



Universidad Nacional de Colombia - sede Bogotá  
Facultad de Ingeniería  
Departamento de Sistemas e Industrial  
Curso: Ingeniería de Software 1 (2016701)

**Profesor: Oscar Eduardo Alvarez Rodriguez**

**Estudiante(s):**

Nelson Ivan Castellanos Betancourt  
Angel Santiago Avendaño Cañon  
David Sebastian Hurtado Sanchez  
Nicolas Zuluaga Galindo

### Clean Code

El código del proyecto se dividió en 3 partes principales y 1 un script sencillo:

- Landing: el landing principal que presenta los servicios y contacto con el centro médico, abierto al público y expuesto a través de internet. Fue implementado en ReactJS con Vite como servidor de desarrollo local.
- Frontend interno: código con la UI que permite la gestión de pacientes y recursos. También se desarrolló usando ReactJS con Vite como servidor de desarrollo local.
- Backend DRF: backend implementado con Django Rest Framework, un framework de Python para el desarrollo de API REST, cuenta con ORM integrado que facilita las consultas con la base de datos.
- Notifications Consumer: script en python encargado de consumir las notificaciones. El backend actúa como Producer y este script se encarga de consumir las notificaciones generadas, enviándolas a los destinatarios.

### SELECCIÓN Y CONFIGURACIÓN DE LINTERS

Pull Request asociada: [add linters and github actions to check them](#)

Para landing y frontend interno se utiliza **ESLint** como linter y **Prettier** como formatter. A continuación se detallan las reglas implementadas; se utilizan las mismas configuraciones en ambos (ver [eslint.conf.js](#)).

- **Configuración general:**
  - Soporte para ECMAScript 2020: Se define esta regla para asegurar la compatibilidad con las últimas características del lenguaje JS.
  - Uso de JSX: se habilitaron las características de ECMAScript para soportar JSX, el cuál es fundamental para desarrollar con ReactJS.
- **Reglas de JavaScript:** reglas recomendadas de eslint/js, aplicadas para garantizar las mejores prácticas en código JS. Estas reglas incluyen:
  - Detección de errores comunes
  - Mejores prácticas de programación
  - Consistencia en el uso de variables y estructuras de lenguaje.

- **Reglas para React:** se incluyeron reglas recomendadas de eslint-plugin-react, que influyen:
  - Propiedades de componentes: todos los componentes que reciben argumentos requieren que tengan definidos sus prop-types.
  - Uso de JSX: se desactivan reglas obsoletas, ya no se requiere el import React from "react".
  - Restricción de componentes anidados inestables.
  - El plugin eslint-plugin-react-hooks para garantizar el uso correcto de Hooks: verifica el correcto uso de dependencias en useEffect, restringe el uso de hooks fuera del ámbito de componentes funcionales.
- **Reglas de accesibilidad:** se utilizó el plugin eslint-plugin-jsx-a11y para mejorar la accesibilidad del código. Se incluyen reglas como:
  - Uso obligatorio del atributo alt en elementos <img>
  - Validación de enlaces <a> para que sean accesibles
  - Manejo de eventos de click con soporte de teclado
- **Reglas de formateo con Prettier:** se utilizó el plugin eslint-plugin-prettier para garantizar el formato consistente del código. Se establecieron las siguientes reglas:
  - Uso de comillas dobles
  - Inclusión de punto y coma al final de las sentencias
  - Inclusión de comas finales en objetos y arrays (trailing comma)
  - Líneas de código con un máximo de 80 caracteres de longitud.
- **Reglas de importación:** se establece una regla para evitar el uso de rutas relativas.
  - Se prohibió el uso de rutas como ".../components/Menu".
  - Se exige el uso de alias "@/" para referirse a los archivos dentro de src/. Ejemplo "@/components/Menu".

Para el Backend en DRF y el script Notifications Consumer se utilizó **flake8** como linter y **black** como formatter. **Flake8** es una herramienta que combina verificaciones de estilo de código siguiendo el estándar PEP8, detección de errores con pyflakes y complejidad de código con mccabe. La configuración utilizada incluye los siguientes ajustes a las reglas predefinidas:

- **Exclusión de archivos:** se excluyeron directorios y archivos que no necesitan ser analizados o su análisis está pendiente (como tests.py) dado el avance del proyecto.
  - .gitignore, \_\_pycache\_\_, migrations, venv, env, docs (se excluye porque es una app que solamente genera las urls para ver la documentación en swagger y redoc).
- **Longitud máxima de línea:** se ajustó la regla que establece un límite máximo de caracteres por línea al valor de max-line-length = 100 en lugar del valor predeterminado de 79.
- **Reglas importantes que incluye flake8:**
  - **E501:** Líneas que exceden la longitud permitida.

- **E231:** Espaciado incorrecto después de una coma.
- **E302:** Se requiere al menos dos líneas en blanco entre funciones de nivel superior (aplica para clases).
- **F401:** Importaciones no utilizadas.
- **F821:** Uso de variables no definidas.

Por otra parte, **black** es una herramienta para el formateo automático dde código para python que se enfoca en establecer un estilo coherente en el proyecto. Sus principales características son:

- Líneas de hasta 88 caracteres por defecto (lo ajustamos a 100 con flake8)
- Uso consistente de comillas dobles en lugar de comillas sencillas
- Conversión de cadenas multilínea en estructuras claras para mejorar la legibilidad
- Indentación de 4 espacios, alineándose con el estándar PEP8.
- Eliminación automática de espacios innecesarios y formateo de listas y diccionarios para mayor legibilidad.

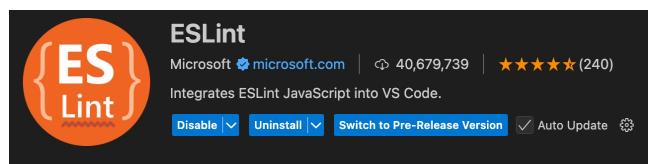
## IMPLEMENTANDO LOS LINTERS

Dividimos la implementación de los linters en dos secciones: la primera se enfoca en su aplicación dentro del entorno de desarrollo, mientras que la segunda verifica su correcto uso antes de fusionar cambios en la rama principal mediante GitHub Actions. A continuación el paso a paso de cada una:

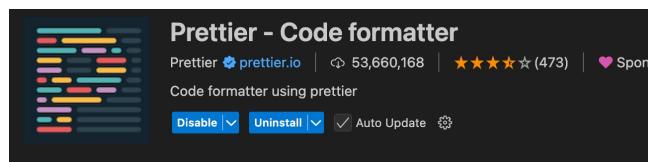
### Implementación en el entorno de desarrollo:

Se utilizaron las siguientes extensiones de VS Code para hacer uso de las configuraciones de los linters durante el desarrollo:

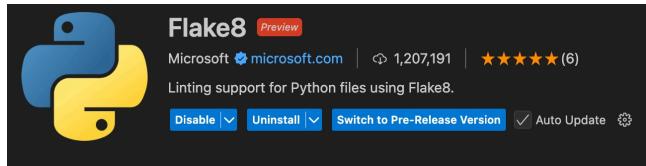
- ESLint: Analiza el código en tiempo real para detectar errores y advertencias, según las reglas que configuramos en el archivo `.eslint.config.json`. Además corrige automáticamente los errores al guardar el archivo.



- Prettier: Formate el código automáticamente cuándo se guarda el archivo. Esta integrado con ESLint y se puede utilizar para formateo en bloque en el menú contextual (Format document, Format selection).



- Flake8: analiza el código en tiempo real, detecta errores de sintaxis y mejores prácticas, como las ya mencionadas. Compatible con configuraciones personalizadas mediante el archivo .flake8. Resalta errores y advertencias en el editor de código.



Para los casos en los que se abría el repositorio completo en VS Code es necesario configurar las extensiones, pues estas buscan el archivo de configuración correspondiente en el directorio raíz del repositorio. Esta es la configuración que se usó en el archivo .vscode/settings.json:

A screenshot of a VS Code terminal window displaying a JSON configuration file. The code is as follows:

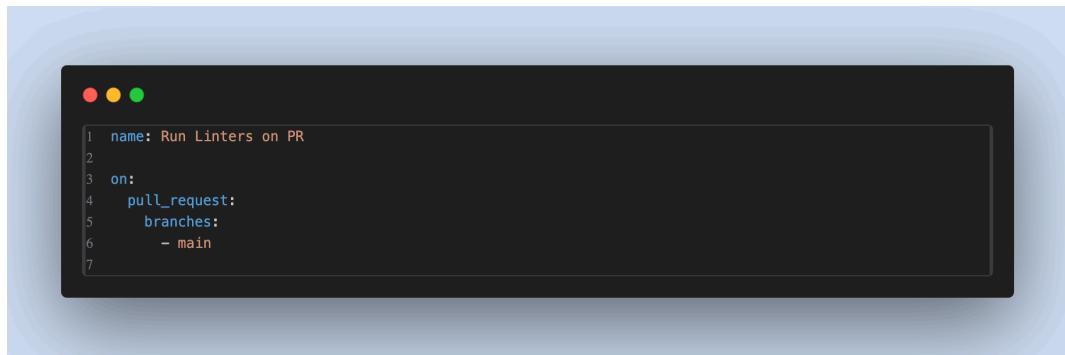
```
1 {
2     "eslint.options": {
3         "cwd": "${workspaceFolder}/Proyecto/landing/"
4     },
5     "eslint.validate": ["javascript", "javascriptreact"],
6     "editor.codeActionsOnSave": {
7         "source.fixAll.eslint": "explicit"
8     },
9     "editor.formatOnSave": true,
10    "flake8.cwd": "${workspaceFolder}/Proyecto/django-backend/centro_medico/",
11    "python.analysis.extraPaths": [
12        "${workspaceFolder}/Proyecto/python_shared"
13    ]
14 }
```

### Implementación de Github Actions:

El archivo [.github/workflows/lint.yml](#) define un flujo de trabajo en Github Actions para ejecutar los linters configurados en los distintos directorios del repositorio, esto con el objetivo de garantizar la calidad del código antes de fusionarlo en la rama principal del repositorio.

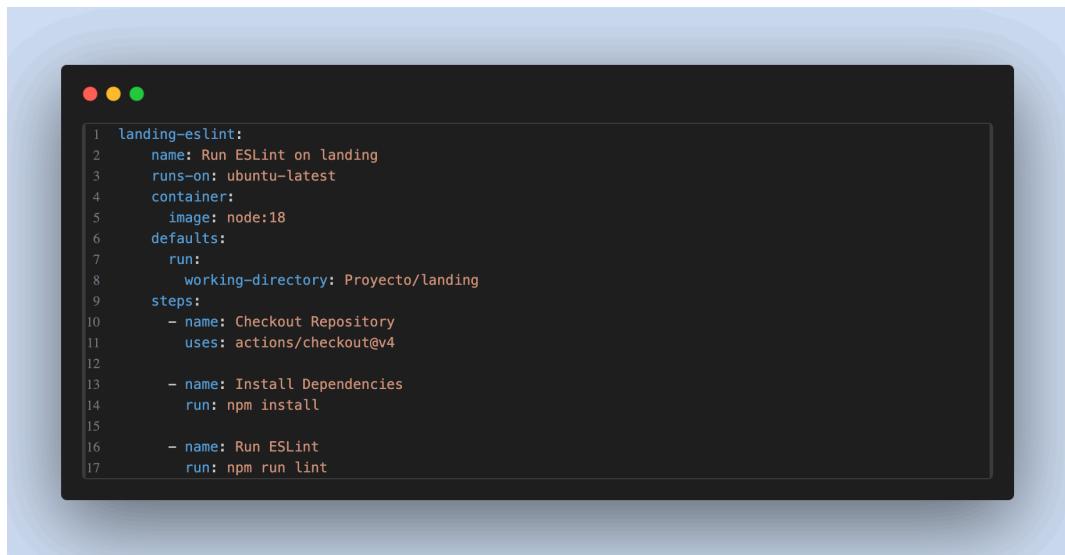
Desglose de la configuración:

- Se define que el flujo de trabajo se active automáticamente cuando se haga pull request a la rama main.



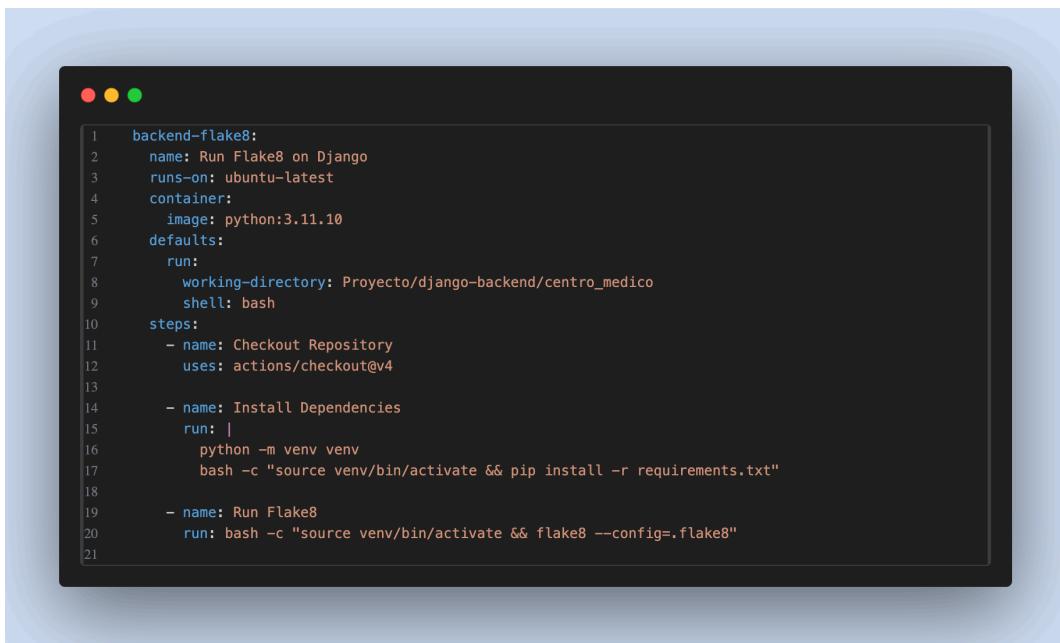
```
1 name: Run Linters on PR
2
3 on:
4   pull_request:
5     branches:
6       - main
7
```

- El flujo de trabajo de esta G. Action está compuesto por tres jobs, cada uno con un contenedor específico y una imagen específica para facilitar la instalación de dependencias, que además permite su ejecución en paralelo.
  1. **Job landing-eslint:** se ejecuta en un contenedor con una imagen de Node.js versión 18 y sigue los siguientes pasos 1. Se mueve al directorio raíz del correspondiente 2. Clona el repositorio y obtiene los cambios de la rama que desencadenó el pull request 3. Instala las dependencias 4. Ejecuta el comando npm run lint.



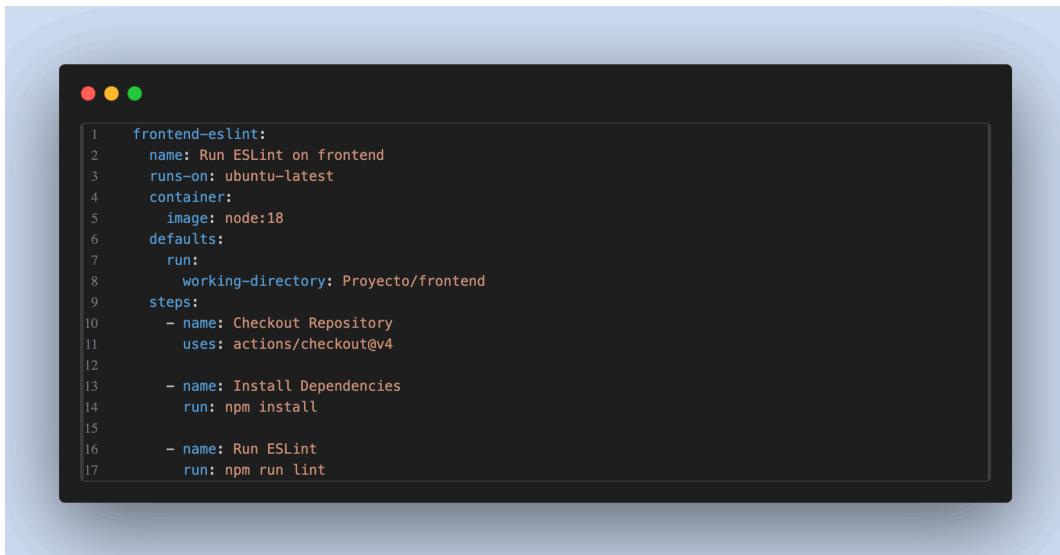
```
1 landing-eslint:
2   name: Run ESLint on landing
3   runs-on: ubuntu-latest
4   container:
5     image: node:18
6   defaults:
7     run:
8       working-directory: Proyecto/landing
9   steps:
10    - name: Checkout Repository
11      uses: actions/checkout@v4
12
13    - name: Install Dependencies
14      run: npm install
15
16    - name: Run ESLint
17      run: npm run lint
```

2. **Job backend-flake8:** se ejecuta en un contenedor con una imagen de python 3.11.10. Sigue estos pasos: 1. Una vez corriendo el contenedor se mueve al directorio del backend 2. Clona el repositorio y obtiene los cambios más recientes de rama que desencadenó el pull request. 3. Crea un entorno virtual de python e instala las dependencias 4. Posteriormente ejecuta el comando flake8 --config=.flake8.



```
1 backend-flake8:
2   name: Run Flake8 on Django
3   runs-on: ubuntu-latest
4   container:
5     image: python:3.11.10
6   defaults:
7     run:
8       working-directory: Proyecto/django-backend/centro_medico
9       shell: bash
10  steps:
11    - name: Checkout Repository
12      uses: actions/checkout@v4
13
14    - name: Install Dependencies
15      run: |
16        python -m venv venv
17        bash -c "source venv/bin/activate && pip install -r requirements.txt"
18
19    - name: Run Flake8
20      run: bash -c "source venv/bin/activate && flake8 --config=.flake8"
21
```

3. **Job frontend-eslint:** del mismo modo que el **job landing-eslint**, se ejecuta en un contenedor con una imagen de Node.js versión 18 y sigue los siguientes pasos 1. Se mueve al directorio raíz del correspondiente 2. Clona el repositorio y obtiene los cambios de la rama que desencadenó el pull request 3. Instala las dependencias 4. Ejecuta el comando `npm run lint`.



```
1 frontend-eslint:
2   name: Run ESLint on frontend
3   runs-on: ubuntu-latest
4   container:
5     image: node:18
6   defaults:
7     run:
8       working-directory: Proyecto/frontend
9   steps:
10    - name: Checkout Repository
11      uses: actions/checkout@v4
12
13    - name: Install Dependencies
14      run: npm install
15
16    - name: Run ESLint
17      run: npm run lint
```

## EJECUTANDO LOS LINTERS

## Ejecutando ESLint:

En el screenshot se muestra que al ejecutar `npm run lint` (se desactivaron las extensiones porque estas corregía automáticamente los errores) se encontraron 118 errores, de los cuales 2 eran advertencias. La mayoría de estos estaban asociados al uso de comillas, algunos sobre la falta de accesibilidad en imágenes y el manejo de eventos click con teclado para elementos que no son botones. Hubo otros errores que mostraron la ausencia de saltos de línea, y otros más sobre el uso incorrecto de hooks como `useEffect`. La mayoría de estos se solucionaron con el comando `npm run lint --fix` y no requirieron de intervención directa de nuestra parte.

## Ejecutando flake8:

```

~/dev/IngenieriaSoftwareI/Proyecto/django-backend/centro_medico git:(linters)±5 (0.315s)
flake8 .

./docs/tests.py:1:1: F401 'django.test.TestCase' imported but unused
./docs/urls.py:1:1: F401 'django.contrib.admin' imported but unused
./docs/urls.py:2:1: F401 'django.urls.include' imported but unused
./docs/urls.py:17:80: E501 line too long (88 > 79 characters)
./docs/views.py:1:1: F401 'django.shortcuts.render' imported but unused
./notifications/admin.py:1:1: F401 'django.contrib.admin' imported but unused
./notifications/models.py:1:1: F401 'django.db.models' imported but unused
./notifications/tests.py:1:1: F401 'django.test.TestCase' imported but unused
./notifications/viewssets.py:1:1: F401 'django.shortcuts.render' imported but unused
./occupancy/admin.py:1:1: F401 'django.contrib.admin' imported but unused
./occupancy/models.py:32:80: E501 line too long (84 > 79 characters)
./occupancy/models.py:48:80: E501 line too long (85 > 79 characters)
./occupancy/models.py:61:80: E501 line too long (97 > 79 characters)
./occupancy/models.py:71:80: E501 line too long (85 > 79 characters)
./occupancy/models.py:75:80: E501 line too long (86 > 79 characters)
./occupancy/models.py:82:80: E501 line too long (92 > 79 characters)
./occupancy/serializers.py:2:80: E501 line too long (88 > 79 characters)
./occupancy/serializers.py:15:80: E501 line too long (84 > 79 characters)
./occupancy/serializers.py:26:80: E501 line too long (84 > 79 characters)
./occupancy/serializers.py:30:80: E501 line too long (84 > 79 characters)
./occupancy/serializers.py:50:80: E501 line too long (84 > 79 characters)
./occupancy/tests.py:1:1: F401 'django.test.TestCase' imported but unused
./occupancy/urls.py:12:80: E501 line too long (81 > 79 characters)
./occupancy/urls.py:13:80: E501 line too long (81 > 79 characters)
./occupancy/urls.py:14:80: E501 line too long (84 > 79 characters)
./occupancy/urls.py:16:80: E501 line too long (81 > 79 characters)
./occupancy/viewssets.py:3:80: E501 line too long (88 > 79 characters)
./occupancy/viewssets.py:42:80: E501 line too long (81 > 79 characters)
./patients/admin.py:1:1: F401 'django.contrib.admin' imported but unused
./patients/models.py:10:80: E501 line too long (81 > 79 characters)
./patients/models.py:14:80: E501 line too long (83 > 79 characters)
./patients/models.py:16:80: E501 line too long (86 > 79 characters)
./patients/models.py:17:80: E501 line too long (80 > 79 characters)
./patients/models.py:41:80: E501 line too long (98 > 79 characters)
./patients/models.py:55:80: E501 line too long (84 > 79 characters)
./patients/models.py:76:80: E501 line too long (86 > 79 characters)
./patients/models.py:77:80: E501 line too long (80 > 79 characters)
./patients/models.py:84:80: E501 line too long (143 > 79 characters)
./patients/serializers.py:24:1: E402 module level import not at top of file
./patients/serializers.py:25:1: E402 module level import not at top of file
./patients/serializers.py:26:1: E402 module level import not at top of file
./patients/serializers.py:27:1: E402 module level import not at top of file
./patients/serializers.py:61:80: E501 line too long (87 > 79 characters)
./patients/serializers.py:65:80: E501 line too long (81 > 79 characters)
./patients/tests.py:1:1: F401 'django.test.TestCase' imported but unused
./patients/urls.py:7:80: E501 line too long (88 > 79 characters)
./patients/viewssets.py:16:80: E501 line too long (90 > 79 characters)
./pre_registrations/admin.py:1:1: F401 'django.contrib.admin' imported but unused
./pre_registrations/models.py:13:80: E501 line too long (81 > 79 characters)
./pre_registrations/models.py:20:80: E501 line too long (86 > 79 characters)
./pre_registrations/models.py:21:80: E501 line too long (80 > 79 characters)
./pre_registrations/models.py:32:80: E501 line too long (80 > 79 characters)
./pre_registrations/models.py:35:80: E501 line too long (83 > 79 characters)
./pre_registrations/models.py:37:80: E501 line too long (86 > 79 characters)
./pre_registrations/models.py:38:80: E501 line too long (80 > 79 characters)
./pre_registrations/models.py:76:80: E501 line too long (86 > 79 characters)
./pre_registrations/models.py:77:80: E501 line too long (80 > 79 characters)
./pre_registrations/serializers.py:66:80: E501 line too long (84 > 79 characters)
./pre_registrations/tests.py:1:1: F401 'django.test.TestCase' imported but unused
./pre_registrations/urls.py:7:80: E501 line too long (84 > 79 characters)
./pre_registrations/viewssets.py:2:1: F401 'rest_framework.permissions.IsAuthenticated' imported but unused
./protocols/admin.py:1:1: F401 'django.contrib.admin' imported but unused
./protocols/tests.py:1:1: F401 'django.test.TestCase' imported but unused
./protocols/views.py:1:1: F401 'django.shortcuts.render' imported but unused
./users/admin.py:1:1: F401 'django.contrib.admin' imported but unused
./users/backends.py:20:80: E501 line too long (82 > 79 characters)
./users/models.py:1:1: F401 'django.db.models' imported but unused
./users/serializers.py:14:80: E501 line too long (87 > 79 characters)
./users/serializers.py:18:80: E501 line too long (87 > 79 characters)
./users/tests.py:1:1: F401 'django.test.TestCase' imported but unused
./users/views.py:5:80: E501 line too long (80 > 79 characters)
./users/views.py:40:80: E501 line too long (86 > 79 characters)
./users/views.py:44:80: E501 line too long (84 > 79 characters)
./users/views.py:57:80: E501 line too long (84 > 79 characters)

★ Fix line length and unused imports in the Python files. ⌘ Enter

```

~/dev/IngenieriaSoftwareI/Proyecto/django-backend/centro\_medico git:(linters)±5 ✘ Agent Mode

Este screenshot muestra el resultado de ejecutar flake8 . (se desactivaron las extensiones porque estas corregía automáticamente los errores). Se encontraron 89 errores, muchos sobre líneas de código demasiado largas que superan los 79 caracteres que tiene definidos por defecto flake8, por lo que ajustamos el límite máximo a 100. También resaltó otros errores cómo la importación de librerías no usadas, e importaciones debajo de código ejecutable cuando el estándar dice que deben hacerse en el inicio del archivo.

```
.env ~/dev/IngenieriaSoftwareI/Proyecto/django-backend/centro_medico git:(main)±13 (0..324s)
black .

reformatted ~/dev/IngenieriaSoftwareI/Proyecto/django-backend/centro_medico/centro_medico/asgi.py
reformatted ~/dev/IngenieriaSoftwareI/Proyecto/django-backend/centro_medico/centro_medico/wsgi.py
reformatted ~/dev/IngenieriaSoftwareI/Proyecto/django-backend/centro_medico/centro_medico/renderers.py
reformatted ~/dev/IngenieriaSoftwareI/Proyecto/django-backend/centro_medico/centro_medico/manage.py
reformatted ~/dev/IngenieriaSoftwareI/Proyecto/django-backend/centro_medico/centro_medico/urls.py
reformatted ~/dev/IngenieriaSoftwareI/Proyecto/django-backend/centro_medico/centro_medico/docs/urls.py
reformatted ~/dev/IngenieriaSoftwareI/Proyecto/django-backend/centro_medico/centro_medico/occupancy/urls.py
reformatted ~/dev/IngenieriaSoftwareI/Proyecto/django-backend/centro_medico/centro_medico/patients/urls.py
reformatted ~/dev/IngenieriaSoftwareI/Proyecto/django-backend/centro_medico/centro_medico/patients/viewsets.py
reformatted ~/dev/IngenieriaSoftwareI/Proyecto/django-backend/centro_medico/centro_medico/occupancy/serializers.py
reformatted ~/dev/IngenieriaSoftwareI/Proyecto/django-backend/centro_medico/pre_registrations/urls.py
reformatted ~/dev/IngenieriaSoftwareI/Proyecto/django-backend/centro_medico/occupancy/viewsets.py
reformatted ~/dev/IngenieriaSoftwareI/Proyecto/django-backend/centro_medico/protocols/urls.py
reformatted ~/dev/IngenieriaSoftwareI/Proyecto/django-backend/centro_medico/pre_registrations/viewsets.py
reformatted ~/dev/IngenieriaSoftwareI/Proyecto/django-backend/centro_medico/centro_medico/settings.py
reformatted ~/dev/IngenieriaSoftwareI/Proyecto/django-backend/centro_medico/centro_medico/occupancy/models.py
reformatted ~/dev/IngenieriaSoftwareI/Proyecto/django-backend/centro_medico/patients/serializers.py
reformatted ~/dev/IngenieriaSoftwareI/Proyecto/django-backend/centro_medico/users/viewsets.py
reformatted ~/dev/IngenieriaSoftwareI/Proyecto/django-backend/centro_medico/protocols/viewsets.py
reformatted ~/dev/IngenieriaSoftwareI/Proyecto/django-backend/centro_medico/protocols/serializers.py
reformatted ~/dev/IngenieriaSoftwareI/Proyecto/django-backend/centro_medico/users/backends.py
reformatted ~/dev/IngenieriaSoftwareI/Proyecto/django-backend/centro_medico/users/urls.py
reformatted ~/dev/IngenieriaSoftwareI/Proyecto/django-backend/centro_medico/protocols/models.py
reformatted ~/dev/IngenieriaSoftwareI/Proyecto/django-backend/centro_medico/patients/models.py
reformatted ~/dev/IngenieriaSoftwareI/Proyecto/django-backend/centro_medico/pre_registrations/serializers.py
reformatted ~/dev/IngenieriaSoftwareI/Proyecto/django-backend/centro_medico/users/serializers.py
reformatted ~/dev/IngenieriaSoftwareI/Proyecto/django-backend/centro_medico/pre_registrations/models.py
reformatted ~/dev/IngenieriaSoftwareI/Proyecto/django-backend/centro_medico/users/views.py
reformatted ~/dev/IngenieriaSoftwareI/Proyecto/django-backend/centro_medico/notifications/urls.py
reformatted ~/dev/IngenieriaSoftwareI/Proyecto/django-backend/centro_medico/notifications/viewsets.py
reformatted ~/dev/IngenieriaSoftwareI/Proyecto/django-backend/centro_medico/notifications/serializers.py
reformatted ~/dev/IngenieriaSoftwareI/Proyecto/django-backend/centro_medico/notifications/models.py
reformatted ~/dev/IngenieriaSoftwareI/Proyecto/django-backend/centro_medico/centro_medico/renderers.py
reformatted ~/dev/IngenieriaSoftwareI/Proyecto/django-backend/centro_medico/notifications/viewsets.py
reformatted ~/dev/IngenieriaSoftwareI/Proyecto/django-backend/centro_medico/patients/serializers.py
reformatted ~/dev/IngenieriaSoftwareI/Proyecto/django-backend/centro_medico/patients/models.py

All done! ✨ 🎉 ✨
28 files reformatted, 51 files left unchanged.

.env ~/dev/IngenieriaSoftwareI/Proyecto/django-backend/centro_medico git:(main)±13 ⓘ Agent Mode
python manage.py runserver ✨
```

Para el formateo de código se utilizó el comando `black .` arreglo un total de 28 archivos, mejorando la legibilidad del código.

### Ejecutando Github Actions en local:

Para la ejecución del flujo de trabajo de github actions en local se utilizó la herramienta `act` disponible para macOS. Esta nos permitió simular el envío de una pull request y ejecutar los contenedores, que comprueban que se realicen buenas prácticas de código.

Este es el resultado simulado cuándo se hace una pull request que no cumple las reglas establecidas en los linters:

```
Would you like me to help you set up the correct command for your specific use case?

~/dev/IngenieriaSoftwareI git:(main)±2 (54.485s)
act pull_request -j landing-eslint --container-architecture linux/amd64
[INFO] Using docker host 'unix:///var/run/docker.sock', and daemon socket 'unix:///var/run/docker.sock'
[Run Linters on PR/Run ESLint on landing] 🚫 Start Image:node:18
[Run Linters on PR/Run ESLint on landing] 🚫 docker create image=node:18 platform:linux/amd64 entrypoint=['tail' '-f' '/dev/null'] cmd=[] network='host'
[Run Linters on PR/Run ESLint on landing] 🚫 docker run image=node:18 platform:linux/amd64 entrypoint=['tail' '-f' '/dev/null'] cmd=[] network='host'
[Run Linters on PR/Run ESLint on landing] 🚫 docker exec cmd=[node --no-warnings -e console.log(process.execPath)] user= workdir=
[Run Linters on PR/Run ESLint on landing] 🚫 Run Main Checkout Repository
[Run Linters on PR/Run ESLint on landing] 🚫 cp -r /Users/ivan/dev/IngenieriaSoftwareI/. dst=/Users/ivan/dev/IngenieriaSoftwareI
[Run Linters on PR/Run ESLint on landing] ✅ Success - Main Checkout Repository
[Run Linters on PR/Run ESLint on landing] ✅ Run Main Install Dependencies
[Run Linters on PR/Run ESLint on landing] 🚫 docker exec cmd=[sh -e /var/run/act/workflow/1.sh] user= workdir=Proyecto/landing
[Run Linters on PR/Run ESLint on landing] 🚫 npm warn EBADENGINE Unsupported engine
[Run Linters on PR/Run ESLint on landing] 🚫 npm WARN EADDENGINE peer dep missing in bundle: react-router@7.1.5
[Run Linters on PR/Run ESLint on landing] 🚫 npm warn EBADENGINE required: { node: '^>2.0.0' }
[Run Linters on PR/Run ESLint on landing] 🚫 npm warn EBADENGINE current: { node: '^10.29.6', npm: '^10.8.2' }
[Run Linters on PR/Run ESLint on landing] 🚫 npm warn EBADENGINE
added 246 packages, and audited 247 packages in 7s
109 packages are looking for funding
  run `npm fund` for details
found 0 vulnerabilities
[Run Linters on PR/Run ESLint on landing] ✅ Success - Main Install Dependencies
[Run Linters on PR/Run ESLint on landing] ✅ Run Main Run ESLint
[Run Linters on PR/Run ESLint on landing] 🚫 docker exec cmd=[sh -e /var/run/act/workflow/2.sh] user= workdir=Proyecto/landing
> landing@0.0.0 lint
> eslint .

/Users/ivan/dev/IngenieriaSoftwareI/Proyecto/landing/src/main.jsx
3:60  error  Insert ';' prettier/prettier
  ✖ 1 problem (1 error, 0 warnings)
  1 error (1 errors, 0 warnings potentially fixable with the '--fix' option.

[Run Linters on PR/Run ESLint on landing] ❌ Failure - Main Run ESLint
[Run Linters on PR/Run ESLint on landing] exitcode: '1', failure
[Run Linters on PR/Run ESLint on landing] ❌ Job failed
Error: Job 'Run ESLint on landing' failed

Fix the ESLint error in main.jsx ⚡ Enter

~/dev/IngenieriaSoftwareI git:(main)±14 [x] Agent Mode
act pull_request -j landing-eslint --container-architecture linux/amd64 --fix ↵
```

Este es otro resultado simulado cuándo la pull request cumple las condiciones:

```
~/dev/IngenieriaSoftwareI git:(main)±22 (28.882s)
act pull_request --container-architecture linux/amd64
| Downloading drf_spectacular-0.28.0-py3-none-any.whl (103 kB)
|   103.9/103.9 kB 1.2 MB/s eta 0:00:00
| Downloading flake8-7.1.1-py2.py3-none-any.whl (57 kB)
|   57.7/57.7 kB 646.2 kB/s eta 0:00:00
| Downloading inflection-0.5.1-py2.py3-none-any.whl (9.5 kB)
| Downloading jsonschema-4.23.0-py3-none-any.whl (88 kB)
|   88.5/88.5 kB 7.3 MB/s eta 0:00:00
| Downloading jsonschema_specifications-2024.10.1-py3-none-any.whl (18 kB)
| Downloading mccabe-0.7.0-py2.py3-none-any.whl (7.3 kB)
| Downloading pycodestyle-2.12.1-py2.py3-none-any.whl (31 kB)
| Downloading pyflakes-3.2.0-py2.py3-none-any.whl (62 kB)
|   62.7/62.7 kB 746.9 kB/s eta 0:00:00
| Downloading PyYAML-6.0.2-cp311-cp311-manylinux_2_17_x86_64.manylinux2014.x86_64.whl (762 kB)
|   763.0/763.0 kB 7.7 MB/s eta 0:00:00
[Run Linters on PR/Run ESLint on frontend] ✅ Success - Main Run ESLint
| Downloading referencing-0.36.2-py3-none-any.whl (26 kB)
[Run Linters on PR/Run ESLint on frontend] Cleaning up container for job Run ESLint on frontend
| Downloading rpds-py-0.22.3-cp311-cp311-manylinux_2_17_x86_64.manylinux2014.x86_64.whl (381 kB)
[Run Linters on PR/Run ESLint on frontend] ❌ Job succeeded
|   381.3/381.3 kB 4.9 MB/s eta 0:00:00
| Downloading sqlparse-0.5.3-py3-none-any.whl (44 kB)
|   44.4/44.4 kB 6.8 MB/s eta 0:00:00
| Downloading typing_extensions-4.12.2-py3-none-any.whl (37 kB)
| Downloading tzdata-2025.1-py2.py3-none-any.whl (346 kB)
|   346.8/346.8 kB 4.6 MB/s eta 0:00:00
| Downloading uritemplate-4.1.1-py2.py3-none-any.whl (10 kB)
| Installing collected packages: uritemplate, tzdata, typing_extensions, sqlparse, rpds-py, PyYAML, pyflakes, pycodestyle, mccabe, inflection, flake8, Django, jsonschema-specifications, djangorestframework, jsonschema, drf-spectacular
[Run Linters on PR/Run ESLint on landing] ✅ Success - Main Run ESLint
[Run Linters on PR/Run ESLint on landing] Cleaning up container for job Run ESLint on landing
[Run Linters on PR/Run ESLint on landing] ❌ Job succeeded
| Successfully installed Django-5.1.5 PyYAML-6.0.2 asgiref-3.8.1 attrs-25.1.0 django-environ-0.12.0 djangorestframework-3.15.2 drf-spectacular-4.23.0 jsonschema-specifications-2024.10.1 mccabe-0.7.0 pycodestyle-2.12.1 pyflakes-3.2.0 referencing-0.36.2 rpds-py-0.22.3 sqlparse-0.1 uritemplate-4.1.1
|
| [notice] A new release of pip is available: 24.0 → 25.0
| [notice] To update, run: pip install --upgrade pip
[Run Linters on PR/Run Flake8 on Django] ✅ Success - Main Install Dependencies
[Run Linters on PR/Run Flake8 on Django] ✅ Run Main Run Flake8
[Run Linters on PR/Run Flake8 on Django] 🚫 docker exec cmd=[bash --noprofile --norc -e -o pipefail /var/run/act/workflow/2.sh] user=o
[Run Linters on PR/Run Flake8 on Django] ✅ Success - Main Run Flake8
[Run Linters on PR/Run Flake8 on Django] Cleaning up container for job Run Flake8 on Django
[Run Linters on PR/Run Flake8 on Django] ❌ Job succeeded

Toggle on / off natural language detection in the command line input. ⚡

~/dev/IngenieriaSoftwareI claudie 3.5 sonnet ✘ 1 block attached
⚡ Ask in natural language
```

## OPINIONES DEL CÓDIGO DE LOS INTEGRANTES DEL GRUPO

### (Sebastian Hurtado)

Aunque no tengo un conocimiento profundo sobre los lenguajes de programación utilizados en este proyecto, trabajar con el código desarrollado en conjunto con mis compañeros ha sido una experiencia sumamente enriquecedora. Gracias a la estructura clara y la aplicación de buenas prácticas de desarrollo, he podido comprender la lógica detrás de muchas implementaciones, lo que me ha permitido aprender considerablemente. La organización del código y el uso de herramientas como ESLint, Prettier y Flake8 han facilitado la legibilidad y han garantizado que el proyecto mantenga estándares de calidad consistentes. Además, la división del código en distintas secciones, como el frontend en ReactJS, el backend con Django Rest Framework y el sistema de notificaciones en Python, me ha permitido familiarizarme con distintas tecnologías y comprender cómo interactúan entre sí. Aprecio mucho el esfuerzo de mis compañeros en documentar y estructurar su código de manera comprensible, ya que esto ha hecho que mi proceso de aprendizaje sea más fluido y que cualquier modificación o ajuste sea más fácil de realizar. Sin duda, esta experiencia ha sido clave para mejorar mis habilidades y ampliar mi conocimiento en el desarrollo de software.

### Nelson Ivan Castellanos Betancourt

Trabajar en el desarrolló de la aplicación ha sido un proceso interesante. Uno de los puntos más destacables es el uso de los linters, los cuales han facilitado bastante la visualización y organización del código. Gracias a eso el código que he visto de mis compañeros es consistente, fácil de leer y de comprender.

En general tengo un buen entendimiento de las tareas en las que trabajan mis compañeros. La comunicación puede mejorar, podríamos usar más herramientas de gestión de tareas como Trello, pero para la constancia que requiere el proyecto no me parece algo tan grave, whatsapp es suficiente.

Ya tengo experiencia en los lenguajes y frameworks utilizados, por lo que el código de mis compañeros es mayormente legible, no tendría problemas para modificarlo si hace falta. Algunos se apoyan de la AI para asistir el desarrolló en el proyecto, lo cuál ha contribuido positivamente en el progreso del proyecto, sin un impacto negativo significativo en la legibilidad del código. Por el contrario, desde mi punto de vista en ocasiones hay un exceso de comentarios en el código. Entiendo que para alguien que está empezando con una tecnología es un apoyo importante, pero para alguien con unas bases más fundamentadas en tales tecnologías es algo diferente, comprender algunas sentencias de código es puramente tener conocimiento de la tecnología y los comentarios sobran.

En general mis compañeros han demostrado ser capaces de trabajar con tecnologías con las que no han tenido contacto anteriormente, el código que escriben funciona y es comprensible.

### Nicolás Zuluaga Galindo

La revisión del código de mis compañeros ha sido una experiencia muy enriquecedora, especialmente porque no tenía experiencia con los frameworks utilizados (Django Rest y

React). A pesar de eso, gracias a la experticia de un compañero en el desarrollo web, se organizó el proyecto en subproyecto que integrante del grupo pueda desarrollar, facilitando el avance y disminuyendo los errores. A su vez, esto disminuyó al mínimo la necesidad de modificar código de otro compañero; cuando es requerido pido una explicación para poder comprender su funcionamiento. A pesar de los desafíos, considero que el grupo ha mantenido buenas prácticas de desarrollo, lo que permitido que la calidad del proyecto sea constante.

**Angel Santiago Avendaño Cañon**

Tras realizar una lectura detallada y cuidadosa del código del proyecto, se puede detallar los diferentes niveles de experiencia al momento de escribir, incluyendo elementos que aportan de manera significativa a la mejora en la escritura de código limpio. Dada mi poca experiencia con Django REST Framework, ver el código de mis compañeros ayudó a tener ejemplos sobre los cuales trabajar y no tener que aprender desde cero. Se notan también las diferencias en la experiencia del uso de los diferentes frameworks, lenguajes y convenciones, a medida que la base de código del proyecto avanza, se ha notado como cada uno se adapta al lenguaje y a diferentes prácticas comunes para las herramientas usadas. Hay algunas partes del código que por su estructura sugieren el uso de herramientas de inteligencia artificial, y adaptación posterior a la base de código de la aplicación. Algo que ha sido usado de buena manera para dar un boost a la productividad del desarrollo.