



UNIVERSIDAD
NACIONAL
DE COLOMBIA

PROYECTO FINAL

Ingeniería de Software I

Wulfredo Javier Barco Godoy - wbarco@unal.edu.co
Brahian Camilo Gómez Carvajal - bgomezca@unal.edu.co
Juan David Rivera Buitrago - jriverabu@unal.edu.co
William Darío Vanegas Marín - wvanegas@unal.edu.co

Profesor
Oscar Eduardo Alvarez Rodríguez

Universidad Nacional de Colombia
Facultad de Ingeniería
Ingeniería de Sistemas y Computación
Bogotá, Colombia
09 de Febrero de 2025

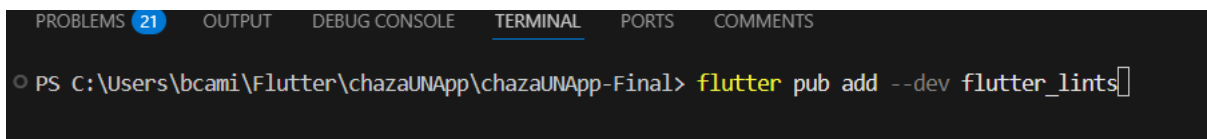
7. Clean code

En el mundo del desarrollo de software, Clean Code se refiere a escribir código de manera clara, legible y fácil de mantener. Este enfoque promueve buenas prácticas como nombrar adecuadamente variables y funciones, limitar la complejidad en los métodos o clases, y eliminar duplicaciones innecesarias. El objetivo es crear un código que sea lo más comprensible posible para los desarrolladores, facilitando la colaboración y reduciendo costos de mantenimiento en el largo plazo. Para asegurar que el código cumpla con estos estándares de calidad, se suelen utilizar herramientas llamadas linters. Un linter revisa automáticamente el código fuente para detectar errores comunes, inconsistencias de formato o incumplimiento de convenciones de estilo.

En nuestro proyecto dado que estamos desarrollando con Flutter vamos a usar 'flutter_lints' como herramienta para el análisis estático del código debido a que es el paquete de linters que nos ofrece este lenguaje como opción más usada, así que vamos a proceder con la instalación :

1. En nuestra terminal procederemos a instalar el paquete de linters con el siguiente comando :

```
Unset  
flutter pub add --dev flutter_lints
```



The screenshot shows a terminal window with a dark background. At the top, there are tabs for 'PROBLEMS' (with a blue circle containing '21'), 'OUTPUT', 'DEBUG CONSOLE', 'TERMINAL' (which is selected and underlined), 'PORTS', and 'COMMENTS'. Below the tabs, the terminal prompt shows the command being executed: 'PS C:\Users\bcami\Flutter\chazaUNApp\chazaUNApp-Final> flutter pub add --dev flutter_lints'.

```

PS C:\Users\bcami\Flutter\chazaUNApp\chazaUNApp-Final> flutter pub add --dev flutter_lints
Resolving dependencies...
Downloading packages...
  async 2.11.0 (2.13.0 available)
  boolean_selector 2.1.1 (2.1.2 available)
  characters 1.3.0 (1.4.0 available)
  clock 1.1.1 (1.1.2 available)
  collection 1.19.0 (1.19.1 available)
  fake_async 1.3.1 (1.3.3 available)
+ flutter_lints 5.0.0
  font_awesome_flutter 9.2.0 (10.8.0 available)
  leak_tracker 10.0.7 (10.0.9 available)
  leak_tracker_flutter_testing 3.0.8 (3.0.9 available)
+ lints 5.1.1

```

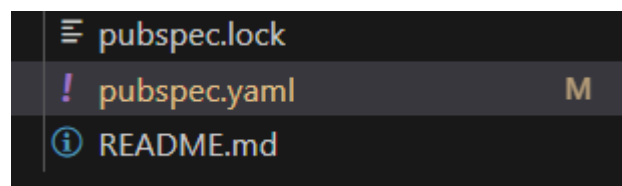
Como resultado obtendremos la descarga de estos dos paquetes y la añadidura a la respectiva dependencia , en nuestro caso dev_dependencies que contiene paquetes para desarrollo, como linters o herramientas de pruebas :

```

60 dev_dependencies:
61   flutter_test:
62     sdk: flutter
63   flutter_lints: ^5.0.0
64
65   # The "flutter_lints" package below contains a set of recommended lints to
66   # encourage good coding practices. The lint set provided by the package is
67   # activated in the `analysis_options.yaml` file located at the root of your
68   # package. See that file for information about deactivating specific lint
69   # rules and activating additional ones.

```

esto a su vez está contenido en el archivo pubspec.yaml que es nuestro archivo principal de configuración de nuestro proyecto en Flutter , aquí definimos dependencias, versiones, paquetes, activos y configuración del proyecto.Cuando agregamos paquetes con ‘flutter pub add’ se registran aquí.



Una vez instalamos los linters procedemos a configurarlos a nuestras necesidades si queremos , en caso de no hacerlo Dart usará sus reglas por defecto. Por consiguiente procederemos a configurar nuestras reglas en el archivo analysis_options.yaml , si no se creó automáticamente lo podemos hacer manualmente y deberá estar en la raíz del proyecto. En nuestro caso se creó automáticamente y se ve así :

```

chazaUNApp-Final > ! analysis_options.yaml
1  # This file configures the analyzer, which statically analyzes Dart code to
2  # check for errors, warnings, and lints.
3  #
4  # The issues identified by the analyzer are surfaced in the UI of Dart-enabled
5  # IDEs (https://dart.dev/tools#ides-and-editors). The analyzer can also be
6  # invoked from the command line by running `flutter analyze`.
7
8  # The following line activates a set of recommended lints for Flutter apps,
9  # packages, and plugins designed to encourage good coding practices.
10 include: package:flutter_lints/flutter.yaml
11
12 linter:
13   # The lint rules applied to this project can be customized in the
14   # section below to disable rules from the `package:flutter_lints/flutter.yaml`
15   # included above or to enable additional rules. A list of all available lints
16   # and their documentation is published at
17   # https://dart-lang.github.io/linter/lints/index.html.
18   #
19   # Instead of disabling a lint rule for the entire project in the
20   # section below, it can also be suppressed for a single line of code
21   # or a specific dart file by using the `// ignore: name_of_lint` and
22   # `// ignore_for_file: name_of_lint` syntax on the line or in the file
23   # producing the lint.
24   rules:
25     # avoid_print: false # Uncomment to disable the `avoid_print` rule
26     # prefer_single_quotes: true # Uncomment to enable the `prefer_single_quotes` rule
27
28   # Additional information about this file can be found at
29   # https://dart.dev/guides/language/analysis-options
30

```

En el apartado de rules es donde procederemos a colocar las reglas que consideremos convenientes para la buena estructura y funcionamiento de nuestro código :

```

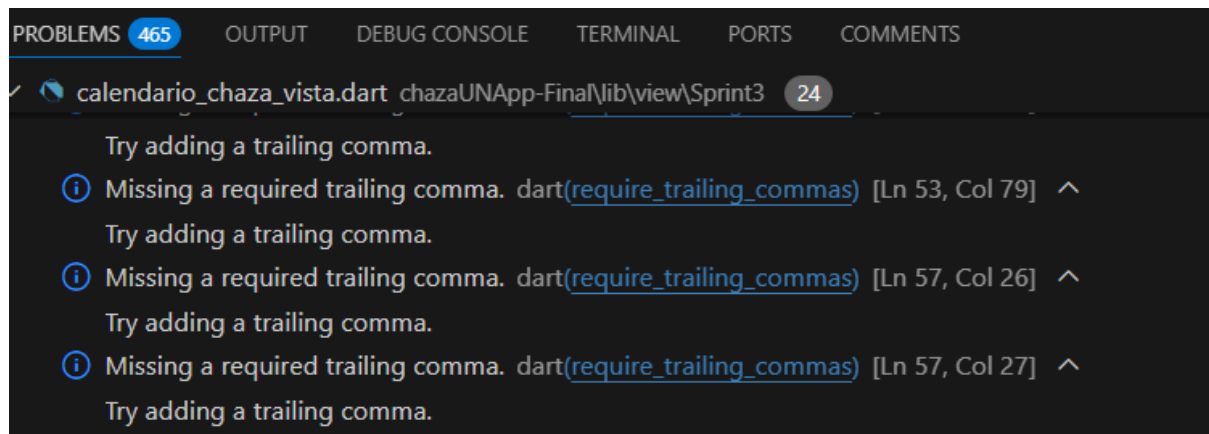
rules:
  # avoid_print: false # Uncomment to disable the `avoid_print` rule
  # prefer_single_quotes: true # Uncomment to enable the `prefer_single_quotes` rule
  - prefer_const_constructors #use const when posible
  - avoid_print #avoid print use log instead
  - prefer_final_fields #use final instead of var when the value is not going to change
  - always_use_package_imports #use package imports instead of relative imports
  - require_trailing_commas #require trailing commas
  - sort_pub_dependencies #sort pub dependencies

```

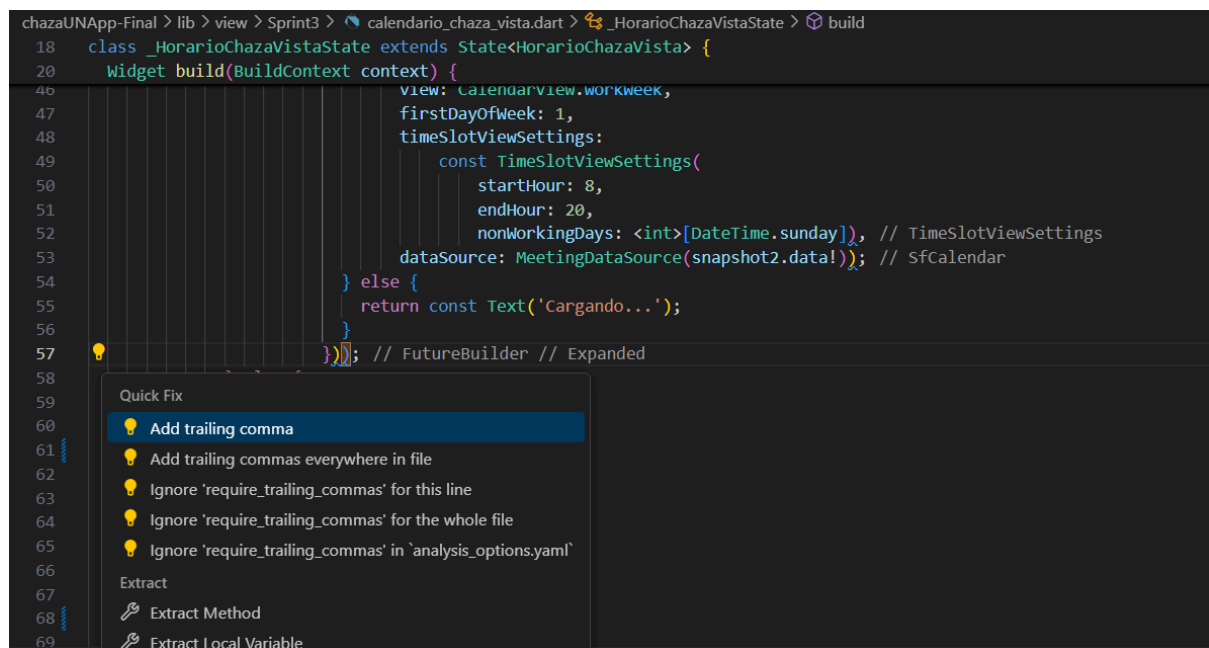
Una vez guardamos estos cambios los linters empezaran a hacer su trabajo e identificaran errores en base a las reglas definidas :



Sorprendentemente pasamos de 21 problemas en nuestro código a 465 una vez los linters empezaron su funcionamiento , pero es preciso mencionar que la inmensa mayoría de estos errores son simples comas que antes no contemplabamos colocar pero que mejoraran la lectura de nuestro código:



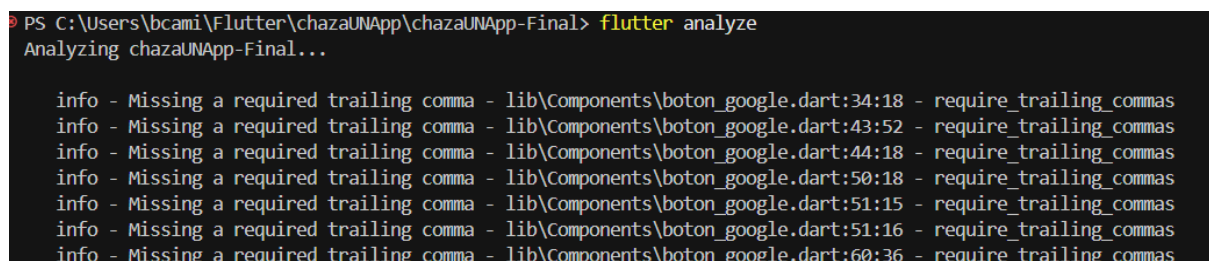
Nos dirigimos a esta línea y el mismo editor nos da la opción de corregir esto rápidamente :



Una vez se coloca la o las respectivas comas , ya no arroja ningún error o al menos no respecto a esta regla xD.



Esta manera de visualización es mediante el bloque problems , si quisiéramos verlo por medio de la consola tendremos que ejecutar el comando 'flutter analyze' y esto nos debería arrojar los mismos problemas :



Y así básicamente sería la manera en que los linters nos ayudarán a gestionar nuestros errores , haciendo de nosotros mejores escritores de código y siguiendo patrones para implementar un mejor y más legible código para otros.

- **Quejas u Opiniones :**

Brahian Camilo : Debo mencionar que gran parte del código hasta el momento ha sido escrito por javier dado que el nos ofreció continuar con esta idea que él ya venía desarrollando pero así como se lo hice saber a el , para mi fue y es difícil entender y leer su código , y adicionalmente entender la estructura de las carpetas pues desde mi humilde opinión las veo un poco desordenadas, por tanto para mi es difícil modificar su código y leerlo , no obstante ya hemos hablado de esto con todos y estamos trabajando en nuestra organización y hoy sumamos un paso añadiendo los linters que antes no teníamos.

Juan Rivera : Como mencionó camilo , ha sido un poco difícil acoplarnos al código de javier sin embargo en mi caso y junto con IA he tratado de hacerlo e igualmente ya hemos hablado de la reestructuración del mismo que es lo que estamos haciendo ahora , por otra parte nos dividimos roles en backend y frontend cada uno para aportar desde nuestras fortalezas.

William Vanegas : Por mi parte debo mencionar que no conocía dart ni flutter por tanto adaptarme me a costado un poquito más y adicionalmente mencionar que no solo ha sido el código de javier por las razones ya mencionadas sino también el de camilo que encuentro un poco difícil de entender , sin embargo me he ayudado de la IA y he aportado mi parte con ayuda de la misma , hemos hecho reuniones comentando esto mismo para poder cubrir estas necesidades como un grupo.

Javier Barco : Después de escuchar a mis compañeros he venido ordenando y tratando de hacer más legible mi código pues eran prácticas que no tenía antes pero que con este proyecto y las clases he venido implementando poco a poco para mejorar mi calidad y mi orden , adicionalmente se delegaron roles de acuerdo a las preferencias de cada uno y se está trabajando desde las mismas esperando poder sacar adelante el proyecto.

