

6/12/23

# BASES DE DATOS

LICENCIATURA EN INFORMÁTICA



ALUMNO: MARTIN VALENTE MOCTEZUMA CUELLO

ASESOR: JUAN MANUEL MARTINEZ FERNANDEZ

## UNIDAD 7: Nuevas tecnologías

### Act. complementaria 1

#### a) Investiga ¿qué es un API? y ¿qué es un microservicio?

R:

Una *API* (Application Programming Interface) es un conjunto **de definiciones y protocolos** que permiten que dos componentes de software se comuniquen entre sí, se utilizan para exponer las funcionalidades de un componente de software a otros componentes, por ejemplo, para que una aplicación pueda acceder a los datos de una base de datos o para que un servicio web pueda realizar una determinada tarea.

Un *microservicio* es un enfoque arquitectónico que **divide una aplicación en servicios pequeños, independientes y altamente especializados**. Cada microservicio resuelve un único problema o realiza una tarea específica. Los microservicios se comunican entre sí a través de API.

#### b) Investiga qué es Swagger y ¿para qué se utiliza?

R:

Swagger es una herramienta de código abierto que **se utiliza para documentar y probar API**. Es una especificación abierta que define cómo se puede describir una API. Swagger se utiliza para describir los recursos, operaciones y parámetros de una API.

Se puede utilizar para:

**Documentación:** Swagger puede generar documentación HTML, JSON o YAML para una API. Esta documentación puede ser utilizada por los desarrolladores para entender cómo utilizar la API.

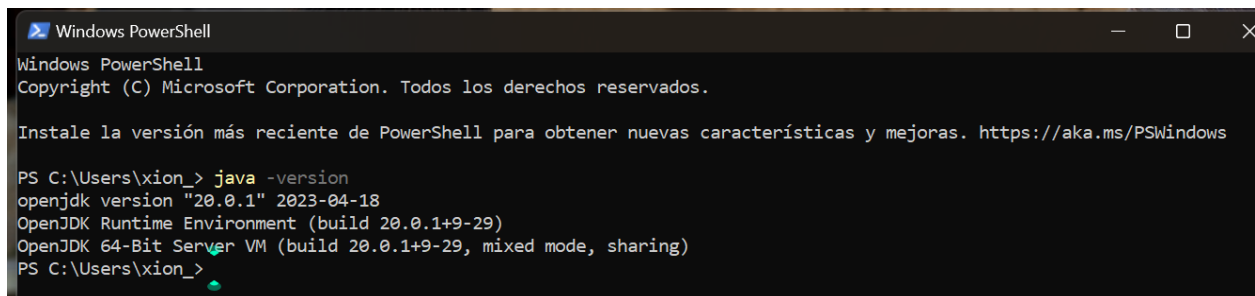
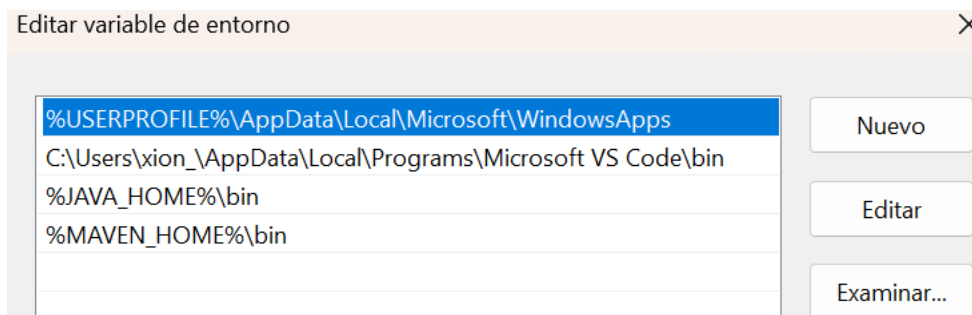
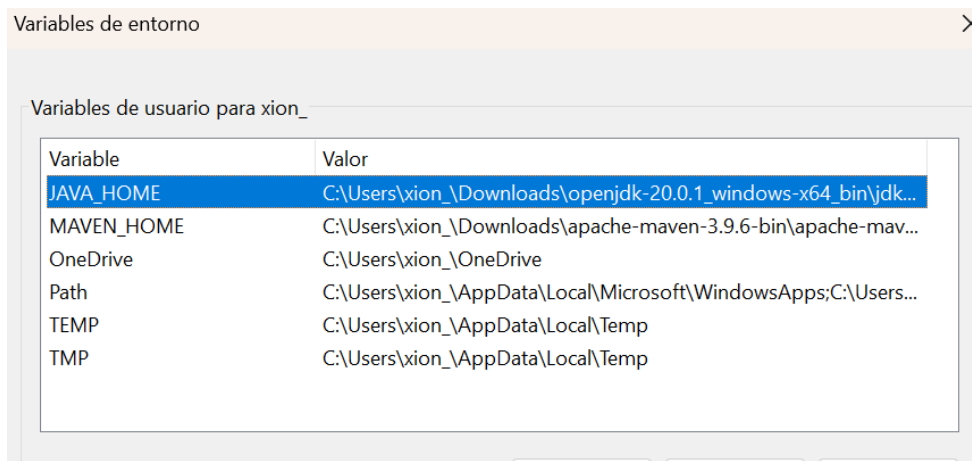
**Pruebas:** Swagger puede utilizarse para generar pruebas unitarias para una API. Estas pruebas pueden ayudar a los desarrolladores a asegurarse de que la API funciona correctamente.

**Autogeneración:** Swagger puede utilizarse para generar código cliente para una API. Este código cliente puede ser utilizado por los desarrolladores para conectarse a la API.

**c) Descarga el proyecto del API que se encuentra en GitHub. En el foro se indicará la dirección de GitHub.**

**d) Completa los endpoints que faltan.**

### 1) Instalación de Java y Maven



```
PS C:\Users\xion_> mvnw --version
Apache Maven 3.9.6 (bc0240f3c744dd6b6ec2920b3cd08dcc295161ae)
Maven home: C:\Users\xion_\Downloads\apache-maven-3.9.6-bin\apache-maven-3.9.6
Java version: 20.0.1, vendor: Oracle Corporation, runtime: C:\Users\xion_\Downloads\openjdk-20.0.1_windows-x64_bin\jdk-20.0.1
Default locale: es_MX, platform encoding: UTF-8
OS name: "windows 11", version: "10.0", arch: "amd64", family: "windows"
PS C:\Users\xion_>
```

Congratulations, you have created a new Quarkus application.

Why do you see this?

This page is served by Quarkus. The source is in `src/main/resources/META-INF/resources/index.html`.

What can I do from here?

If not already done, run the application in dev mode using: `mvn compile quarkus:dev`.

- Add REST resources, Servlets, functions and other services in `src/main/java`.
- Your static assets are located in `src/main/resources/META-INF/resources`.
- Configure your application in `src/main/resources/application.properties`.

Do you like Quarkus?

Go give it a star on [GitHub](#)

How do I get rid of this page?

Just delete the `src/main/resources/META-INF/resources/index.html` file.

Application

GroupId: la.kingtide  
ArtifactId: starter-quakus  
Version: 1.0-SNAPSHOT  
Quarkus Version: 1.9.0.Final

Next steps

[Set up your IDE](#)  
[Getting started](#)  
[Quarkus Web Site](#)

```
PS C:\Users\xion_\Downloads\projecto-api-fca-unam\projecto-api-fca-unam> mvnw.cmd quarkus:dev
[INFO] Scanning for projects...
[INFO] -----< la.kingtide:starter-quakus >-----
[INFO] Building starter-quakus 1.0-SNAPSHOT
[INFO] -----[ jar ]-----
[INFO] --- quarkus-maven-plugin:1.10.3.Final:dev (default-cli) @ starter-quakus ---
[INFO] Using 'UTF-8' encoding to copy filtered resources.
[INFO] Copying 3 resources
[INFO] Nothing to compile - all classes are up to date
Listening for transport dt_socket at address: 5005

2023-12-07 19:20:58,390 INFO [io.quarkus] (Quarkus Main Thread) starter-quakus 1.0-SNAPSHOT on JVM (powered by Quarkus 1.10.3.Final) started in 2.400s. Listening on: http://localhost:8090
2023-12-07 19:20:58,396 INFO [io.quarkus] (Quarkus Main Thread) Profile dev activated. Live Coding activated.
2023-12-07 19:20:58,396 INFO [io.quarkus] (Quarkus Main Thread) Installed features: [agroal, cdi, jdbc-postgresql, liquibase, mutiny, narayana-jta, resteasy, resteasy-jsonb, security, smallrye-context-propagation, smallrye-jwt, smallrye-openapi, swagger-ui, vertx, vertx-web]
```

c) Ejecuta el endpoint GET /products/all. ¿Qué devuelve y en qué formato?

```
Curl
curl -X GET "http://localhost:8090/products/all" -H "accept: application/json"

Request URL
http://localhost:8090/products/all

Server response
Code      Details
200
Response body
{
  "data": [
    {
      "id": 380,
      "name": "Computadora HP 280 G3",
      "price": 34750,
      "quantity": 10,
      "sku": "7877771"
    },
    {
      "id": 382,
      "name": "Computadora Dell Vostro 3470",
      "price": 31700,
      "quantity": 5,
      "sku": "7877772"
    },
    {
      "id": 388,
      "name": "Computadora Dell OptiPlex 3860",
      "price": 31700,
      "quantity": 5,
      "sku": "7877773"
    }
  ]
}
```

Devuelve la información de cada producto, con sus id, nombre, precio, etc. En un formato similar al de cuando ejecutamos un código usando PgAdmin pero en Inglés.

d) Prueba hacer una inserción a la tabla de productos, ejecutando el endpoint *POST* `/product/add`

Request body

```
{
  "description": "Computadora Lenovo Ideapad 3, Intel Core i5-8400 2.80GHz, 12GB, 250 GB, Windows 11 Pro 64-bit",
  "id": 777,
  "name": "Laptop Lenovo para estudiantes",
  "price": 777,
  "quantity": 9,
  "sku": "V347SF15s41TW10P1M"
}
```

Responses

Curl

```
curl -X POST "http://localhost:8090/products/add" -H "accept: application/json" -H "Content-Type: application/json" -d '{"description":"Computadora Lenovo Ideapad 3, Intel Core i5-8400 2.80GHz, 12GB, 250 GB, Windows 11 Pro 64-bit","id":777,"name":"Laptop Lenovo para estudiantes","price":777,"quantity":9,"sku":"V347SF15s41TW10P1M"}'
```

Request URL

```
http://localhost:8090/products/add
```

Server response

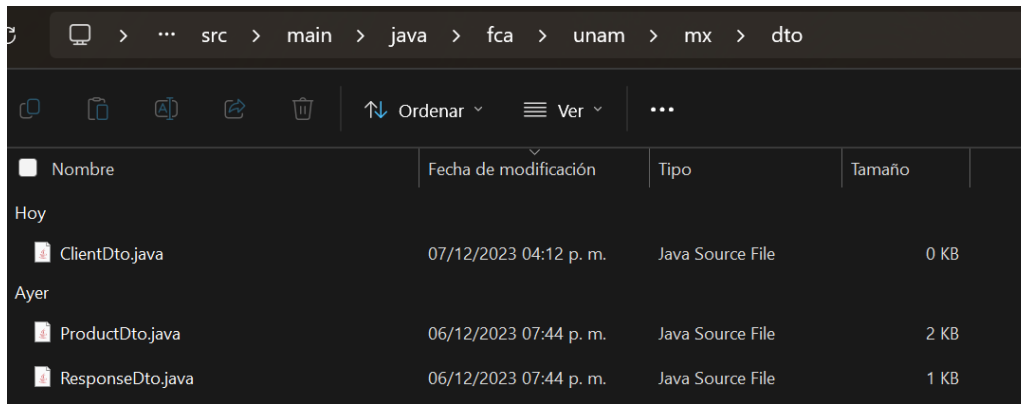
Code	Details
200	<div>Response body</div> <div><pre>{   "data": "ok",   "success": true }</pre></div> <div>Response headers</div> <div><pre>content-length: 28 content-type: application/json</pre></div>

Si se agregó al verificar con PgAdmin

ordenes	326	Computadora Lenovo ThinkCentre M715	Computadora Lenovo ThinkCentre M715, AMD Ryzen 5 PRO 2400G 3.20GHz, 8GB, 256GB SSD, Win
productos	327	Computadora Lenovo ThinkCentre M720	Computadora Lenovo ThinkCentre M720, Intel Core i5-8400 2.80GHz, 8GB, 1TB, Windows 10 Pro 64-b
Columns (6)	328	Computadora Lenovo ThinkCentre M725s	Computadora Lenovo ThinkCentre M725s, AMD Ryzen 5 PRO 2400G 3.30GHz, 8GB, 1TB, Windows 1
nombre	329	Computadora Dell OptiPlex 3060	Computadora Dell OptiPlex 3060, Intel Core i5-8500T 2.10GHz, 8GB, 500GB, Windows 10 Pro 64-bit -
descripcion	330	Laptop Lenovo para estudiantes	Computadora Lenovo Ideapad 3, Intel Core i5-8400 2.80GHz, 12GB, 250 GB, Windows 11 Pro 64-bit
precio			

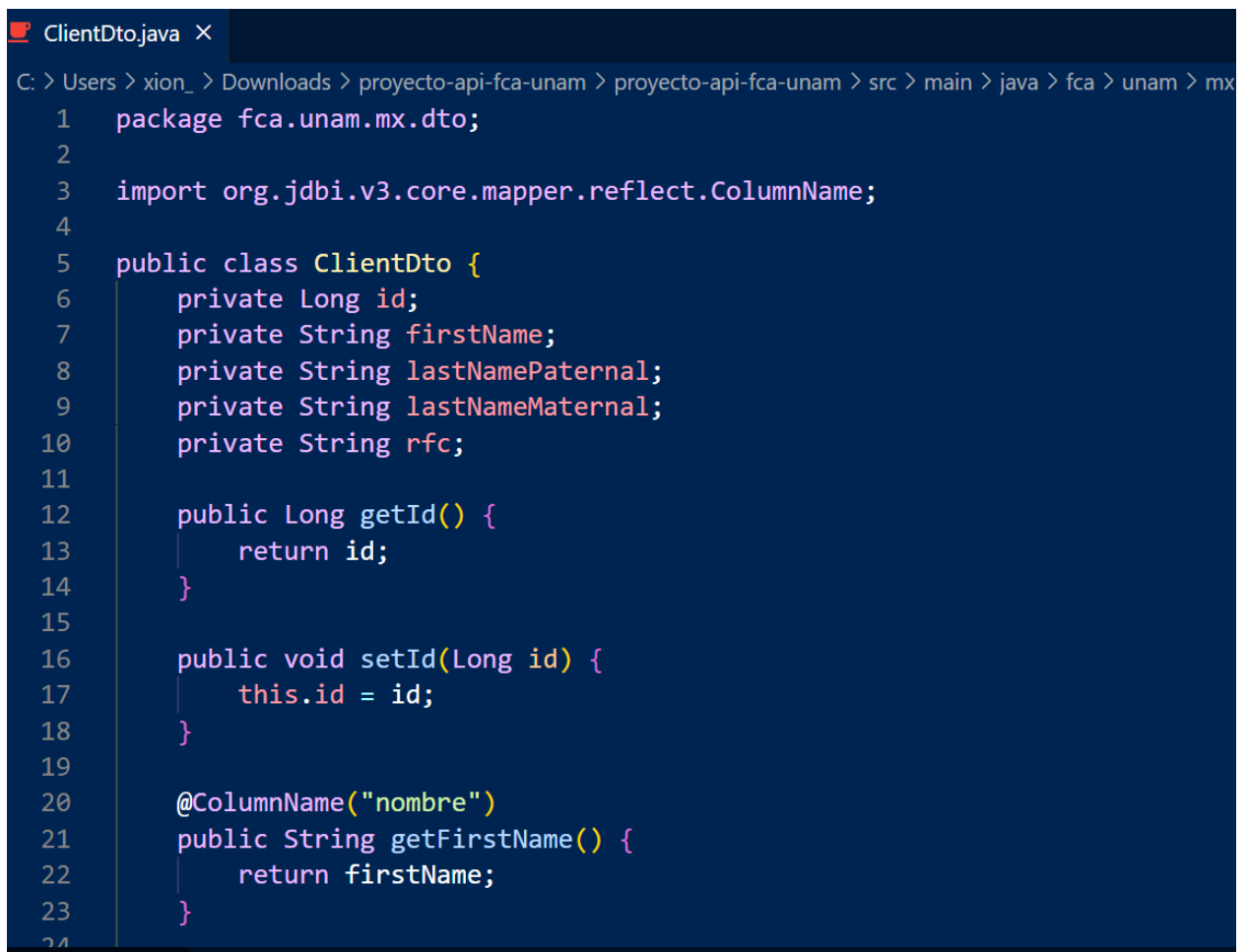
e) Crea el endpoint `/clients/all` que muestre todos los clientes. Deberás tomar como referencia el endpoint `GET /products/all`. Para ello deberás:

e.1) Crear el archivo `ClientDto.java` en la carpeta `dto`.



The screenshot shows a file explorer window with the path `src > main > java > fca > unam > mx > dto`. The table below lists the files in this directory.

Nombre	Fecha de modificación	Tipo	Tamaño
Hoy			
ClientDto.java	07/12/2023 04:12 p. m.	Java Source File	0 KB
Ayer			
ProductDto.java	06/12/2023 07:44 p. m.	Java Source File	2 KB
ResponseDto.java	06/12/2023 07:44 p. m.	Java Source File	1 KB



```
ClientDto.java X
C: > Users > xion_ > Downloads > proyecto-api-fca-unam > proyecto-api-fca-unam > src > main > java > fca > unam > mx
1  package fca.unam.mx.dto;
2
3  import org.jdbi.v3.core.mapper.reflect.ColumnName;
4
5  public class ClientDto {
6      private Long id;
7      private String firstName;
8      private String lastNamePaternal;
9      private String lastNameMaternal;
10     private String rfc;
11
12     public Long getId() {
13         return id;
14     }
15
16     public void setId(Long id) {
17         this.id = id;
18     }
19
20     @ColumnName("nombre")
21     public String getFirstName() {
22         return firstName;
23     }
24 }
```

e.2) Modificar el archivo `StoreDao.java` para incluir el nuevo query `getClientes()`. Deberás mapear `ClienteDto` de manera similar a:

`@RegisterBeanMapper(ProductDto.class)`

`@SqlQuery("SELECT * FROM productos")`

`List<ProductDto> getProducts();`

```
10 import java.util.List;
11
12 public interface StoreDao {
13
14     @RegisterBeanMapper(ProductDto.class)
15     @SqlQuery("SELECT * FROM productos")
16     List<ProductDto> getProducts();
17
18     @RegisterBeanMapper(ClienteDto.class)
19     @SqlQuery("SELECT * FROM clientes")
20     List<ClienteDto> getClientes();
21
22     @SqlUpdate("INSERT INTO productos (nombre, descripcion, precio, cantidad, sku) VALUES(:p.name, :p.descripcion, :p.precio, :p.cantidad, :p.sku)")
23     void addProduct(@BindBean("p") ProductDto productDto);
24 }
25
```

e.2.1) ¿Para qué sirve la directiva `@RegisterBeanMapper` ([https://jdbi.org/#\\_beanmapper](https://jdbi.org/#_beanmapper))?

Se utiliza en **JDBI** para registrar un mapa de objetos de base de datos a objetos **Java**. Este mapa se utiliza para convertir los resultados de una consulta SQL a objetos Java.

La directiva `@RegisterBeanMapper` tiene dos parámetros:

- El primer parámetro es la clase de objeto Java que se utilizará para mapear los resultados de la consulta.
- El segundo parámetro es opcional y se puede utilizar para especificar un prefijo para los nombres de las columnas de la base de datos.

Por ejemplo, el siguiente código registra un mapa de objetos de base de datos a objetos `ProductDto`:

Java

```
@RegisterBeanMapper(ProductDto.class)
```

```
@SqlQuery("SELECT * FROM productos")
```

```
List<ProductDto> getProducts();
```

### e.2.2) ¿Qué deberá devolver el método getClients?

Deberá devolver una lista de objetos ClientDto, la cual contendrá todos los clientes registrados en la base de datos.

e.3) Deberás modificar el archivo StoreDal.java y crear el método getClients() de manera similar a:

```
public ResponseDto<List<ProductDto>> getProducts() {  
  
    ResponseDto responseDto = new ResponseDto<List<ProductDto>>();  
  
    responseDto.setSuccess(true);  
  
    Jdbi jdbi = jdbiService.getInstance();  
  
    var products = jdbi.withExtension(StoreDao.class, dao -> dao.getProducts());  
  
    responseDto.setData(products);  
  
    return responseDto;  
  
}
```

```
30     responseDto.setData(products);  
31     return responseDto;  
32 }  
33  
34 public ResponseDto<List<ClientDto>> getClients() {  
35     ResponseDto<List<ClientDto>> responseDto = new ResponseDto<>();  
36     responseDto.setSuccess(true);  
37  
38     Jdbi jdbi = jdbiService.getInstance();  
39     List<ClientDto> clients = jdbi.withExtension(StoreDao.class, dao -> dao.getClients());  
40  
41     responseDto.setData(clients);  
42     return responseDto;  
43 }  
44  
45 public ResponseDto<String> addProduct(final ProductDto productDto) {
```



e.4) Deberás crear el archivo ClientsResource.java en la carpeta src/main/java/resources junto a ProductsResources.java El archivo ClientsResource.java deberá contener un método getClients() similar a:

```
@GET

@Path("/all")

@Produces(MediaType.APPLICATION_JSON)

@Operation(summary = "Get all products")

@APIResponses(value = {

    @APIResponse(responseCode = "200", content = @Content(mediaType =

MediaType.APPLICATION_JSON)),

})

public Response getProducts() {

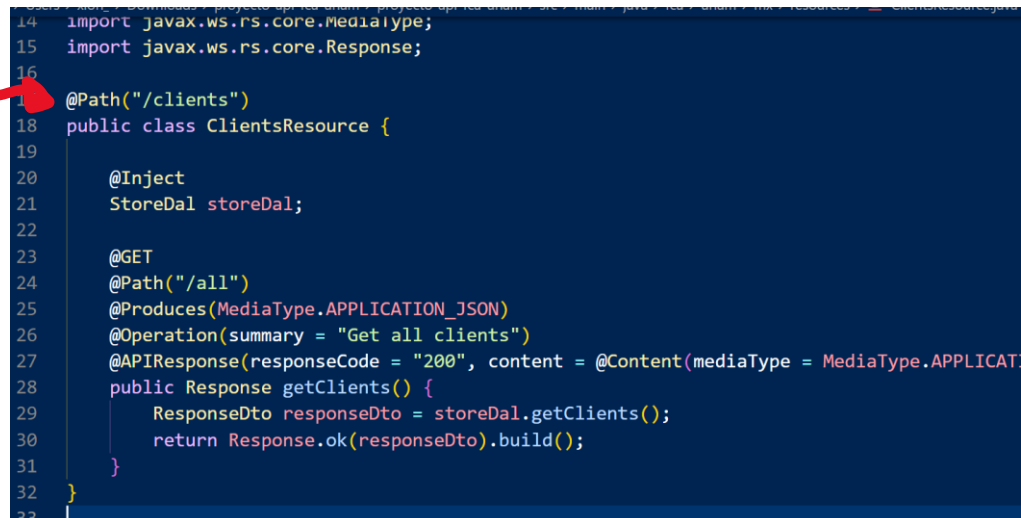
    var responseDto = storeDal.getProducts();

    return Response.ok(responseDto).build();

}
```

Nota, el path , @Path("/products")

deberá ser modificado a clients en el archivo ClientsResource.java



```
14 import javax.ws.rs.core.MediaType;
15 import javax.ws.rs.core.Response;
16
17 @Path("/clients")
18 public class ClientsResource {
19
20     @Inject
21     StoreDal storeDal;
22
23     @GET
24     @Path("/all")
25     @Produces(MediaType.APPLICATION_JSON)
26     @Operation(summary = "Get all clients")
27     @APIResponse(responseCode = "200", content = @Content(mediaType = MediaType.APPLICATION_JSON))
28     public Response getClients() {
29         ResponseDto responseDto = storeDal.getClients();
30         return Response.ok(responseDto).build();
31     }
32 }
33
```

**e.5) Compila el proyecto con el comando**

```
./mvnw.cmd clean compile
```

```

PS C:\Users\xion\Downloads\projecto-api-fca-unam\projecto-api-fca-unam> ./mvnw.cmd clean compile
[INFO] Scanning for projects...
[INFO]
[INFO] -----< la.kingtide:starter-quakus >-----
[INFO] Building starter-quakus 1.0-SNAPSHOT
[INFO]
[INFO] -----[ jar ]-----
[INFO]
[INFO] --- maven-clean-plugin:2.5:clean (default-clean) @ starter-quakus ---
[INFO] Deleting C:\Users\xion\Downloads\projecto-api-fca-unam\projecto-api-fca-unam\target
[INFO]
[INFO] --- quarkus-maven-plugin:1.10.3.Final:generate-code (default) @ starter-quakus ---
[INFO]
[INFO] --- maven-resources-plugin:2.6:resources (default-resources) @ starter-quakus ---
[INFO] Using 'UTF-8' encoding to copy filtered resources.
[INFO] Copying 3 resources
[INFO]
[INFO] --- maven-compiler-plugin:3.8.1:compile (default-compile) @ starter-quakus ---
[INFO] Changes detected - recompiling the module!
[INFO] Compiling 9 source files to C:\Users\xion\Downloads\projecto-api-fca-unam\projecto-api-fca-unam\target\classes
[INFO]
[INFO] BUILD SUCCESS
[INFO]
[INFO] -----
[INFO] Total time: 6.536 s
[INFO] Finished at: 2023-12-07T18:45:22-06:00
[INFO]
[INFO] -----

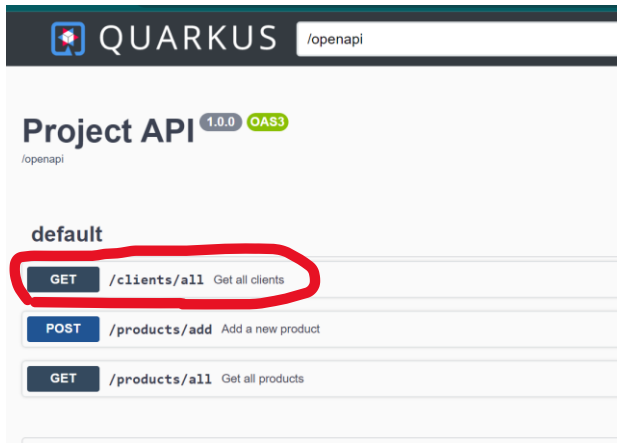
```

### e.6) Y ejecútalo

```
./mvnw.cmd quarkus:dev
```

[illegible]

e.7) Navega a <http://localhost:8090/docs> y verifica que se muestra el nuevo endpoint.



The screenshot shows the Quarkus Project API documentation. The 'default' section lists three endpoints: GET /clients/all (Get all clients), POST /products/add (Add a new product), and GET /products/all (Get all products). The GET /clients/all endpoint is highlighted with a red circle.

Curl

```
curl -X GET "http://localhost:8090/clients/all" -H "accept: application/json"
```

Request URL

```
http://localhost:8090/clients/all
```

Server response

Code Details

200

Response body

```
{
  "data": [
    {
      "firstName": "Juan",
      "id": 1,
      "lastNameMaternal": "González",
      "lastNamePaternal": "Pérez",
      "rfc": "JUPG800825"
    },
    {
      "firstName": "Laura",
      "id": 2,
      "lastNameMaternal": "López",
      "lastNamePaternal": "Martínez",
      "rfc": "LAML900515"
    },
    {
      "firstName": "Carlos",
      "id": 3,
      "lastNameMaternal": "Sánchez",
      "lastNamePaternal": "Hernández",
      "rfc": "CAHS750310"
    }
  ]
}
```

f) Crea un repositorio en GitHub y sube tus cambios.

g) En un documento en PDF, anexa las capturas de pantalla del proyecto y el enlace a tu repositorio: <https://github.com/ValenteMoctezuma/UNIDAD-7-Nuevas-Tecnolog-as-Act.-complementaria-1-Por-Valente-Moctezuma>

**Bibliografía:**

ChatGPT 3.5

Bard AI