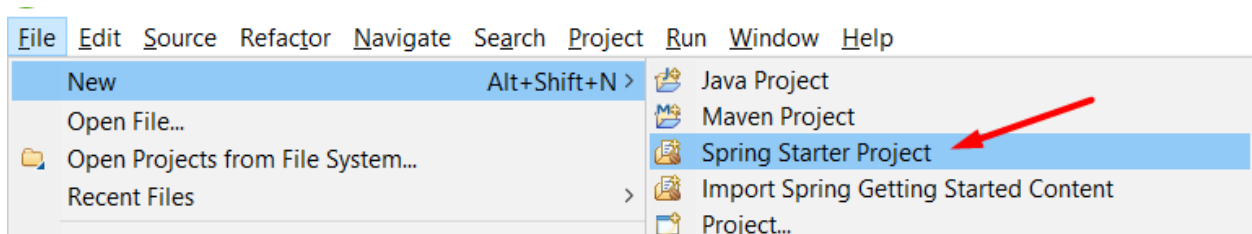


PROYECTO “Bcrypt”

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:** bcrypt-demo
- **Type:** Maven
- **Java Version:** 17
- **Group:** com.example
- **Artifact:** bcryptdemo
- **Package:** com.example.bcryptdemo
- **Packaging:** Jar

A screenshot of the 'New Spring Starter Project' dialog box in STS. The 'Service URL' is set to 'https://start.spring.io'. The 'Name' is 'bcrypt-demo-1'. The 'Use default location' checkbox is checked. The 'Location' is 'C:\Users\JR\Documents\workspace\bcrypt-demo-1'. The 'Type' is 'Maven', 'Packaging' is 'Jar', 'Java Version' is '17', and 'Language' is 'Java'. The 'Group' is 'com.example', 'Artifact' is 'bcryptdemo', 'Version' is '0.0.1-SNAPSHOT', 'Description' is 'Demo project for Spring Boot', and 'Package' is 'com.example.bcryptdemo'.

Seleccionar las dependencias necesarias:

- **Spring Web**
- **Spring Boot DevTools**
- **Spring Security**
- **Lombok**

A screenshot of the dependency selection checkboxes in the STS dialog. The checkboxes are arranged in a grid. The checked dependencies are: 'Spring Boot DevTools', 'Spring Security', 'Lombok', and 'Spring Web'. The unchecked dependencies are: 'H2 Database', 'Spring Cache Abstraction', 'PostgreSQL Driver', 'Spring Data JPA', and 'Spring Web Services'.

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

Paso 2: Crear un DTO para recibir contraseñas

Creamos la clase dto de PasswordRequest



```
package com.example.bcryptdemo.dto;

public class PasswordRequest {

    private String password;

    private String hash; // solo se usará en verificación

    public String getPassword() {

        return password;

    }

    public void setPassword(String password) {

        this.password = password;

    }

    public String getHash() {

        return hash;

    }

    public void setHash(String hash) {

        this.hash = hash;

    }

}
```

Explicación:

- **Atributos:**

- **private String password;** Almacena la contraseña que se va a cifrar o verificar.
- **private String hash;** Almacena el hash de la contraseña, que se utiliza solo en la verificación.

- **Métodos:**

- **getPassword() y setPassword(String password):** Métodos para obtener y establecer la contraseña.
- **getHash() y setHash(String hash):** Métodos para obtener y establecer el hash.

Paso 3: Crear el controlador REST

Creamos la clase controlador BcryptController



```
package com.example.bcryptdemo.controller;

import com.example.bcryptdemo.dto.PasswordRequest;
import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;
import org.springframework.web.bind.annotation.*;

@RestController
@RequestMapping("/api/bcrypt")
public class BcryptController {

    private final BCryptPasswordEncoder passwordEncoder = new BCryptPasswordEncoder();

    @PostMapping("/encrypt")
    public String encrypt(@RequestBody PasswordRequest request) {
        return passwordEncoder.encode(request.getPassword());
    }

    @PostMapping("/verify")
    public String verify(@RequestBody PasswordRequest request) {
        boolean match = passwordEncoder.matches(request.getPassword(), request.getHash());
        return match ? "Contraseña válida" : "Contraseña inválida";
    }
}
```

Explicación:

- **Anotaciones:**
 - **@RestController:** Indica que esta clase es un controlador REST.
 - **@RequestMapping("/api/bcrypt"):** Define la ruta base para todos los endpoints en este controlador.
- **Atributo:**
 - **private final BCryptPasswordEncoder passwordEncoder = new BCryptPasswordEncoder();:** Crea una instancia de BCryptPasswordEncoder, que se utiliza para cifrar y verificar contraseñas.
- **Métodos:**
 - **@PostMapping("/encrypt"):** Este método recibe una solicitud POST en la ruta /api/bcrypt/encrypt. Toma un objeto PasswordRequest en el cuerpo de la solicitud, cifra la contraseña utilizando BCrypt y devuelve el hash resultante.
 - **@PostMapping("/verify"):** Este método recibe una solicitud POST en la ruta /api/bcrypt/verify. Toma un objeto PasswordRequest que contiene una contraseña y un hash, y verifica si la contraseña coincide con el hash. Devuelve un mensaje indicando si la contraseña es válida o no.

Paso 4: Desactivar la seguridad del spring security (no lo usaremos en este ejemplo)

Agrega una clase de configuración para decirle a Spring que **no proteja los endpoints** en la clase **SecurityConfig**:



```
package com.example.bcryptdemo.config;

import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.security.config.Customizer;
import org.springframework.security.web.SecurityFilterChain;
import org.springframework.security.config.annotation.web.builders.HttpSecurity;

@Configuration
public class SecurityConfig {

    @Bean
    public SecurityFilterChain securityFilterChain(HttpSecurity http) throws Exception {
        http
            .csrf(csrf -> csrf.disable()) // Desactivar CSRF para pruebas POST con Postman
            .authorizeHttpRequests(auth -> auth
                .anyRequest().permitAll() // Permitir todos los endpoints
            )
            .httpBasic(Customizer.withDefaults()); // Desactiva login form

        return http.build();
    }
}
```

Explicación:

- **Anotaciones:**
 - **@Configuration:** Indica que esta clase es una clase de configuración de Spring.
 - **@Bean:** Indica que el método `securityFilterChain` devuelve un objeto que debe ser registrado como un bean en el contexto de Spring.
- **Método `securityFilterChain`:**
 - Este método configura las reglas de seguridad para las solicitudes HTTP.
 - **`http.csrf(csrf -> csrf.disable());`** Desactiva la protección CSRF (Cross-Site Request Forgery). Esto es útil para pruebas con herramientas como Postman, pero en un entorno de producción, es recomendable mantenerlo habilitado.
 - **`authorizeHttpRequests(auth -> auth.anyRequest().permitAll());`** Permite todas las solicitudes HTTP sin necesidad de autenticación. Esto significa que cualquier endpoint en la aplicación es accesible sin restricciones.
 - **`httpBasic(Customizer.withDefaults());`** Configura la autenticación básica HTTP, pero en este caso, no se está utilizando ya que todas las solicitudes están permitidas.

Paso 5: Ejecutar los endpoints en Postman

POST `http://localhost:8080/api/bcrypt/encrypt`



```
{  
  "password": "clave123"  
}
```

bcrypt / Encriptar contraseña

POST http://localhost:8080/api/bcrypt/encrypt

Params Authorization Headers (9) Body Scripts Settings

none form-data x-www-form-urlencoded raw binary GraphQL JSON

Body

```
1 {  
2   "password": "clave123"  
3 }
```

200 OK • 115 ms • 394 B

Raw Preview Visualize

1 \$2a\$10\$61M8n7Rmh5LwR5vUW2y08uCbF31wQ1ufQc7ovdXPYDRwUPRiCfM2m

POST http://localhost:8080/api/bcrypt/verify



```
{  
  "password": "clave123",  
  "hash": "{{bcrypt_hash}}"  
}
```

bcrypt / Verificar contraseña

POST http://localhost:8080/api/bcrypt/verify

Params Authorization Headers (9) Body Scripts Settings

none form-data x-www-form-urlencoded raw binary GraphQL JSON

Body

```
1 {  
2   "password": "clave123",  
3   "hash": "{{bcrypt_hash}}"  
4 }
```

200 OK • 102 ms • 353 B

Raw Preview Visualize

1 Contraseña válida