Informe - Código limpio

Materia:

ingeniería de software I

Presentado por:

Adrian Ramirez Gonzalez

Adrian Alexander Benavides

Andrés Hernando Borda Muñoz

Brayan Alejandro Muñoz Pérez

Profesor:

Oscar Eduardo Álvarez Rodríguez

Universidad Nacional de Colombia

Facultad de ingeniería

Viernes 31 de enero del 2025

Implementación de Linters

Ya que este proyecto usa tanto Javascript como Python, se usará un Linter por cada lenguaje de programación. Específicamente, para Python se usará Pylint y para Javascript se usará ESLint.

Implementación de Pylint

1. Entrar al entorno virtual creado para python

```
C:\Users\adria\OneDrive\Desktop\ingenieria_de_software_1\Proyecto>cd venv

C:\Users\adria\OneDrive\Desktop\ingenieria_de_software_1\Proyecto\venv>cd Scripts

C:\Users\adria\OneDrive\Desktop\ingenieria_de_software_1\Proyecto\venv\Scripts>activate

(venv) C:\Users\adria\OneDrive\Desktop\ingenieria_de_software_1\Proyecto\venv\Scripts>
```

2. Instalar Pylint

```
(venv) C:\Users\adria\OneDrive\Desktop\ingenieria_de_software_1\Proyecto>pip install pylint
```

Con estos dos pasos, pylint ya se encuentra listo para ser ejecutado sobre la carpeta de interés que en este caso serán la carpetas “backend_djangoREST” e “inventario”.

ESLint

En este caso, al crear el proyecto en Vuejs se puede decidir si instalar ESLint desde el inicio del proyecto. Nosotros escogimos esta configuración desde el principio, por lo cual aquí mostraremos el paso a paso para crear un proyecto en Vue que incluya ESLint

1. Ejecutar el comando vue create <<nombre del proyecto>>

```
C:\Users\adria\Desktop\Prueba>vue create prueba
```

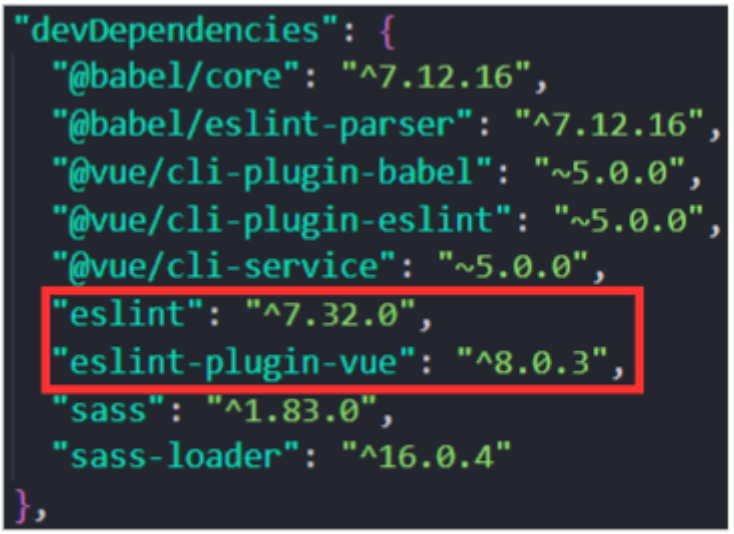
2. Se escoge la opcion Default ([Vue 3] babel, eslint)

```
? Please pick a preset:
> Default ([Vue 3] babel, eslint)
  Default ([Vue 2] babel, eslint)
  Manually select features
```

3. Se crea el proyecto

```
Vue CLI v5.0.8
✨  Creating project in C:\Users\adria\Desktop\Prueba\prueba.
🗃️ Initializing git repository...
⚙️  Installing CLI plugins. This might take a while...
```

4. Una vez creado el proyecto, se puede verificar la instalación de ESLint dentro del archivo package.json que trae el proyecto



Ejecución de las herramientas

Pylint

Para ejecutar Pylint se usará el comando **python -m pylint backend_djangoREST** y **python -m pylint inventario.** Como resultado de el uso del comando se tiene la siguiente salida:

1. **python -m pylint backend_djangoREST**

```
(venv) C:\Users\adria\OneDrive\Desktop\ingenieria_de_software_1\Proyecto\backend_djangoREST>python -m pylint backend_djangoREST
************* Module backend_djangoREST
backend_djangoREST\__init__.py:1:0: C0103: Module name "backend_djangoREST" doesn't conform to snake_case naming style (invalid-name)

-------------------------------------------------------------------
Your code has been rated at 9.71/10 (previous run: 9.70/10, +0.01)
```

Como se puede ver, el único error que tiene este código según el Linter es el nombre de la carpeta **backend_djangoREST** ya que según Pylint el nombre debería ser **backend_django_rest**, sin embargo, no se modifica el nombre de la carpeta ya que este fue el nombre creado durante la inicialización del proyecto, por lo cual varias partes del código necesitar buscar este nombre y cambiarlo podría implicar problemas.

2. **python -m pylint inventario**

Al ejecutar este comando se obtienen los siguientes errores

```
************* Module inventario.models
inventario\models.py:16:4: R0903: Too few public methods (0/2) (too-few-public-methods)
inventario\models.py:141:18: E1101: Instance of 'ForeignKey' has no 'nombre' member (no-member)
************* Module inventario.serializers
inventario\serializers.py:16:4: R0903: Too few public methods (0/2) (too-few-public-methods)
inventario\serializers.py:32:4: R0903: Too few public methods (0/2) (too-few-public-methods)
************* Module inventario.views
inventario\views.py:62:12: E1101: Class 'Organizacion' has no 'objects' member (no-member)
inventario\views.py:68:26: E1101: Class 'Organizacion' has no 'objects' member (no-member)
inventario\views.py:70:11: E1101: Class 'Organizacion' has no 'DoesNotExist' member (no-member)
inventario\views.py:96:9: W0613: Unused argument 'request' (unused-argument)
inventario\views.py:104:0: R0901: Too many ancestors (11/7) (too-many-ancestors)
inventario\views.py:109:15: E1101: Class 'Producto' has no 'objects' member (no-member)
inventario\views.py:149:27: E1101: Class 'Producto' has no 'objects' member (no-member)
inventario\views.py:145:15: W0613: Unused argument 'request' (unused-argument)
************* Module inventario.migrations.0001_initial
inventario\migrations\0001_initial.py:52:0: C0301: Line too long (146/100) (line-too-long)
inventario\migrations\0001_initial.py:65:0: C0301: Line too long (160/100) (line-too-long)
inventario\migrations\0001_initial.py:71:0: C0301: Line too long (108/100) (line-too-long)
inventario\migrations\0001_initial.py:72:0: C0301: Line too long (111/100) (line-too-long)
inventario\migrations\0001_initial.py:1:0: C0103: Module name "0001_initial" doesn't conform to snake_case naming style (invalid-name)
************* Module inventario.migrations.0002_alter_usuario_id_organizacion
inventario\migrations\0002_alter_usuario_id_organizacion.py:1:0: C0103: Module name "0002_alter_usuario_id_organizacion" doesn't conform to snake_case naming style (invalid-name)
************* Module inventario.migrations.0003_remove_movimiento_id_usuario_and_more
inventario\migrations\0003_remove_movimiento_id_usuario_and_more.py:1:0: C0103: Module name "0003_remove_movimiento_id_usuario_and_more" doesn't conform to snake_case naming style (invalid
-name)
************* Module inventario.migrations.0004_precios
inventario\migrations\0004_precios.py:1:0: C0103: Module name "0004_precios" doesn't conform to snake_case naming style (invalid-name)
************* Module inventario.migrations.__init__
inventario\migrations\__init__.py:1:0: R0801: Similar lines in 2 files
==inventario.migrations.0001_initial:[27:32]
==inventario.migrations.0004_precios:[24:29]
            fields=[
                ('id', models.BigAutoField(auto_created=True,
                                           primary_key=True,
                                           serialize=False,
                                           verbose_name='ID')), (duplicate-code)
```

Adicionalmente se obtiene la siguiente calificación

```
--------------------------------------------------------------------
Your code has been rated at 7.44/10 (previous run: 7.33/10, +0.11)
```

Los errores que aparecen (junto a la justificación de por qué no se resuelven) son los siguientes:

1. **too-few-public-methods:** Este error se da debido a que existe una clase que no tiene suficientes métodos públicos. En este caso el error se produce debido a la clase Meta, la cual es una clase usada para añadir información extra sobre otras clases, por lo cual nunca contiene funciones. Por esta razón no se puede resolver este problema, ya que implicaría crear funciones dentro de la clase Meta que no se usarán.
2. **no-member:** Este error surge debido a que se está intentando obtener un atributo de una entidad por medio de la Foreign Key y, aunque pylint cree que este atributo no existe, realmente el atributo si existe.
3. **unused-argument:** Este error se da debido a que Pylint considera que algunas variables no están siendo usadas, sin embargo, Django realmente si usa estas variables.
4. **line-too-long:** Este error aparece debido a que Pylint detecta líneas de código muy largas. En este caso estas líneas se encuentran en los archivos de migraciones (archivos creados por django cuando se realizan modificaciones a la base de datos). Estos archivos a veces pueden contener strings muy largos los cuales son detectados por Pylint.
5. **invalid-name:** Este error surge debido a que Pylint detecta nombres de archivos que siguen el formato snake_case, sin embargo, en este caso los archivos que no están siguiendo este formato son los archivos de migraciones los cuales son creados directamente por django, por lo cual modificar estos nombres estaría mal ya que es el estándar que maneja el framework.

ESLint

Para ejecutar el linter nos ubicamos en la carpeta donde está el proyecto de Vue y allí ejecutaremos el comando **npm run lint.** Como resultado de este comando tenemos la siguiente salida:

```
C:\Users\adria\Desktop\ingenieria_de_software_1\Proyecto\frontend_vuejs>npm run lint

> frontend_vuejs@0.1.0 lint
> vue-cli-service lint

 DONE  No lint errors found!
```

Como se puede ver, el linter nos confirma que no ha encontrado ningún error de estilo o formato.

Opinión respecto al código de sus compañeros

Adrian Ramirez Gonzalez

De acuerdo al código que he visto, puedo entender en que se ha estado trabajando gracias principalmente al nombramiento de variables y funciones que permiten rápidamente intuir cual es el propósito de cada sección de código. Por otro lado, si tuviera que trabajar con alguna sección de código que yo no he hecho, creo que en poco tiempo podría entenderlo y empezar trabajar sobre la misma.

Adrian Alexander Benavides

El código de mis compañeros es fácil de leer y es intuitivo en cuanto a poder leerlo, sin embargo en mi caso al no tener tanto conocimiento en algunas herramientas usadas se me dificultó al momento de realizar modificaciones, pero en cuanto al código de mis compañeros era legible, organizado y se logra comprender qué hace cada línea de código.

Andrés Hernando Borda Muñoz

Desde la perspectiva de clean code, aunque el código muestra una estructura funcional básica con una buena organización, presenta oportunidades de mejora en términos de consistencia y separación de responsabilidades.

Por ejemplo, la ausencia de un manejo de errores (alternando entre console.error en InventoryService y alert en el router), y la combinación de lógica de negocio con presentación (como se observa en la parte del inventario donde la obtención de datos y la presentación están en el mismo archivo) sugieren la necesidad de implementar patrones más consistentes y un enfoque más robusto en el manejo de estados y errores. Aparte ayudaría a la seguridad de la app, lo cual es fundamental en todo proyecto para asegurar a su vez la mantenibilidad.

Por lo demás si en algún momento un desarrollador ajeno al proyecto quisiera leer el código encontraría una comprensión sencilla del mismo dada su estructura y consistencia.

Brayan Alejandro Muñoz Pérez

El código que revisé está bien organizado y claro, lo que hace que sea bastante fácil de entender y mantener. Las funciones y variables tienen nombres descriptivos, así que no hace falta depender de comentarios largos para saber qué hace cada parte. La estructura modular y la separación de responsabilidades están bien logradas, lo que ayuda a mantener

el código limpio y muy fácil de modificar. Estoy de acuerdo con mi compañero Adrían en que si tuviese que trabajar en una parte que no escribí, sería fácil de entender y hacer cambios sin ningún problema. En general, siguen bien los principios de Clean Code que aprendimos en clase.