Trabajo en grupo - Calidad en software generado por IA agéntica - Aplicación de división de cuentas

Rubén Oliva Zamora
Artur Vargas Carrión

2 de mayo de 2025

1. Motivación

La creciente adopción de herramientas de inteligencia artificial (IA) generativa, como GitHub Copilot o Cursor, en el ciclo de vida del desarrollo de software plantea nuevos desafíos y oportunidades. Si bien estas herramientas prometen acelerar el desarrollo, surge la pregunta fundamental sobre la calidad, mantenibilidad y testabilidad del código generado.

Este proyecto tiene como objetivo principal investigar precisamente esto: evaluar la calidad del código producido por IA agéntica en el contexto de una aplicación web full-stack completa, aunque sencilla. Buscamos aplicar sistemáticamente las técnicas de pruebas aprendidas para analizar la robustez, fiabilidad y las posibles debilidades del código generado por IA. La aplicación propuesta, un divisor de cuentas de restaurante basado en el reconocimiento de tickets, sirve como un caso de estudio práctico y autocontenido para esta investigación.

2. Descripción y funcionalidades del software a probar

El software a desarrollar y probar será una aplicación web full-stack denominada provisionalmente *TicketSplitter*. Su propósito es simplificar el proceso de dividir la cuenta de un restaurante entre varias personas.

Tecnologías:

- **Backend:** Python con el framework FastAPI. Se eligió Python por su robusto ecosistema para IA/visión por computador y FastAPI por su simplicidad y rendimiento.
- **Frontend:** HTML, CSS con el framework Bulma y JavaScript vanilla. Se busca simplicidad y claridad en la interfaz.
- **IA/Procesamiento:** Se utilizará alguna biblioteca de Python para el reconocimiento óptico de caracteres (OCR) (ej. Tesseract a través de 'pytesseract') para extraer texto del ticket. El procesamiento posterior para identificar ítems y precios será lógica implementada en Python junto a la IA agéntica.

Funcionalidades principales:

1. **Carga de ticket:** El usuario puede subir una imagen (foto) de un ticket de restaurante a través de la interfaz web.
2. **Procesamiento y extracción:** El backend recibe la imagen, utiliza OCR para extraer el texto, y aplica lógica para identificar los ítems consumidos y sus respectivos precios.
3. **Visualización y edición:** El frontend muestra la lista de ítems y precios extraídos, permitiendo al usuario corregir posibles errores del OCR o del procesamiento.

4. **Entrada de comensales:** El usuario introduce los nombres de las personas entre las que se dividirá la cuenta.

5. **Asignación de ítems:** El usuario asigna cada ítem (o fracciones del mismo) a una o varias personas.

6. **Cálculo y resultado:** La aplicación calcula automáticamente cuánto debe pagar cada persona.

7. **Visualización del reparto:** Se muestra un resumen claro del reparto final de la cuenta.

El desarrollo del código será realizado íntegramente por una IA agéntica (tipo Copilot) a partir de nuestros prompts. El núcleo de este trabajo consistirá en la ejecución y análisis exhaustivo de las pruebas de software por nuestra parte, aplicando las técnicas vistas en la asignatura sobre el código generado por la IA. Se procurará que la IA siga buenas prácticas y genere un diseño simple.

3. Prompt inicial para la generación de la aplicación

El desarrollo comenzará utilizando un prompt inicial dirigido a la IA generativa para establecer la estructura base del proyecto. Un ejemplo de este prompt sería:

```
Genera la estructura básica de un proyecto para una aplicación web full-stack
utilizando Python con FastAPI para el backend y HTML/CSS con el framework Bulma
y JavaScript vanilla para el frontend.
El propósito de la aplicación es procesar imágenes de tickets de restaurante y
ayudar a dividir la cuenta entre varias personas.

IMPORTANTE: La estructura del proyecto y el código inicial deben estar
diseñados pensando en la TESTEABILIDAD. Prepara la estructura para facilitar
la escritura y ejecución de pruebas unitarias, de integración y E2E, así como
la medición de la cobertura de código.

Crea los siguientes directorios y archivos iniciales, colocando el código
fuente principal dentro de un directorio 'src' para separarlo de los tests y
la configuración:
- Un directorio raíz del proyecto.
- Un directorio `src` para todo el código fuente de la aplicación:
    - `src/main.py`: Aplicación FastAPI principal.
    - `src/routers/`: Directorio para los routers (ej. `receipts.py`).
    - `src/core/`: Directorio para la lógica de negocio (ej. `processing.py`).
    - `src/models/`: Directorio para modelos Pydantic (ej. `schemas.py`).
- Un directorio `static` (fuera de `src`) para archivos CSS y JS.
- Un directorio `templates` (fuera de `src`) para plantillas HTML (ej. `index.html`).
    - `templates/index.html`: Debe incluir Bulma CSS desde CDN y estructura básica.
- Un directorio `tests` (fuera de `src`) para alojar todos los ficheros de prueba.
    - Incluye subdirectorios `tests/unit` y `tests/integration`.
    - Incluye un archivo vacío `tests/unit/test_example_unit.py` y
      `tests/integration/test_example_integration.py` como marcadores.
- Un archivo `requirements.txt` con las dependencias básicas y de prueba:
  fastapi, uvicorn, python-multipart, pydantic, Jinja2, pytest, pytest-cov, httpx.
- Un archivo `Makefile` básico con objetivos para:
    - Instalar dependencias: `install` (usando `pip install -r requirements.txt`).
```

```
    - Ejecutar todos los tests: ‘test‘ (usando ‘pytest tests/‘).
    - Calcular la cobertura de código: ‘coverage‘ (usando ‘pytest --cov=src tests/‘).

Asegúrate de que ‘src/main.py‘ configure la aplicación FastAPI, incluya el router
de ‘src/routers/receipts.py‘, configure el montaje de directorios estáticos y
plantillas Jinja2 (considerando que están fuera de ‘src‘). El endpoint
raíz ("/") debe renderizar ‘templates/index.html‘.
El archivo ‘src/routers/receipts.py‘ debe tener un endpoint POST vacío para
‘/upload/‘ que espere un archivo, diseñado de forma que sea fácilmente testeable
(ej., usando inyección de dependencias de FastAPI para la lógica de negocio).
El archivo ‘templates/index.html‘ debe tener un <input type="file"> y un
botón de envío.
Configura el ‘Makefile‘ para que los comandos ‘make install‘, ‘make test‘ y
‘make coverage‘ funcionen correctamente con la estructura de directorios ‘src‘
y ‘tests‘ propuesta.
```

Nota: Los prompts subsecuentes se definirán de forma iterativa durante el desarrollo para implementar las funcionalidades específicas, refinar el código y corregir errores, siempre guiados por la IA.

4. Tipos de pruebas a realizar

Aplicaremos un enfoque exhaustivo de pruebas, cubriendo distintos niveles y tipos vistos en la asignatura, para evaluar la calidad del código generado por IA:

- **Pruebas unitarias:**
  - **Objetivo:** verificar el correcto funcionamiento de las unidades de código más pequeñas y aisladas (funciones, métodos) en el backend, como las funciones de cálculo o validación. Se buscará una buena cobertura de código.
  - **Ejemplos:** comprobar funciones de cálculo de división, validación de datos de entrada, funciones auxiliares de procesamiento de texto post-OCR.
- **Pruebas de caja blanca:**
  - **Objetivo:** analizar la estructura interna del código generado por la IA, diseñando casos de prueba basados en caminos lógicos, condiciones y bucles.
  - **Ejemplos:** probar diferentes ramas condicionales en la lógica de asignación de ítems, verificar el manejo de bucles en el procesamiento de líneas del ticket. Evaluar la complejidad del código.
- **Pruebas de integración:**
  - **Objetivo:** comprobar la correcta interacción entre diferentes módulos o componentes del sistema.
  - **Ejemplos:** verificar la comunicación Frontend ->Backend (API) ->Lógica de procesamiento ->Respuesta al Frontend. Probar la integración con la biblioteca OCR. Se usarán clientes de prueba específicos del framework backend.
- **Pruebas de carga:**
  - **Objetivo:** evaluar el rendimiento y la estabilidad de la aplicación (principalmente el backend) bajo condiciones de carga simulada.

    - **Ejemplos:** simular múltiples usuarios subiendo tickets y calculando divisiones concurrentemente para medir tiempos de respuesta y tasa de errores. Se emplearán herramientas estándar de pruebas de carga.

- **Pruebas end-to-end (E2E):**
    - **Objetivo:** simular flujos de usuario completos a través de la interfaz gráfica para verificar que el sistema funciona como un todo.
    - **Ejemplos:** automatizar el flujo completo: abrir la web ->subir imagen ->introducir nombres ->asignar ítems ->verificar el resultado final mostrado en la página. Se utilizarán herramientas de automatización de navegador.

El análisis de los resultados se centrará en identificar patrones de errores, problemas de mantenibilidad, cobertura de pruebas y la eficacia general de usar IA para generar este tipo de aplicación.