Pasos para crear un proyecto completo usando **Angular**, **Spring Boot**, **MongoDB** y **JWT**. Desde la creación del proyecto hasta cómo hacer la conexión entre el frontend y el backend.

Paso 1: Configuración del Backend (Spring Boot + MongoDB + JWT)

1.1. Instalación de herramientas necesarias

- **Java**: Asegúrate de tener **Java 11 o superior** instalado. Puedes descargarlo desde aquí.
- **Maven**: Necesitarás Maven para gestionar las dependencias del proyecto.
- **MongoDB**: Si no tienes MongoDB instalado, puedes instalarlo desde aquí. Si prefieres no instalarlo localmente, puedes usar **MongoDB Atlas** (una base de datos en la nube gratuita).

1.2. Crear un proyecto Spring Boot

1. Ve a Spring Initializr.
2. Configura el proyecto con las siguientes opciones:
    - **Project**: Maven Project
    - **Language**: Java
    - **Spring Boot**: 2.7.4 (o la última versión estable)
    - **Project Metadata**:
        - **Group**: com.example
        - **Artifact**: comentariosapp
        - **Name**: ComentariosApp
        - **Description**: Proyecto de comentarios con Spring Boot y MongoDB
        - **Package Name**: com.example.comentariosapp
        - **Packaging**: Jar
        - **Java**: 11 (o la versión que tengas instalada)
3. En **Dependencies**, selecciona:
    - Spring Web
    - Spring Data MongoDB
    - Spring Security
    - Spring Boot DevTools
    - Lombok (opcional pero ayuda a reducir el código)
4. Haz clic en **Generate**, descarga el archivo ZIP y descomprímelo en tu computadora.

1.3. Configuración de MongoDB

1. Asegúrate de que MongoDB esté corriendo en tu máquina o utiliza MongoDB Atlas (si decides usar Atlas, crea una base de datos en la nube y obtén la URL de conexión).
2. En el archivo `src/main/resources/application.properties`, agrega la URL de conexión de MongoDB:

```
spring.data.mongodb.uri=mongodb://localhost:27017/comentariosapp
```

1.4. Crear modelos y repositorios

1. **Comentario.java**: Crea una clase en `src/main/java/com/example/comentariosapp/model/Comentario.java`:

```
package com.example.comentariosapp.model;
```

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

@Document
public class Comentario {
    @Id
    private String id;
    private String mensaje;
    private String usuario;

    // Getters and setters
    public String getId() {
        return id;
    }

    public void setId(String id) {
        this.id = id;
    }

    public String getMensaje() {
        return mensaje;
    }

    public void setMensaje(String mensaje) {
        this.mensaje = mensaje;
    }

    public String getUsuario() {
        return usuario;
    }

    public void setUsuario(String usuario) {
        this.usuario = usuario;
    }
}
```

2. **ComentarioRepository.java**: Crea una interfaz en `src/main/java/com/example/comentariosapp/repository/ComentarioRepository.java`:

```
package com.example.comentariosapp.repository;

import com.example.comentariosapp.model.Comentario;
import org.springframework.data.mongodb.repository.MongoRepository;

public interface ComentarioRepository extends MongoRepository<Comentario,
String> {
}
```

1.5. Configuración de JWT

1. **JwtUtil.java**: Crea una clase en `src/main/java/com/example/comentariosapp/security/JwtUtil.java` para gestionar la creación y validación de tokens JWT:

```
package com.example.comentariosapp.security;

import io.jsonwebtoken.Jwts;
import io.jsonwebtoken.SignatureAlgorithm;
import java.util.Date;
```

```
public class JwtUtil {

    private String secretKey = "miSecreto"; // Cambiar este valor por algo más
seguro

    public String generateToken(String username) {
        return Jwts.builder()
                .setSubject(username)
                .setIssuedAt(new Date())
                .setExpiration(new Date(System.currentTimeMillis() + 1000 * 60 *
60 * 10)) // 10 horas
                .signWith(SignatureAlgorithm.HS256, secretKey)
                .compact();
    }

    public boolean validateToken(String token, String username) {
        String extractedUsername = extractUsername(token);
        return (username.equals(extractedUsername) && !isTokenExpired(token));
    }

    public String extractUsername(String token) {
        return Jwts.parser()
                .setSigningKey(secretKey)
                .parseClaimsJws(token)
                .getBody()
                .getSubject();
    }

    private boolean isTokenExpired(String token) {
        return extractExpiration(token).before(new Date());
    }

    private Date extractExpiration(String token) {
        return Jwts.parser()
                .setSigningKey(secretKey)
                .parseClaimsJws(token)
                .getBody()
                .getExpiration();
    }
}
```

2. **SecurityConfig.java**: Configura la seguridad de Spring para usar JWT en `src/main/java/com/example/comentariosapp/security/SecurityConfig.java`:

```
package com.example.comentariosapp.security;

import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.security.config.annotation.web.builders.HttpSecurity;
import
org.springframework.security.config.annotation.web.configuration.EnableWebSecuri
ty;
import
org.springframework.security.config.annotation.web.configuration.WebSecurityConf
igurerAdapter;

@Configuration
@EnableWebSecurity
public class SecurityConfig extends WebSecurityConfigurerAdapter {

    @Override
```

```
    protected void configure(HttpSecurity http) throws Exception {
        http.csrf().disable()
            .authorizeRequests()
            .antMatchers("/auth/**").permitAll() // Rutas públicas para login
            .antMatchers("/api/comentarios/admin").hasRole("ADMIN") // Solo
acceso para admin
            .anyRequest().authenticated() // Resto de rutas protegidas
            .and().addFilter(new
JwtAuthorizationFilter(authenticationManager()));
    }

}
```

1.6. Crear Controladores

1. **AuthController.java**: En `src/main/java/com/example/comentariosapp/controller/AuthController.java` para el login:

```
package com.example.comentariosapp.controller;

import com.example.comentariosapp.security.JwtUtil;
import org.springframework.web.bind.annotation.*;

@RestController
@RequestMapping("/auth")
public class AuthController {

    private JwtUtil jwtUtil;

    public AuthController(JwtUtil jwtUtil) {
        this.jwtUtil = jwtUtil;
    }

    @PostMapping("/login")
    public String login(@RequestBody UserCredentials credentials) {
        // Aquí puedes validar las credenciales
        return jwtUtil.generateToken(credentials.getUsername());
    }
}

class UserCredentials {
    private String username;
    private String password;

    // Getters and setters
}
```

2. **ComentarioController.java**: En `src/main/java/com/example/comentariosapp/controller/ComentarioController.java` para manejar los comentarios:

```
package com.example.comentariosapp.controller;

import com.example.comentariosapp.model.Comentario;
import com.example.comentariosapp.repository.ComentarioRepository;
import org.springframework.web.bind.annotation.*;

import java.util.List;

@RestController
```

```
@RequestMapping("/api/comentarios")
public class ComentarioController {

    private ComentarioRepository comentarioRepository;

    public ComentarioController(ComentarioRepository comentarioRepository) {
        this.comentarioRepository = comentarioRepository;
    }

    @PostMapping
    public Comentario enviarComentario(@RequestBody Comentario comentario) {
        return comentarioRepository.save(comentario);
    }

    @GetMapping("/admin")
    public List<Comentario> obtenerComentarios() {
        return comentarioRepository.findAll();
    }
}
```

Paso 2: Frontend (Angular)

2.1. Crear el Proyecto Angular

1. Asegúrate de tener **Node.js** y **npm** instalados. Puedes descargarlos desde aquí.
2. Instala Angular CLI globalmente (si no lo tienes):

```
npm install -g @angular/cli
```

3. Crea el proyecto Angular:

```
ng new comentarios-frontend
cd comentarios-frontend
```

4. Instala las dependencias necesarias para manejar JWT:

```
npm install @auth0/angular-jwt
```

2.2. Crear el Servicio de Autenticación

En `src/app/auth.service.ts`, crea un servicio para manejar el login y la verificación del token JWT:

```
import { Injectable } from '@angular/core';
import { HttpClient } from '@angular/common/http';
import { Observable } from 'rxjs';
import { JwtHelperService } from '@auth0/angular-jwt';

@Injectable({
  providedIn: 'root'
})
export class AuthService {
  private apiUrl = 'http://localhost:8080/auth/login';
  private jwtHelper = new JwtHelperService();

  constructor(private http: HttpClient) { }

  login(username: string, password: string): Observable<any> {
    return this.http.post<any>(this.apiUrl, { username, password });
  }
```

```
  isAuthenticated(): boolean {
    const token = localStorage.getItem('token');
    return token && !this.jwtHelper.isTokenExpired(token);
  }

  getToken(): string {
    return localStorage.getItem('token')!;
  }
}
```

2.3. Crear Componentes de Login y Usuario

1. **LoginComponent** en `src/app/login/login.component.ts`:

```
import { Component } from '@angular/core';
import { AuthService } from '../auth.service';
import { Router } from '@angular/router';

@Component({
  selector: 'app-login',
  template: `
    <form (ngSubmit)="login()">
      <input [(ngModel)]="username" placeholder="Usuario" name="username" />
      <input [(ngModel)]="password" placeholder="Contraseña" type="password"
name="password" />
      <button type="submit">Iniciar sesión</button>
    </form>
  `
})
export class LoginComponent {
  username: string = '';
  password: string = '';

  constructor(private authService: AuthService, private router: Router) {}

  login() {
    this.authService.login(this.username, this.password).subscribe(response => {
      localStorage.setItem('token', response.token);
      this.router.navigate(['/usuario']);
    });
  }
}
```

2. **UsuarioComponent** en `src/app/usuario/usuario.component.ts` para enviar comentarios:

```
import { Component } from '@angular/core';
import { AuthService } from '../auth.service';
import { HttpClient } from '@angular/common/http';

@Component({
  selector: 'app-usuario',
  template: `
    <form (ngSubmit)="enviarComentario()">
      <textarea [(ngModel)]="comentario" name="comentario"></textarea>
      <button type="submit">Enviar Comentario</button>
    </form>
  `
})
export class UsuarioComponent {
  comentario: string = '';
```

```
  constructor(private authService: AuthService, private http: HttpClient) {}

  enviarComentario() {
    const token = this.authService.getToken();
    this.http.post('http://localhost:8080/api/comentarios', { mensaje:
this.comentario }, {
      headers: { 'Authorization': `Bearer ${token}` }
    }).subscribe();
  }
}
```

2.4. Configurar las Rutas

En `src/app/app-routing.module.ts`:

```
import { NgModule } from '@angular/core';
import { RouterModule, Routes } from '@angular/router';
import { LoginComponent } from './login/login.component';
import { UsuarioComponent } from './usuario/usuario.component';

const routes: Routes = [
  { path: '', redirectTo: '/login', pathMatch: 'full' },
  { path: 'login', component: LoginComponent },
  { path: 'usuario', component: UsuarioComponent }
];

@NgModule({
  imports: [RouterModule.forRoot(routes)],
  exports: [RouterModule]
})
export class AppRoutingModule { }
```

Paso 3: Ejecutar el Proyecto

1. **Backend**:
    - Navega a la carpeta del proyecto Spring Boot y ejecuta:

```
mvn spring-boot:run
```

2. **Frontend**:
    - Navega a la carpeta del proyecto Angular y ejecuta:

```
ng serve
```

Esto debería levantar el frontend en `http://localhost:4200` y el backend en `http://localhost:8080`.

Paso 4: Probar la Aplicación

1. Accede a la aplicación en el navegador.
2. Inicia sesión usando el formulario en la página de login.
3. En la página de usuario, envía un comentario.
4. Verifica que los comentarios estén almacenados en MongoDB y que el administrador pueda acceder a ellos.

Ahora tienes una aplicación básica que utiliza **Angular**, **Spring Boot**, **MongoDB** y **JWT** para autenticación. Puedes seguir ampliando este proyecto con más características según lo necesites.