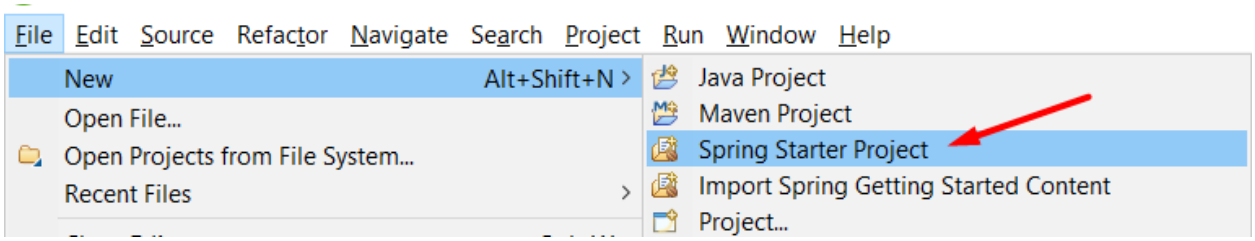
PROYECTO “gardening-store”

Paso 1: Crear el Proyecto en STS

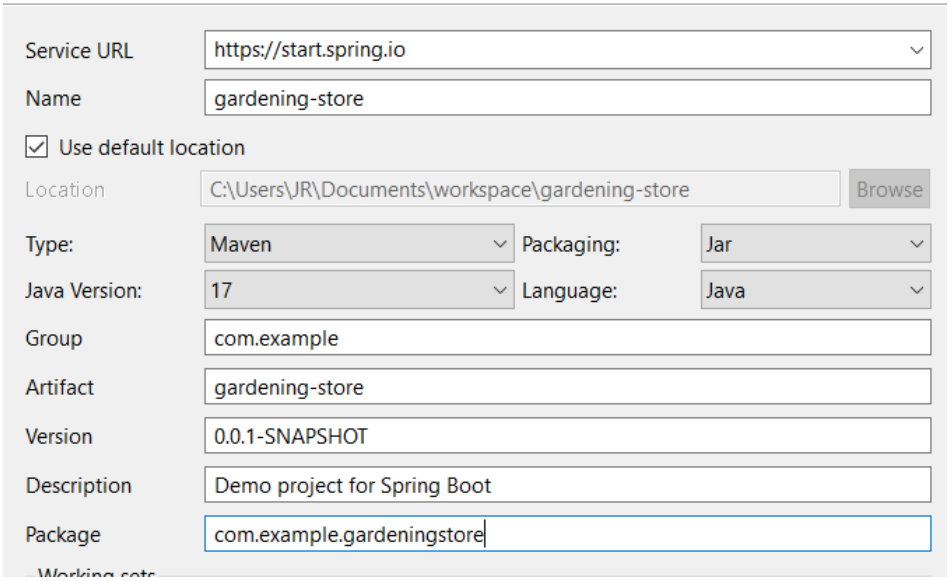
Abrir Spring Tool Suite (STS).

Ir a **File → New → Spring Starter Project**.

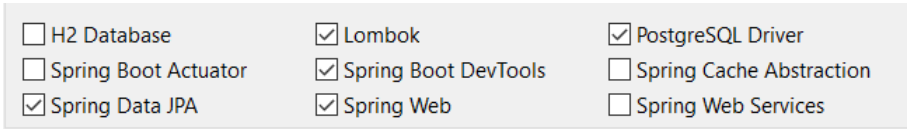


Completar los datos del proyecto:

- **Name:** gardening-store
- **Type:** Maven
- **Java Version:** 17
- **Group:** com.example
- **Artifact:** gardening-store
- **Packaging:** Jar



- **Dependencies:**
  - **Spring Web** (Para la API REST)
  - **Spring Boot DevTools** (Para recarga automática)
  - **Spring Data JPA** (Para manejar base de datos con Hibernate)
  - **PostgreSQL Driver** (Para conectarse a PostgreSQL)
  - **Lombok** (Para reducir código repetitivo)



Hacer clic en **Finish** para generar el proyecto.

Paso 2: Configurar application.properties

Abre el archivo **src/main/resources/application.properties** y agrega:

```
spring.application.name=gardening-store
```

Explicación:

- **spring.application.name**: Define el nombre de la aplicación Spring Boot. En este caso, la aplicación se llama gardening-store. Este nombre puede ser utilizado para identificar la aplicación en logs, métricas, etc.

```
spring.datasource.url=jdbc:postgresql://aws-0-us-west-1.pooler.supabase.com:6543/postgres
spring.datasource.username=postgres.wluwoborplxwkdfppdck
spring.datasource.password=admin
spring.datasource.driver-class-name=org.postgresql.Driver
```

Explicación:

- **spring.datasource.url**: Especifica la URL de conexión a la base de datos PostgreSQL.
- **spring.datasource.username**: Es el nombre de usuario para conectarse a la base de datos.
- **spring.datasource.password**: Es la contraseña para el usuario de la base de datos.
- **spring.datasource.driver-class-name**: Especifica el driver JDBC que se utilizará para conectarse a la base de datos.

```
spring.jpa.generate-ddl=true
spring.jpa.hibernate.ddl-auto=update
spring.jpa.show-sql=true
spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.PostgreSQLDialect
```

Explicación:

- **spring.jpa.generate-ddl**: Habilita la generación automática del esquema de la base de datos a partir de las entidades JPA.
- **spring.jpa.hibernate.ddl-auto**: Define el comportamiento de Hibernate con respecto a la generación del esquema de la base de datos. update significa que Hibernate actualizará el esquema de la base de datos si es necesario, pero no borrará datos existentes.
- **spring.jpa.show-sql**: Habilita la impresión de las sentencias SQL generadas por Hibernate en la consola.
- **spring.jpa.properties.hibernate.dialect**: Especifica el dialecto de SQL que Hibernate debe usar para generar las consultas SQL.

```
spring.datasource.hikari.maximum-pool-size=5
spring.datasource.hikari.minimum-idle=5
spring.datasource.hikari.idle-timeout=30000
spring.datasource.hikari.max-lifetime=60000
```

Explicación:

- **spring.datasource.hikari.maximum-pool-size**: Define el número máximo de conexiones en el pool de conexiones. Aquí, el máximo es 5.
- **spring.datasource.hikari.minimum-idle**: Define el número mínimo de conexiones inactivas que se mantendrán en el pool. En este caso, es 5.
- **spring.datasource.hikari.idle-timeout**: Especifica el tiempo máximo (en milisegundos) que una conexión puede estar inactiva antes de ser cerrada. Aquí, es 30000 ms (30 segundos).
- **spring.datasource.hikari.max-lifetime**: Define el tiempo máximo (en milisegundos) que una conexión puede estar en el pool antes de ser cerrada. En este caso, es 60000 ms (60 segundos).

```
spring.datasource.hikari.data-source-properties.cachePrepStmts=false
spring.datasource.hikari.data-source-properties.useServerPrepStmts: false
spring.datasource.hikari.data-source-properties.prepareThreshold: 0
```

Explicación:

- **cachePrepStmts**: Deshabilita el caché de sentencias preparadas (false).
- **useServerPrepStmts**: Deshabilita el uso de sentencias preparadas en el servidor (false).
- **prepareThreshold**: Establece el umbral para el uso de sentencias preparadas. Un valor de 0 significa que no se usarán sentencias preparadas.

Paso 3: Crear las Entidades JPA

Entidad Category (Categorías de productos):

```
package com.example.gardeningstore.model;

import jakarta.persistence.*;
import lombok.*;

@Entity
@Table(name = "categories")
@Getter @Setter
@NoArgsConstructor @AllArgsConstructor
public class Category {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @Column(nullable = false, unique = true)
    private String name;
}
```

Explicación:

- **@Entity:** Indica que esta clase es una entidad JPA y será mapeada a una tabla en la base de datos.
- **@Table(name = "categories"):** Especifica el nombre de la tabla en la base de datos que representará a esta entidad. Si no se define, JPA usaría el nombre de la clase (Category) por defecto.
- **@Id:** Marca el campo id como la clave primaria de la tabla.

- **@GeneratedValue(strategy = GenerationType.IDENTITY):** Indica que el valor del id se generará automáticamente usando una estrategia de autoincremento de la base de datos.
- **@Column(nullable = false, unique = true):** Define restricciones para la columna name
  - nullable = false: No permite valores null.
  - unique = true: No permite valores repetidos en la columna.

Entidad Customer (Clientes):

```java
package com.example.gardeningstore.model;

import jakarta.persistence.*;
import lombok.*;

@Entity
@Table(name = "customers")
@Getter @Setter
@NoArgsConstructor @AllArgsConstructor
public class Customer {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @Column(nullable = false)
    private String name;

    @Column(nullable = false, unique = true)
    private String email;
}
```

Entidad Product (Productos):

```java
package com.example.gardeningstore.model;

import jakarta.persistence.*;
import lombok.*;

@Entity
@Table(name = "products")
@Getter @Setter
@NoArgsConstructor @AllArgsConstructor
public class Product {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @Column(nullable = false)
    private String name;

    @Column(nullable = false)
    private Double price;

    @ManyToOne
    @JoinColumn(name = "category_id", nullable = false)
    private Category category;
}
```

Entidad Order (Pedidos, con PK compuesta):

```java
package com.example.gardeningstore.model;

import jakarta.persistence.*;
import lombok.*;

import java.util.Date;
import java.util.List;

@Entity
@Table(name = "orders")
@Getter @Setter
@NoArgsConstructor
@AllArgsConstructor
public class Order {
    @EmbeddedId
    private OrderPK id;

    @Column(nullable = false)
    private Date orderDate;

    @ManyToOne
    @MapsId("customerId")
    @JoinColumn(name = "customer_id", nullable = false)
    private Customer customer;


    @ManyToMany
    @JoinTable(
        name = "order_products",
        joinColumns = {
            @JoinColumn(name = "order_id", referencedColumnName = "id"),
            @JoinColumn(name = "customer_id", referencedColumnName = "customer_id")
        },
        inverseJoinColumns = @JoinColumn(name = "product_id")
    )
    private List<Product> products;
}
```

Explicación:

- **@EmbeddedId:** Indica que id es una clave primaria compuesta (definida en otra clase llamada OrderPK). OrderPK debe estar anotada con @Embeddable y contener los atributos que forman la clave primaria.
- **@MapsId("customerId"):** Como la clave primaria (OrderPK) tiene un campo customerId, esta anotación indica que el campo customer en Order debe **mapearse** a customerId en OrderPK. Se usa en combinación con @EmbeddedId.

- **@ManyToMany:** Indica que **una orden puede tener muchos productos y un producto puede estar en muchas órdenes**.
- **@JoinTable(...):** Define la **tabla intermedia** que gestionará la relación Order ↔ Product. La tabla intermedia se llamará **order_products**.
- **joinColumns = {...}:** Especifica las columnas en order_products que referencian a la tabla orders. Como Order tiene una **clave compuesta**, se incluyen:
  - order_id (mapea id de Order)
  - customer_id (mapea customer_id de OrderPK)
- **inverseJoinColumns = @JoinColumn(name = "product_id"):** Especifica que product_id en order_products apunta a la clave primaria de la tabla products.

Y la **clave compuesta** debe definirse correctamente en OrderPK:

```
package com.example.gardeningstore.model;

import java.io.Serializable;

import jakarta.persistence.Embeddable;
import lombok.AllArgsConstructor;
import lombok.Getter;
import lombok.NoArgsConstructor;
import lombok.Setter;

@Embeddable
@Getter @Setter
@NoArgsConstructor @AllArgsConstructor
public class OrderPK implements Serializable {
    private static final long serialVersionUID = 1L;


   private Long id;
   private Long customerId;
}
```

Explicación:

- **@Embeddable:** Indica que esta clase **no es una entidad por sí misma**, sino que se usará como una clave primaria compuesta dentro de otra entidad (Order). Se usará en la anotación @EmbeddedId dentro de Order.
- **implements Serializable:** JPA requiere que las claves primarias compuestas sean **serializables**. serialVersionUID se usa para asegurar la compatibilidad en la serialización.
- **private Long id:** Representa el identificador del pedido en la tabla orders.
- **private Long customerId:** Representa el identificador del cliente asociado al pedido.

Paso 4: Crear los Repositorios JPA

Crea los siguientes **repositorios** en el paquete repository:

CategoryRepository:

```java
package com.example.gardeningstore.repository;

import com.example.gardeningstore.model.Category;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.stereotype.Repository;

@Repository
public interface CategoryRepository extends JpaRepository<Category, Long> {
}
```

ProductRepository:

```java
package com.example.gardeningstore.repository;

import com.example.gardeningstore.model.Product;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.stereotype.Repository;

import java.util.List;

@Repository
public interface ProductRepository extends JpaRepository<Product, Long> {
    List<Product> findByCategoryId(Long categoryId);
}
```

CustomerRepository:

```java
package com.example.gardeningstore.repository;

import com.example.gardeningstore.model.Customer;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.stereotype.Repository;

@Repository
public interface CustomerRepository extends JpaRepository<Customer, Long> {
}
```

OrderRepository:

```java
package com.example.gardeningstore.repository;

import com.example.gardeningstore.model.Order;
import com.example.gardeningstore.model.OrderPK;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.data.jpa.repository.Query;
import org.springframework.data.repository.query.Param;
import org.springframework.stereotype.Repository;

import java.util.List;

@Repository
public interface OrderRepository extends JpaRepository<Order, OrderPK> {


    // Obtener todas las órdenes de un cliente específico
    List<Order> findByCustomerId(Long customerId);

    // Obtener el total gastado por un cliente en pedidos
    @Query("SELECT SUM(p.price) FROM Order o JOIN o.products p WHERE o.customer.id = :customerId")
    Double getTotalSpentByCustomer(@Param("customerId") Long customerId);
}
```

Explicación:

- **Usa @Query** para definir una consulta en JPQL (Java Persistence Query Language)**.**
- **Consulta explicada:**
  - JOIN o.products p → Une Order con su lista de products.
  - WHERE o.customer.id = :customerId → Filtra por el cliente especificado.
  - SUM(p.price) → Suma el precio de todos los productos en las órdenes del cliente.
- **Resultado:** Devuelve el total gastado por un cliente en todas sus órdenes.

Paso 5: Crear Controladores REST

Crea los **controladores** en el paquete controller:

CategoryController:

```java
package com.example.gardeningstore.controller;

import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.*;

import com.example.gardeningstore.model.Category;
import com.example.gardeningstore.repository.CategoryRepository;

import java.util.List;

@RestController
@RequestMapping("/categories")
public class CategoryController {
    private final CategoryRepository categoryRepository;

    public CategoryController(CategoryRepository categoryRepository) {
        this.categoryRepository = categoryRepository;
    }

    @GetMapping
    public List<Category> getAll() {
        return categoryRepository.findAll();
    }

    @PostMapping
    public Category create(@RequestBody Category category) {
        return categoryRepository.save(category);
    }

    @DeleteMapping("/{id}")
    public ResponseEntity<?> delete(@PathVariable Long id) {
        categoryRepository.deleteById(id);
        return ResponseEntity.ok().build();
    }
}
```

CustomerController:

```java
package com.example.gardeningstore.controller;

import com.example.gardeningstore.model.Customer;
import com.example.gardeningstore.repository.CustomerRepository;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.*;

import java.util.List;

@RestController
@RequestMapping("/customers")
public class CustomerController {
    private final CustomerRepository customerRepository;

    public CustomerController(CustomerRepository customerRepository) {
        this.customerRepository = customerRepository;
    }

    @GetMapping
    public List<Customer> getAllCustomers() {
        return customerRepository.findAll();
    }

    @PostMapping
    public ResponseEntity<Customer> createCustomer(@RequestBody Customer customer) {
        if (customer.getName() == null || customer.getEmail() == null) {
            return ResponseEntity.badRequest().body(null);
        }
        return ResponseEntity.ok(customerRepository.save(customer));
    }
}
```

OrderController:

```java
package com.example.gardeningstore.controller;

import com.example.gardeningstore.model.Customer;
import com.example.gardeningstore.model.Order;
import com.example.gardeningstore.model.OrderPK;
import com.example.gardeningstore.model.Product;
import com.example.gardeningstore.repository.CustomerRepository;
import com.example.gardeningstore.repository.OrderRepository;
import com.example.gardeningstore.repository.ProductRepository;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.*;

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

@RestController
@RequestMapping("/orders")
public class OrderController {
    private final OrderRepository orderRepository;
    private final CustomerRepository customerRepository;
    private final ProductRepository productRepository;

    public OrderController(OrderRepository orderRepository, CustomerRepository customerRepository, ProductRepository productRepository) {
        this.orderRepository = orderRepository;
        this.customerRepository = customerRepository;
        this.productRepository = productRepository;
    }

    @GetMapping
    public List<Order> getAllOrders() {
        return orderRepository.findAll();
    }

    @GetMapping("/customer/{customerId}")
    public List<Order> getOrdersByCustomer(@PathVariable Long customerId) {
        return orderRepository.findByCustomerId(customerId);
    }

    @PostMapping
    public ResponseEntity<?> createOrder(@RequestBody Order orderRequest) {

        Optional<Customer> customerOpt =
```

```
        customerRepository.findById(orderRequest.getCustomer().getId());
        if (customerOpt.isEmpty()) {
            return ResponseEntity.badRequest().body("Customer does not exist.");
        }

        List<Product> productList = productRepository.findAllById(
            orderRequest.getProducts().stream().map(Product::getId).toList()
        );

        if (productList.size() != orderRequest.getProducts().size()) {
            return ResponseEntity.badRequest().body("One or more products do not exist.");
        }

        OrderPK orderPK = new OrderPK();
        orderPK.setId(orderRequest.getId().getId());
        orderPK.setCustomerId(orderRequest.getCustomer().getId());

        Order newOrder = new Order();
        newOrder.setId(orderPK);
        newOrder.setOrderDate(new Date());
        newOrder.setCustomer(customerOpt.get());
        newOrder.setProducts(productList);

        Order savedOrder = orderRepository.save(newOrder);
        return ResponseEntity.ok(savedOrder);
    }
}
```

ProductController:

```java
package com.example.gardeningstore.controller;

import com.example.gardeningstore.model.Category;
import com.example.gardeningstore.model.Product;
import com.example.gardeningstore.repository.CategoryRepository;
import com.example.gardeningstore.repository.ProductRepository;

import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.*;

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

@RestController
@RequestMapping("/products")
public class ProductController {
    private final ProductRepository productRepository;
    private final CategoryRepository categoryRepository;

    public ProductController(ProductRepository productRepository, CategoryRepository categoryRepository) {
        this.productRepository = productRepository;
        this.categoryRepository = categoryRepository;
    }

    @GetMapping
    public List<Product> getAllProducts() {
        return productRepository.findAll();
    }


    @GetMapping("/category/{categoryId}")
    public ResponseEntity<List<Product>> getProductsByCategory(@PathVariable Long categoryId) {
        List<Product> products = productRepository.findByCategoryId(categoryId);
        if (products.isEmpty()) {
            return ResponseEntity.notFound().build();
        }
        return ResponseEntity.ok(products);
    }

    @PostMapping
    public ResponseEntity<?> createProduct(@RequestBody Product product) {
        if (product.getCategory() == null || product.getCategory().getId() == null) {
            return ResponseEntity.badRequest().body("Category ID is required.");
```

```
        }

        // Buscar la categoría en la base de datos
        Optional<Category> category = categoryRepository.findById(product.getCategory().getId());
        if (category.isEmpty()) {
            return ResponseEntity.badRequest().body("Category does not exist.");
        }

        product.setCategory(category.get());

        Product savedProduct = productRepository.save(product);
        return ResponseEntity.ok(savedProduct);
    }
}
```

Paso 6: Ejecutar y Probar

Probar los endpoints en Postman: