

Actividad 6 – Aplicativo de Arquitectura de Software

Samuel García Torres

Diego Alejandro Salgado González

Geraldine de los ángeles Ríos Lamedá

Facultad de Ingeniería

Ingeniería de Software

Corporación Universitaria Iberoamericana

Arquitectura de Software

Joaquin Sanchez

13 – 10 – 2024

Introducción

Biblioteca es un servicio web que suplirá las necesidades de muchas personas al momento de necesitar y poder requerir un libro de manera virtual para aquellos usuarios de todas las edades, ya sean estudiantes, personas cotidianas o profesionales; es así como en este documento se explica de manera detallada el proyecto, se aclara el modelo del producto donde se representa como funciona el software, se hace una descripción técnica del producto, se hace una propuesta de herramientas y tecnologías a utilizar, se aclara las ventajas de usar el tipo de arquitectura que se lleva a cabo; todo esto explicado de una manera detallada con el fin de tener una mejor comunicación y una planificación concisa.

Link del Repositorio

<https://github.com/Alejo-119/Arq.-Software-Act-6.git>

Tipo de Arquitectura

El tipo de arquitectura de software que se va a utilizar es la **Arquitectura M.V.C** (modelo vista controlador), esta arquitectura separa la aplicación en tres componentes, Modelo, Vista y Controlador, lo que permite una mejor organización del código, cada componente se encarga de una responsabilidad específica.

Modelo: Maneja los datos y la lógica de negocio.

Vista: Gestiona la interfaz gráfica para el proyecto.

Controlador: Actúa de intermediario entre el usuario y el modelo, gestionando la interacción.

Ventajas

Facilita la mantenibilidad y el desarrollo colaborativo, ya que distintos desarrolladores pueden trabajar en distintas partes del proyecto.

Es más accesible hacer pruebas unitarias y funcionales, tiene flexibilidad en la interfaz porque podemos modificar la interfaz para web o aplicación móvil sin necesidad de modificar el código del modelo.

La Escalabilidad es un factor importante, la separación de los componentes permite hacer cambios en el código sin tener que modificar o reescribir todo el código.

Planteamiento del Modelo del Producto:

Objetivo del Proyecto: El objetivo de esta aplicación web es proporcionar una solución eficiente para la gestión de una biblioteca, permitiendo a los administradores y usuarios gestionar libros, préstamos y cuentas de usuario de manera sencilla e intuitiva. La aplicación está diseñada

para facilitar el acceso a los recursos bibliográficos y optimizar la experiencia del usuario en la búsqueda y el préstamo de libros.

Tecnologías Utilizadas:

Editor de Código: Visual Studio Code

Framework: Python Flask

Base de Datos: Firebase Firestore

Frontend: HTML/CSS y JavaScript

Repositorio: GitHub

Funcionalidades Principales:

1. Gestión de Libros:

Agregar, eliminar y consultar libros disponibles en la biblioteca y mostrar detalles de los libros y su disponibilidad actual.

2. Gestión de Usuarios:

Registro y modificación de cuentas de usuario y opción de eliminación de cuentas de usuario.

3. Gestión de Préstamos:

Registro de préstamos de libros por parte de los usuarios y registro de devoluciones de libros.

4. Gestión de Administradores:

Capacidad para bloquear usuarios en caso de infracciones o problemas y acceso exclusivo para agregar libros a la base de datos.

Estructura del Código: Clases y Métodos

1. Clase *Administrador*

Atributos:

id: Identificador único del administrador.

usuario: Nombre de usuario del administrador.

contraseña: Contraseña del administrador (almacenada de forma segura).

Métodos:

agregar_libro(libro): Permite al administrador añadir un nuevo libro a la base de datos.

bloquear_usuario(usuario_id): Permite al administrador bloquear a un usuario en caso de infracciones.

Imagen 1: Clase Admin

```
class Admin:

    def __init__(self, id_admin, usuario, contraseña):
        self.id_admin = id_admin
        self.usuario = usuario
        self.contraseña = contraseña

    # Método para agregar un libro a la base de datos
    def agregar_libro(self, libro):
        return libro.mostrar_detalle()

    # Método para eliminar un libro
    def eliminar_libro(self, id_libro):
        return f"Libro con ID {id_libro} ha sido eliminado."

    # Método para bloquear a un usuario
    def bloquear_usuario(self, id_usuario):
        return f"Usuario con ID {id_usuario} ha sido bloqueado."
```

2. Clase Usuario

Atributos:

id: Identificador único del usuario.

nombre: Nombre completo del usuario.

contraseña: Contraseña del usuario (almacenada de forma segura).

email: Correo electrónico del usuario.

Métodos:

registrar_usuario(): Permite a un nuevo usuario registrarse en la aplicación.

modificar_usuario(nuevos_datos): Permite al usuario actualizar su información personal.

eliminar_usuario(): Permite al usuario eliminar su cuenta.

Imagen 2: Clase Usuario

```
class Usuario:

    def __init__(self, id_usuario, nombre, contraseña, email):
        self.id_usuario = id_usuario
        self.nombre = nombre
        self.contraseña = contraseña
        self.email = email

    # Método para registrar un usuario
    def registrar_usuario(self):
        print(f"Usuario {self.nombre} registrado con éxito, su ID de usuario es: {self.id_usuario} :)")

    # Método para modificar los datos de usuario
    def modificar_usuario(self, nombre=None, contraseña=None, email=None):
        if nombre:
            self.nombre = nombre
        if contraseña:
            self.contraseña = contraseña
        if email:
            self.email = email
        print(f"Usuario {self.id_usuario} modificado con éxito.")

    # Método para eliminar el usuario

    def eliminar_usuario(self):
        print(f"Usuario {self.id_usuario} ha sido eliminado :( ")
```

3. Clase Libro

Atributos:

id: Identificador único del libro.

autor: Nombre del autor del libro.

título: Título del libro.

editorial: Nombre de la editorial del libro.

disponibilidad: Estado del libro (disponible/no disponible).

Métodos:

mostrar_detalle(): Muestra toda la información relevante sobre el libro.

mostrar_disponibilidad(): Indica si el libro está disponible para préstamo.

mostrar_no_disponible(): Indica que el libro no está disponible para préstamo.

Imagen 3: Clase Libro

```
class Libro:

    def __init__(self, id_libro, autor, titulo, editorial, disponible):
        self.id_libro = id_libro
        self.autor = autor
        self.titulo = titulo
        self.editorial = editorial
        self.disponible = disponible

    # Método para mostrar detalles del libro
    def mostrar_detalle(self):
        return{
            "ID": self.id_libro,
            "Autor": self.autor,
            "Título": self.titulo,
            "Editorial": self.editorial,
            "Disponible": self.disponible
        }

    # Método para obtener libros disponibles desde Firebase
    @staticmethod
    def obtener_libros_disponibles():
        libros_disponibles = db.child("libros").order_by_child("Disponible").equal_to(True).get()
        return [libro.val() for libro in libros_disponibles.each()]

    # Método para obtener libros no disponibles desde Firebase
    @staticmethod
    def obtener_libros_no_disponibles():
        libros_no_disponibles = db.child("libros").order_by_child("Disponible").equal_to(False).get()
        return [libro.val() for libro in libros_no_disponibles.each()]
```

4. Clase Préstamo

Atributos:

id_prestamo: Identificador único del préstamo.

fecha_prestamo: Fecha en la que se realiza el préstamo.

fecha_devolucion: Fecha en la que se devuelve el libro.

Métodos:

registrar_prestamo(): Registra el préstamo de un libro por parte de un usuario.

registrar_devolucion(): Registra la devolución de un libro.

Imagen 4: Clase Prestamo

```
class Prestamo:

    def __init__(self, id_prestamo, id_libro, id_usuario, fecha_prestamo, fecha_entrega):
        self.id_prestamo = id_prestamo
        self.id_libro = id_libro
        self.id_usuario = id_usuario
        self.fecha_prestamo = fecha_prestamo
        self.fecha_entrega = fecha_entrega

    # Método para registrar un préstamo
    def registrar_prestamo(self):
        return {
            "ID Préstamo": self.id_prestamo,
            "ID Libro": self.id_libro,
            "Fecha Préstamo": self.fecha_prestamo,
            "Fecha Entrega": self.fecha_entrega,
            "ID Usuario": self.id_usuario
        }

    # Método para registrar devolución
    def registrar_devolución(self, fecha_devolucion):
        self.fecha_entrega = fecha_devolucion
        return f"Préstamo {self.id_prestamo} registrado como devuelto el {self.fecha_entrega}."
```


Interacción del Usuario

Interfaz de Usuario: La aplicación contará con una interfaz web amigable donde los usuarios podrán registrarse, iniciar sesión, consultar libros y realizar préstamos, mientras que los administradores tendrán su propia sección para gestionar libros y usuarios.

Validación de Datos: Se implementarán validaciones en los formularios de entrada para asegurar que los datos sean correctos y completos.

Imagen 5: Diagrama de Caso de Usos

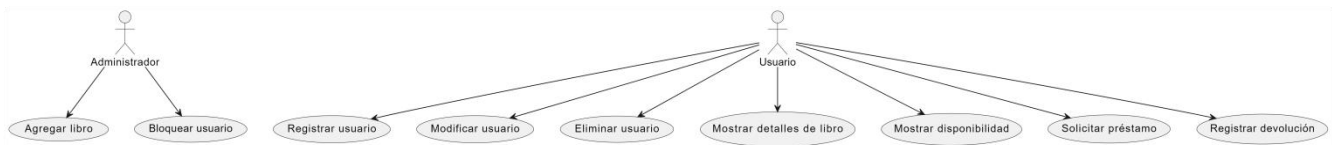
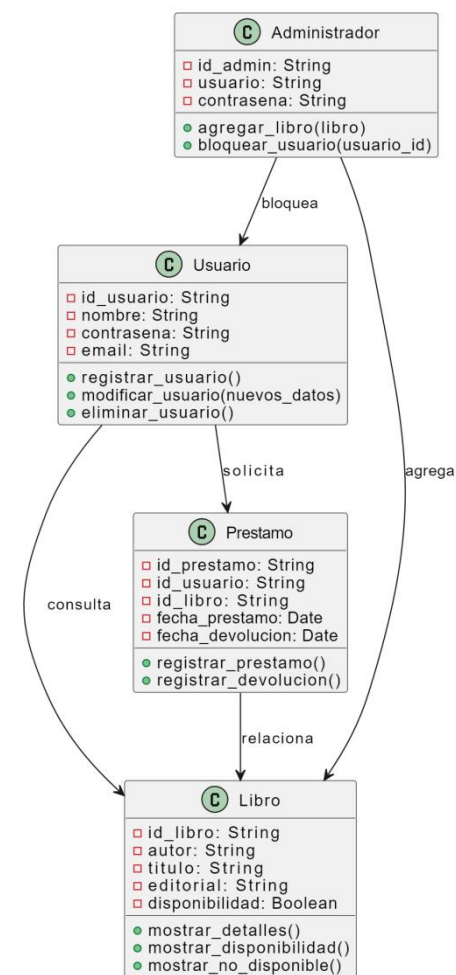


Imagen 6: Diagrama de Clases







Conclusiones

El desarrollo de "Letras Mágicas" ha permitido demostrar la importancia de una arquitectura bien definida, en este caso, la Arquitectura Modelo Vista Controlador (MVC). Esta separación de responsabilidades en componentes facilita la mantenibilidad del sistema, permitiendo que distintos desarrolladores trabajen en paralelo y que se puedan realizar cambios en el código sin afectar otras partes del software. La elección de tecnologías como Python y MySQL, junto con el uso de Visual Studio Code como IDE y GitHub como gestor de versiones, proporciona un entorno de desarrollo moderno y eficiente.

Además, los diagramas UML han sido fundamentales para visualizar y estructurar el sistema de manera clara, ayudando a comprender la interacción entre los diferentes componentes del sistema y planificar el desarrollo de manera más efectiva. La inclusión de estos diagramas ha permitido una mejor comunicación entre los miembros del equipo y una planificación más precisa.

En resumen, el proyecto "Letras Mágicas" se ha beneficiado de una arquitectura flexible y escalable, así como de herramientas y tecnologías modernas que facilitarán su evolución futura y su adaptación a nuevas necesidades.

Bibliografía

- Arciniegas Herrera, J. L., Collazos Ordóñez, C. A., Fernández de Valdenebro, M. V., Hormiga Juspian, M. A., Tulande Arroyo, A. (2010). Patrones arquitectónicos sobre usabilidad en el dominio de las aplicaciones web. *Ingeniería e Investigación*, 30 (1), 52-55. <https://revistas.unal.edu.co/index.php/ingeinv/article/view/15207/16001>
- Li, P., Jiang, P., Liu, J. (2019). Mini-MES: A Microservices-Based Apps System for Data Interconnecting and Production Controlling in Decentralized Manufacturing. *Applied Sciences* , 9 (18), 3675. <https://www.mdpi.com/2076-3417/9/18/3675>
- Pantaleo, G. y Rinaudo, I. (2015). *Ingeniería de Software*. Buenos Aires: Alfaomega. https://www-alfaomegacloud-com.iberobasededatosezproxy.com/auth/ip?intended_url=https://www-alfaomegacloud-com.iberobasededatosezproxy.com/library/publication/ingenieria-de-software
- Roldán Martínez, D. Valderas Aranda, P. J. y Torres Bosch, V. (2018). *Microservicios: un enfoque integrado*. Paracuellos de Jarama, Madrid, RA-MA Editorial. <https://elibro.net/es/lc/biblioibero/titulos/106522>
- Bass, L., Clements, P., & Kazman, R. (2012). *Software Architecture in Practice* (3rd ed.). Addison-Wesley Professional.
- Henao, C. [@CristianDavidHenao]. (n.d.).  *Como crear un Repositorio y Subir Proyecto a*  *GITHUB*  *Paso a Paso* . Youtube. Retrieved September 29, 2024, from <https://www.youtube.com/watch?v=eQMciGVc8N0>