

1. Planteamiento del problema

En la actualidad el desarrollo de servicios web se ha convertido en un estándar en la programación al momento que se requiere compartir o intercambiar información entre cliente, servidor; de allí nació la necesidad de usar mecanismo que permitan gestionarlos y monitorizarlos, para esto se necesita de alguna herramienta la cual facilite la invocación de dichas herramientas.

Por otro lado los servicios basados en el estilo REST están aumentando debido a que se trata de una arquitectura mucho más amigable con el usuario, más ligera y más simple de utilizar que su predecesor el SOAP. Debido a esa motivación y buscando la innovación se plantea la alternativa de implementar un servicio web rest con el uso de la infraestructura de Spring Boot, la cual elimina la necesidad de configurar la aplicación a trabajar con el uso de archivos XML.

2. Objetivos

a) Objetivo general

Desarrollar un sistema basado en la creación de un Servicio Web Rest con el uso del framework Spring boot para verificar su versatilidad a la hora de crear servicios web.

b) Objetivos específicos

- Identificar las características y funcionamiento de un servicio web con un modelo de arquitectura de software (Rest) para desarrollar dicho entorno a partir de un modelo determinado y de esta manera establecer la relación cliente servidor.
- Determinar la utilidad de Spring Boot mediante la investigación de sus dependencias de tal manera que permita la creación de un servicio web rest.
- Desarrollar un servicio web restfull con Spring Boot con acceso a una base de datos (Firebase) para lograr obtener los datos requeridos por el usuario, utilizando el IDE Netbeans .

3. Estado del Arte

El término servicios web según la W3C (World Wide Web Consistorium) proporciona un medio estándar de interoperabilidad entre diferentes aplicaciones de software que se ejecutan en una variedad de plataformas y/o marcos de trabajo. Un servicio web es un sistema de software diseñado para soportar la interacción entre máquinas a través de una red. Tiene una interfaz descrita en un formato procesable por máquina. Se

podría entender como un conjunto de aplicaciones o tecnologías que pueden interoperar en la web, su principal función es la de intercambiar datos entre sí; tanto los proveedores ofrecen sus servicios y los usuarios solicitan el servicio llamando a estos procedimientos utilizando la web (David Booth, s.f.).

En el desarrollo de software propuesto en el documento de (Nguyen, 2015), los servicios IDE se utilizan para ayudar a los desarrolladores en las tareas de programación. Sin embargo, en las aplicaciones web dinámicas, dado que el código del lado del cliente está incrustado en el programa del lado del servidor como literales de cadenas, proporcionar servicios IDE para dicho código incrustado es un desafío.

Según (Terpak, 2014) REST es un tipo de arquitectura orientada a recursos para el desarrollo de servicios web reemplazando especificaciones como SOAP y gracias al desarrollo de los servicios web REST, no es necesario realizar una programación exclusiva para cada plataforma, ni limitada por el lenguaje, el hardware la ejecución sino que es suficiente con exponer el servicio web para ser consumido por los clientes, debido a ello empresas como Spring ha desarrollado frameworks que facilitan el desarrollo para ser expuestos para su utilización.

(Gutierrez, 2016) Menciona que Spring Boot es un framework que posee las bibliotecas necesarias para la creación de aplicaciones ejecutables basadas en Spring. Los componentes que posee Spring Boot permiten simplificar los pasos de la selección de las dependencias necesarias para el proyecto a desarrollar y su despliegue en el servidor. No requiere de configuración usando archivos en XML/JSON y permite la creación de servicios REST en Java de manera fácil, centrando al desarrollador en los elementos críticos de desarrollo específico de su aplicación.

4. Marco Teórico

El término REST (Representational State Transfer) se originó en el año 2000, descrito en la tesis de Roy Fielding, padre de la especificación HTTP. Un servicio REST no es una arquitectura software, sino un conjunto de restricciones con las que podemos crear un estilo de arquitectura software, la cual podremos usar para crear aplicaciones web respetando HTTP.

Hoy en día la mayoría de las empresas utilizan API REST para crear servicios. Esto se debe a que es un estándar lógico y eficiente para la creación de servicios web. Por poner algún ejemplo tenemos los sistemas de identificación de Facebook, la autenticación en los servicios de Google (hojas de cálculo, Google Analytics, ...).

Según (Alvarez, 2013) las restricciones que definen a un sistema RESTful serían:

Cliente/Servidor : Esta restricción mantiene al cliente y al servidor débilmente acoplados. Esto quiere decir que el cliente no necesita conocer los detalles de implementación del servidor y el servidor se “despreocupa” de cómo son usados los datos que envía al cliente.

Sin estado : Aquí se dice que cada petición que recibe el servidor debería ser independiente, es decir, no es necesario mantener sesiones.

Cacheable : Debe admitir un sistema de almacenamiento en caché. La infraestructura de red debe soportar una caché de varios niveles. Este almacenamiento evitará repetir varias conexiones entre el servidor y el cliente para recuperar un mismo recurso.

Interfaz Uniforme : Define una interfaz genérica para administrar cada interacción que se produzca entre el cliente y el servidor de manera uniforme, lo cual simplifica y separa la arquitectura. Esta restricción indica que cada recurso del servicio REST debe tener una única dirección, “URI”.

Sistema de Capas: El servidor puede disponer de varias capas para su implementación. Esto ayuda a mejorar la escalabilidad, el rendimiento y la seguridad.

A continuación algunas características de una API REST:

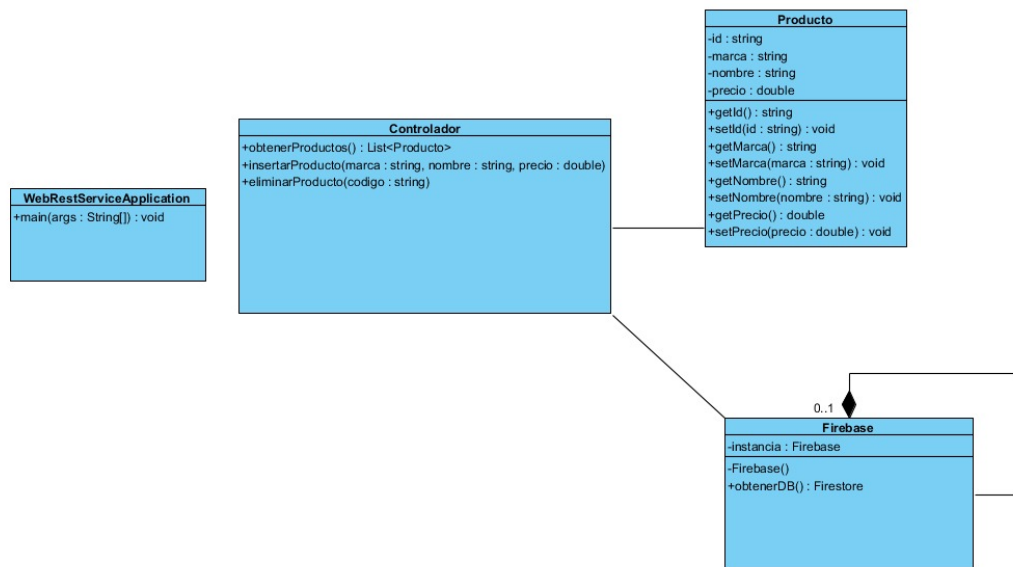
Las operaciones más importantes que permitirán manipular los recursos son cuatro: GET para consultar y leer, POST para crear, PUT para editar y DELETE para eliminar. El uso de hipermedios (término que en el ámbito de las páginas web define el conjunto de procedimientos para crear contenidos que contengan texto, imagen, vídeo, audio y otros métodos de información) para permitir al usuario navegar por los distintos recursos de una API REST a través de enlaces HTML. Para terminar, comentar que lo más importante a tener en cuenta al crear nuestro servicio o API REST no es el lenguaje en el que se implemente sino que las respuestas a las peticiones se hagan en XML o JSON, ya que es el lenguaje de intercambio de información más usado. Algunos frameworks con los que podremos implementar nuestras APIs: Las más usadas son JAX-RS y Spring Boot para Java, Django REST framework para Python, Laravel para PHP o Restify para Node.js(BBVA, 2016)

Ahora se definirá lo que es Web Services, un Web Service, o Servicio Web, es un método de comunicación entre dos aparatos electrónicos en una red. Es una colección de protocolos abiertos y estándares usados para intercambiar datos entre aplicaciones o sistemas. Las aplicaciones escritas en varios lenguajes de programación que funcionan

en plataformas diferentes pueden utilizar web service para intercambiar información a través de una red. La interoperatividad, por ejemplo entre Java y Python o Windows y Linux se debe al uso de estándares abiertos. (Lazaro, 2015)

5. Diagramas

Diagrama UML



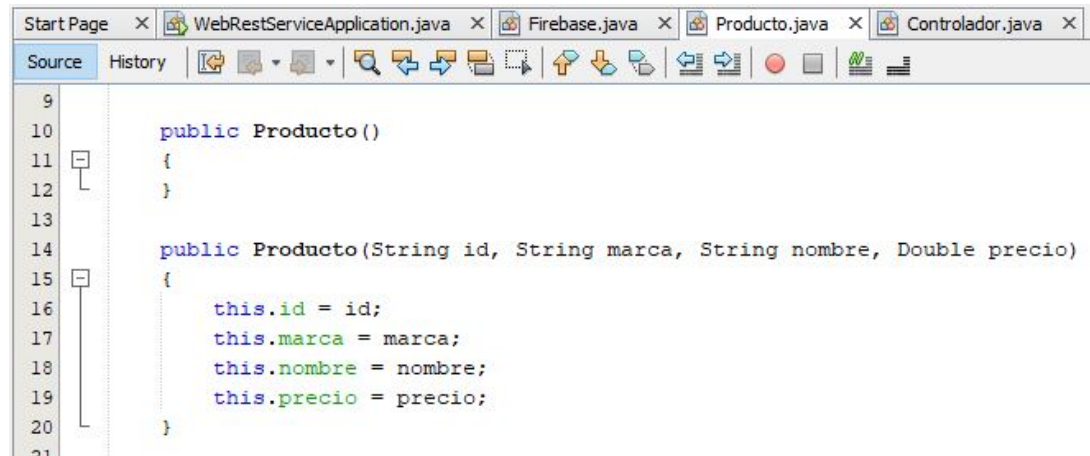
6. Explicación del código fuente

Explicación del código en Java

- Se debe crear una clase llamada “Producto” la cual tiene los atributos de un producto como su (id, Marca, Nombre, Precio).

```
Start Page  X  WebRestServiceApplication.java  X  Firebase.java  X  Producto.java  X
Source  History  [Icons]
1  package bbbs.controlador.WebRestService;
2
3  public class Producto
4  {
5      private String id;
6      private String marca;
7      private String nombre;
8      private Double precio;
9  }
```

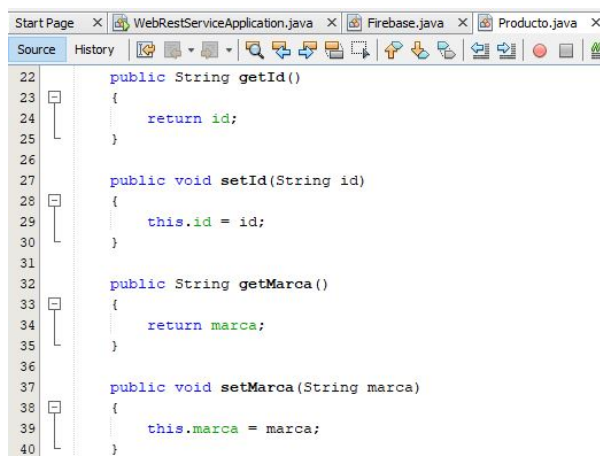
- Se crea los constructores



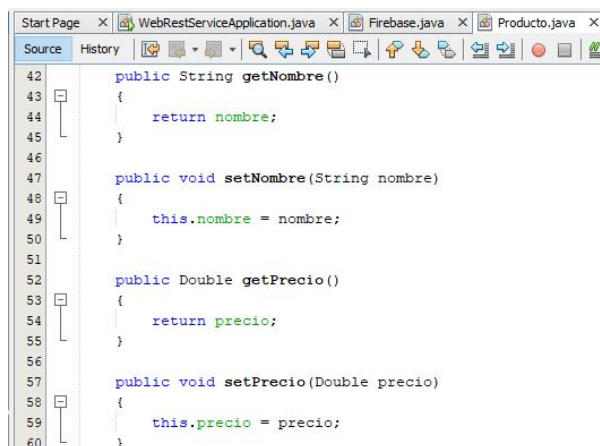
```
Start Page x WebRestServiceApplication.java x Firebase.java x Producto.java x Controlador.java x
Source History
9
10 public Producto()
11 {
12 }
13
14 public Producto(String id, String marca, String nombre, Double precio)
15 {
16     this.id = id;
17     this.marca = marca;
18     this.nombre = nombre;
19     this.precio = precio;
20 }
21
```

Nota: Cabe recalcar que esta es una clase de persistencia ya que se va a tener los mismos atributos que existirán en la base de datos en este caso Fire Base.

- Se crea los getters y setters para posteriormente poder ser llamados.

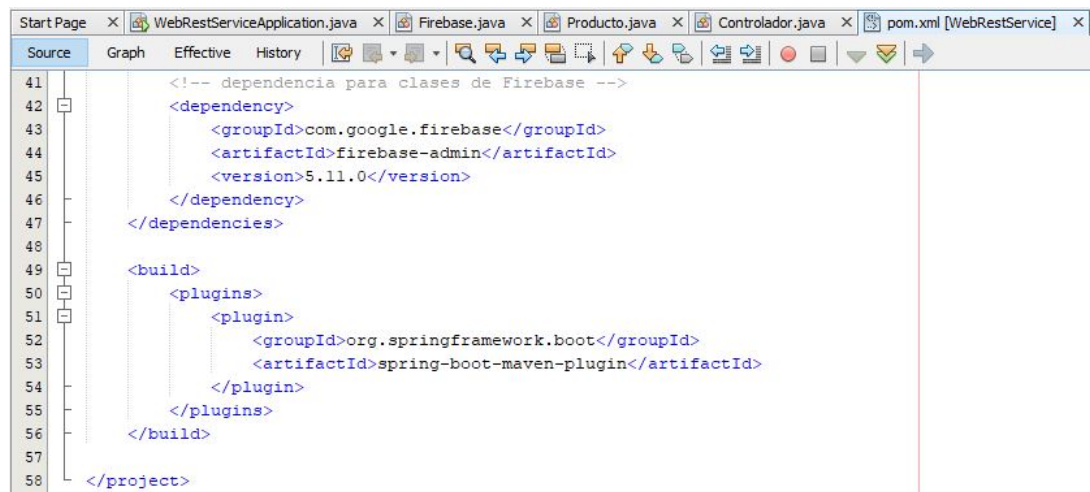


```
Start Page x WebRestServiceApplication.java x Firebase.java x Producto.java x
Source History
22 public String getId()
23 {
24     return id;
25 }
26
27 public void setId(String id)
28 {
29     this.id = id;
30 }
31
32 public String getMarca()
33 {
34     return marca;
35 }
36
37 public void setMarca(String marca)
38 {
39     this.marca = marca;
40 }
```



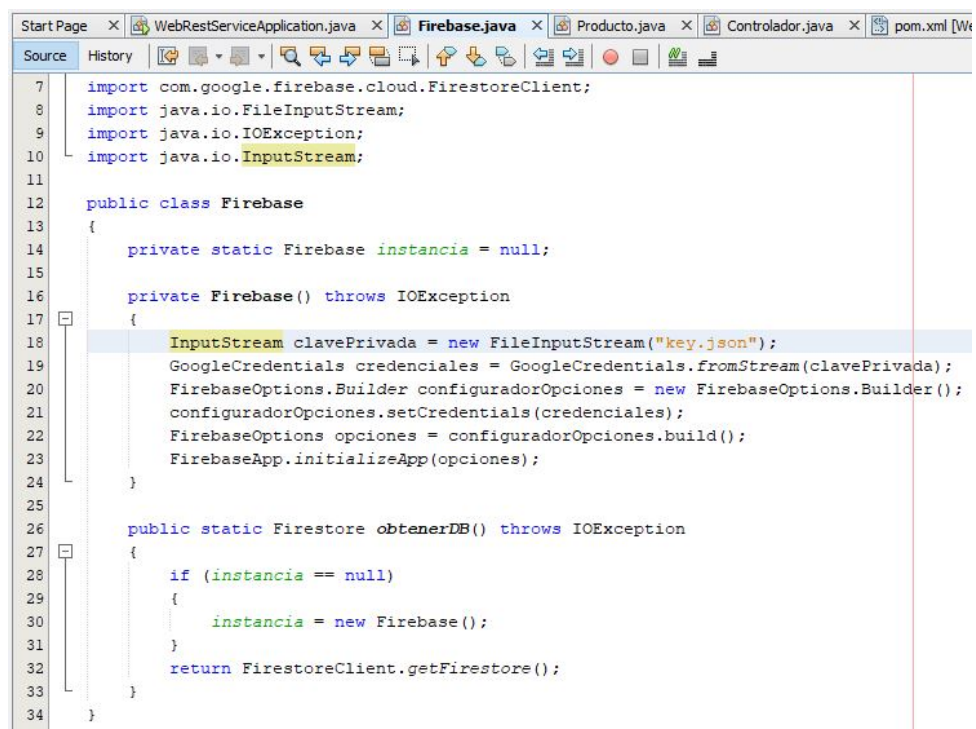
```
Start Page x WebRestServiceApplication.java x Firebase.java x Producto.java x
Source History
42 public String getNombre()
43 {
44     return nombre;
45 }
46
47 public void setNombre(String nombre)
48 {
49     this.nombre = nombre;
50 }
51
52 public Double getPrecio()
53 {
54     return precio;
55 }
56
57 public void setPrecio(Double precio)
58 {
59     this.precio = precio;
60 }
```

- Luego se debe crear una clase llamada FireBase, la cual servirá para realizar la conexión a la base de datos, pero antes se debe colocar las dependencias para poder manejar todas las clases de Firebase.



```
41 <!-- dependencia para clases de Firebase -->
42 <dependency>
43     <groupId>com.google.firebase</groupId>
44     <artifactId>firebase-admin</artifactId>
45     <version>5.11.0</version>
46 </dependency>
47 </dependencies>
48
49 <build>
50     <plugins>
51         <plugin>
52             <groupId>org.springframework.boot</groupId>
53             <artifactId>spring-boot-maven-plugin</artifactId>
54         </plugin>
55     </plugins>
56 </build>
57
58 </project>
```

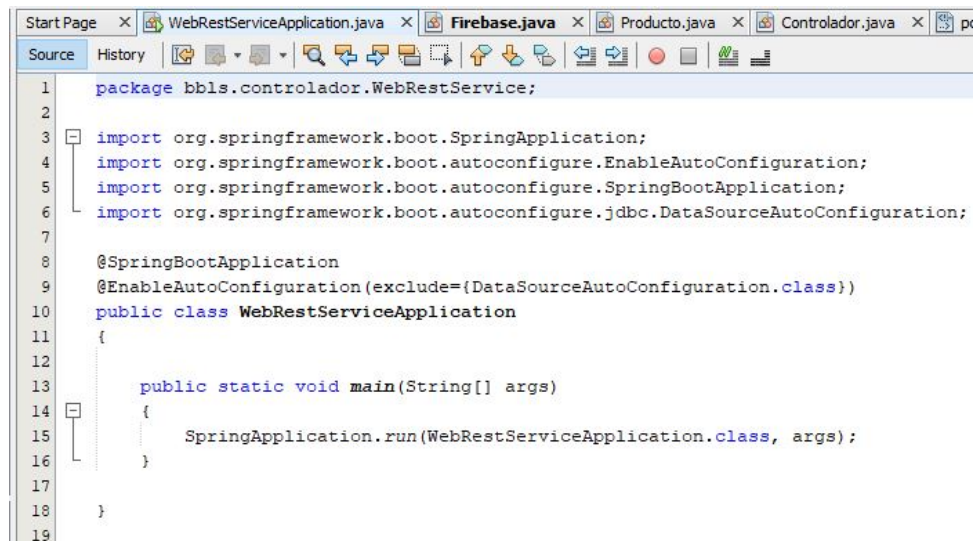
- Dentro de la clase FireBase se crea un método, en este caso llamado obtenerDB(), este método permite obtener la instancia, cuando no hay nada esta nulo y cuando existe algo retorna a la instancia y asegura que se conecte solo a una base de datos.



```
7 import com.google.firebase.cloud.FirestoreClient;
8 import java.io.FileInputStream;
9 import java.io.IOException;
10 import java.io.InputStream;
11
12 public class Firebase
13 {
14     private static Firebase instancia = null;
15
16     private Firebase() throws IOException
17     {
18         InputStream clavePrivada = new FileInputStream("key.json");
19         GoogleCredentials credenciales = GoogleCredentials.fromStream(clavePrivada);
20         FirebaseOptions.Builder configuradorOpciones = new FirebaseOptions.Builder();
21         configuradorOpciones.setCredentials(credenciales);
22         FirebaseOptions opciones = configuradorOpciones.build();
23         FirebaseApp.initializeApp(opciones);
24     }
25
26     public static Firestore obtenerDB() throws IOException
27     {
28         if (instancia == null)
29         {
30             instancia = new Firebase();
31         }
32         return FirestoreClient.getFirestore();
33     }
34 }
```

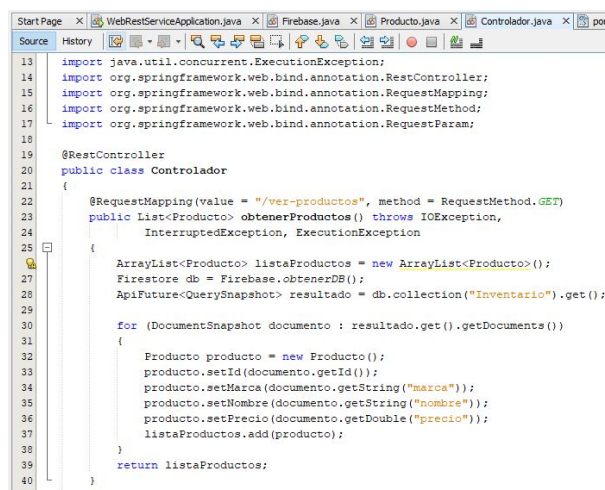
- En la clase principal creada con el nombre de WebRestServiceApplication, se

puede observar el método que ejecuta el servicio; el framework está diseñado para que ejecute ya una base de datos y gestione automáticamente la conexión, en otras palabras para evitar error de compilación por datasource inexistente.



```
1 package bbbs.controlador.WebRestService;
2
3 import org.springframework.boot.SpringApplication;
4 import org.springframework.boot.autoconfigure.EnableAutoConfiguration;
5 import org.springframework.boot.autoconfigure.SpringBootApplication;
6 import org.springframework.boot.autoconfigure.jdbc.DataSourceAutoConfiguration;
7
8 @SpringBootApplication
9 @EnableAutoConfiguration(exclude={DataSourceAutoConfiguration.class})
10 public class WebRestServiceApplication
11 {
12
13     public static void main(String[] args)
14     {
15         SpringApplication.run(WebRestServiceApplication.class, args);
16     }
17
18 }
19
```

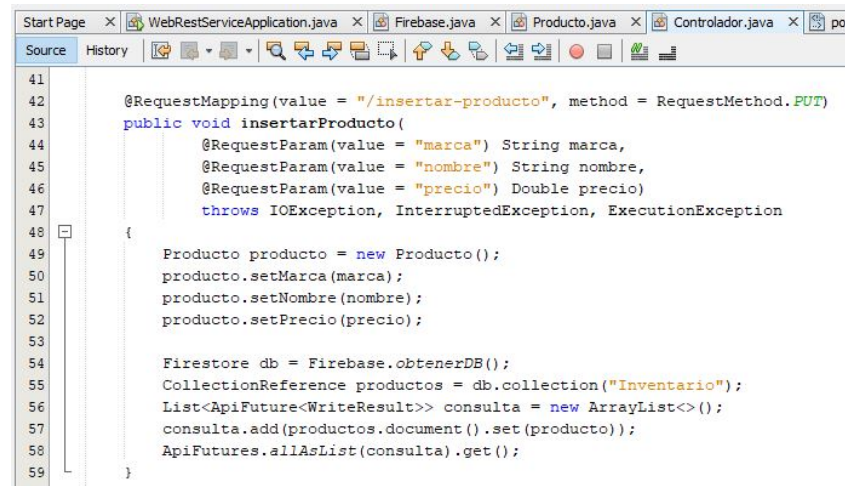
- A continuación se crea la clase Controlador en la cual se va a desarrollar el servicio rest. El primer fragmento de código sirve para ingresar todos los atributos que contiene un producto para que así se guarden en la base de datos a través del objeto collection. Para consultar todos los atributos que se han ingresado y guardado en la base de datos, luego se procede a hacer una instancia del servicio creando un método, es así que se crea el método obtenerProductos el mismo que retorna todos los productos que están ingresados, se obtiene la instancia de la base de datos y esa base de datos va a tener una colección los cuales se guardan en “resultado”.



```
13 import java.util.concurrent.ExecutionException;
14 import org.springframework.web.bind.annotation.RestController;
15 import org.springframework.web.bind.annotation.RequestMapping;
16 import org.springframework.web.bind.annotation.RequestMethod;
17 import org.springframework.web.bind.annotation.RequestParam;
18
19 @RestController
20 public class Controlador
21 {
22     @RequestMapping(value = "/ver-productos", method = RequestMethod.GET)
23     public List<Producto> obtenerProductos() throws IOException,
24         InterruptedException, ExecutionException
25     {
26         ArrayList<Producto> listaProductos = new ArrayList<Producto>();
27         Firestore db = Firebase.obtenerDB();
28         ApiFuture<QuerySnapshot> resultado = db.collection("Inventario").get();
29
30         for (DocumentSnapshot documento : resultado.get().getDocuments())
31         {
32             Producto producto = new Producto();
33             producto.setId(documento.getId());
34             producto.setMarca(documento.getString("marca"));
35             producto.setNombre(documento.getString("nombre"));
36             producto.setPrecio(documento.getDouble("precio"));
37             listaProductos.add(producto);
38         }
39         return listaProductos;
40     }
41 }

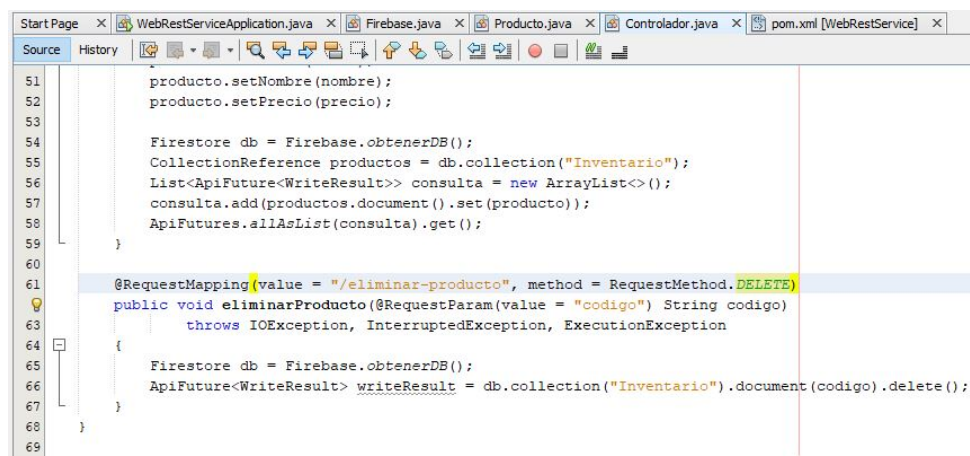
```

- Luego de ello dentro Controlador se crea el método insertarProducto que como su nombre lo indica será el encargado de recibir producto por producto y transportarlos a una lista de productos llamado inventario.



```
41
42 @RequestMapping(value = "/insertar-producto", method = RequestMethod.PUT)
43 public void insertarProducto(
44     @RequestParam(value = "marca") String marca,
45     @RequestParam(value = "nombre") String nombre,
46     @RequestParam(value = "precio") Double precio)
47     throws IOException, InterruptedException, ExecutionException
48 {
49     Producto producto = new Producto();
50     producto.setMarca(marca);
51     producto.setNombre(nombre);
52     producto.setPrecio(precio);
53
54     Firestore db = Firebase.obtenerDB();
55     CollectionReference productos = db.collection("Inventario");
56     List<ApiFuture<WriteResult>> consulta = new ArrayList<>();
57     consulta.add(productos.document().set(producto));
58     ApiFutures.allAsList(consulta).get();
59 }
```

- Finalmente dentro de la misma clase Controlador, se crea el método llamado eliminarProducto el cual se encarga de eliminar el producto del inventario a través de la id correspondiente a cada producto.



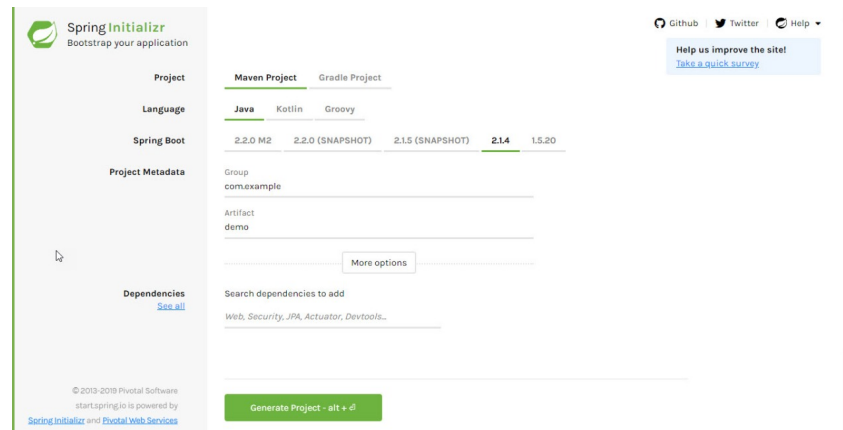
```
51     producto.setNombre(nombre);
52     producto.setPrecio(precio);
53
54     Firestore db = Firebase.obtenerDB();
55     CollectionReference productos = db.collection("Inventario");
56     List<ApiFuture<WriteResult>> consulta = new ArrayList<>();
57     consulta.add(productos.document().set(producto));
58     ApiFutures.allAsList(consulta).get();
59 }
60
61 @RequestMapping(value = "/eliminar-producto", method = RequestMethod.DELETE)
62 public void eliminarProducto(@RequestParam(value = "codigo") String codigo)
63     throws IOException, InterruptedException, ExecutionException
64 {
65     Firestore db = Firebase.obtenerDB();
66     ApiFuture<WriteResult> writeResult = db.collection("Inventario").document(codigo).delete();
67 }
68
69 }
```

7. Descripción de Prerrequisitos y Configuración

Para desarrollar este servicio, se necesita tener instalado un IDE, en este caso NetBeans con el JDK de 64 bits, se va a necesitar también en programa Postman para posteriormente probar el servicio que se ha creado. Se necesita una cuenta en Google para tener acceso a Firebase y poder guardar y recuperar de esta base de datos.

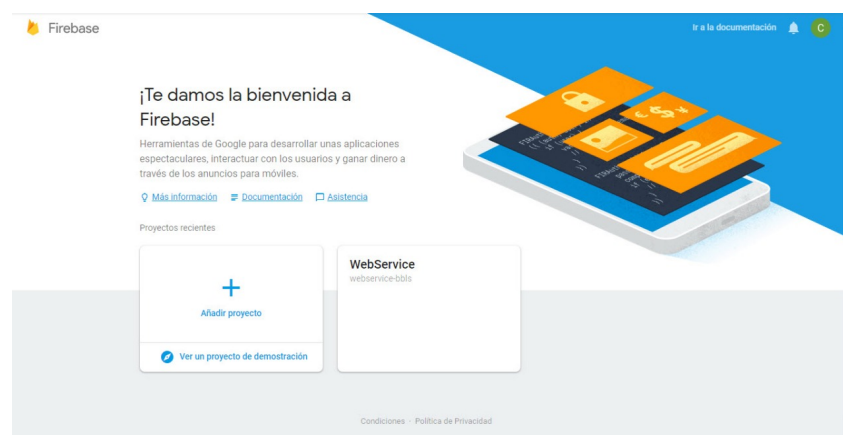
Es necesario contar con un JDK de bits, ya que con el de 32 bits no se podrá establecer una conexión con la base de datos.

SpringBoot: Para iniciar con la creación del proyecto, se ingresa a <https://start.spring.io/>

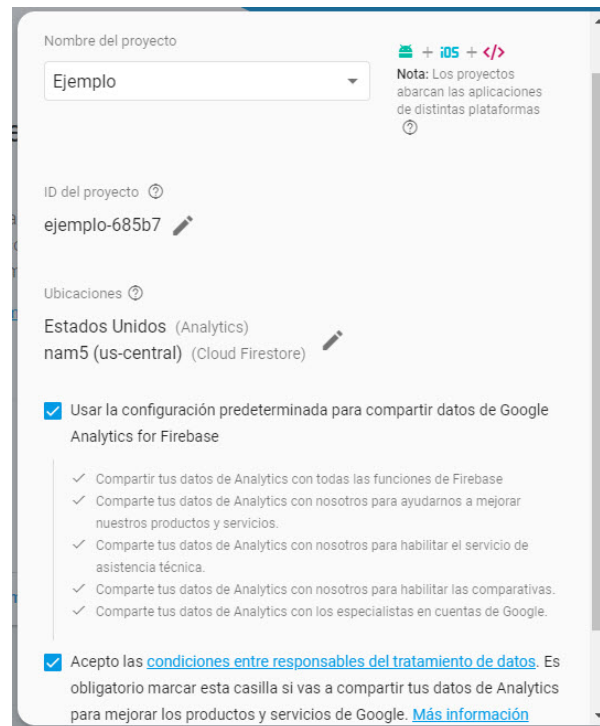
The screenshot shows the Spring Initializr web application. On the left, there's a sidebar with links for Project, Language, Spring Boot, Project Metadata, and Dependencies. The main area is divided into sections for Project (Maven Project), Language (Java), Spring Boot version (2.1.4), and Project Metadata (Group: com.example, Artifact: demo). There's a 'Generate Project' button at the bottom. The footer mentions '© 2019-2018 Pivotal Software' and 'start.spring.io is powered by Spring Initializr and Pivotal Web Services'.

En esta ventana se realiza la configuración inicial del proyecto, selecciona Maven Project, posteriormente el lenguaje de programación en el cual se va a trabajar, la versión de Spring que se utilizará, luego se ingresa el nombre del proyecto y el nombre de la clase que se creará por defecto. Por último, se selecciona las dependencias, inicialmente seleccionamos Web. Clic en Generate Project, y se descargará un archivo .zip que contiene el proyecto, se exporta y se abre el proyecto con NetBeans.

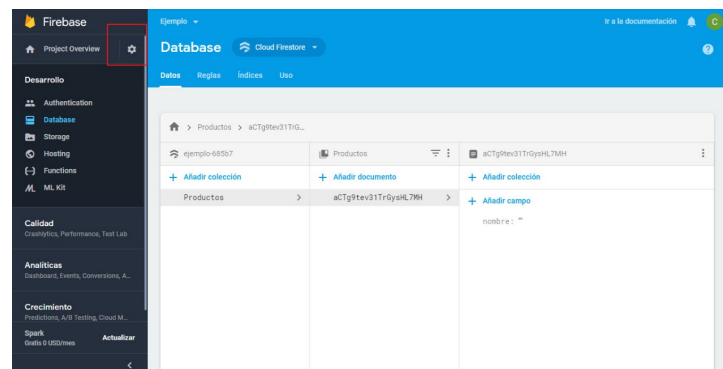
FireBase: Para configurar el acceso a la base de datos, se accede a <https://console.firebase.google.com> y se inicia sesión con una cuenta de Google. Realizado esto se presenta la siguiente página:



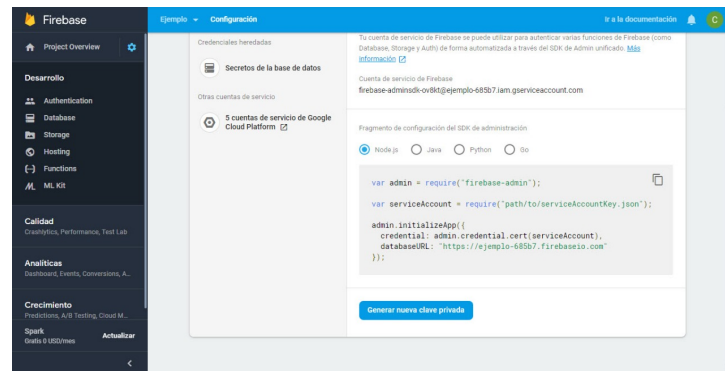
Se añade un proyecto y se aceptan las condiciones.



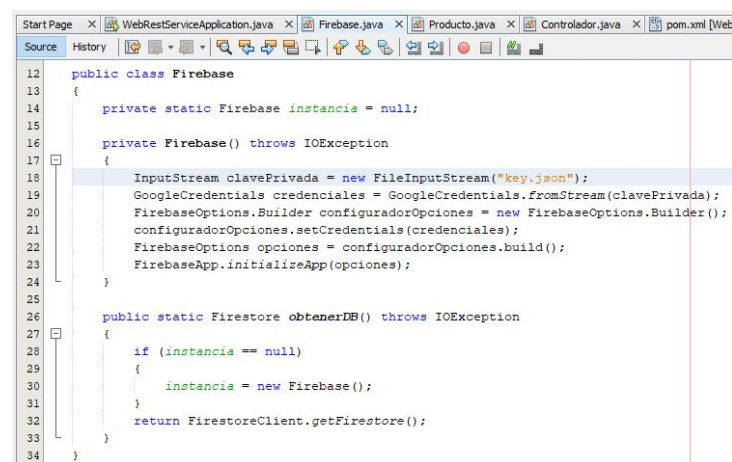
Una vez dentro de Firebase, se procede a crear la base de datos que se va a utilizar, con todos los campos que se requieran manejar.



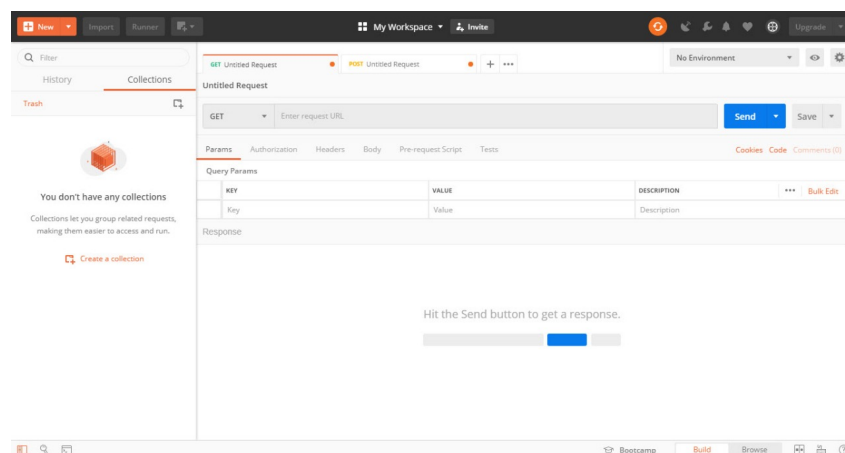
El paso siguiente es ir a Configuraciones, posteriormente a Configuración del Proyecto, Cuentas de Servicio y al final de esta pestaña clic en crear nueva clave privada, dentro de esta pestaña se encuentra el código en Java para la configuración del SDK de administración.



Con este código y la clave que se descarga en un archivo .JSON se procede a crear un clase en el proyecto para establecer la conexión entre el servicio que se va a crear y la base de datos a utilizar para este propósito.



Postman: Basta con descargar Postman e instalarlo en la computadora, servirá para poder probar los servicios que se han desarrollado.



8. Aportaciones

Firestore: Firestore es una plataforma desarrollada por Google que facilita el desarrollo de apps, proporcionando un servidor backend para las aplicaciones. Además, el mismo backend puede ser utilizado de forma común en diversas plataformas: Android, IOS y web.



Postman: Es una herramienta que está orientada a desarrolladores web, dentro de esta herramienta se pueden realizar peticiones HTTP. Resulta de mucha ayuda este programa a la hora de programar e ir probando el funcionamiento de los avances que se realicen.



9. Conclusiones

- Luego de realizar la investigación se puede concluir que la importancia de un servicio web Rest se basa en que en la actualidad no existe proyecto o aplicación que no disponga de una API REST para la creación de servicios profesionales a partir de ese software. Hay cientos de empresas que generan negocio gracias a REST y las APIs REST. Sin ellas, todo el crecimiento en horizontal sería prácticamente imposible. Esto es así porque REST es el estándar más lógico, eficiente y habitual en la creación de APIs para servicios de Internet.
- Utilizar Spring Boot es de gran ayuda para los desarrolladores web, ya que le permite al programador preocuparse en el desarrollo de la aplicación, la creación de dependencias y el despliegue en servidor es cosa del framework.

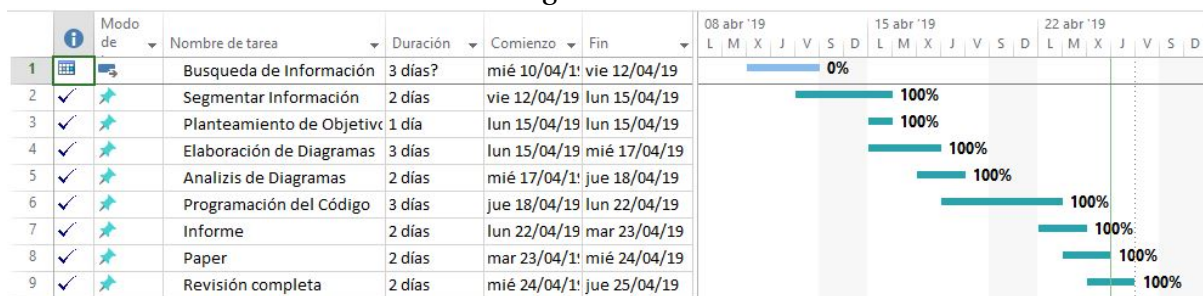
- El tener acceso a una base de datos (Firebase) para el servicio web creado con Netbeans y con la ayuda de Spring Boot es de gran ayuda e importancia ya que permite verificar que los métodos del servicio web se están ejecutando adecuadamente.

10. Recomendaciones

- Se recomienda verificar todos los prerequisites establecidos, en especial la versión del JDK, ya que, sin la versión necesaria, de 64 bits, no se podrá establecer la conexión con la base de datos utilizada.
- Existen otros frameworks como Node.js, pero se recomienda trabajar con Spring Boot ya que se ha desarrollado con Java durante años y en cuanto a seguridad es más robusto.
- Es recomendable trabajar el consumo del servicio web RESTful a través de un programa que nos permita la interacción con el usuario como lo es Postman ya que es mucho más amigable y entendible que trabajar mediante el navegador.

11. Cronograma

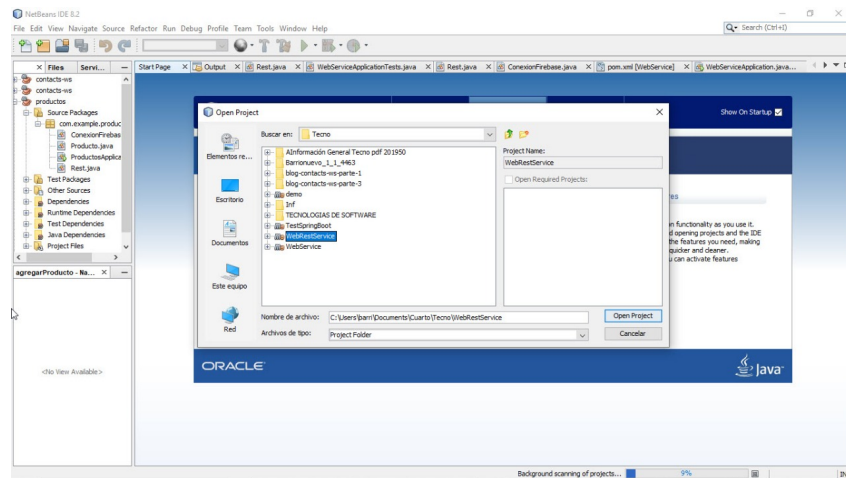
Diagrama Gantt



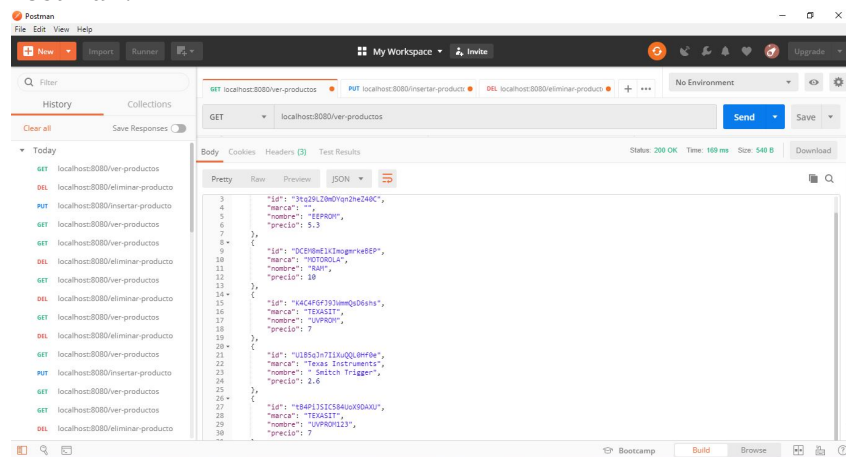
12. Anexos

a) Manual de Usuario

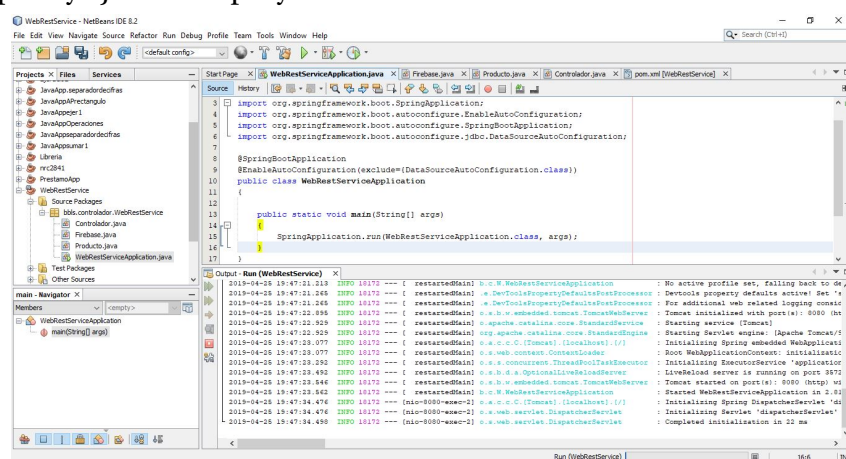
- Abrir NetBeans y abrir el proyecto WebRestService:



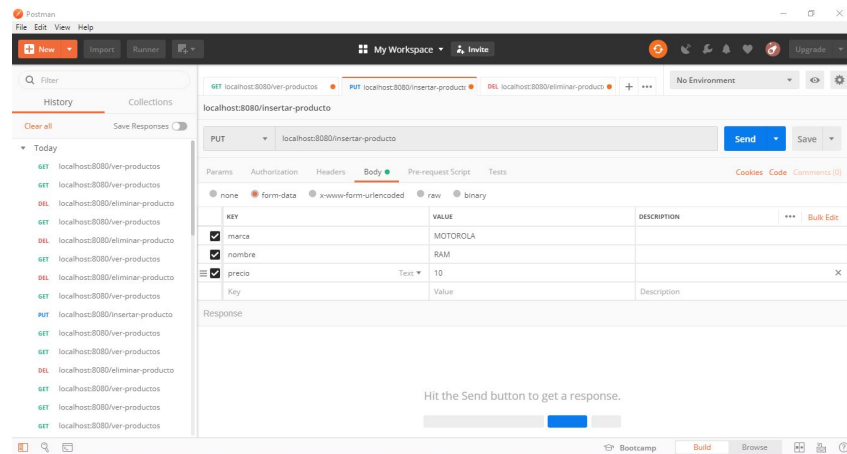
■ Abrir Postman:



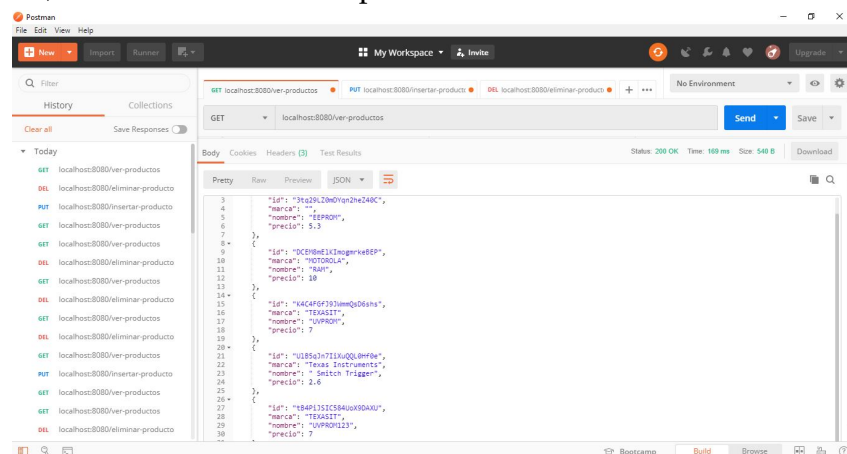
■ Compilar y ejecutar el proyecto:



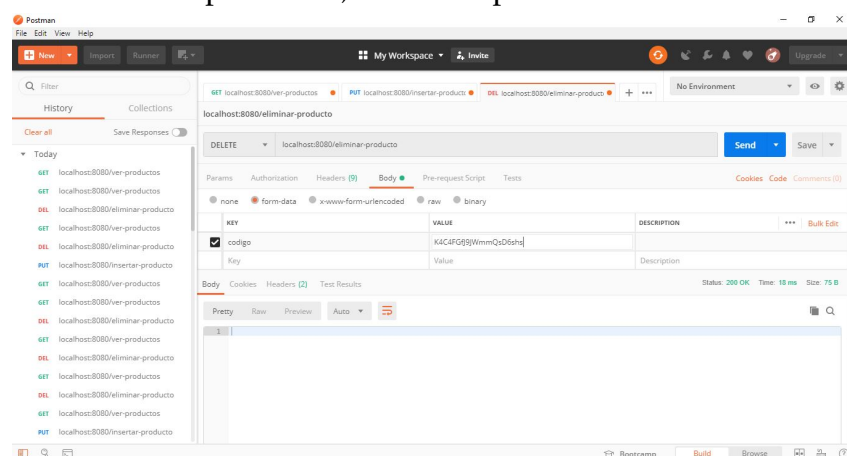
- Abrir Postman, seleccionar la petición HTTP requerida en este caso PUT, y el url: localhost:8090/Insertar-producto , en este paso se va a agregar un producto a la base de datos.



- Seleccionar la petición HTTP requerida en este caso GET, y el url: localhost:8090/ ver-productos , retorna todos los productos guardados en la base de datos, se muestra también el producto añadido recientemente:

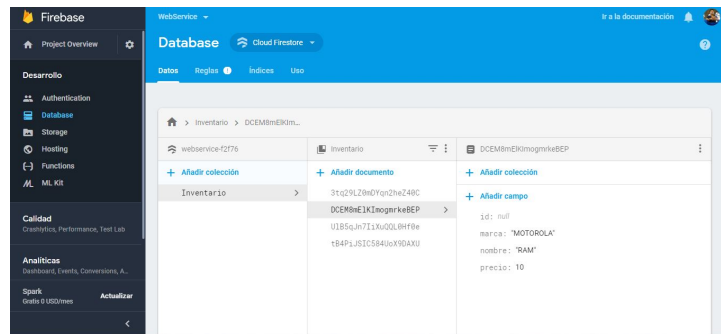


- Seleccionar la petición HTTP requerida en este caso DELETE, y el url: localhost: 8090/eliminar-productos, elimina el producto de acuerdo al código:



- Seleccionar la petición HTTP requerida en este caso GET, y el url: local-

host:8090/ ver-productos, despliega nuevamente todos los productos y se observa que el producto eliminado en el punto anterior ya no se encuentra en la base de datos:



Nota: Si se desea ver la base de datos, se accede a la página de Firebase con la cuenta que previamente se ha creado la base de datos

Referencias

- Alvarez, C. (2013). Introducción a servicios rest. *ArquitecturaJava*, 4(3), 34–35, Recuperado de: <https://www.arquitecturajava.com/servicios-rest/>.
- BBVA. (2016). Api rest. *API MARKET*, 4(3), 2–3, <https://bbvaopen4u.com/es/actualidad/api-rest-que-es-y-cuales-son-sus-ventajas-en-el-desarrollo-de-proyectos>.
- David Booth, H. H. (s.f.). Arquitectura de servicios web.
- Gutierrez, F. (2016). *Pro spring boot*.
- Lazaro, D. (2015). Introducción a los web services. *IEEE*, 5(1).
- Nguyen, H. V. (2015). Varis: Soporte ide para el código de cliente incorporado en aplicaciones web php. *IEEE*, 8(3), 896–906, Recuperado de: <https://ieeexplore.ieee.org/document/7203045/authorsauthors>.
- Terpak, J. (2014). La propuesta de servicio web para soporte de cálculos termoquímicos específicos. *IEEE*, 5(1), 63–75, Recuperado de: <https://ieeexplore.ieee.org/document/6843677/authorsauthors>.