API REST Ventas SPRING BOOT.

Vamos a crear el APIREST que permita hacer la facturación en el backend Spring Boot y como base de datos MongoDB, además vamos a documentar el api con swagger. El ejercicio contiene el siguiente procedimiento:

1. Creación del Proyecto.
2. Agregar Swagger al Proyecto.
3. Creación de la Estructura del Proyecto.
4. Configuración de la base de datos MongoDB
5. Creación del modelo de Datos
6. Cree un repositorio de datos de Spring - VentasRepositorio.java
7. Capa de servicio (usa repositorio)
8. Creación de las API: VentasControlador
9. Ejecución de la aplicación de arranque Spring
10. Pruebe las API REST con Swagger.

1. Creación del Proyecto (Backend).

Vamos a crear un proyecto Spring boot con las agregando las dependencias Spring web, devTools y Spring Data MongoDB.

2. Agregar Swagger al Proyecto.

Agregamos dos dependencias nuevas para gestionar el uso de swagger:

```
<dependency>
	<groupId>io.springfox</groupId>
	<artifactId>springfox-swagger2</artifactId>
	<version>2.9.2</version>
</dependency>

<dependency>
	<groupId>io.springfox</groupId>
	<artifactId>springfox-swagger-ui</artifactId>
	<version>2.9.2</version>
</dependency>
```

En la clase principal se agrega la anotación "@EnableSwagger2" para activar el uso de la herramienta, además se debe agregar un código que nos permite agregar una arquitectura MVC con swagger al archivo de de ejecución, la documentación de la herramienta se encuentra en el siguiente url:

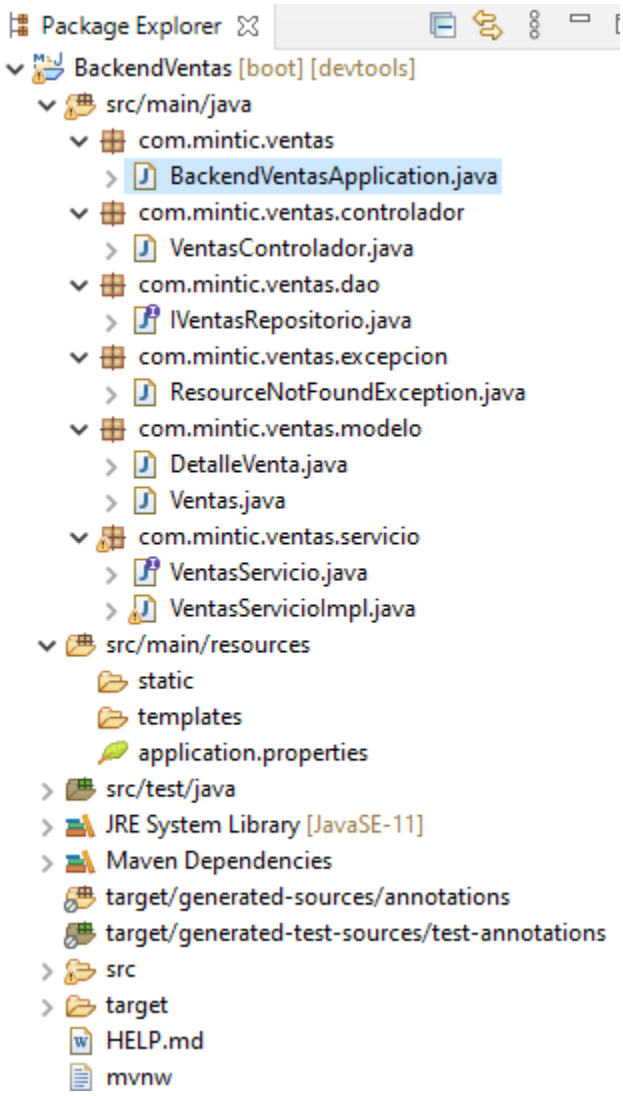
En el siguiente gráfico se ilustra el contenido de la página en la sección de implementación de spring boot:

El contenido del archivo sería el siguiente:

```java
package com.mintic.ventas;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.context.annotation.Bean;

import springfox.documentation.builders.PathSelectors;
import springfox.documentation.builders.RequestHandlerSelectors;
import springfox.documentation.spi.DocumentationType;
import springfox.documentation.spring.web.plugins.Docket;
import springfox.documentation.swagger2.annotations.EnableSwagger2;

@SpringBootApplication
@EnableSwagger2
public class BackendVentasApplication {

    @Bean
    public Docket ventasApi() {
        return new Docket(DocumentationType.SWAGGER_2)
                .select()
                .apis(RequestHandlerSelectors.any())
                .paths(PathSelectors.any())
                .build();
    }


    public static void main(String[] args) {
        SpringApplication.run(BackendVentasApplication.class, args);
    }

}
```

3. Creación de la Estructura del Proyecto.

Agregamos los packages para crear los archivos necesarios en el proyecto como se aprecia en la figura:



4. Configuración de la base de datos MongoDB.

Spring Boot configura automáticamente la mayoría de las cosas según las dependencias que ha agregado en el archivo pom.xml.

Dado que hemos agregado una dependencia spring-boot-starter-mongodb, Spring Boot intenta construir una conexión con MongoDB leyendo la configuración de la base de datos del archivo application.properties.

Respetando las recomendaciones de la universidad en el documento guía del proyecto como se muestra en la figura se dará una estructura a la base de como se aprecia a continuación:

db_ventas	Ventas {
	cedula_cliente (integer, optional),
	codigo_venta (integer, optional),
	detalle_venta (Array[DetalleVentas], optional),
	ivaventa (number, optional),
	total_venta (number, optional),
	valor_venta (number, optional)
	}
	DetalleVentas {
	cantidad_producto (integer, optional),
		{
		"cedula_cliente": 0,
		"codigo_venta": 0,
		"detalle_venta": [
		{
		"cantidad_producto": 0,
		"codigo_producto": 0,
		"valor_total": 0,
		"valor_venta": 0,
		"valoriva": 0
		}
		],
		"ivaventa": 0,
		"total_venta": 0,
		"valor_venta": 0
		}

Abra el archivo application.properties y agregue las siguientes propiedades de MongoDB :

```
# Propiedades MongoDB
spring.data.mongodb.uri=mongodb://localhost:27017/db_ventas00
server.port=8081

```

Según nuestra configuración, Mongo DB se ejecuta localmente en el puerto predeterminado 27017.

Tenga en cuenta que necesitamos crear la base de datos "db_clientes" usando el siguiente comando: **use db_ventas00**.

Si la base de datos " db_ventas00 " no está presente en MongoDB , creará una nueva.

5. Creación del modelo de Datos

Ahora creemos el modelo de Datos, debemos entender que la clase Ventas contiene un arreglo dentro llamado detalle_venta en donde se deben agregar los productos que son vendidos en la factura.

- **DetalleVenta.**

Dentro del paquete modelo se agrega un archivo DetalleVenta.java con el siguiente contenido:

```java
package com.mintic.ventas.modelo;

public class DetalleVenta {

    private int cantidad_producto;
    private int codigo_producto;
    private double valor_total;
    private double valor_venta;
    private double valoriva;

    public DetalleVenta() {
    }

    public DetalleVenta(int cantidad_producto, int codigo_producto,
            double valor_total, double valor_venta,
            double valoriva) {
        this.cantidad_producto = cantidad_producto;
        this.codigo_producto = codigo_producto;
        this.valor_total = valor_total;
        this.valor_venta = valor_venta;
        this.valoriva = valoriva;
    }

    public int getCantidad_producto() {
        return cantidad_producto;
    }

    public void setCantidad_producto(int cantidad_producto) {
        this.cantidad_producto = cantidad_producto;
    }

    public int getCodigo_producto() {
        return codigo_producto;
    }

    public void setCodigo_producto(int codigo_producto) {
        this.codigo_producto = codigo_producto;
    }

    public double getValor_total() {
        return valor_total;
    }

    public void setValor_total(double valor_total) {
        this.valor_total = valor_total;
    }

    public double getValor_venta() {
        return valor_venta;
    }

    public void setValor_venta(double valor_venta) {
        this.valor_venta = valor_venta;
    }

    public double getValoriva() {
        return valoriva;
    }

    public void setValoriva(double valoriva) {
        this.valoriva = valoriva;
    }
}
```

- **Ventas.**

Dentro del paquete modelo se agrega un archivo Ventas.java, agregué un atributo llamado sucursal que nos facilite la identificación de la ciudad a quien pertenece la venta (Bogotá, Medellín o Cali) con el siguiente contenido:

```java
package com.mintic.ventas.modelo;

import java.util.List;

import org.springframework.data.annotation.Id;
import org.springframework.data.mongodb.core.mapping.Document;

@Document(collection = "ventas")
public class Ventas {

    @Id
    private String _id;

    private String sucursal;
    private int cedula_cliente;
    private int codigo_venta;
    private List<DetalleVenta> detalle_venta;
    private double ivaventa;
    private double total_venta;
    private double valor_venta;

    public Ventas() {
    }

    public Ventas(String _id, String sucursal, int cedula_cliente,
            int codigo_venta, List<DetalleVenta> detalle_venta,
            double ivaventa, double total_venta, double valor_venta) {
        super();
        this._id = _id;
        this.sucursal = sucursal;
        this.cedula_cliente = cedula_cliente;
        this.codigo_venta = codigo_venta;
        this.detalle_venta = detalle_venta;
        this.ivaventa = ivaventa;
        this.total_venta = total_venta;
        this.valor_venta = valor_venta;
    }

    public String getSucursal() {
        return sucursal;
    }

    public void setSucursal(String sucursal) {
        this.sucursal = sucursal;
    }
```

Agregar set y get

6. Cree un repositorio de datos de Spring - VentasRepositorio.java.

A continuación, necesitamos crear VentasRepositorio para acceder a los

datos de la base de datos, en esta interface se agrega como extensión una clase "MongoRepository" que contiene los métodos básicos de gestión y ejecución de la Base de Datos.

```java
package com.mintic.ventas.dao;

import org.springframework.data.mongodb.repository.MongoRepository;
import org.springframework.stereotype.Repository;

import com.mintic.ventas.modelo.Ventas;

@Repository
public interface IVentasRepositorio extends MongoRepository<Ventas, String> {

}
```

7. Capa de servicio (usa repositorio).

La capa de servicio es opcional; aún se recomienda realizar lógica de negocio adicional, si corresponde. Generalmente, nos conectaremos con el repositorio aquí para operaciones CRUD. Vamos a crear la interface con las cabeceras de los métodos a implementar.

```java
package com.mintic.ventas.servicio;

import java.util.List;

import com.mintic.ventas.modelo.Ventas;

public interface VentasServicio {

    // Cabeceras de métodos a utilizar en la colección Ventas
    // Crear
    Ventas crearVentas(Ventas ventas);

    //Actualizar
    Ventas updateVentas(Ventas ventas);

    //Listado de Ventas
    List<Ventas> getAllVentas();

    //Buscar Ventas por _id
    Ventas getVentasById(String ventasId);

    //Eliminar un Ventas
    void deleteVentas(String ventasId);


}
```

Clase "VentasServicioImpl".

Ahora implementamos los métodos declarados en una clase llamada "VentasServicioImpl", los métodos se desarrollan instanciando un objeto de la interface de servicio VentasServicio.

```java
package com.mintic.ventas.servicio;

import java.util.List;
import java.util.Optional;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
import org.springframework.transaction.annotation.Transactional;

import com.mintic.ventas.dao.IVentasRepositorio;
import com.mintic.ventas.excepcion.ResourceNotFoundException;
import com.mintic.ventas.modelo.Ventas;
import com.mintic.ventas.dao.IVentasRepositorio;

@Service
@Transactional
public class VentasServicioImpl implements VentasServicio{

    @Autowired
    private IVentasRepositorio ventaRepo;

    @Override
    public Ventas crearVentas(Ventas ventas) {
        return ventaRepo.save(ventas);
    }

    @Override
    public Ventas updateVentas(Ventas ventas) {
        // Se busca el cliente con el Id del objeto cliente recibido
        Optional<Ventas> ventasDb = this.ventaRepo.findById(ventas.get_id());

        if(ventasDb.isPresent()) {
            // Se crea un objeto tipo Cliente con los datos recuperados
            Ventas ventasUpdate = ventasDb.get();
            // Se actualiza el valor de cada atributo con el get correspondiente
            // de cliente
            ventasUpdate.set_id(ventas.get_id());
            ventasUpdate.setCedula_cliente(ventas.getCedula_cliente());
            ventasUpdate.setCodigo_venta(ventas.getCodigo_venta());;
            ventasUpdate.setDetalle_venta(ventas.getDetalle_venta());
            ventasUpdate.setIvaventa(ventas.getIvaventa());
            ventasUpdate.setTotal_venta(ventas.getTotal_venta());
            ventasUpdate.setValor_venta(ventas.getValor_venta());
            // Se actualizan los valores del objeto cliente
            ventaRepo.save(ventasUpdate);
            return ventasUpdate;
        }else {
            throw new ResourceNotFoundException("Registro no Encontrado con el Id:"+ventas.get_id());
        }
    }

    @Override
    public List<Ventas> getAllVentas() {
        return ventaRepo.findAll();
    }

```

```
57      @Override
58      public Ventas getVentasById(String ventasId) {
59          // Búsqueda de cliente por _id
60          Optional<Ventas> ventasDb = this.ventaRepo.findById(ventasId);
61          if(ventasDb.isPresent()) {
62              return ventasDb.get();
63          }else {
64              throw new ResourceNotFoundException("Registro no Encontrado con el Id:"+ventasId);
65          }
66      }
67
68      @Override
69      public void deleteVentas(String ventasId) {
70          // Busca primero
71          Optional<Ventas> ventaDb = this.ventaRepo.findById(ventasId);
72          if(ventaDb.isPresent()) {
73              // Si lo encuentra lo borra
74              this.ventaRepo.delete(ventaDb.get());
75          }else {
76              throw new ResourceNotFoundException("Registro no Encontrado con el Id:"+ventasId);
77          }
78      }
79  }
```

- **Clase "ResourceNotFoundException".**

Para poder administrar las excepciones creamos una clase que permita emitir un mensaje en tiempo de ejecución, el contenido se puede apreciar en las siguientes líneas, se crea sobre un packages excepcion.

```
1   package com.mintic.clientes.excepcion;
2
3   import org.springframework.web.bind.annotation.ResponseStatus;
4
5   @ResponseStatus
6   public class ResourceNotFoundException extends RuntimeException{
7
8       private static final long serialVersionUID = 1L;
9
10      public ResourceNotFoundException(String message) {
11          super(message);
12      }
13
14      public ResourceNotFoundException(String message, Throwable throwable) {
15          super(message, throwable);
16      }
17
18  }
```

8. Creación de las API: VentasControlador.

En el controlador creamos las API que van a poder utilizar los clientes con los métodos requeridos usando la interface del servicio e implementando las soluciones que permitan construir el CRUD completo de clientes.

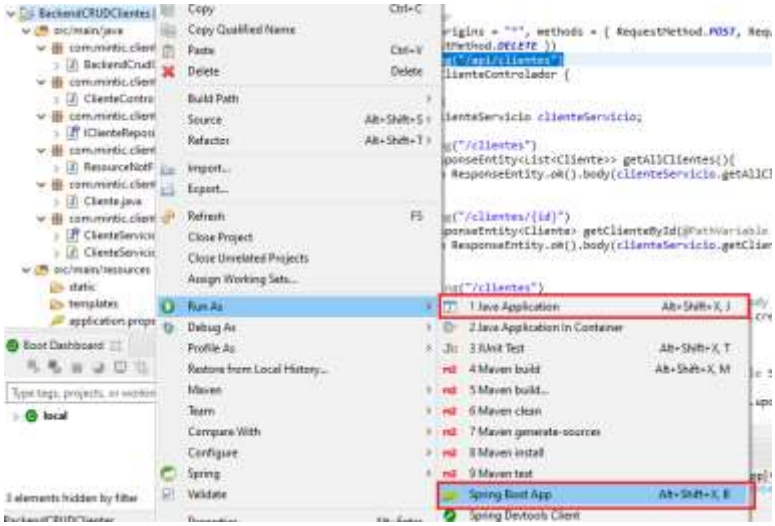
Creamos una clase llamada VentasControlador con las anotaciones de @RestController, @CrossOrigin y @RequestMapping("/api/ventas"), este es el puente para que el cliente llame a cada uno de los métodos.

```java
package com.mintic.ventas.controlador;

import java.util.List;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.CrossOrigin;
import org.springframework.web.bind.annotation.DeleteMapping;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.PutMapping;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;
import org.springframework.web.bind.annotation.RestController;

import com.mintic.ventas.modelo.Ventas;
import com.mintic.ventas.servicio.VentasServicio;


@RestController
@CrossOrigin(origins = "*", methods = { RequestMethod.POST, RequestMethod.GET, RequestMethod.PUT,
        RequestMethod.DELETE })
@RequestMapping("/api/ventas")
public class VentasControlador {

    @Autowired
    private VentasServicio ventasServicio;

    @GetMapping("/ventas")
    public ResponseEntity<List<Ventas>> getAllVentas(){
        return ResponseEntity.ok().body(ventasServicio.getAllVentas());
    }

    @GetMapping("/ventas/{id}")
    public ResponseEntity<Ventas> getVentasById(@PathVariable String id) {
        return ResponseEntity.ok().body(ventasServicio.getVentasById(id));
    }

    @PostMapping("/ventas")
    public ResponseEntity < Ventas > crearVenta(@RequestBody Ventas ventas){
        return ResponseEntity.ok().body(this.ventasServicio.crearVentas(ventas));
    }

    @PutMapping("/ventas/{id}")
    public ResponseEntity<Ventas> updateVenta(@PathVariable String id, @RequestBody Ventas ventas){
        ventas.set_id(id);
        return ResponseEntity.ok().body(this.ventasServicio.updateVentas(ventas));
    }

    @DeleteMapping("/ventas/{id}")
    public HttpStatus deleteVenta(@PathVariable String id) {
        this.ventasServicio.deleteVentas(id);
        return HttpStatus.OK;
    }

}
```

9. Ejecución de la aplicación de arranque Spring.

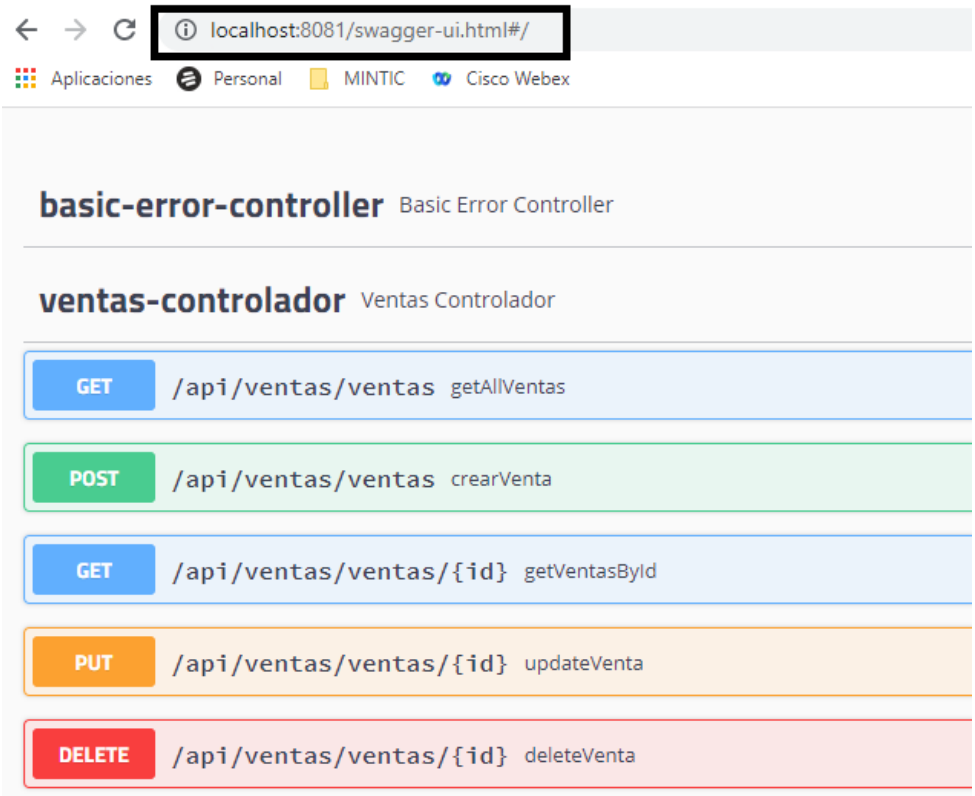
Cambiemos el puerto de ejecución a 8081 y después para ejecutar y probar el proyecto podemos acudir al archivo de inicio del mismo y

ejecutar como una aplicación java o en su defecto podemos "Run As" Spring Boot App como se puede observar en la figura:



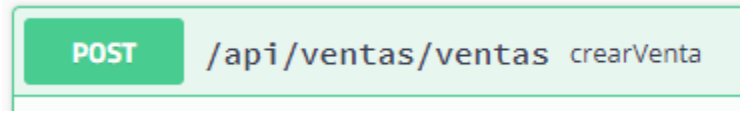
10. Pruebe las API REST con Swagger.

Para probar el proyecto nos vamos a un navegador y escribimos http://localhost:8081/swagger-ui.html#/ nos debe aparecer una figura como la siguiente:



Podemos probar cada uno de los métodos en forma directa en el navegador y reemplazamos con ello el uso de Postman.

- **Método POST: crearVenta.**

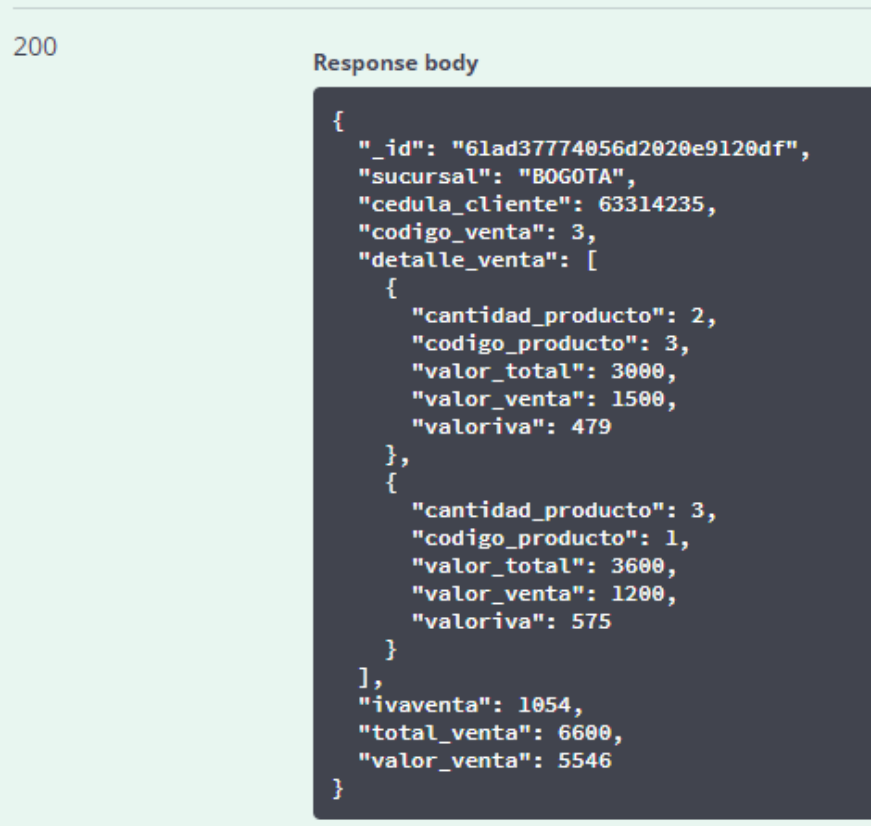


Vamos a iniciar con el método post para crear una venta, debemos seleccionar el botón de Try it Out, escribimos los datos del nuevo cliente sin escribir algún Id con el fin de que MongoDb lo genere:

```
{
  "cedula_cliente": 63314235,
  "codigo_venta": 3,
  "detalle_venta": [
    {
      "cantidad_producto": 2,
      "codigo_producto": 3,
      "valor_total": 3000,
```

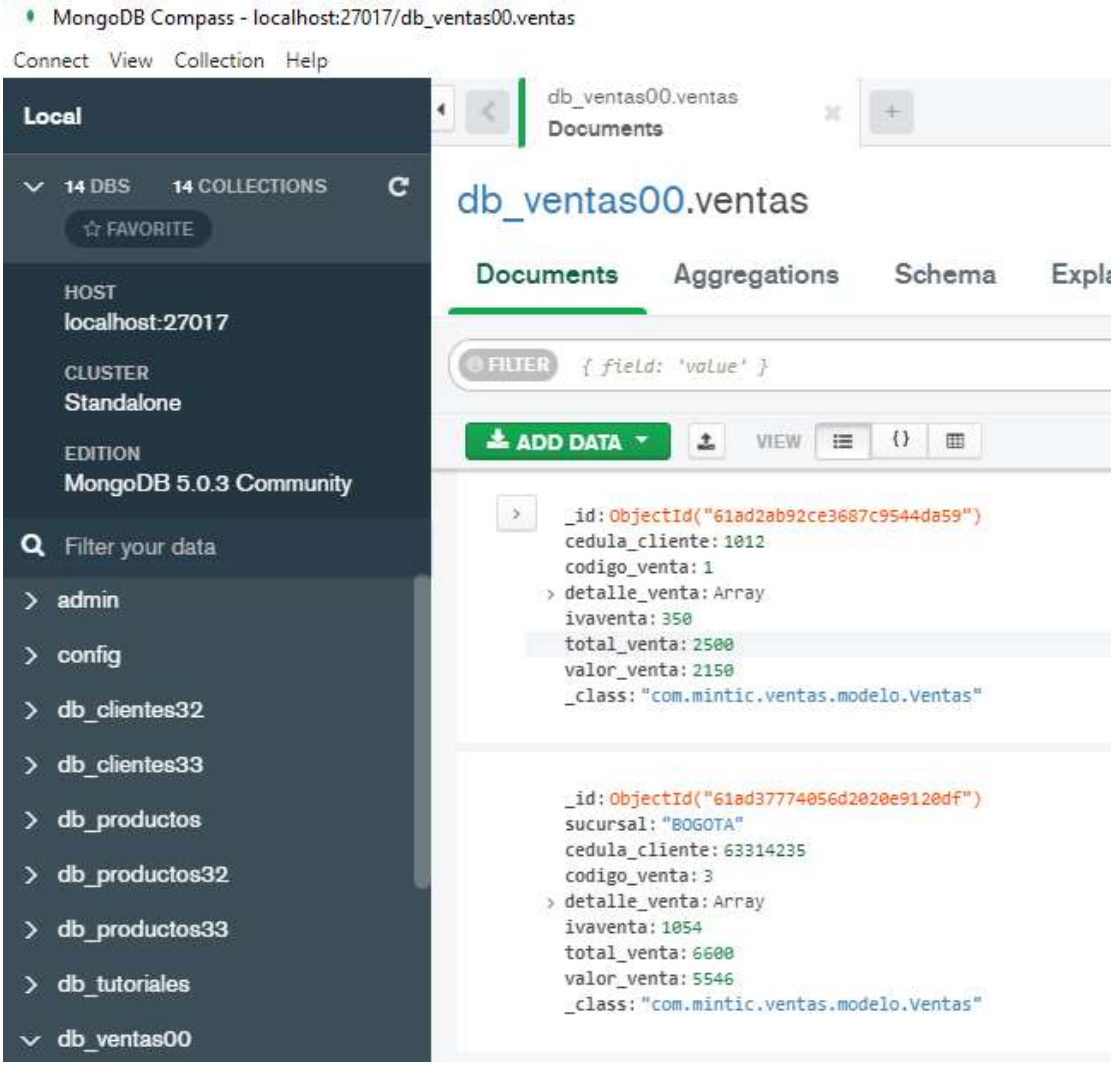
```
    "valor_venta": 1500,
    "valoriva": 479
   },
   {
    "cantidad_producto": 3,
    "codigo_producto": 1,
    "valor_total": 3600,
    "valor_venta": 1200,
    "valoriva": 575
   }
  ],
  "ivaventa": 1054,
  "sucursal": "BOGOTA",
  "total_venta": 6600,
  "valor_venta": 5546
}
```


Después presionamos el botón de ejecutar (Execute), debajo aparecerá la respuesta correspondiente.

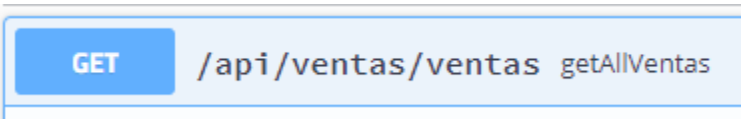


Podemos revisar sobre la Base de Datos de MongoDb en la consola o usando

la interfaz de Compass.



- **Método GET: getAllVentas.**



Al probar el método getAllVentas podemos visualizar el resultado con la lista de todos las ventas:

```
[
  {
    "_id": "61ad2ab92ce3687c9544da59",
    "sucursal": null,
    "cedula_cliente": 1012,
    "codigo_venta": 1,
    "detalle_venta": [
      {
        "cantidad_producto": 3,
        "codigo_producto": 1,
        "valor_total": 1500,
        "valor_venta": 500,
        "valoriva": 200
      },
      {
        "cantidad_producto": 2,
        "codigo_producto": 2,
        "valor_total": 1000,
        "valor_venta": 400,
        "valoriva": 150
      }
    ],
    "ivaventa": 350,
    "total_venta": 2500,
    "valor_venta": 2150
  },
  {
    "_id": "61ad37774056d2020e9120df",
    "sucursal": "BOGOTA",
    "cedula_cliente": 63314235,
    "codigo_venta": 3,
    "detalle_venta": [
      {
        "cantidad_producto": 2,
        "codigo_producto": 3,
        "valor_total": 3000,
        "valor_venta": 1500,
        "valoriva": 479
      },
      {
        "cantidad_producto": 3,
        "codigo_producto": 1,
        "valor_total": 3600,
        "valor_venta": 1200,
        "valoriva": 575
      }
    ],
    "ivaventa": 1054,
    "total_venta": 6600,
    "valor_venta": 5546
  }
]
```