Asignatura	**Proyecto Construcción de Software**
Carrera	**Tec. en Programación de Computadoras**
Ciclo	**2024**
Cuatrimestre	**Segundo Cuatrimestre**
Trabajo Práctico	**Escenario**

Grupo

ID/Matrícula	APELLIDO, Nombres	Correo Electrónico
	Tupac, José	jose.tupac@comunidad.ub.edu.ar
	Sol Sol Cespedes, Jose	jose.solsol@comunidad.ub.edu.ar
	Park, Lucas	lucas.park@comunidad.ub.edu.ar
000162244	Lozano, Ricardo	ricardo.lozano@comunidad.ub.edu.ar

Grilla de calificación

Concepto	Funcionalidad	Contenidos	Resolución	Formato	Ampliaciones
Sobresaliente (10)					
Distinguido (9-8)					
Bueno (7-6)					
Aprobado (5-4)					
Insuficiente (3-2-1)					
Reprobado (0)					
NOTA					

Comentario adicional del Profesor:

Metodología Ágil y Arquitectura 4+1

Metodología Ágil

La metodología ágil es un enfoque de desarrollo de software que se centra en la entrega iterativa e incremental de productos, con ciclos cortos que permiten adaptarse rápidamente a los cambios. En lugar de definir todos los requisitos al principio, como ocurre en metodologías tradicionales, los equipos ágiles buscan obtener retroalimentación constante para ajustar el desarrollo del proyecto.

Algunos de los principios fundamentales de la metodología ágil incluyen:

- Entregas incrementales: El software se entrega en pequeños incrementos funcionales al final de cada iteración o sprint.
- Colaboración continua: Los desarrolladores trabajan estrechamente con los stakeholders, adaptando el producto en función de su feedback.
- Flexibilidad: Los cambios en los requisitos son bienvenidos, incluso en etapas avanzadas del desarrollo.
- Individuos e interacciones: El énfasis está en la colaboración del equipo más que en los procesos o herramientas.

Arquitectura 4+1

La arquitectura 4+1 es un modelo diseñado por Philippe Kruchten para organizar y representar la arquitectura de un sistema de software desde múltiples perspectivas, permitiendo a los desarrolladores, usuarios y stakeholders tener una visión más clara de cómo se compone y funciona el sistema. Este modelo se divide en cuatro vistas principales y un conjunto de escenarios que validan estas vistas.

Las vistas de la arquitectura 4+1 son:

1. Vista Lógica: Describe la estructura estática del sistema en términos de sus componentes principales y cómo se relacionan entre sí para cumplir con los requisitos funcionales. Esta vista es especialmente útil para los desarrolladores que necesitan entender cómo está organizado el sistema desde una perspectiva funcional.
2. Vista de Desarrollo: Esta vista muestra la estructura del software desde la perspectiva del programador. Organiza el sistema en módulos, paquetes o componentes de código fuente, y define cómo se distribuyen las responsabilidades entre ellos. Está orientada a los aspectos de implementación y gestión de versiones.
3. Vista de Procesos: Se centra en los aspectos dinámicos del sistema, como la concurrencia, la sincronización y la comunicación entre procesos. Es útil para diseñar sistemas que requieren escalabilidad y alto rendimiento, ya que muestra cómo interactúan los diferentes procesos en tiempo de ejecución.

4. Vista Física: Describe cómo el software se despliega en la infraestructura física, incluyendo servidores, redes y dispositivos. Esta vista es crucial para entender cómo los componentes del sistema se distribuyen en el hardware, y se utiliza para diseñar la infraestructura del sistema.

Escenarios en la Arquitectura 4+1

Los **escenarios**, también conocidos como **casos de uso**, son fundamentales para validar las cuatro vistas de la arquitectura. Estos escenarios permiten analizar cómo interactúan las distintas partes del sistema en situaciones reales y aseguran que la arquitectura sea capaz de cumplir con los requisitos funcionales y no funcionales. A continuación, se detallan varios aspectos críticos que deben considerarse al desarrollar y validar estos escenarios:

Importancia de los Escenarios

1. **Validación de Requisitos**: Los escenarios actúan como un mecanismo para verificar que el sistema cumple con los requisitos establecidos. Proporcionan una representación tangible de cómo se espera que el sistema funcione en situaciones reales, permitiendo a los desarrolladores y stakeholders asegurarse de que el diseño satisface las necesidades del usuario.
2. **Puesta en Escena**: La puesta en escena implica recrear las condiciones en las que el sistema será utilizado, incluyendo el entorno técnico y las características del usuario. Esto es esencial para evaluar el rendimiento del sistema en situaciones diversas y para garantizar que se tenga en cuenta el contexto del usuario.
3. **Experiencia de Usuario (UX) y Diseño de Interfaz (UI)**: Los escenarios son cruciales para evaluar la UX y la UI, ya que validan la facilidad de uso y la efectividad de la interfaz a través de la que los usuarios interactúan con el sistema. Esto incluye el análisis de la navegación, la claridad de las interacciones y la respuesta del sistema a las acciones del usuario.
4. **Sincronización del Diseño (Design Syncing)**: Los escenarios también permiten comprobar que todos los elementos del sistema están bien sincronizados, desde la interfaz de usuario hasta los componentes del backend. La correcta sincronización garantiza una experiencia de usuario fluida y coherente, y minimiza la posibilidad de errores en la interacción entre diferentes partes del sistema.
5. **Usabilidad**: La usabilidad es un aspecto crítico que se evalúa a través de los escenarios. Se refiere a qué tan fácil y eficiente es para los usuarios completar tareas en el sistema. Los escenarios ayudan a identificar posibles obstáculos que puedan dificultar el uso del sistema y aseguran que los usuarios puedan realizar acciones comunes sin problemas.
6. **Accesibilidad**: Los escenarios deben considerar la accesibilidad para garantizar que el sistema sea usable por personas con diferentes capacidades, incluidas aquellas con discapacidades visuales, auditivas o motoras. Esto implica evaluar si el sistema cumple

con normas de accesibilidad y si permite interacciones adecuadas para todos los usuarios.

7. **Interoperabilidad**: La interoperabilidad se refiere a la capacidad del sistema para integrarse con otros sistemas y plataformas. Los escenarios deben validar que el sistema pueda compartir y recibir datos de manera eficiente, asegurando que los usuarios puedan interactuar con otras aplicaciones o servicios sin inconvenientes.

Casos de Uso

Los **casos de uso** son descripciones detalladas de cómo los usuarios (o actores) interactúan con un sistema para lograr un objetivo específico. Son herramientas fundamentales en el desarrollo de software, ya que ayudan a capturar y comunicar los requisitos funcionales del sistema de manera clara y comprensible. Los casos de uso no solo se centran en las acciones del usuario, sino que también incluyen la interacción entre el usuario y el sistema, así como las respuestas del sistema a estas acciones.

Componentes de un Caso de Uso

1. **Actor**: Un actor es cualquier entidad externa que interactúa con el sistema. Puede ser un usuario humano, otro sistema o un dispositivo. Los actores son quienes inician los casos de uso y buscan completar un objetivo.
2. **Objetivo**: Cada caso de uso tiene un objetivo específico que el actor desea lograr mediante la interacción con el sistema. Este objetivo define el resultado esperado y es el motivo por el cual se inicia la interacción.
3. **Descripción**: La descripción de un caso de uso detalla los pasos que se siguen durante la interacción entre el actor y el sistema. Esto incluye las acciones que realiza el actor y las respuestas del sistema. Esta sección debe ser lo suficientemente clara y precisa para que cualquier desarrollador o interesado pueda entender cómo debería funcionar la interacción.
4. **Flujos de Eventos**:
    - **Flujo Principal**: Describe la secuencia de pasos ideal que se siguen cuando el caso de uso se lleva a cabo de manera exitosa.
    - **Flujos Alternativos**: Estos son escenarios que pueden ocurrir bajo ciertas condiciones, como errores o excepciones. Los flujos alternativos son importantes para prever situaciones que pueden afectar la experiencia del usuario.
5. **Precondiciones y Postcondiciones**:
    - **Precondiciones**: Son las condiciones que deben cumplirse antes de que el caso de uso pueda iniciarse. Estas pueden incluir el estado del sistema o la disponibilidad de ciertos recursos.
    - **Postcondiciones**: Son las condiciones que se espera que se cumplan después de que el caso de uso se haya ejecutado. Esto asegura que el sistema haya cambiado su estado de acuerdo con el objetivo del actor.

Importancia de los Casos de Uso

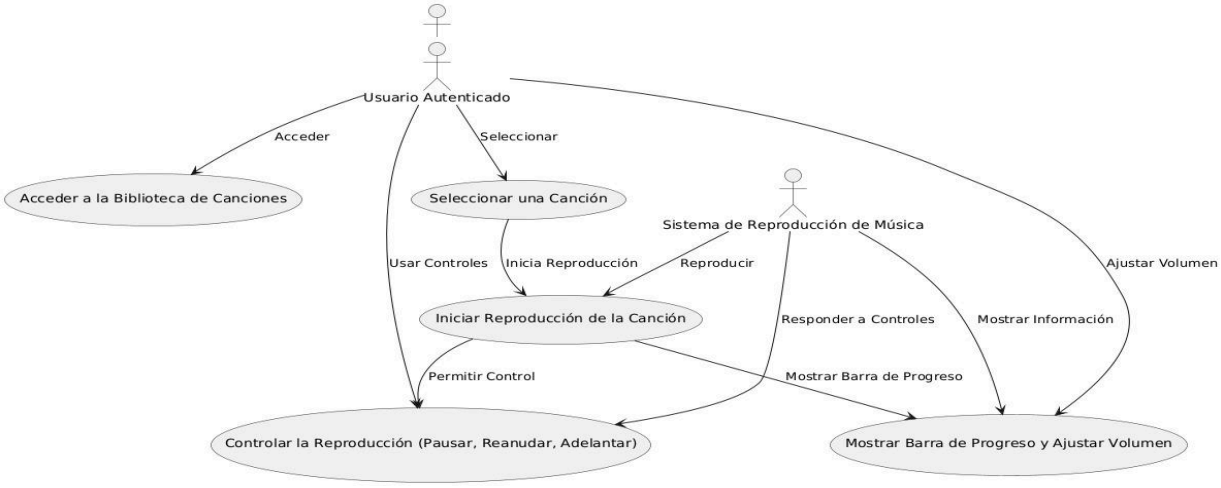
- **Comunicación**: Los casos de uso sirven como un lenguaje común entre los interesados, desarrolladores y diseñadores, facilitando la comprensión de lo que el sistema debe hacer.
- **Captura de Requisitos**: Ayudan a identificar y documentar los requisitos funcionales del sistema, asegurando que todos los aspectos necesarios se consideren durante el desarrollo.
- **Diseño y Pruebas**: Proporcionan una base sólida para la planificación del diseño del sistema y las pruebas, asegurando que se validen las funcionalidades que se han definido en los casos de uso.

Casos de Uso Extendidos y Reducidos

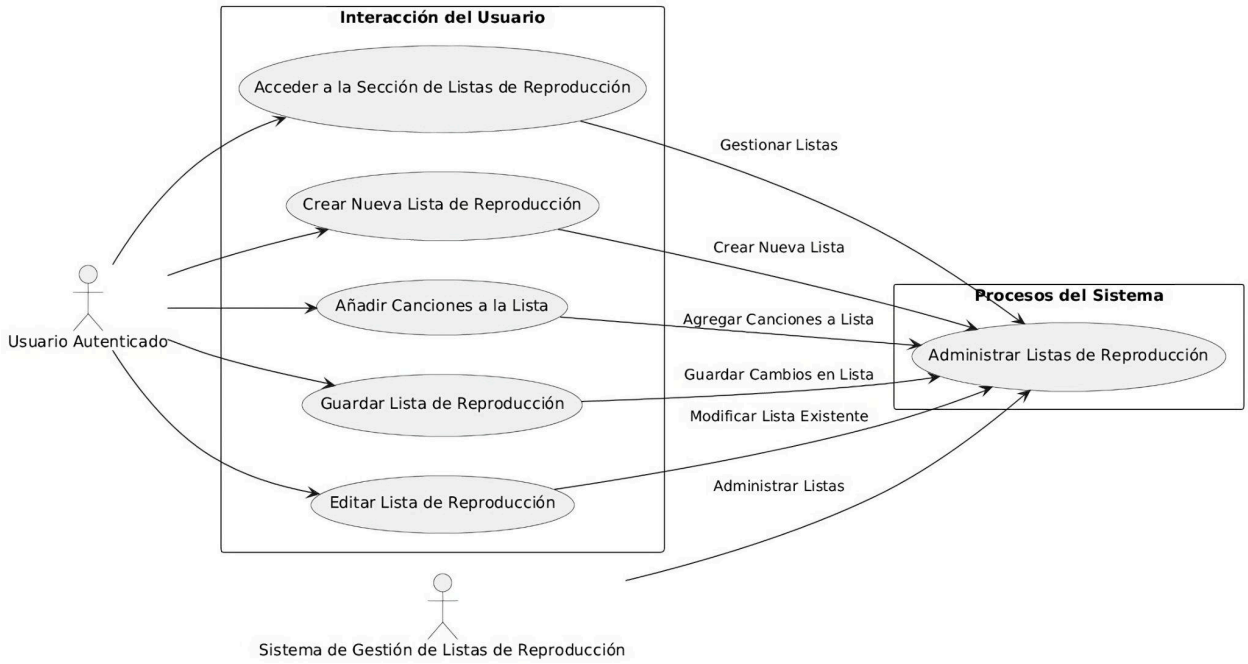
- **Casos de Uso Extendidos**: Se centran en interacciones más complejas o que involucran múltiples actores o procesos. Estos pueden incluir situaciones como la gestión de grandes bibliotecas de música o la integración con otros sistemas.
- **Casos de Uso Reducidos**: Se enfocan en las acciones más básicas y comunes que un usuario puede realizar en el sistema, como reproducir una canción o buscar un artista. Estos casos suelen ser más simples y sirven para validar funcionalidades fundamentales.

Casos de Uso en 'UB MUSIC'

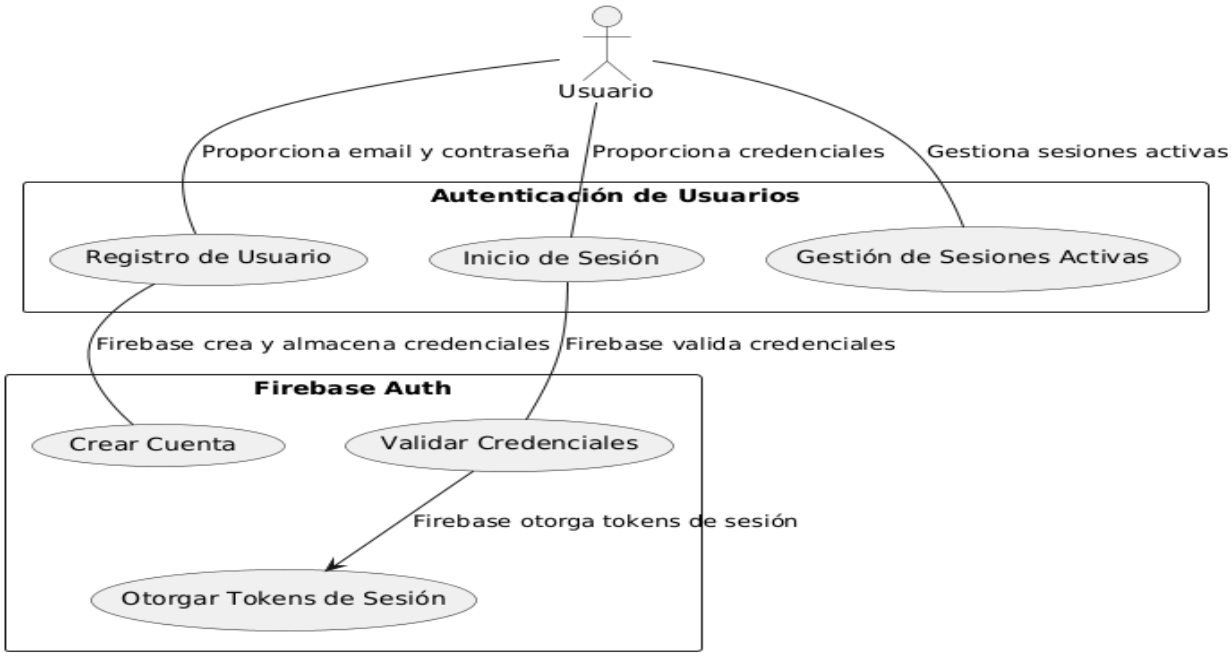
Reproducción de una Canción



Gestión de Listas de Reproducción



Autenticación de Usuarios



Generación de Recomendaciones Personalizadas