Proyecto Final

Presentado por:
Stiven Aguirre Granada - *staguirreg@unal.edu.co*
Andrés Felipe Perdomo Uruburu - *aperdomou@unal.edu.co*
Cesar Camilo Velandia Cubillos - *cevelandiac@unal.edu.co*
Manuel Santiago Mori Ardila - *mmori@unal.edu.co*

Profesor:
Oscar Eduardo Alvarez Rodriguez
oalvarezr@unal.edu.co

Viernes 8 de Febrero

Universidad Nacional de Colombia
Facultad de Ingeniería
Departamento de Ingeniería Sistemas e Industrial
2024

Anexo de Documentos Relacionados:

- Proyecto_Final

PUNTO 7: Clean Code

Código limpio (Clean Code)

En este proyecto, una de nuestras prioridades es mantener un código limpio y organizado para que sea fácil de entender y mejorar. En el desarrollo de software, mantener un código limpio y bien estructurado es esencial para asegurar la calidad, facilitar el mantenimiento y mejorar la colaboración entre los desarrolladores. Por eso, decidimos aplicar algunas reglas de **Clean Code** usando herramientas que nos faciliten la vida, tanto en el **backend** como en el **frontend**.

En este proyecto, se implementarán herramientas de análisis estático y formateo automático tanto para el **backend** como para el **frontend**. Estas herramientas nos ayudarán a identificar problemas en el código y a aplicar buenas prácticas de programación de forma consistente.

Backend:

Orden y control con ESLint y Prettier

En el backend, como estamos trabajando con **Nest.js** (que funciona sobre **Node.js**), vamos a usar dos herramientas muy conocidas: **ESLint** y **Prettier**. Estas nos van a ayudar a mantener el código en línea con las buenas prácticas, detectar errores y asegurarnos de que todo el proyecto se vea bonito y ordenado (porque el código también tiene que verse bien).

- **ESLint** se encarga de revisar el código y marcar cualquier cosa rara o fuera de lugar, como errores de sintaxis o reglas de estilo que no se estén cumpliendo.
- **Prettier**, por su parte, se enfoca en el formato: alinea el código, pone los espacios correctos.

  Configuración

  Cuando creamos un proyecto de **Nest.js**, ESLint y Prettier ya vienen casi listos para usar. Solo tenemos que asegurarnos de tener todo instalado y configurar algunas cosa:

  - Primero instalamos Nest.js:
    npm install -g @nestjs/cli

  - Luego, creamos el proyecto:
    nest new proyecto-ing.-soft-i

  Escogimos npm como gestor de paquetes y el proyecto ya tiene ESLint y Prettier integrados.

Uso

Para revisar si el código cumple con las reglas de ESLint, ejecutamos: *npm run lint*
Esto nos muestra todos los errores o advertencias del código. Si algo no está bien, lo arreglamos antes de seguir.

Con **Prettier** ejecutamos: *npm run format*

Esto le da formato a todo el código. Así se evita escribir código desordenado o con inconsistencias de estilo.

Luego usamos **Visual Studio Code** con las extensiones de ESLint y Prettier para que nos marque los errores directamente mientras escribimos. Esto ahorra mucho tiempo y facilita corregir cualquier error en el código.

Frontend:

Limpieza con flutter_lints

En el frontend estamos trabajando con **Flutter**, así que aquí usamos **flutter_lints**, que es una colección de reglas específicas para proyectos en Flutter. Esto nos ayuda a mantener el código limpio y en línea con las buenas prácticas recomendadas para este framework.

flutter_lints se configura desde el archivo *pubspec.yaml*. Una vez listo, el comando *flutter analyze* nos permite revisar todo el código y marcar las cosas que necesitan corrección.

Uso práctico

Si nos olvidamos de usar *const* en los constructores de nuestros widgets, el linter nos avisa con una advertencia (*prefer_const_constructors*). Podemos solucionarlo manualmente o usar el comando:

```
dart fix --apply
```

Esto corrige automáticamente todas las ocurrencias en el archivo.
También nos da la opción de ignorar ciertas reglas si es necesario. Por ejemplo, si queremos ignorar la advertencia de *avoid_print*, podemos añadir un comentario:

```
// ignore: avoid_print

print('Mensaje de prueba');
```

Buenas prácticas:

Además de usar estas herramientas, aplicamos varias reglas básicas para mantener el código limpio:

- **Nombres claros:** Nada de *x, y o variable1*. Usamos nombres descriptivos para que cualquiera entienda el código.
- **Funciones pequeñas:** Si una función empieza a crecer demasiado, la dividimos. Así, cada función hace solo una cosa y es más fácil de mantener.
- **Código muerto:** Eliminamos código que no usamos o comentarios innecesarios.
- **Consistencia:** Usamos el mismo estilo de código en todo el proyecto. Esto hace que se vea profesional y fácil de leer.