Universidad Nacional de Colombia - sede Bogotá
Facultad de Ingeniería
Departamento de Sistemas e Industrial
Curso: Ingeniería de Software 1

Integrantes

- Daniel Alejandro Duitama Correa
- Edwin Felipe Pinilla Peralta
- Miguel Angel Martinez Fernandez (miamartinezfe@unal.edu.co)
- Juan Sebastián Umaña Camacho (juumanac@unal.edu.co)

Backend

Descripción

Para la consistencia y buenas prácticas del código en el backend (java con spring boot), se decidió usar Checkstyle para verificar convenciones de código y estilo , además de usar SonarQube para un análisis estático más avanzado (bugs, vulnerabilidades, code smells).

Checkstyle

1. Primero agregamos el plugin en el archivo POM.<plugin>

```
<groupId>org.apache.maven.plugins</groupId>
<artifactId>maven-checkstyle-plugin</artifactId>
<version>3.2.0</version>
<executions>
   <execution>
      <phase>verify</phase>
      <goals>
         <goal>check</goal>
      </goals>
   </execution>
</executions>
<configuration>
   <configLocation>google_checks.xml</configLocation>
   <consoleOutput>true</consoleOutput>
   <failOnViolation>true</failOnViolation>
```

```
    </configuration>
</plugin>
```

2. Tener en cuenta que se usa los checks o el standard que definió google, una vez agregado usamos el siguiente comando de Maven ‘mvn checkstyle:check ‘, este comando revisa archivo por archivo revisando que se cumplan los estándares configurados en ‘google_checks.xml’ y mostrará advertencias si alguno se incumple o directamente no dejara iniciar el codigo si hay una violacion.
3. Aquí podemos ver la auditoria con el proyecto vacío el cual marca que los directorios no cumplen con los estándares

```
[INFO] --- checkstyle:3.2.0:check (default-cli) @ union-backend ---
[INFO] Comenzando auditoría...
[WARN] C:\Users\Miguel
Nuvu\Desktop\repositorio-grupal\Proyecto\Backend\union-backend\src\main\java\
com\UNion\union_backend\UnionBackendApplication.java:1:9: Package name
'com.UNion.union_backend' must match pattern '^[a-z]+(\.[a-z][a-z0-9]*)*$'.
[PackageName]
Auditoría concluida.
[INFO] You have 0 Checkstyle violations.
[INFO]
------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO]
------------------------------------------------------------------------
```

4. Luego hacemos otra prueba poniendo una linea que es muy larga para los checks:

```
@SpringBootApplication
public class UnionBackendApplication {

  public static void main(String[] args) {
    System.out.println("Hola Mundosssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssss"); // Linea muy larga
    SpringApplication.run(UnionBackendApplication.class, args);
  }

}
```

Lo cual da como resultado (que indica se pasan los 100 caracteres definidos en el estándar):

```
[INFO] --- checkstyle:3.2.0:check (default-cli) @ union-backend ---
[INFO] Comenzando auditoría...
[WARN] C:\Users\Miguel
Nuvu\Desktop\repositorio-grupal\Proyecto\Backend\union-backend\src\main\java\
com\UNion\union_backend\UnionBackendApplication.java:1:9: Package name
'com.UNion.union_backend' must match pattern '^[a-z]+(\.[a-z][a-z0-9]*)*$'.
[PackageName]
[WARN] C:\Users\Miguel
Nuvu\Desktop\repositorio-grupal\Proyecto\Backend\union-backend\src\main\java\
```

```
com\UNion\union_backend\UnionBackendApplication.java:14: La línea es mayor de
100 caracteres (encontrado 130). [LineLength]
Auditoría concluida.
[INFO] You have 0 Checkstyle violations.
[INFO]
------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO]
------------------------------------------------------------------------
```

Frontend

Descripción

Para garantizar la consistencia y las buenas prácticas en el código del frontend (TypeScript-Next.js 14), se decidió utilizar **ESLint** para verificar las convenciones de código y estilo.

ESLint

1. Para la creación del proyecto se utilizo la libreria create-next-app la cual incluye en el proyecto de manera automática y con la configuración predeterminada el linter **ESLint**, esta es la configuración predeterminada:

```
{
      "extends": ["next/core-web-vitals", "next/typescript"]
}
```

Esta configuración predeterminada emplea dos paquetes de reglas, los cuales a su vez emplean el paquete de reglas padre “next”. En total suman casi 100 reglas, las cuales se pueden consultar en la documentación de Next.js sobre el plugin de ESLint.

2. Aunque ESLint funciona en tiempo real (mientras se escribe el código) por defecto, al crear el proyecto con create-next-app, se crea un script específico para ejecutar ESLint manualmente en el proyecto:

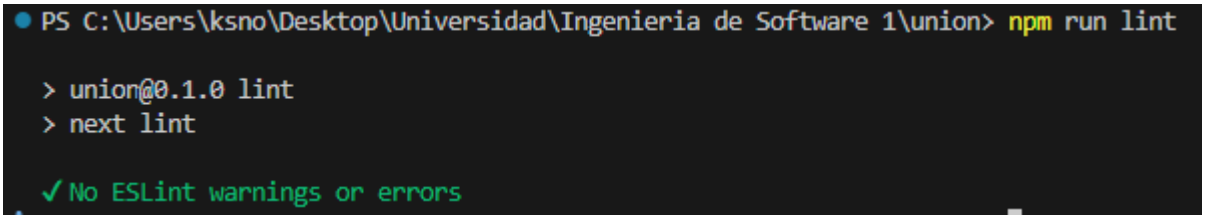
```
"scripts": {
    "dev": "next dev",
    "build": "next build",
    "start": "next start",
    "lint": "next lint"
```

```
  },
```

Por lo que para realizar una auditoría de código limpio de manera manual ejecutamos el comando:

```
npm run lint
```

3.Después de realizar la ejecución del comando, vemos en consola los resultados



Aquí podemos ver que el proyecto hasta ahora (50% aprox) cumple con todas las reglas de código limpio establecidas por ESLint.

4.A manera de prueba, si creamos una variable que nunca se usa en el código, ESLint nos mostrará un mensaje de error ya que esto viola una de sus reglas configuradas (@typescript-eslint/no-unused-vars)