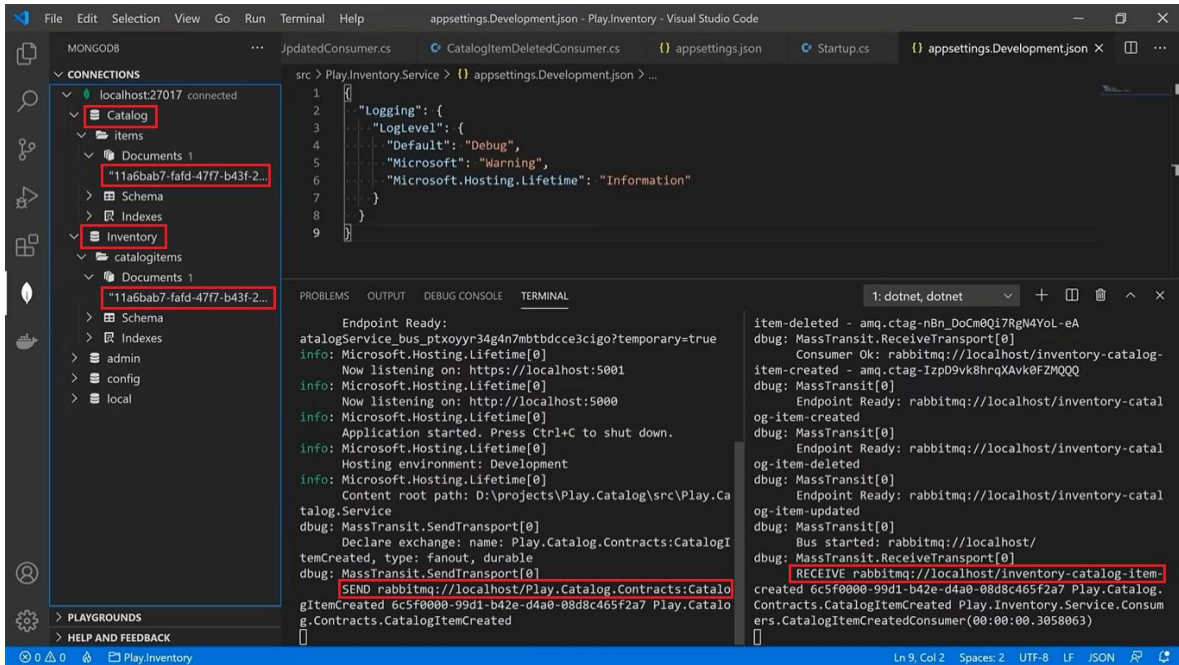
Churn statistics

Ports and contexts

Export definitions

Import definitions

## Enviar y recibir mensajes entre microservicios a través de RabbitMQ

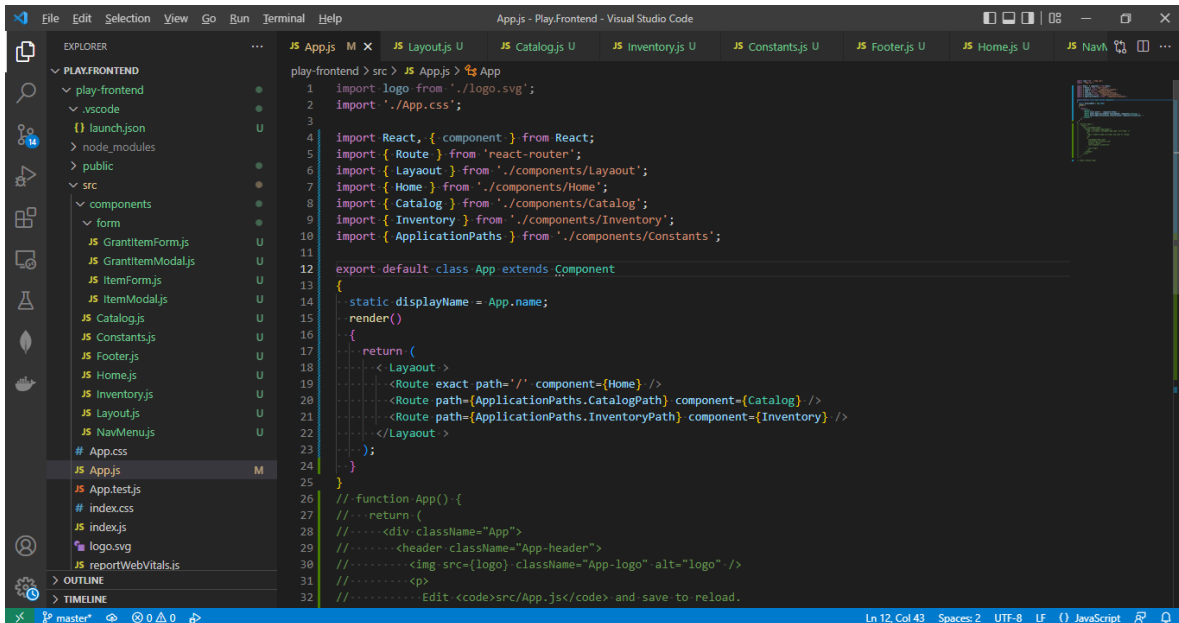


The screenshot shows the Visual Studio Code interface with the Play.Inventory service configuration in the Explorer and the terminal output.

**Explorer:** The left sidebar shows the project structure. The `CONNECTIONS section is expanded, showing a connection to localhost:27017 with a database named Catalog. The Inventory collection is highlighted.`

**Terminal:** The terminal shows the output of the `dotnet` command. It displays the application's configuration, including the logging level (Debug), the host name (localhost:5001), and the content root path. The terminal also shows the application's startup logs, including the message `SEND rabbitmq://localhost/Play.Catalog.Contracts.CatalogItemCreated` and the receipt of a message `RECEIVE rabbitmq://localhost/inventory-catalog-item-created`.

## Frontend (React)

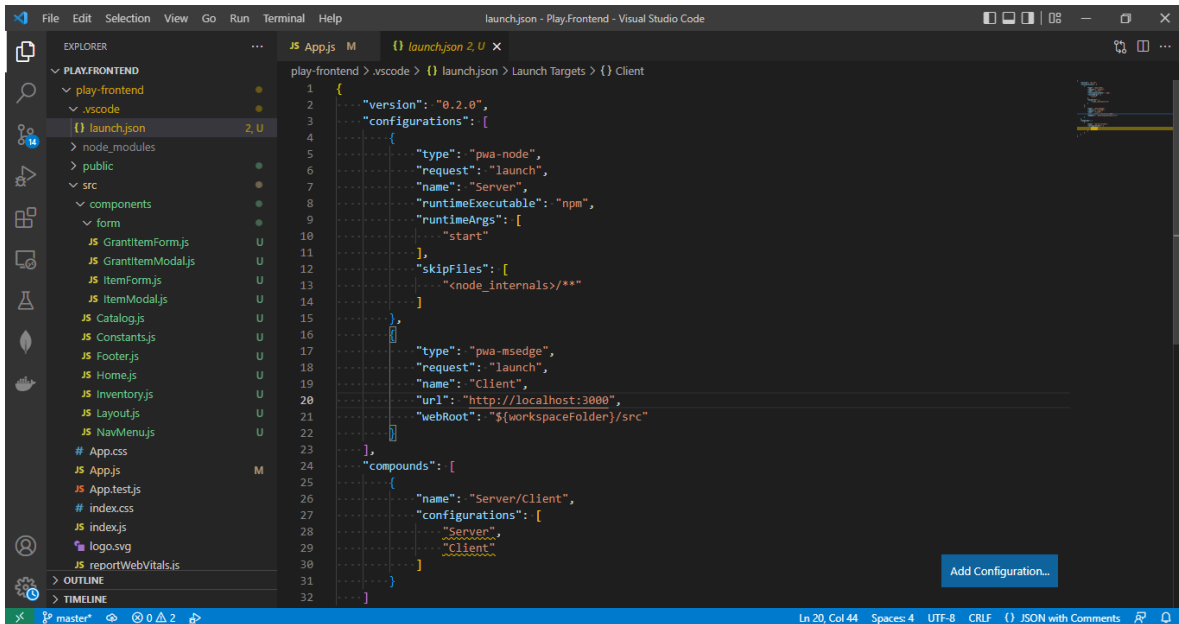


The screenshot shows the Visual Studio Code interface with the Play.Frontend application code in the Explorer and the terminal output.

**Explorer:** The left sidebar shows the project structure. The `PLAYFRONTEND` section is expanded, showing the `src` directory. The `App.js` file is highlighted.

**Terminal:** The terminal shows the output of the `dotnet` command. It displays the application's configuration, including the logging level (Debug), the host name (localhost:5001), and the content root path. The terminal also shows the application's startup logs, including the message `SEND rabbitmq://localhost/Play.Catalog.Contracts.CatalogItemCreated` and the receipt of a message `RECEIVE rabbitmq://localhost/inventory-catalog-item-created`.

## Hosted (Node.js) Web Server



```

1 {
2   "version": "0.2.0",
3   "configurations": [
4     {
5       "type": "pwa-node",
6       "request": "launch",
7       "name": "Server",
8       "runtimeExecutable": "npm",
9       "runtimeArgs": [
10        "start"
11      ],
12      "skipFiles": [
13        "**<node_internals>/**"
14      ],
15    },
16    {
17      "type": "pwa-msedge",
18      "request": "launch",
19      "name": "Client",
20      "url": "http://localhost:3000",
21      "webRoot": "${workspaceFolder}/src"
22    }
23  ],
24  "compounds": [
25    {
26      "name": "Server/Client",
27      "configurations": [
28        "Server",
29        "Client"
30      ]
31    }
32  ]
33 }

```

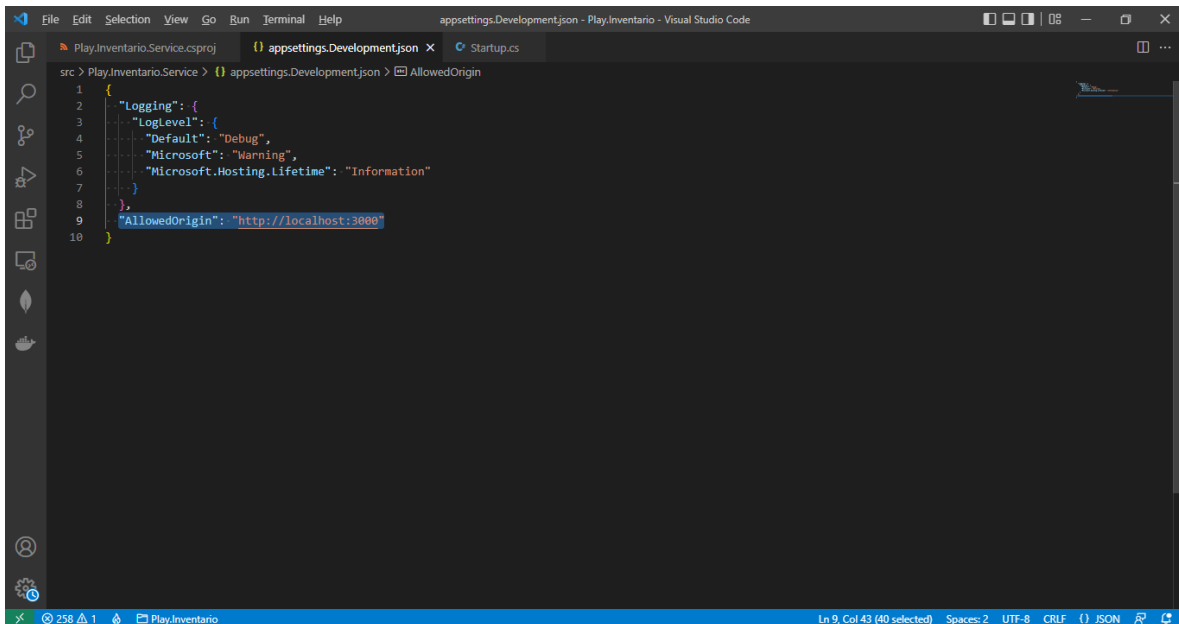
## CORS (Cross-Origin Resource Sharing)

Para la comunicación entre el frontend y los microservicios

### CORS

Permite a un servidor indicar cualquier otro origen distinto del suyo desde el cual un navegador debería permitir la carga de recursos.

Permitir origen desde otras fuentes

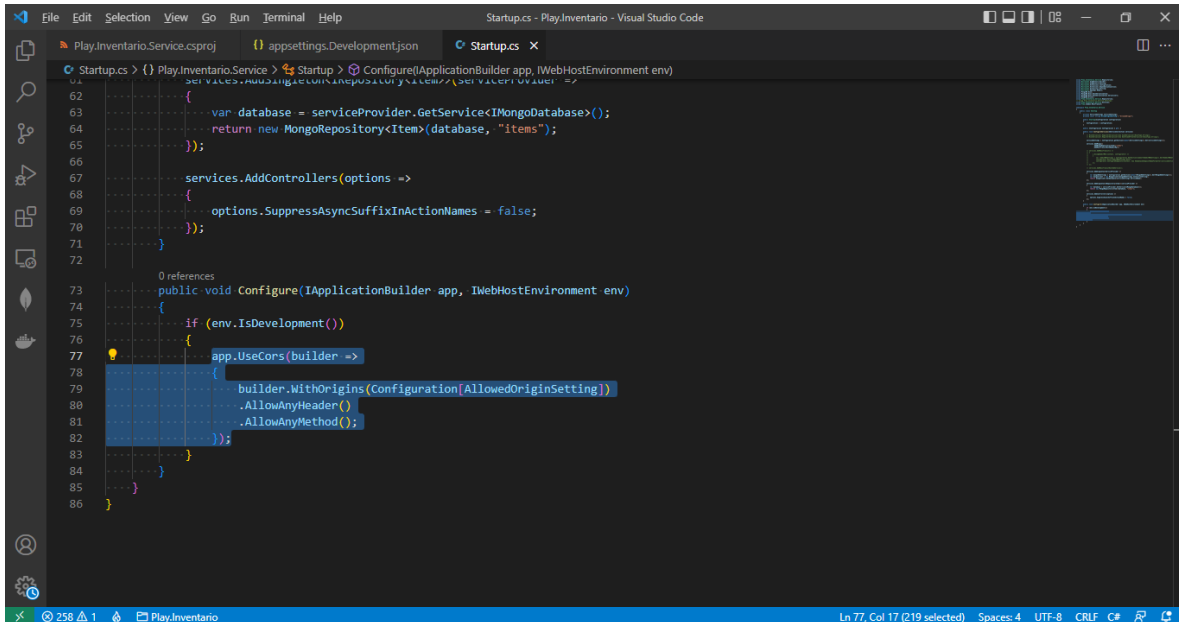


```

1 {
2   "Logging": {
3     "LogLevel": {
4       "Default": "Debug",
5       "Microsoft": "Warning",
6       "Microsoft.Hosting.Lifetime": "Information"
7     }
8   },
9   "AllowedOrigin": "http://localhost:3000"
10 }

```

## Permitir otros orígenes



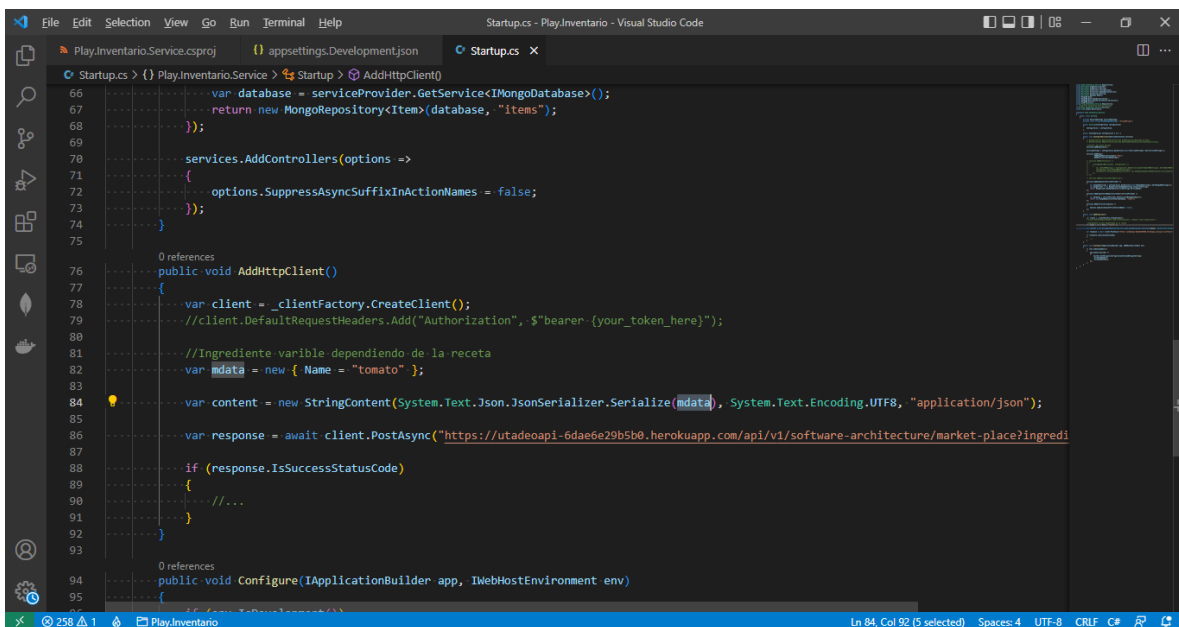
```

1  ...services.AddSingleton<IMongoRepository>(() => {
2  ...    return new MongoRepository<Item>(database, "items");
3  ...});
4
5  ...services.AddControllers(options =>
6  ...{
7  ...    options.SuppressAsyncSuffixInActionNames = false;
8  ...});
9
10 ...
11
12 0 references
13 public void Configure(IApplicationBuilder app, IWebHostEnvironment env)
14 {
15     if (env.IsDevelopment())
16     {
17         app.UseCors(builder =>
18         {
19             builder.WithOrigins(Configuration[AllowedOriginSetting])
20                 .AllowAnyHeader()
21                 .AllowAnyMethod();
22         });
23     }
24 }
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86

```

Consumir API plaza de mercado desde API propia

<https://utadeoapi-6dae6e29b5b0.herokuapp.com/api/v1/software-architecture/market-place?ingredient=tomato>



```

66 ...var database = serviceProvider.GetService<IMongoDatabase>();
67 ...return new MongoRepository<Item>(database, "items");
68 ...});
69
70 ...services.AddControllers(options =>
71 ...{
72 ...    options.SuppressAsyncSuffixInActionNames = false;
73 ...});
74
75
76 0 references
77 public void AddHttpClient()
78 {
79     var client = _clientFactory.CreateClient();
80     //client.DefaultRequestHeaders.Add("Authorization", $"bearer {your_token_here}");
81     //Ingrediente variable dependiendo de la receta
82     var mdata = new { Name = "tomato" };
83
84     var content = new StringContent(System.Text.Json.JsonSerializer.Serialize(mdata), System.Text.Encoding.UTF8, "application/json");
85
86     var response = await client.PostAsync("https://utadeoapi-6dae6e29b5b0.herokuapp.com/api/v1/software-architecture/market-place?ingredient=tomato", content);
87
88     if (response.IsSuccessStatusCode)
89     {
90         //...
91     }
92
93
94 0 references
95 public void Configure(IApplicationBuilder app, IWebHostEnvironment env)
96 {
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200

```

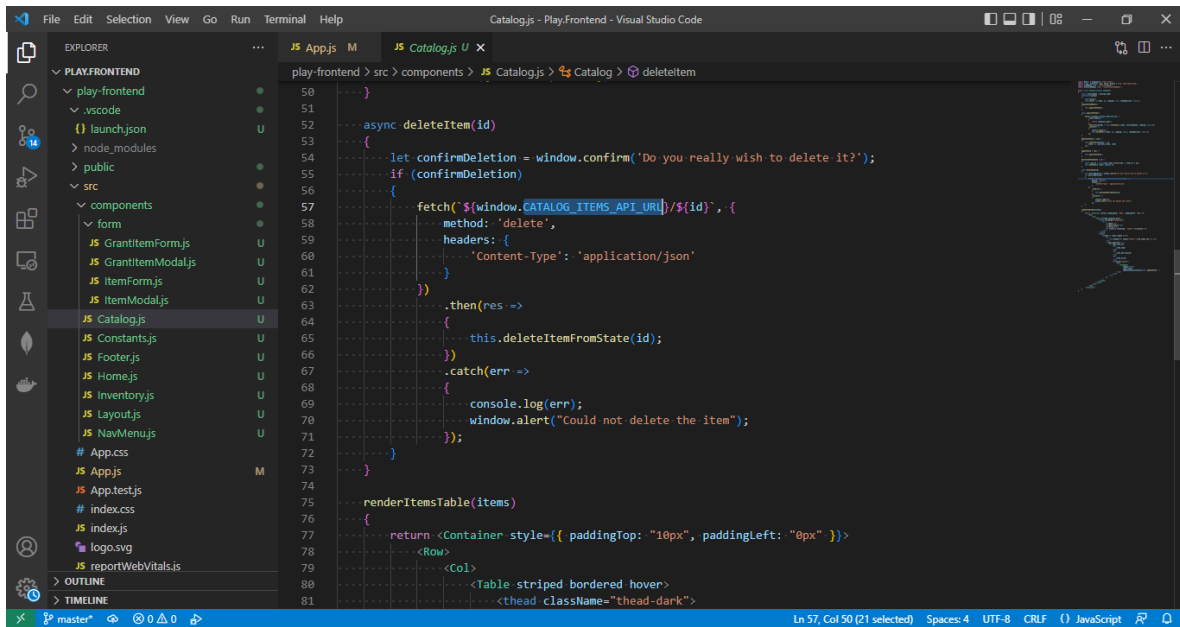


```

<  >  ↻  🏠  utadeoapi-6dae6e29b5b0.herokuapp.com/api/v1/software-architecture/market-place?ingredient=tomato
{
  "message": "Thanks for your purchase",
  "data": {
    "tomato": 3
  }
}

```

## Interconexión entre microservicios



## Recetas

### Recetas x 6

#### (Ingredientes):

Tomato  
Lemon  
Potato  
Rice  
Ketchup  
Lettuce  
Onion  
Cheese  
Meat  
Chicken

A continuación, se describen 6 recetas ficticias utilizando los ingredientes proporcionados.

#### 1. Ensalada de Tomate y Queso:

- Ingredientes:

- Tomato (2 unidades)
- Lettuce (1 unidad)
- Cheese (2 unidades)

- Preparación: Corta los tomates en rodajas, mezcla con la lechuga y el queso. Sirve con tu aderezo favorito.

## **2. Arroz con Pollo al Limón:**

- Ingredientes:

- Rice (1 unidad)
- Chicken (1 unidad)
- Lemon (2 unidades)

- Preparación: Cocina el arroz. En una sartén, cocina el pollo con el jugo de limón. Sirve el pollo sobre el arroz.

## **3. Papas Gratinadas con Queso:**

- Ingredientes:

- Potato (4 unidades)
- Cheese (2 unidades)

- Preparación: Corta las papas en rodajas finas. Coloca en capas en una fuente para horno con queso entre cada capa. Hornea hasta que las papas estén tiernas.

## **4. Salsa de Tomate Casera:**

- Ingredientes:

- Tomato (5 unidades)
- Onion (1 unidad)
- Garlic (2 unidades)

- Preparación: Hierva los tomates y pélalos. Sofríe la cebolla y el ajo, luego agrega los tomates triturados. Cocina a fuego lento hasta obtener una salsa espesa.

## **5. Hamburguesa con Salsa de Ketchup:**

- Ingredientes:

- Meat (1 unidad)
- Lettuce (1 unidad)

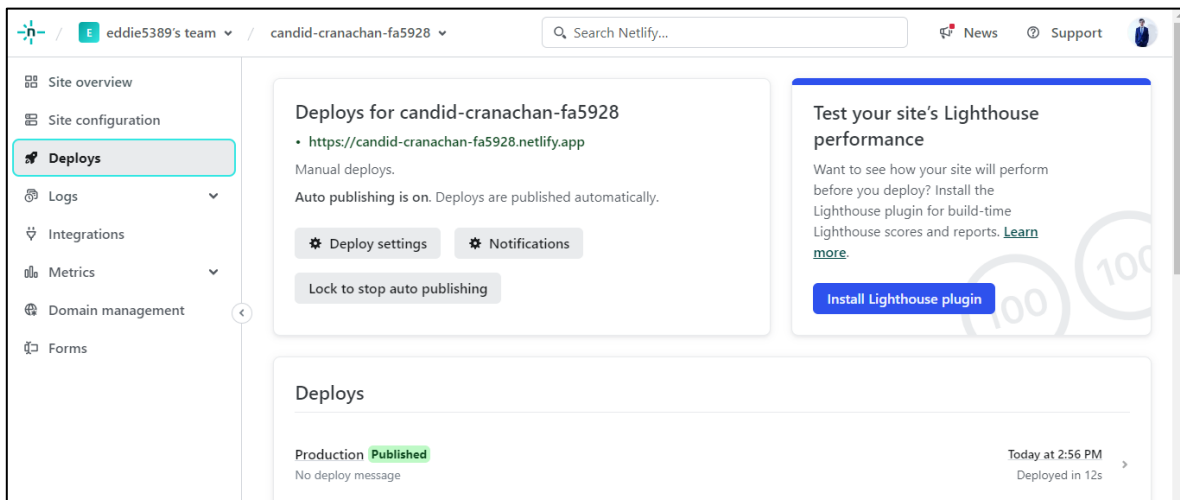
- Tomato (1 unidad)
- Ketchup (1 unidad)
- Preparación: Forma las hamburguesas y cocínalas. Arma las hamburguesas con lechuga, tomate y salsa de ketchup.

## 6. Pollo con Limón y Queso:

- Ingredientes:
  - Chicken (1 unidad)
  - Lemon (1 unidad)
  - Cheese (2 unidades)
- Preparación: Cocina el pollo a la parrilla o en una sartén. Exprime limón sobre el pollo y agrega queso rallado antes de servir.

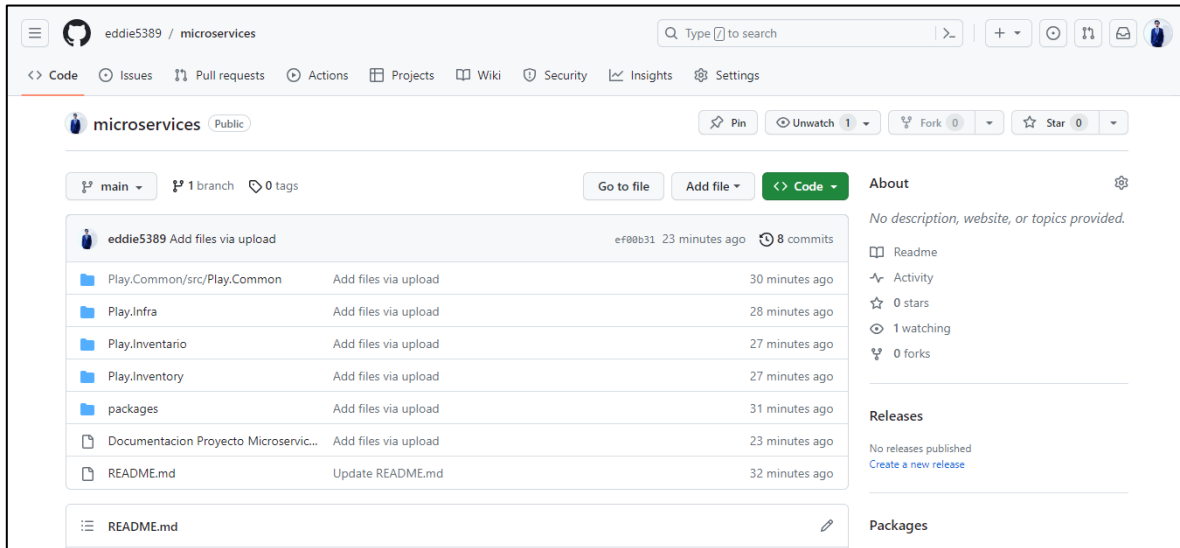
## Deploy App NETLIFY

URL: <https://candid-cranachan-fa5928.netlify.app/>



## GitHub Repositorio

URL: <https://github.com/eddie5389/microservices>



**Nota.** Github no permitió subir archivo frontend (react) ya que excede el tamaño permitido (25MB) por lo cual se adjunta el archivo comprimido en “.zip” directamente en Avata.

