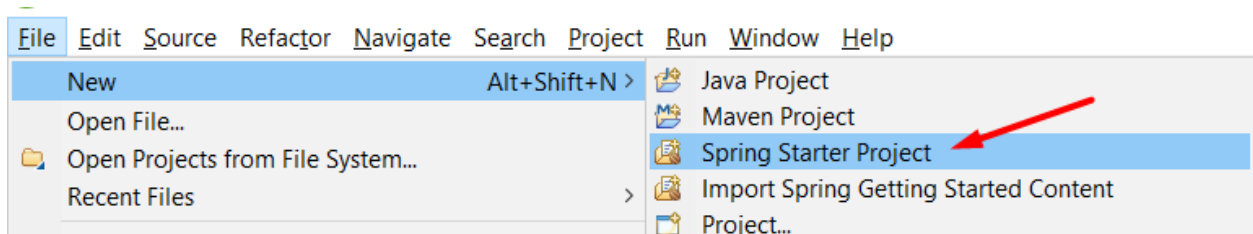


PROYECTO "SHA-256"

Paso 1: Crear el Proyecto en STS

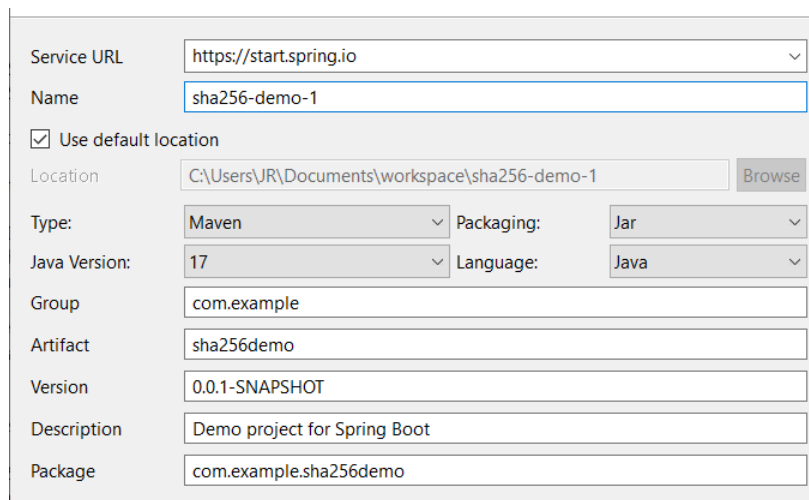
Abrir Spring Tool Suite (STS).

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



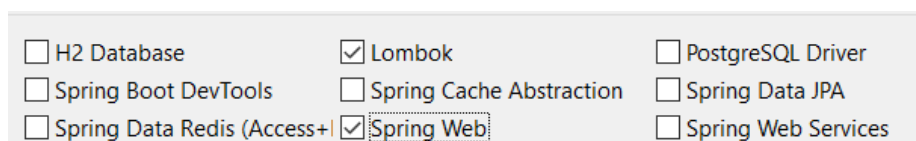
Completar los datos del proyecto:

- **Name:** sha256-demo
- **Type:** Maven
- **Java Version:** 17
- **Group:** com.example
- **Artifact:** sha256demo
- **Package:** com.example.sha256demo
- **Packaging:** Jar



Seleccionar las dependencias necesarias:

- **Spring Web**
- **Lombok** (Para reducir código innecesario).



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

Paso 2: Crear el SHA-256

Creamos la clase de utilidad para SHA-256



```
package com.example.sha256demo.util;

import java.security.MessageDigest;
import java.security.NoSuchAlgorithmException;

public class HashUtil {

    public static String aplicarSHA256(String input) {
        try {
            MessageDigest digest = MessageDigest.getInstance("SHA-256");
            byte[] hashBytes = digest.digest(input.getBytes());
            StringBuilder sb = new StringBuilder();

            for (byte b : hashBytes) {
                sb.append(String.format("%02x", b)); // convierte a hexadecimal
            }

            return sb.toString();
        } catch (NoSuchAlgorithmException e) {
            throw new RuntimeException("Algoritmo SHA-256 no disponible", e);
        }
    }
}
```

Explicación:

- Usamos la **clase MessageDigest** de Java para obtener una instancia del algoritmo SHA-256. Es como decirle al sistema: "quiero aplicar SHA-256 sobre un texto".
- **input.getBytes()** convierte el texto (por ejemplo "clave123") en un arreglo de bytes.
- **digest.digest(...)** aplica el algoritmo SHA-256 sobre esos bytes.
- El resultado es otro arreglo de bytes que representa el **hash binario** (no legible).
- Creamos un objeto StringBuilder para **ir armando la versión en texto del hash**.
- Recorremos cada byte del hash binario.
 - `String.format("%02x", b)` convierte cada byte en un **número hexadecimal de dos dígitos**.
 - `%02x` → rellena con ceros a la izquierda si es necesario.
 - Ejemplo: el byte 15 se convierte en "0f".
 - El `append()` va juntando todos los hexadecimales en un solo string.

Paso 3: Crear el DTO de entrada

Creamos la clase HashRequest



```
package com.example.sha256demo.dto;

public class HashRequest {
    private String texto;

    public String getTexto() {
        return texto;
    }

    public void setTexto(String texto) {
        this.texto = texto;
    }
}
```

Explicación:

- Contiene un campo texto y los métodos **getTexto** y **setTexto** para acceder y modificar este campo.
- Este DTO se utiliza en el controlador para recibir la cadena de texto que se va a hashear.

Paso 4: Crear el controlador REST

Creamos la clase controller HashController



```
package com.example.sha256demo.controller;

import com.example.sha256demo.dto.HashRequest;
import com.example.sha256demo.util.HashUtil;
import org.springframework.web.bind.annotation.*;

@RestController
@RequestMapping("/api/hash")
public class HashController {

    @PostMapping("/sha256")
    public String obtenerHashSHA256(@RequestBody HashRequest request) {
        return HashUtil.aplicarSHA256(request.getTexto());
    }
}
```

Explicación:

- Está anotado con **@RestController**, lo que indica que es un controlador que maneja solicitudes REST.
- La anotación **@RequestMapping("/api/hash")** define la ruta base para todas las solicitudes manejadas por este controlador.
- El método obtenerHashSHA256 está anotado con **@PostMapping("/sha256")**, lo que indica que maneja solicitudes POST en la ruta /api/hash/sha256.

- Este método toma un objeto HashRequest en el cuerpo de la solicitud (anotado con **@RequestBody**), extrae la cadena de texto y utiliza la clase HashUtil para calcular y devolver el hash SHA-256.

Paso 5: Ejecutar los endpoints en Postman

POST http://localhost:8080/api/hash/sha256

