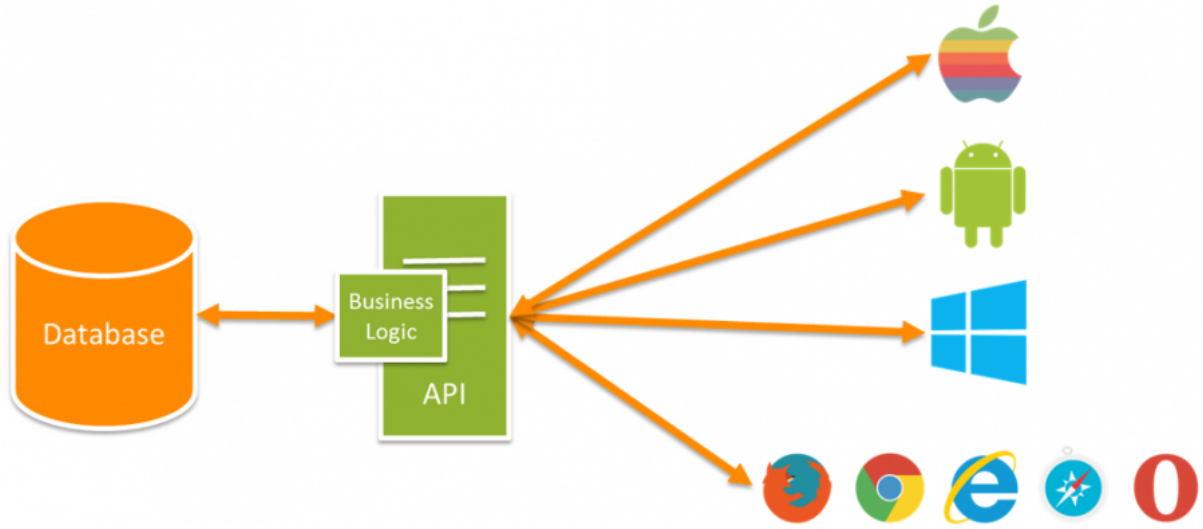
Tema 3. Práctica 3.

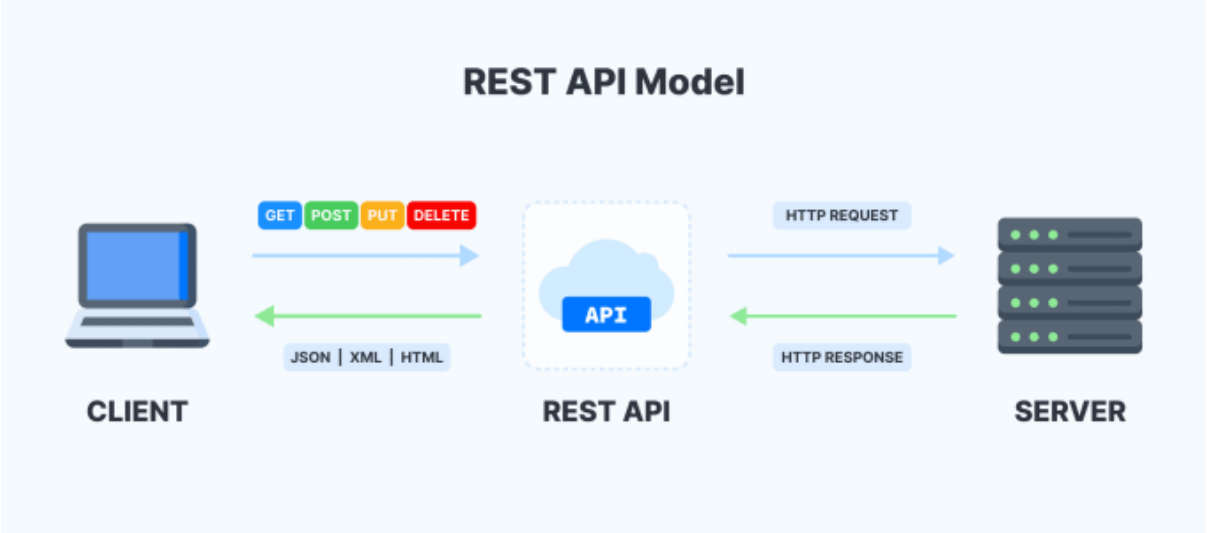
Spring: Creación de servicios web (APIs REST).

1. Servicios web y REST API

Un servicio web permite diseñar aplicaciones para diferentes plataformas reutilizando la lógica de negocio y el acceso a los datos facilitando un punto de entrada común a todos. De este modo se reducen los tiempos de desarrollo y se facilita el mantenimiento de las aplicaciones.



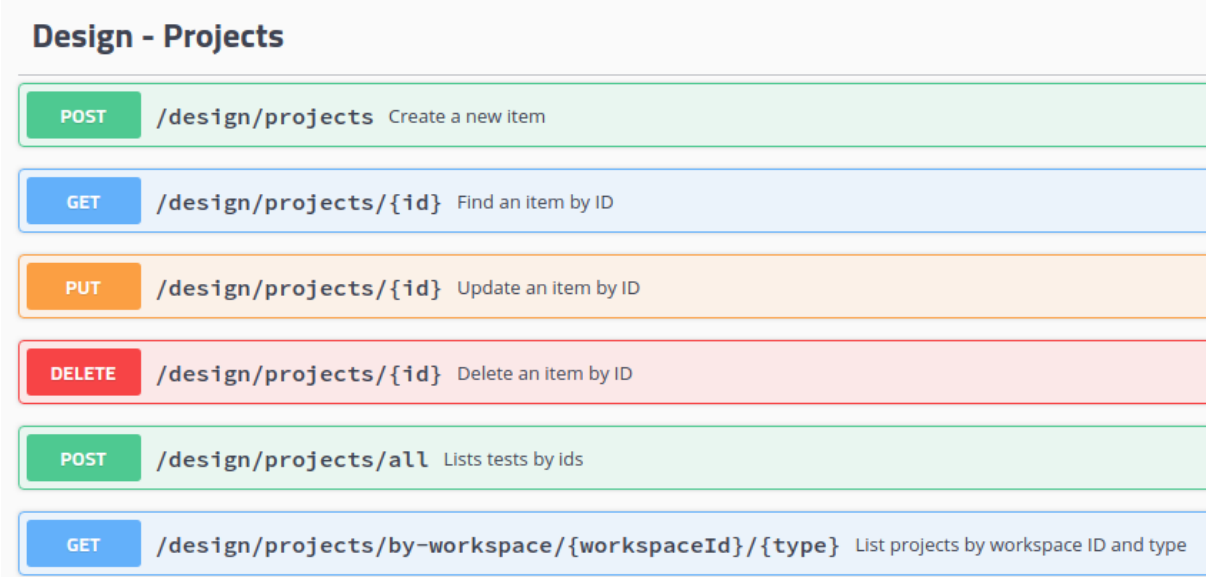
Una API REST es un servicio web que utiliza el protocolo HTTP. Pero en lugar de servir páginas web HTML, se sirven ficheros (habitualmente JSON y XML).



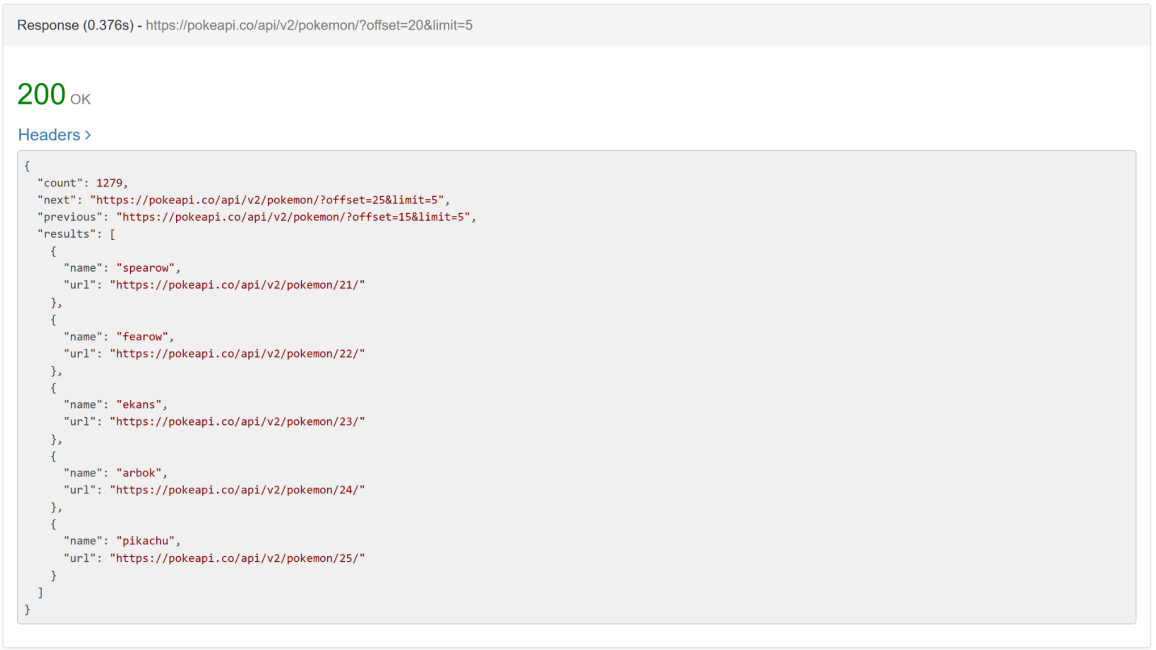
Tema 3. Práctica 3. Creación de servicios web.

Al usar una API REST tenemos los siguientes conceptos básicos:

- ➔ **Petición**. Realizada por el cliente o consumidor de la API. La interfaz web o aplicación.
  - ➔ **End Point**. Es la URL del recurso al que estamos accediendo.
  - ➔ **Método HTTP**. Es la acción que queremos hacer sobre el recurso: podemos listar recursos, borrarlos, actualizarlos, etc.



- ➔ **Respuesta**. Enviada por el servidor HTTP, realiza las acciones solicitadas en la petición (proporcionar un dato, borrar un registro, etc.).
  - ➔ **Header**. Es la respuesta del protocolo HTTP, puede devolver un código HTTP válido (200 que todo ha ido bien, 404 si no se ha encontrado el recurso, entre otros).
  - ➔ **Body**. Es el contenido del recurso al que estamos accediendo. Habitualmente un fichero JSON o XML (nosotros trabajaremos con JSON).



2. Servicios web en Spring

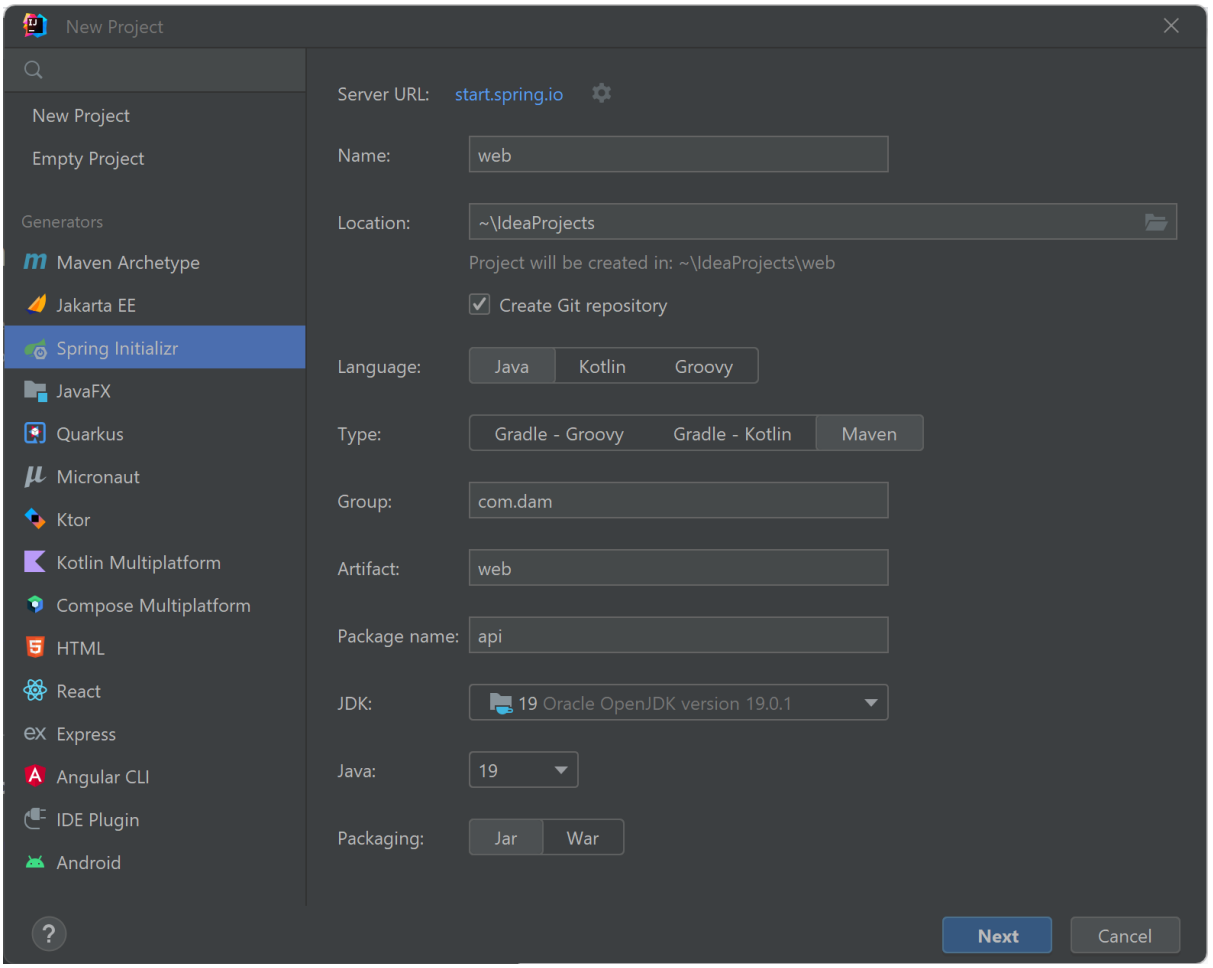
Spring es un framework para el desarrollo de aplicaciones. Spring permite desarrollar aplicaciones en entorno servidor de manera ágil con sus múltiples herramientas. Nos vamos a centrar en la creación de APIs. Spring nos permite crear servicios web con diferentes aproximaciones. En esta práctica vamos a crear una API REST tradicional, en próximas prácticas haremos APIs reactivas.

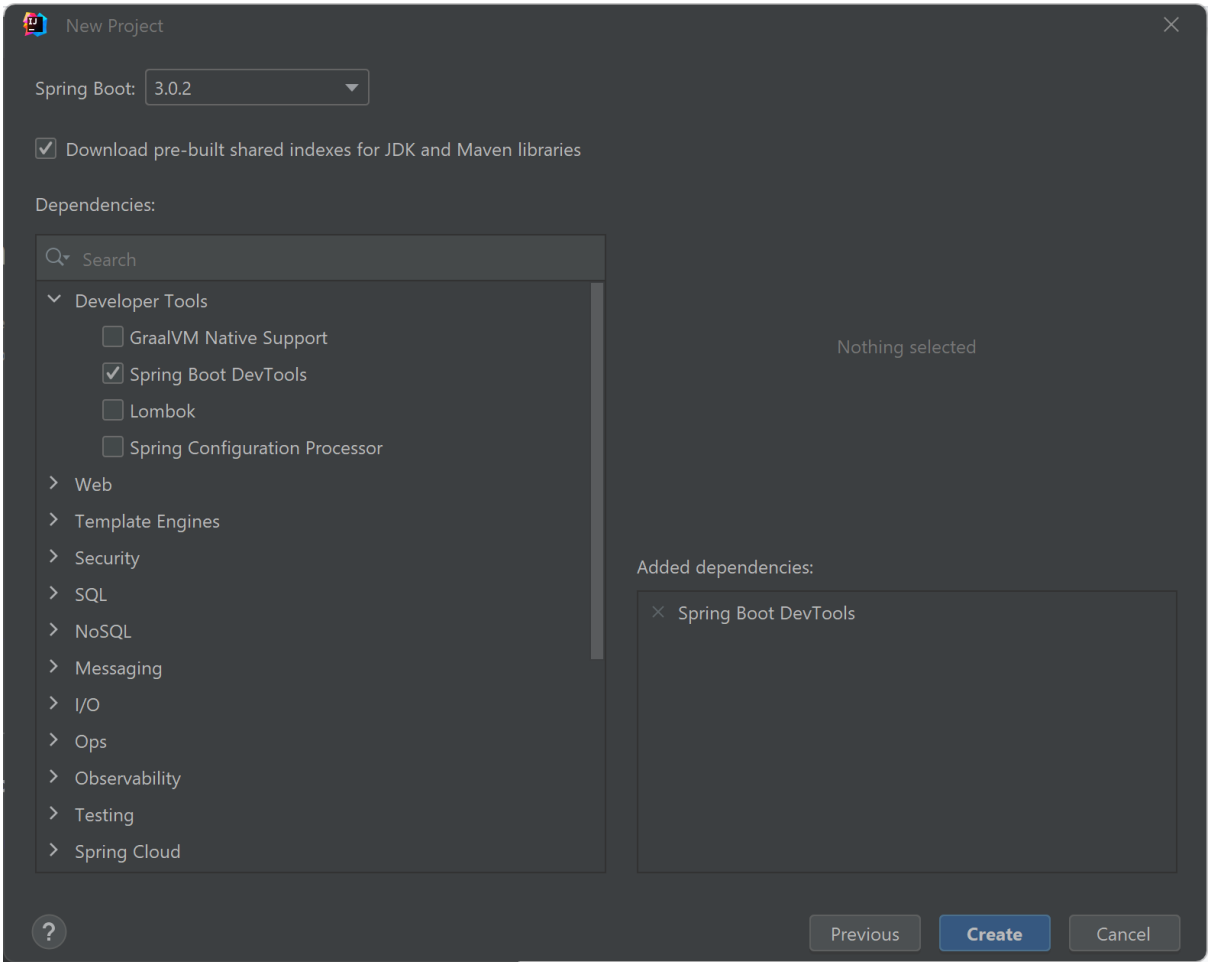
Los web services en Spring están compuestos por los siguientes elementos:

- Los modelos, donde están almacenados los datos.
- Los servicios, que implementan la forma de acceder a los datos de los modelos.
- Los controladores, que relacionan End Point con los modelos usando los servicios anteriores.

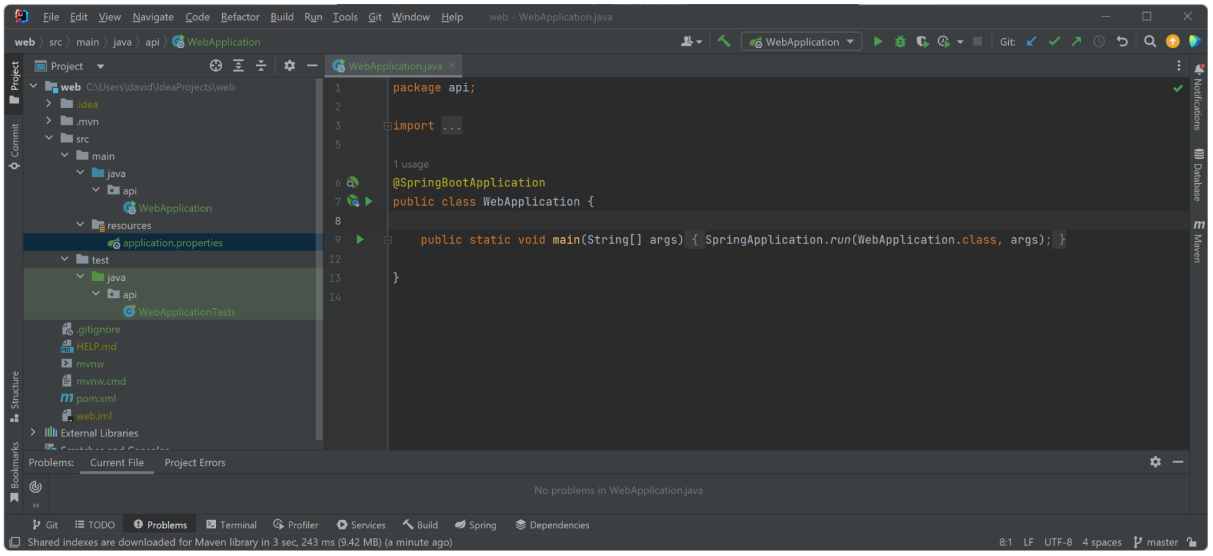
3. Mi primera API REST

Vamos a inicializar el proyecto de Spring para nuestra API.





Nos ha creado el siguiente proyecto.



3.1. El modelo

Vamos a empezar a implementar el modelo. Crearemos una clase Product.java con el código a continuación:

```
public class Product {
   private int id;
   private String name;
   private double price;

   public Product(int id, String pname, double price) {
       this.id = id;
       this.name = pname;
       this.price = price;
   }
}
```

En una aplicación este modelo tendría asociado algún método de persistencia: MongoDB, MariaDB, entre otros por ejemplo haciendo uso de decorators. En este ejemplo para simplificar no habrá persistencia.

3.2. El servicio.

El servicio se encarga de proporcionarnos acceso a los datos haciendo uso del modelo y la persistencia asociada. Como en este caso no tenemos persistencia crearemos varias instancias del modelo. Habitualmente se crea una interfaz y se implementa en una clase. Vamos a crear la siguiente interfaz:

```
public interface IProductService {
   Set<Product> findAll();
}
```

Y a continuación la siguiente clase que la implementa:

```
@Service
public class ProductService implements IProductService
{
   @Override
   public Set<Product> findAll()
   {
       Set<Product> products = new HashSet();
       products.add(new Product(100, "Mobile", 300.00));
       products.add(new Product(101, "Smart TV", 600.00));
       products.add(new Product(102, "Washing Machine", 400.00));
       products.add(new Product(103, "Laptop", 450.00));
```

```
        products.add(new Product(104, "Air Conditioner", 1500.00));
        products.add(new Product(105, "Refrigerator ", 800.00));
        return products;
    }
}
```

3.3. El controlador.

Ahora nos falta configurar cómo será el acceso a nuestro servicio web, qué end point tendrá y el formato de la respuesta. Para ello se implementa un controlador o controller. Implementa el siguiente controlador que contendrá todos los end points relacionados con los productos.

```
@RestController
public class ProductController
{
    @Autowired
    private IProductService productService;

    @GetMapping(value = "/product")
    public ResponseEntity<Set<Product>> getProduct()
    {
        Set<Product> products = productService.findAll();
        return new ResponseEntity<>(products, HttpStatus.OK);
    }
}
```

3.4. Comprobación del funcionamiento de la API.

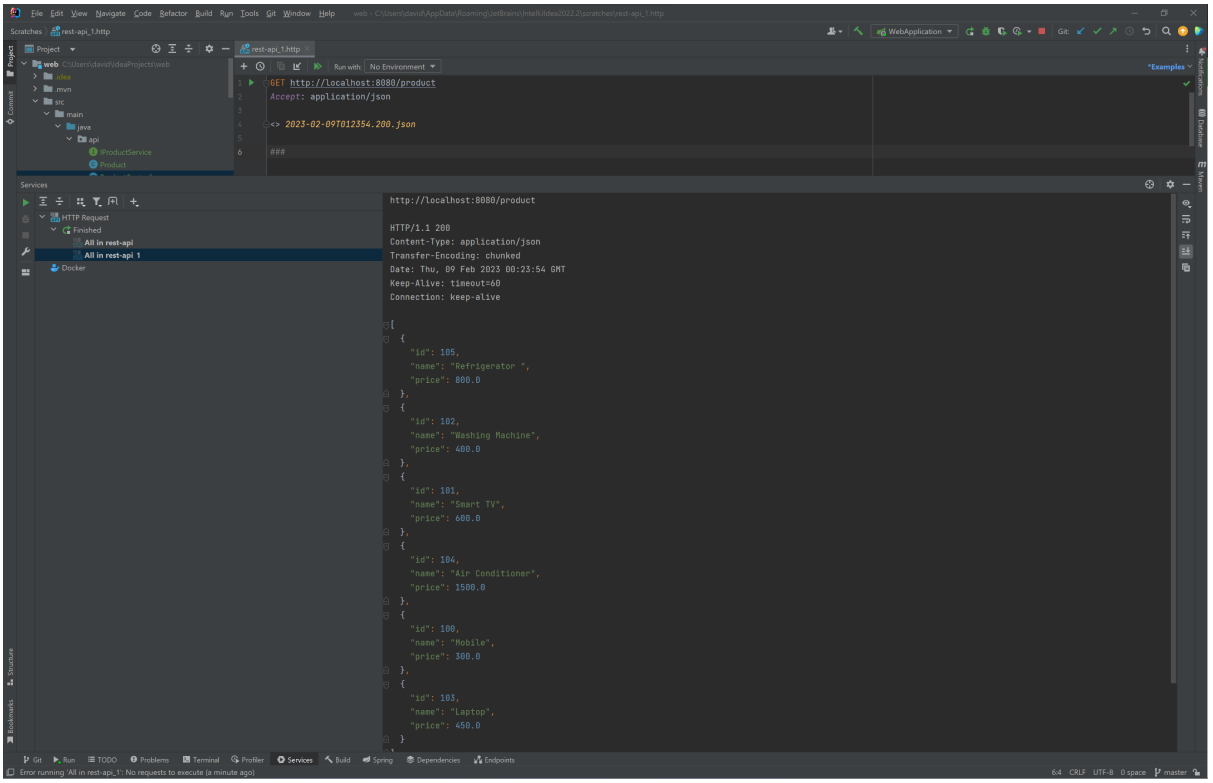
Ahora vamos a comprobar el funcionamiento de la API. Ejecutamos la aplicación y debería haberse publicado nuestro servicio:

http://localhost:8080/product

Para comprobar el correcto funcionamiento tenemos varias herramientas:

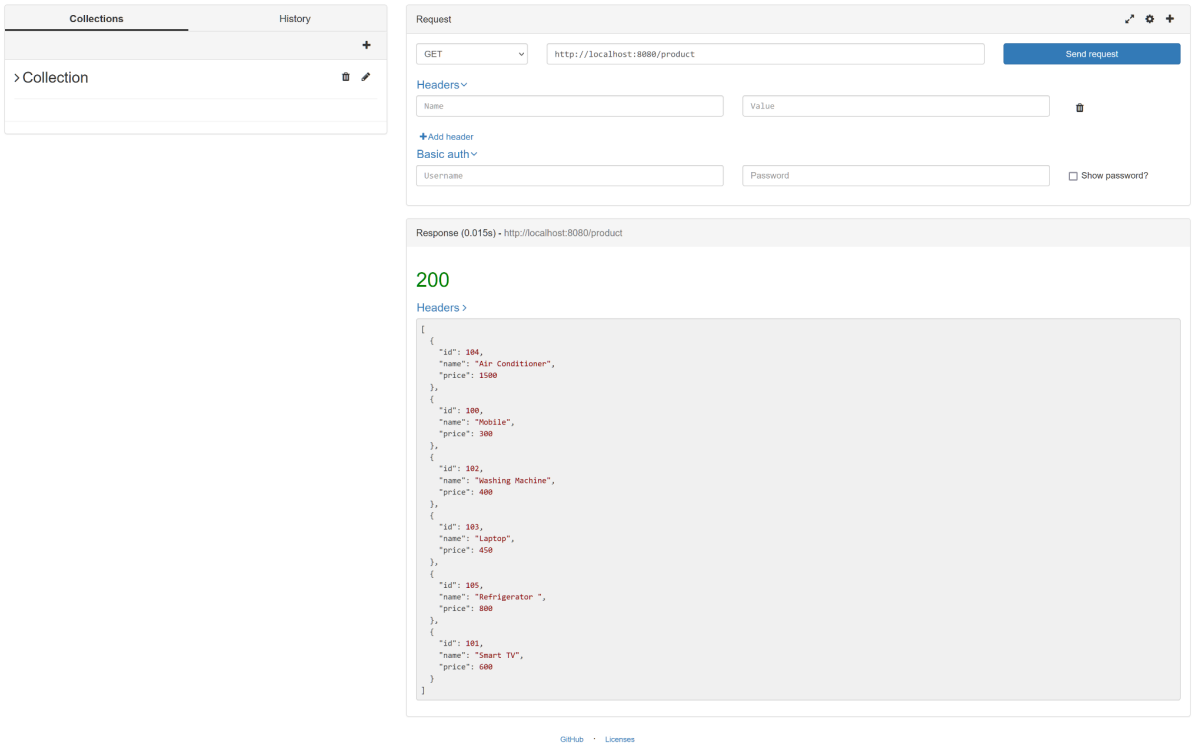
- ➔ **Navegador**. No es recomendable pues no nos proporciona demasiada información, pero abrir la URL desde el navegador nos mostrará la respuesta de la petición.

- ➔ **Cliente HTTP de Intellij**. Nos facilita la depuración y guarda historial de las peticiones. Se encuentra en Tools / HTTP Client.



➔ **RESTED**. Utilizando un complemento para el navegador que facilita el añadir parámetros a las peticiones tanto HTTP como dentro del cuerpo de la petición. Es la herramienta más flexible y la más recomendable. En Firefox existe un ADD-ON llamado Rested que es sencillo de utilizar (aunque hay muchos más con funcionalidades similares).

</> RESTED



Ya tenemos nuestra primera API creada, accede a ella utilizando los 3 métodos descritos.

3.4. Comprobación del funcionamiento de la API.