Fase 4: Desarrollo del proyecto "Encriptación de Contraseña Segura"

Introducción

La fase de desarrollo implica la implementación práctica del diseño establecido previamente. En este momento, se construye el sistema mediante el uso de herramientas, tecnologías y lenguajes de programación seleccionados, siguiendo las especificaciones de los algoritmos y la arquitectura planteada. Esta etapa es crucial, ya que debe garantizarse que el código no solo sea funcional, sino también eficiente, seguro y mantenible.

Objetivo del desarrollo

El objetivo principal de esta fase es construir un sistema completamente funcional que:

1. Evalúe de forma segura y precisa la calidad de una contraseña ingresada por el usuario.
2. Genere contraseñas seguras siguiendo los estándares definidos.
3. Asegure que el sistema sea robusto frente a amenazas como ataques cibernéticos o mal uso.

Actividades principales del desarrollo

1. Configuración del entorno de desarrollo

- **Herramientas de programación:**
    - Lenguaje principal: **Python** (para lógica y backend) o **JavaScript** (en caso de una implementación web interactiva).
    - Frameworks: Flask o Django (en caso de Python) o Node.js/Express (para backend en JavaScript).
    - Interfaz: HTML, CSS y JavaScript (para aplicaciones web).
    - Gestión de dependencias: pipenv (Python) o npm (JavaScript).
- **Control de versiones:** Usar **Git** para el seguimiento de cambios en el código y colaboración del equipo. El repositorio se mantendrá en **GitHub**.

2. Desarrollo de módulos principales

a) Módulo de evaluación

- Implementación de un algoritmo para:
    - Verificar la longitud mínima de la contraseña (ej. ≥ 12 caracteres).
    - Comprobar la presencia de caracteres variados: letras mayúsculas, minúsculas, números y símbolos.

- Validar la entropía mediante fórmulas matemáticas (por ejemplo, Shannon Entropy).
- Comparar la contraseña ingresada con una base de datos de contraseñas comprometidas.

b) Módulo de generación de contraseñas

- Desarrollo de un algoritmo que:
  - Use generadores aleatorios seguros (por ejemplo, secrets en Python o crypto en Node.js).
  - Genere contraseñas de alta entropía con parámetros configurables (longitud, caracteres especiales, etc.).
  - Garantice la unicidad de cada contraseña.

c) Módulo de comunicación con bases de datos

- Si se integra una base de datos local o se utiliza una API de terceros:
  - Configurar consultas seguras para evitar inyecciones SQL.
  - Implementar cacheo para reducir tiempos de respuesta en búsquedas recurrentes.

d) Interfaz de usuario

- Implementar una interfaz web donde los usuarios puedan:
  - Ingresar una contraseña para evaluación.
  - Recibir retroalimentación visual (por ejemplo, un indicador de fuerza).
  - Generar una nueva contraseña si la ingresada es débil.

3. Seguridad del sistema

- **Validación de datos:**
  - Garantizar que todas las entradas del usuario sean validadas para evitar ataques de inyección.
  - Limitar la longitud máxima de contraseñas ingresadas para prevenir desbordamientos de búfer.
- **Protección de contraseñas:**
  - No almacenar ni registrar contraseñas ingresadas en ningún momento.
  - Usar funciones de hash seguras como bcrypt o Argon2 para cualquier simulación.
- **Comunicación segura:**
  - Implementar HTTPS para todas las conexiones entre cliente y servidor.

- Usar bibliotecas de cifrado como PyCrypto o crypto-js en caso de operaciones sensibles.

Amenazas en la fase de desarrollo

1. **Código inseguro:** Puede incluir errores o vulnerabilidades que los atacantes podrían explotar.
2. **Errores en la implementación del algoritmo:** Si los criterios de evaluación o generación no se aplican correctamente, el sistema podría generar contraseñas débiles o clasificar erróneamente las ingresadas.
3. **Manejo inadecuado de datos del usuario:** Las contraseñas ingresadas podrían ser almacenadas o transmitidas sin cifrado.

Soluciones a las amenazas

1. **Código inseguro:**
   - Realizar revisiones de código periódicas para detectar y corregir errores.
   - Utilizar herramientas de análisis estático de código, como SonarQube, para identificar vulnerabilidades.
2. **Errores en la implementación del algoritmo:**
   - Escribir pruebas unitarias para verificar que los algoritmos funcionan según lo especificado.
   - Comparar los resultados con herramientas de evaluación de contraseñas ya existentes.
3. **Manejo inadecuado de datos:**
   - Asegurarse de que las contraseñas nunca se almacenen en texto plano.
   - Usar cifrado extremo a extremo para cualquier dato sensible.

Prácticas recomendadas

1. **Escribir código limpio:** Seguir principios como SOLID y patrones de diseño para garantizar que el código sea mantenible y legible.
2. **Documentar el código:** Agregar comentarios y generar documentación técnica para facilitar el entendimiento por parte de otros desarrolladores.
3. **Automatizar pruebas:** Usar herramientas de prueba continua para validar cada cambio en el código.

Entregables del desarrollo

1. Código fuente completo y funcional, organizado por módulos.
2. Archivos de configuración para el entorno (por ejemplo, requirements.txt para dependencias en Python).
3. Documentación técnica que explique el funcionamiento del sistema.
4. Registros de pruebas unitarias que demuestren el correcto funcionamiento de los módulos.

Resultados esperados del desarrollo

- Un sistema funcional que pueda:
    - Evaluar contraseñas según los criterios definidos.
    - Generar contraseñas seguras de alta entropía.
    - Proteger la información sensible de los usuarios.
- Un código limpio, seguro y bien documentado que facilite futuras mejoras o mantenimiento.