Selección del programa a desarrollar / Generación de diagramas funcionales y Arquitectura de Software

Juan Francisco Morán Gortaire

Universidad Internacional del Ecuador

Sistemas de Información

Lógica de Programación

Enero 2025

Contenido

[1. Detalle del software “Desarrollar un generador seguro de contraseñas” 3](#_Toc188125062)

[Documentación del proyecto 3](#_Toc188125063)

[2. Diagramas de caso de uso y de arquitectura 5](#_Toc188125064)

[Diagrama de caso de uso 5](#_Toc188125065)

[Diagrama arquitectura 5](#_Toc188125066)

[3. Enlace video explicativo 6](#_Toc188125067)

[4. Bibliografía 6](#_Toc188125068)

[5. Diagramas 7](#_Toc188125069)

## Detalle del software “Desarrollar un generador seguro de contraseñas”

### Documentación del proyecto

1. Objetivo del Proyecto

Desarrollar una aplicación de consola en Python que genere contraseñas seguras basadas en los parámetros especificados por el usuario. La aplicación deberá ser fácil de usar y proporcionar opciones para personalizar la longitud y complejidad de las contraseñas.

2. Razones para Usar la Arquitectura Monolítica

• Simplicidad en el Desarrollo y el Mantenimiento: La arquitectura monolítica es ideal para proyectos pequeños como un generador de contraseñas debido a su simplicidad. Permite mantener todo el código en un solo archivo, facilitando el desarrollo, la prueba y el mantenimiento del software sin la complejidad de manejar múltiples servicios o componentes.

• Facilidad de Implementación: Dado que la aplicación consiste en un solo archivo de script ejecutable, la implementación se simplifica considerablemente. No hay necesidad de configurar y coordinar múltiples servicios o contenedores, lo que es ideal para aplicaciones de escala pequeña.

• Menor Sobrecarga de Gestión: Las arquitecturas más complejas, como los microservicios o serverless, requieren una gestión adicional en términos de configuración de red, balanceo de carga y comunicación entre servicios. En una aplicación monolítica, estos no son problemas ya que toda la funcionalidad reside en un solo proceso.

• Optimización de Recursos: Para una aplicación que no requiere escalabilidad a gran escala, como un generador de contraseñas en consola, la arquitectura monolítica utiliza los recursos de manera eficiente sin la sobrecarga que implican las arquitecturas distribuidas.

3. Funcionalidades del Sistema

• Generación de Contraseñas: Capacidad para generar contraseñas que pueden incluir letras mayúsculas, minúsculas, números y símbolos especiales.

• Configuración de Usuario: Permitir al usuario elegir la longitud de la contraseña y los tipos de caracteres que incluirá.

• Interfaz de Usuario en Consola: Proporcionar una interfaz de línea de comandos simple para facilitar la interacción del usuario con la aplicación.

4. Estructura de la Aplicación

La aplicación estará dividida en diferentes secciones dentro de un único archivo de script para mantener la simplicidad:

• Funciones de Generación de Contraseñas: Incluirá lógica para crear contraseñas según los criterios especificados (longitud, inclusión de números, símbolos, etc.).

• Interfaz de Usuario: Manejará todas las interacciones del usuario, como la toma de decisiones y la entrada de datos.

• Validaciones: Asegurará que las entradas del usuario sean válidas y manejará los errores de manera adecuada para evitar fallos en la generación de contraseñas.

5. Flujo de Trabajo de la Aplicación

• Inicio de la Aplicación: El usuario ejecuta el script y es recibido con un menú de opciones.

• Selección de Opciones: El usuario selecciona la longitud de la contraseña y los tipos de caracteres a incluir a través de la consola.

• Generación y Visualización: La aplicación genera la contraseña basada en las preferencias del usuario y la muestra en la consola.

• Repetición o Salida: El usuario puede elegir generar otra contraseña o salir de la aplicación.

6. Consideraciones de Desarrollo

• Mantenibilidad: Estructura clara del código con comentarios adecuados y nombres de variables descriptivos para facilitar futuras actualizaciones o modificaciones.

• Seguridad: Asegurarse de que la generación de contraseñas es realmente aleatoria y segura utilizando las bibliotecas adecuadas de Python.

• Pruebas: Incluir pruebas básicas para verificar que la generación de contraseñas cumple con los requisitos especificados.

7. Herramientas y Tecnologías

• Lenguaje de Programación: Python.

• Librerías de Python: random para la generación de caracteres aleatorios, sys para interacciones de línea de comandos.

8. Despliegue

• La aplicación será ejecutable desde cualquier terminal que tenga Python instalado, sin necesidad de instalaciones adicionales.

## Diagramas de caso de uso y de arquitectura

### Diagrama de caso de uso

Es una herramienta visual que se utiliza en el Lenguaje Unificado de Modelado (UML) para mostrar cómo los usuarios (actores) interactúan con un sistema. Este tipo de diagrama es útil para entender los requisitos funcionales del sistema, es decir, lo que el sistema debe hacer.

Elementos principales:

1. **Actores**: Son los usuarios o sistemas externos que interactúan con el sistema. No se refieren a personas específicas, sino a roles o perfiles.
2. **Casos de uso**: Representan las funcionalidades o servicios que el sistema ofrece a los actores.
3. **Relaciones**: Muestran cómo los actores y los casos de uso están conectados.

Propósitos:

* **Identificar requisitos funcionales**: Ayuda a definir claramente lo que el sistema debe hacer.
* **Facilitar la comunicación**: Proporciona una visión clara y común del sistema para desarrolladores y clientes, lo que facilita la comprensión y el desarrollo del proyecto.

### Diagrama arquitectura

Es una imagen que muestra cómo se organiza y funciona un sistema de software. Es muy útil en el desarrollo de software porque ayuda a entender cómo se conectan y trabajan juntos los diferentes componentes del sistema.

Tipos de diagramas de arquitectura:

1. **Diagrama de arquitectura de software**: Muestra las partes del software y cómo se relacionan.
2. **Diagrama de arquitectura de sistemas**: Enseña los diferentes componentes de un sistema y cómo interactúan.
3. **Diagrama de arquitectura de aplicaciones**: Representa la estructura de una aplicación, incluyendo sus partes, interacciones y flujo de datos.
4. **Diagrama de arquitectura de integración**: Se enfoca en los componentes, datos y tecnología que se usan para integrar diferentes sistemas.
5. **Diagrama de arquitectura de despliegue**: Muestra cómo se relacionan los componentes de la aplicación con sus entornos de despliegue.

Propósitos:

* **Facilitar la colaboración**: Mejora la comunicación entre los desarrolladores y diseñadores.
* **Reducir riesgos**: Ayuda a identificar problemas potenciales desde el principio.
* **Aumentar la eficiencia**: Proporciona una visión clara de la estructura del sistema, lo que permite identificar y resolver problemas rápidamente.
* **Escalabilidad**: Ayuda a encontrar formas eficientes de hacer crecer un sistema.

## Enlace video explicativo

<https://www.youtube.com/watch?v=YnCicpK3rx8>

## Bibliografía

Booch, G., Rumbaugh, J., & Jacobson, I. (2005). *The Unified Modeling Language User Guide* (2nd ed.). Addison-Wesley.

Fowler, M. (2004). *UML Distilled: A Brief Guide to the Standard Object Modeling Language* (3rd ed.). Addison-Wesley.

Rumbaugh, J., Jacobson, I., & Booch, G. (2004). *The Unified Modeling Language Reference Manual* (2nd ed.). Addison-Wesley.

Bass, L., Clements, P., & Kazman, R. (2012). Software Architecture in Practice (3rd ed.). Addison-Wesley.

Richards, M. (2015). Software Architecture Patterns. O'Reilly Media.

Rozanski, N., & Woods, E. (2011). Software Systems Architecture: Working with Stakeholders Using Viewpoints and Perspectives (2nd ed.). Addison-Wesley.

## Diagramas