Objetivos Principal
Desarrollar un sistema de alistamiento de tareas mediante Spring Boot y MySQL, y desplegarlo en una interfaz React.

Objetivos Específicos
- Desarrollar endpoints.
- Utilizar ORM para facilitar consultas a MySQL.
- Permitir la comunicación Frontend y respuestas del cliente.

Crearemos nuestro entorno Spring mediante Spring initializr

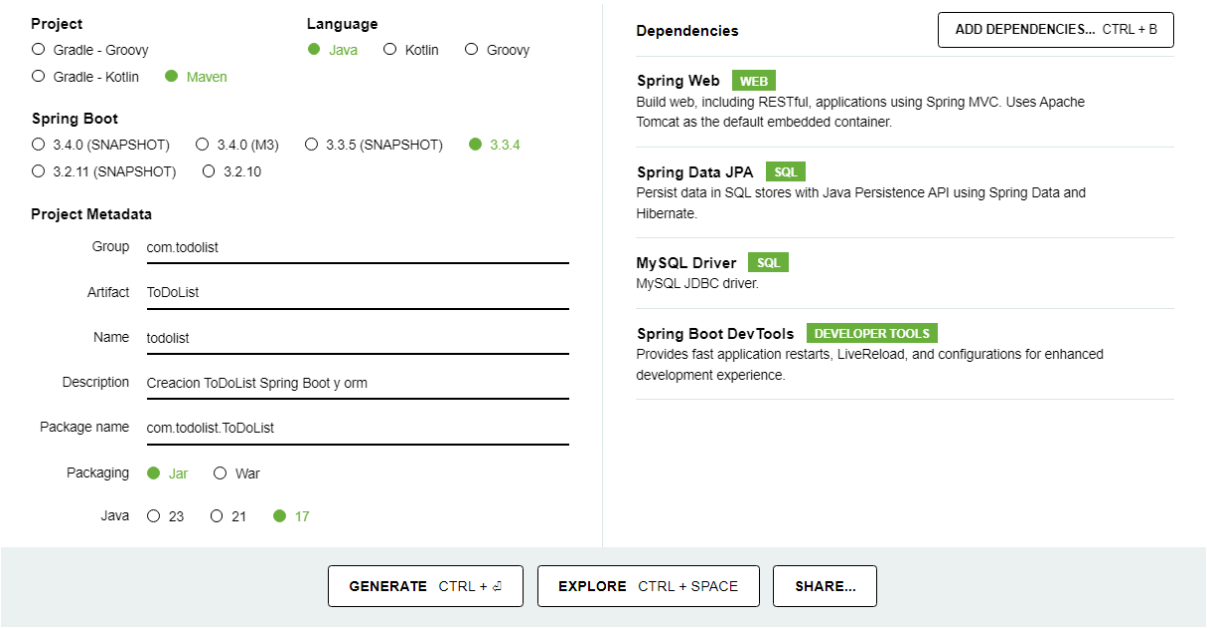

Figura 1: Inicialización de proyecto Spring Boot.

Para nuestro proyecto, ocupamos las dependencias de:
- **Spring web:** Nos ayudará a correr nuestro entorno localmente mediante Apache y Tomcat, además de nuestros servicios Restful y aplicación MVC.
- **Spring Data JPA:** Para nuestro propósito de ocupar ORM, ocupamos esta dependencia para que maneje las solicitudes Queries automáticamente y sin necesidad de tener que ingresarlas nosotros, ya que lo hace internamente.
- **MySQL Driver:** Esta dependencia nos ayudará a comunicarnos con base de datos, en este caso MySQL.
- **Spring Boot Dev Tools:** Nos facilitará las herramientas y las configuraciones en spring.

Otros datos:
- Ocupamos un proyecto de tipo maven.
- Nuestro programa funcionará en Java 17.

Configuramos nuestra conexión a MySQL, dependiendo de tu usuario, cambiamos los valores de username y password.

```
spring.application.name=todolist

#Configuración MySQL
spring.datasource.url=jdbc:mysql://localhost:3306/todolist_db
spring.datasource.username=root
spring.datasource.password=''
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver

#Configuración JPA
spring.jpa.hibernate.ddl-auto=update
spring.jpa.show-sql=true
spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.MySQLDialect
```

Figura 2: Comunicación a la base de datos MySQL

Usaremos el entorno MySQL Workbench, donde ingresamos la siguiente línea para crear la base de datos.

```
create database todolist_db
```

Unos detalles a considerar de nuestro proyecto To Do List, consiste en el estructuramiento de nuestras carpetas.

Al usar Spring Boot, nosotros trabajamos dividiendo el proyecto en 4 partes o componentes que separa el código y se comunican para dar funcionalidad, claro, al separar por clases e interfaces podemos tratarlas mejor para el propósito que cumplan cada uno.

- **Model:** En esta carpeta manejamos más el diseño y estructura de los datos que enviaremos a la base de datos MySQL En este caso, creamos nuestros Getters y Setters para trabajarlos..

Nota: En el futuro, si desean ahorrarse la creación de Getters y Setters pueden usar el decorador @Data, el cual, trabajará internamente los métodos Getters y Setters.

```java
package com.todolist.ToDoList.model;

import jakarta.persistence.*;

@Entity  17 usages
@Table(name = "tareas")
public class Tarea {
    @Id  2 usages
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    private String titulo;  2 usages
    private String descripcion;  2 usages
    private boolean completada;  2 usages
```

Figura 3: Estructura Tareas.java

- **Repository:** Esta ruta permitirá controlar mediante una interfaz la comunicación entre el servicio (parte lógica) y model (modelo de datos), este componente utiliza una interfaz que trabajara con JPA para traducir las peticiones por ORM del servicio a nuestra base de datos en un formato adecuado, así evitamos ingresar dentro de esta interfaz los queries de la base de datos y obtener respuestas al servicio.

```java
1  package com.todolist.ToDoList.repository;
2
3  import org.springframework.data.jpa.repository.JpaRepository;
4  import com.todolist.ToDoList.model.Tarea;
5  public interface TareaRepository extends JpaRepository<Tarea,Long> {  2 usages
6
7  }
8
```

Figura 4: TareaRepository.java

- **Service:** Esta carpeta permitirá manejar la lógica del negocio y cómo tratará las peticiones que en esta ingresen del controlador. En este caso ocupamos para el registro, el despliegue de las tareas y su eliminación.

```java
public class TareaService {

    @Autowired  4 usages
    private TareaRepository tareaRepository;

    public List<Tarea> obtenerTodas(){  1 usage
        return tareaRepository.findAll();
    }

    public Optional<Tarea> obtenerPorId(Long id){  3 usages
        return tareaRepository.findById(id);
    }

    public Tarea guardarTarea(Tarea tarea){  2 usages
        return tareaRepository.save(tarea);
    }

    public void  eliminarTarea(Long id){  1 usage
        tareaRepository.deleteById(id);
    }
}
```

Figura 5: TareaService.java

Controller: Esta última carpeta, maneja los request y response del Frontend y Backend. Además de las rutas URI a acceder.

```java
@CrossOrigin(origins = "http://localhost:3000")  no usages
@RestController
@RequestMapping("/api/tareas")
public class TareaController {
    @Autowired  7 usages
    private TareaService tareaService;

    @GetMapping  no usages
    public List<Tarea> listarTareas(){
        return tareaService.obtenerTodas();
    }

    @GetMapping("/{id}")  no usages
    public ResponseEntity<Tarea> obtenerTarea(@PathVariable Long id) {
        Optional<Tarea> tarea = tareaService.obtenerPorId(id);
        return tarea.map(ResponseEntity::ok).orElse(ResponseEntity.notFound().build());
    }
```

Figura 6: TareaController.java

Pruebas Postman

Primero, para hacer nuestras peticiones, será necesario especificar que las solicitudes se trate con formato tipo JSON.

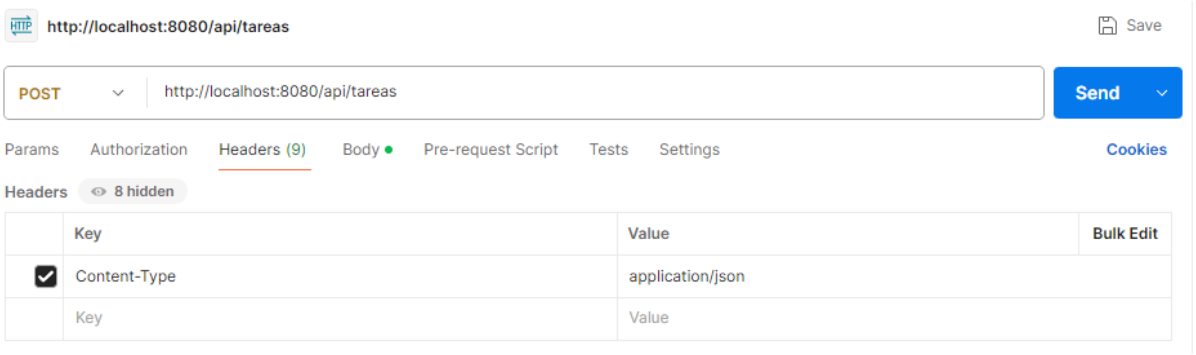

Figura 7: Configuración Headers tipo JSON.

Crear Tareas

Ingresamos a la siguiente estructura, para crear información a la base de datos, mediante el método POST..

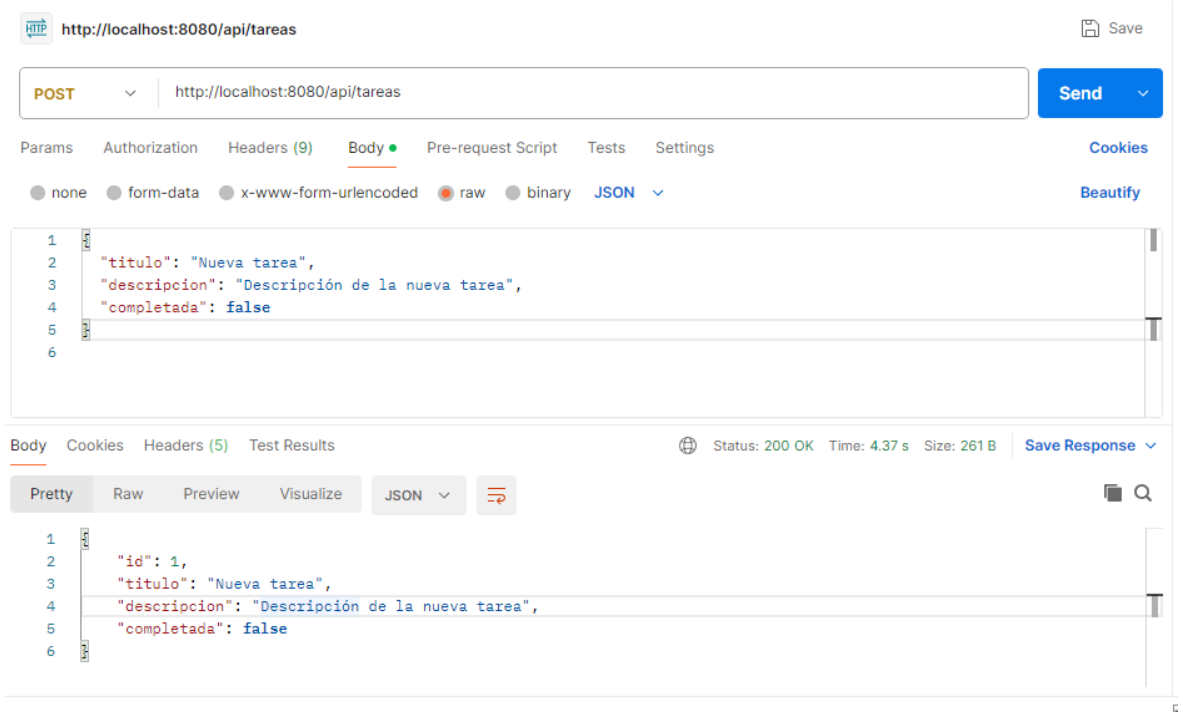

Figura 8: Creación de Tareas.

Obtener Tareas

Utilizamos un método GET para obtener todas las tareas registradas.

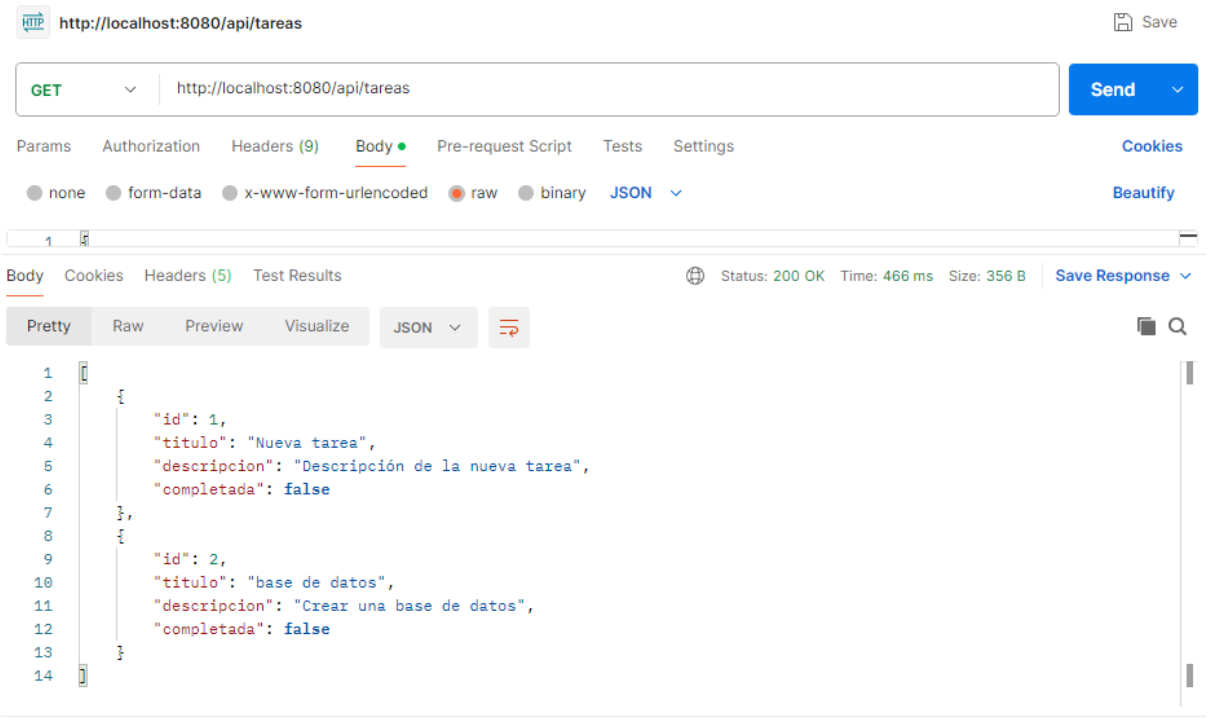

Figura 9: Obtener tareas.

Obtener tarea por id

Ocuparemos el método GET, además de obtener una tarea según su id.

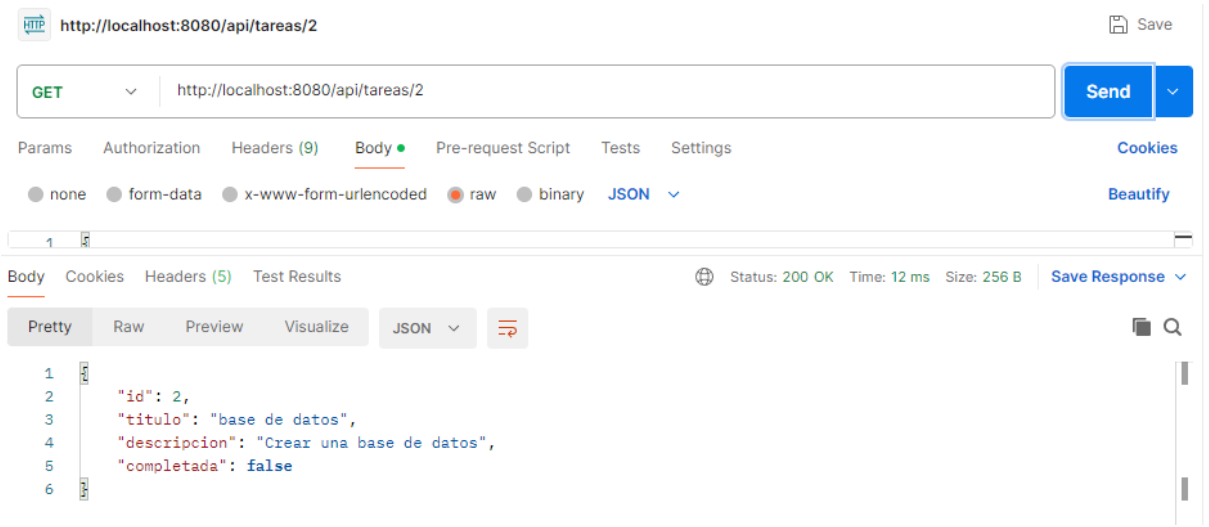

Figura 10: Obtención de una tarea específica

Actualizar Tareas.

.Ocuparemos un método PUT para actualizar nuestras listas en base al id.

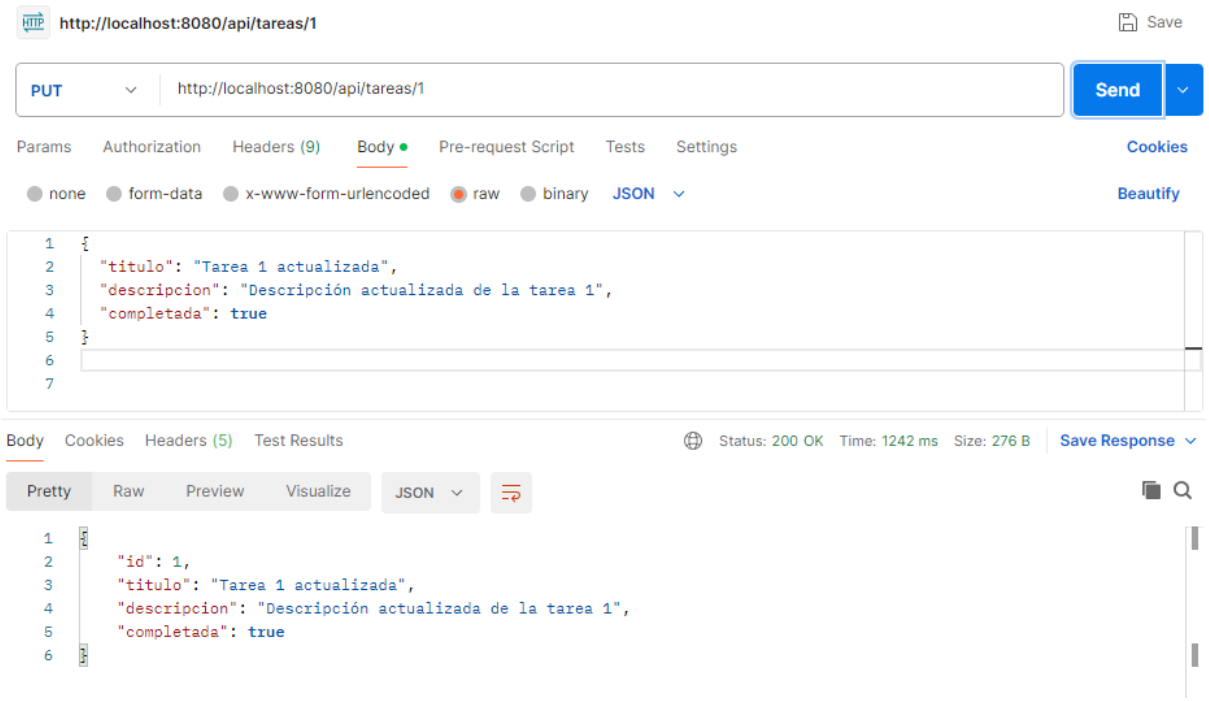

Figura 11: Actualizar Tareas.

Eliminación de Tareas

Podemos eliminar tareas en base a su ID, mediante el método DELETE.

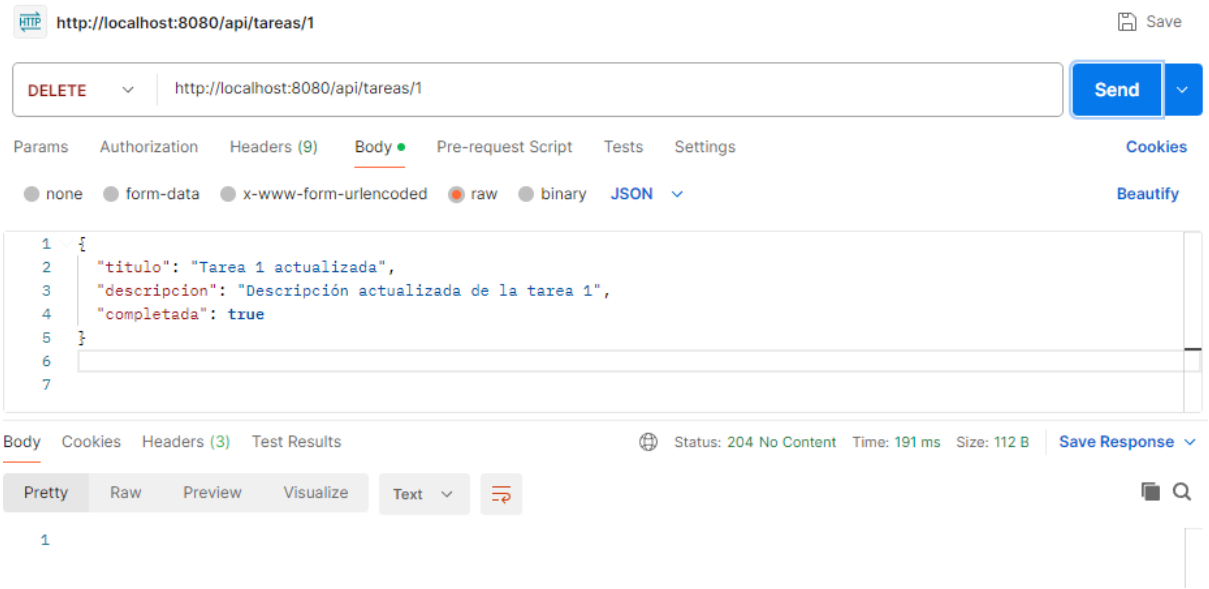

Figura 12: Eliminar Tareas.

Para verificar que se eliminó una tarea podremos desplegar las tareas.

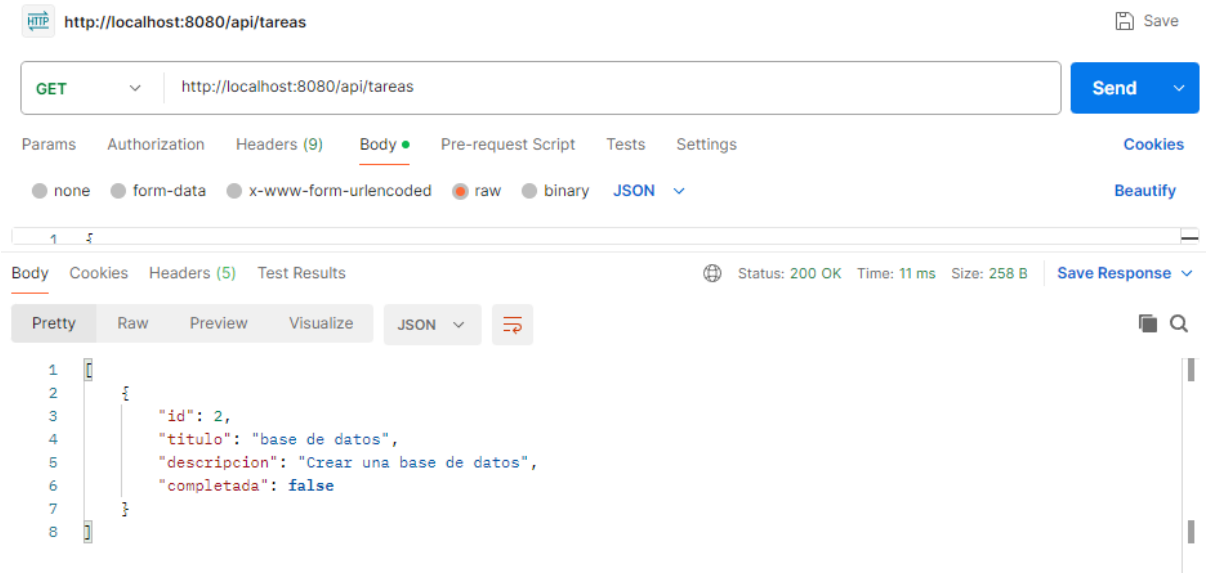

Figura 13: Verificar eliminación de tareas.

Una vez verificado nuestro código backend y sus peticiones, ocuparemos un despliegue por interfaz por el cliente.

Frontend

Usaremos React para desplegar la interfaz de nuestro proyecto. Por tanto, crearemos nuestro proyecto.

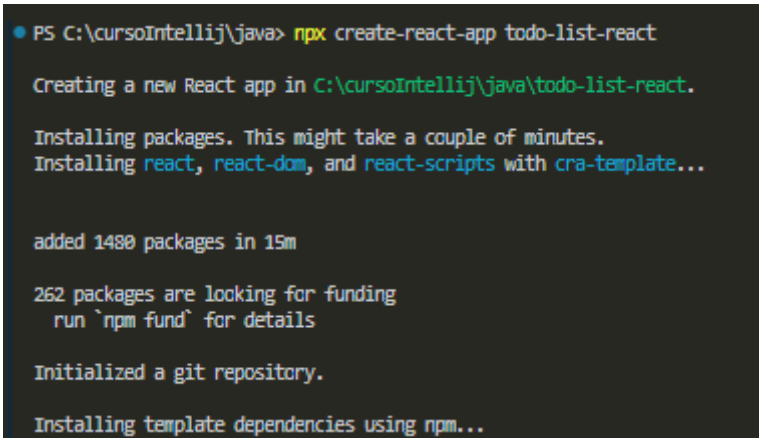

Figura 14: Creación de proyecto.

Para la comunicación, ocuparemos axios para las peticiones http.

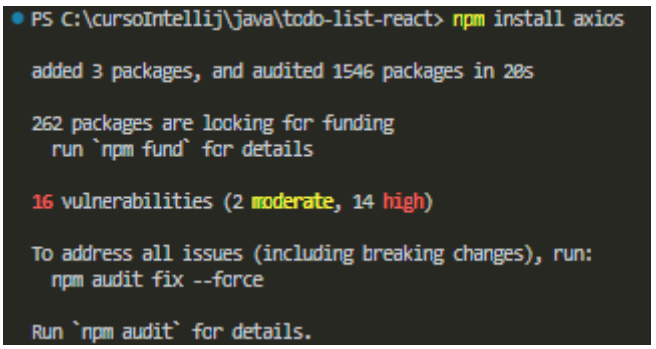

Figura 15: Instalación axios.