Costantín, María Belén

Taller: Integración y Entrega continua
Materia: Ingeniería y Calidad de Software
Fecha de entrega: 12 de Junio de 2025

Introducción

La integración continua (CI) es una práctica muy utilizada en el desarrollo de software porque permite subir cambios de código de forma frecuente a un repositorio, donde automáticamente se ejecutan pruebas y análisis que ayudan a detectar errores rápido. De esta manera, se mejora la calidad del código y se acelera el proceso de entrega. En este trabajo práctico el objetivo fue aplicar esos conceptos en un proyecto real, usando herramientas como GitHub Actions, SonarCloud y Render.

Desarrollo del trabajo

El proyecto que desarrollé fue un turnero muy simple. Un sistema que permite generar turnos según el tipo de atención. Elegí Python por ser el lenguaje con el que más familiarizada estoy y conocer mis limitaciones con otros lenguajes. La lógica del sistema está encapsulada en una clase que mantiene un contador por tipo de turno, y se desarrolló una interfaz web simple con HTML y botones conectados mediante fetch a un backend con Flask, lo cual permitió simular un uso real en la web. El proyecto fue subido a GitHub y se configuró integración continua con GitHub Actions. Además, se utilizó SonarCloud para análisis de código y Render como entorno de entrega web.

Herramientas utilizadas:

- Repositorio: GitHub.
- Servidor de IC: GitHub Actions configurado con un archivo .yml
- Entorno de desarrollo local: Trabajé en Visual Studio Code, ejecutando pytest y app.py para ver cómo funcionaba todo localmente.
- Build local: Incluye la ejecución de tests (pytest) y el servidor (python app.py).
- Despliegue en la web: Utilicé Render para poder ver la app funcionando online.
- Inspección de código: SonarCloud

Pruebas automatizadas:

Se implementaron pruebas para:

- Verificar el formato correcto del turno generado.
- Probar que se incrementan los turnos según el tipo.
- Verificar que se rechacen turnos inválidos.
- Incluir fallo intencional para validar el pipeline.

Despliegue:

Para que Render funcionara correctamente, agregué un archivo Procfile y definí las dependencias en requirements.txt. La interfaz web (index.html) hace pedidos a Flask usando fetch, y se muestra el turno generado junto con un contador por tipo.

Problemáticas y soluciones

- **Confusión entre herramientas:** Al principio me costó entender qué herramientas usar y para qué. Por ejemplo, intenté subir la app a GitHub Pages sin saber que no permite backend. También quise usar Tkinter pensando en una interfaz, pero luego entendí que eso solo sirve localmente, no en la web. Ahí fue cuando pasé a usar Flask y HTML.

- **Render vs GitHub Pages:** Render sí permite ejecutar Python, a diferencia de GitHub Pages. Migré la app ahí para poder mostrarla funcionando online.

- **Render se suspende:** Al ser gratuito, Render pone el proyecto en pausa si no se usa por unos minutos. Esto hace que tarde en cargar la primera vez (no más de 1 minuto). No impide su uso pero es una limitación.

- **Configuración de SonarCloud:** Fue una de las partes más técnicas. Tuve que generar varios tokens y agregar secrets en GitHub. Además, me dio un error por haber puesto mal el nombre de la organización en el secret. Lo solucioné copiando bien el identificador desde SonarCloud.

- **Entender el pipeline:** Me costó bastante entender cómo se ejecuta el flujo en GitHub Actions. Al principio me fallaban los pasos porque estaban mal ubicados o no estaban todos los comandos. Tuve que probar y ajustar varias veces.

- **Error en el test de calidad del código en Sonarcloud:** se integró cobertura de código (pytest-cov) y se configuró su reporte para SonarCloud. Se visualizó un 0% de cobertura al principio por errores de configuración, que fueron corregidos luego agregando --cov=turnero --cov-report=xml que es un archivo xml donde se guarda el porcentaje de cobertura de las pruebas al código.

Conclusión

Personalmente, este trabajo me resultó bastante desafiante porque si bien conozco muchas de estas herramientas desde el lado funcional, nunca las había aplicado desde el lado técnico. Hubo cosas que me costaron entender, aunque parezcan obvias: cómo usar GitHub Pages vs Render, cuándo usar Tkinter o Flask, cómo desplegar algo en la web y no solo en local, y sobre todo cómo se arma una pipeline real.

Mi rol actual en lo laboral está más orientado al análisis funcional y de procesos, y generalmente estoy más lejos del desarrollo en sí. Sin embargo, este trabajo me permitió acercarme a ese mundo, practicar con Git y Python (que elegí justamente por su simpleza) y entender mejor cómo se integran estas herramientas en un flujo real de trabajo. Fue una buena oportunidad para aprender haciendo, y me llevo nuevos conocimientos que sin duda me van a servir.

Referencias

- Fowler, M. (2006). *Continuous Integration*. Martin Fowler. https://martinfowler.com/articles/continuousIntegration.html

Costantín, María Belén

- GitHub. (2024). *Understanding GitHub Actions*. GitHub Docs. https://docs.github.com/en/actions

- SonarCloud. (2024). *Analyzing with GitHub Actions*. SonarCloud Documentation. https://sonarcloud.io/documentation/integrations/github/

- Render. (2024). *Deploy a Flask App*. Render Documentation. https://render.com/docs/deploy-flask

- Code with Ahsan. (2021, 21 julio).*Improving code quality with SonarCloud and GitHub*. YouTube. https://www.youtube.com/watch?v=od4RaBQOXrA

- Luis Vanegas. (2021, 26 agosto). *4. Integración y Configuración de Test y Coverage en Pipeline y Sonarcloud.* YouTube. https://www.youtube.com/watch?v=2nZnBIWIU34

- Luis Vanegas. (2021, 26 agosto). *¿Qué es la integración continua? CI/CD explicada con ejemplo*. YouTube. https://www.youtube.com/watch?v=mLse85mPdb4