

UNIVERSIDAD AUTÓNOMA GABRIEL RENÉ MORENO



SEGUNDO PARCIAL: BIBLIOTECA VIRTUAL UAGRM

Materia: INF512 – Ingeniería de Software II

Docente: Martínez Canedo Rolando Antonio

Grupo: 13

Integrantes:

- | | |
|------------------------------|-----------|
| • Montaña Anivarro Andres | 218154526 |
| • Soliz Supayabe José Daniel | 218075881 |

Santa Cruz – Bolivia

Contenido

1. Fundamentación teórica.....	1
1.1. Introducción.....	1
1.2. Objetivo general	1
1.3. Objetivo específico	1
1.4. Alcance	1
1.5. Microservicios	2
1.5.1. Características.....	2
1.5.2. Ventajas.....	3
1.5.3. Desventajas.....	4
1.5.4. Sugerencias y recomendaciones.....	5
1.6. Gestión de contenedores	5
1.6.1. Kubernetes	6
1.6.2. Docker	7
1.7. Reconocimiento Facial.....	9
2. Proceso de desarrollo de software.....	9
2.1. Scrum.....	9
2.1.1. Roles	9
2.1.2. Product Backlog	10
2.1.3. Sprint 1	11
2.1.4. Sprint 2	12
3. Modelado C4.....	15
3.1. Contexto.....	15
3.2. Modelo de Arquitectura	16
3.2.1. Diagrama de contenedores	16
3.2.2. Diagrama de componentes.....	17
3.3. Modelo de datos.....	18
3.3.1. Diseño conceptual	18
3.3.2. Diseño lógico	18
3.3.3. Diseño físico	19
4. Manual de Usuario	23
5. Conclusiones.....	29
6. Bibliografía	29



1. Fundamentación teórica

1.1. Introducción

En el presente proyecto se desarrollará una aplicación web para el almacenamiento de archivos de tipo PDF, haciendo uso de la arquitectura microservicios. Que contara con las siguientes funcionalidades principales: pasarela de pagos mediante PayPal, autenticación facial y almacenamiento de archivos en la nube.

1.2. Objetivo general

Desarrollar una aplicación web como biblioteca virtual, usando la arquitectura de microservicios.

1.3. Objetivo específico

- Realizar la captura de requerimientos del software mediante el estudio de antecedentes y casos de estudio.
- Analizar los requisitos de software identificados durante la etapa de captura de requerimientos.
- Realizar el diseño del sistema mediante modelos y diagramas, con el fin de definir los detalles de alto nivel y de implementación que guiarán a la construcción del sistema.
- Implementar el software, basándose en el diseño previamente realizado, utilizando un enfoque de calidad.
- Realizar las pruebas del sistema para verificar que se encuentre libre de errores, deficiencias y/o vulnerabilidades, verificando las características y que se cumplan los factores de calidad.

1.4. Alcance

A continuación, definiremos el alcance del proyecto:

Registrar usuarios: El software permitirá el registro de usuario de forma libre, sin embargo, este no podrá realizar ninguna acción hasta que se haya suscrito.

Realizar Suscripción: El software contara con 3 tipos de suscripciones: diarias,

mensuales y anuales.

Realizar pagos en línea: El software permitirá realizar el pago de su suscripción mediante la pasarela de pagos como ser PayPal.

Subir documentos: Se permitirá subir archivos de tipo PDF que serán categorizados de acuerdo a su área, para su visualización de los suscritos.

Autenticación fácil: El software contará con un servicio de autenticación facial, para los usuarios previamente registrados y suscritos.

Bloquear capturas de pantallas: El software contará con seguridad anti plagio, como ser el bloqueo de capturas de pantallas entre otros.

1.5. Microservicios

1.5.1. Características

Los microservicios surgen como una alternativa a la manera tradicional de desarrollar aplicaciones, que se denomina arquitectura monolítica. Este tipo de aplicaciones es muy dependiente del hardware sobre el que se instala, y se configura como un “bloque” único compuesto por tres niveles: el front end, la lógica de business y los datos. Esta gran dependencia del hardware se materializa en largos tiempos de inactividad a la hora de realizar actualizaciones del software o despliegues de nuevo hardware para añadir más capacidad.

- **Están enfocados a escenarios empresariales:** a diferencia del enfoque monolítico, cada microservicio se desarrolla a partir de un escenario concreto de cliente o negocio, por lo cual son más simples y menos condicionados por la tecnología.
- **Son desarrollados por equipos pequeños:** su enfoque a producto, permite a un equipo de desarrollo seguir vinculado al servicio que ha desarrollado a lo largo de todo su ciclo de vida – en contraposición con el modelo enfocado a proyecto, donde el equipo es únicamente responsable del desarrollo del software que a continuación se traslada a otro equipo para su mantenimiento.
- **Permiten un alto nivel de desacoplamiento:** cada microservicio mantiene su propio dominio lógico, comunicándose y compartiendo datos con los demás, con un enfoque RESTful y utilizando los protocolos HTTP y TCP, y XML o JSON como formato de serialización.
- **Son independientes de lenguajes concretos:** cada microservicio puede ser desarrollado en un distinto lenguaje de programación (C++, C#, Java, etc.) y

apoyarse a diversas tecnologías de almacenamiento a segunda de las ventajas que aportan para el objetivo del servicio.

- **Son gestionados de forma independiente:** tanto su implementación, actualización y escalado se realizan de forma independiente, lo que aplica también al control de versiones y a al almacenamiento de los estados de los servicios.
- **Tienen nombres únicos:** cada microservicio tiene un nombre único para que se pueda resolver su ubicación; un registro de servicios se ocupa de recopilar un directorio.
- **Son resilientes:** en caso de error, un microservicio puede reiniciarse en otra máquina para seguir estando disponible, evitando la pérdida de datos y manteniendo su coherencia.

1.5.2. Ventajas

- **Agilidad:** Los microservicios fomentan una organización de equipos pequeños e independientes que se apropian de los servicios. Los equipos actúan en un contexto pequeño y bien comprendido, y están facultados para trabajar de forma más independiente y más rápida. Esto acorta los tiempos del ciclo de desarrollo. Usted se beneficia significativamente del aumento de rendimiento de la organización.
- **Escalado flexible:** Los microservicios permiten que cada servicio se escale de forma independiente para satisfacer la demanda de la característica de la aplicación que respalda. Esto permite a los equipos adecuarse a las necesidades de la infraestructura, medir con precisión el costo de una característica y mantener la disponibilidad si un servicio experimenta un aumento en la demanda.
- **Implementación sencilla:** Los microservicios permiten la integración y la entrega continuas, lo que facilita probar nuevas ideas y revertirlas si algo no funciona. El bajo costo de los errores permite experimentar, facilita la actualización del código y acelera el tiempo de comercialización de las nuevas características.
- **Libertad tecnológica:** Las arquitecturas de microservicios no siguen un enfoque de "diseño único". Los equipos tienen la libertad de elegir la mejor herramienta

para resolver sus problemas específicos. Como consecuencia, los equipos que crean microservicios pueden elegir la mejor herramienta para cada trabajo.

- **Código reutilizable:** La división del software en módulos pequeños y bien definidos les permite a los equipos usar funciones para diferentes propósitos. Un servicio escrito para una determinada función se puede usar como un componente básico para otra característica. Esto permite que una aplicación arranque por sí sola, ya que los desarrolladores pueden crear nuevas capacidades sin tener que escribir código desde cero.
- **Resistencia:** La independencia del servicio aumenta la resistencia de una aplicación a los errores. En una arquitectura monolítica, un error en un solo componente, puede provocar un error en toda la aplicación. Con los microservicios, si hay un error en todo el servicio, las aplicaciones lo manejan degradando la funcionalidad sin bloquear toda la aplicación.

1.5.3. Desventajas

- **Diseño:** debe invertir tiempo en identificar las dependencias entre los servicios. Y debe estar atento, porque cuando se termina un diseño, puede surgir la necesidad de muchos otros debido a esas dependencias. También debe considerar los efectos de los microservicios en sus datos.
- **Pruebas:** las pruebas de integración, así como las pruebas finales, pueden tornarse más complejas e importantes que nunca. Tenga en cuenta que una falla en una parte de la arquitectura puede producir un verdadero error, y esto depende de la manera en que haya diseñado la arquitectura de sus servicios para que sean compatibles entre sí.
- **Control de versiones:** cuando actualice sus sistemas a las nuevas versiones, tenga en cuenta que corre el riesgo de anular la compatibilidad con las versiones anteriores. Se puede diseñar en función de una lógica condicional para manejar este problema, pero se torna una tarea engorrosa y desagradable. Otra opción es implementar múltiples versiones en vivo para distintos clientes, pero esto puede ser más complejo durante el mantenimiento y la gestión.
- **Implementación:** efectivamente también es un desafío, al menos durante la configuración inicial. Para simplificarla, primero debe invertir mucho en la

automatización, ya que la complejidad de los microservicios resulta agobiante para la implementación humana. Tenga en cuenta la manera y el orden en que implementará los servicios.

- **Registro:** con los sistemas distribuidos, se necesitan registros centralizados para integrar todos los elementos. De lo contrario, es imposible controlar la expansión.
- **Monitoreo:** es indispensable tener una vista centralizada del sistema para identificar las causas de los problemas.
- **Depuración:** la depuración remota no es opción y no funcionará en decenas ni cientos de servicios. Desgraciadamente, no hay una única respuesta sobre cómo realizar la depuración en este momento.
- **Conectividad:** considere la detección de servicios, así sea centralizada o integrada.

1.5.4. Sugerencias y recomendaciones

En síntesis, los microservicios tienen numerosas ventajas que ayudan en todos los procesos a la hora de implementarlos y comenzar a utilizarlos. Aun así, como es algo “nuevo” todavía mucha gente se ve en la duda de empezar con algo novedoso o continuar con lo que ya se conocía. Sin embargo, hemos podido ver que la implantación de los microservicios ayuda enormemente en los resultados de las empresas de manera totalmente positiva. Un ejemplo de ello lo vemos como hemos dicho anteriormente en Amazon o Netflix que ya utilizan esta tecnología.

1.6. Gestión de contenedores

Los contenedores son una opción común para implementar y gestionar software en la nube. Se están convirtiendo rápidamente en una parte fundamental de la estrategia de nube de muchas organizaciones. Los contenedores se utilizan para abstraer las aplicaciones del entorno físico en el que se ejecutan. Un contenedor empaqueta todas las dependencias relacionadas con un componente de software y las ejecuta en un entorno aislado. Esto alivia los dolores de cabeza de la gestión y facilita el traslado de las cargas de trabajo, si es necesario.

Las aplicaciones suelen estar formadas por componentes contenedorizados de manera individual (a menudo llamados microservicios), que deben organizarse a nivel de

conectividad de red para que la aplicación se ejecute según lo previsto. El proceso de gestionar múltiples contenedores de esta manera se conoce como organización de contenedores.

En otras palabras, en el desarrollo moderno, las aplicaciones ya no son monolíticas, sino que están compuestas por decenas o cientos de componentes contenedorizados y débilmente acoplados, que necesitan trabajar juntos para permitir que una aplicación determinada funcione tal como ha sido diseñada. La organización de contenedores se refiere al proceso de gestionar el trabajo de componentes individuales y capas de aplicaciones.

Utilice la organización en contenedores para automatizar y gestionar las siguientes tareas:

- Preparación e implementación
- Configuración y programación
- Asignación de recursos
- Disponibilidad de los contenedores
- Ajuste o eliminación de contenedores según el equilibrio de las cargas de trabajo en la infraestructura
- Equilibrio de carga y enrutamiento del tráfico
- Supervisión del estado de los contenedores
- Configuración de aplicaciones en función del contenedor en el que se vayan a ejecutar
- Protección de las interacciones entre contenedores

1.6.1. Kubernetes

Las aplicaciones en contenedores pueden ser complicadas. Durante la producción, muchas pueden requerir cientos o miles de contenedores independientes. Es en este punto donde los entornos en tiempo de ejecución de contenedores, como Docker, se benefician del uso de otras herramientas para orquestar o gestionar todos los contenedores en funcionamiento.

Una de las herramientas más populares para este fin es Kubernetes, un orquestador de contenedores que reconoce varios entornos en tiempo de ejecución de contenedores,

incluido Docker.

Kubernetes orquesta el funcionamiento de varios contenedores juntos de forma armónica. Gestiona áreas como el uso de recursos de infraestructura subyacentes para aplicaciones en contenedores (por ejemplo, la cantidad de recursos de computación, red y almacenamiento necesarios). Las herramientas de orquestación como Kubernetes facilitan la automatización y el escalado de cargas de trabajo basadas en contenedores para entornos de producción activos.

-Principales elementos de Kubernetes:

- **Clúster:** un plano de control y una o varias máquinas informáticas o nodos.
- **Plano de control:** conjunto de procesos que controlan los nodos de Kubernetes. Aquí se originan todas las asignaciones de tareas.
- **Kubelet:** este servicio, el cual se ejecuta en los nodos, lee los manifiestos del contenedor y garantiza el inicio y el funcionamiento de los contenedores definidos.
- **Pod:** un grupo de uno o más contenedores implementados en un solo nodo. Todos los contenedores de un pod comparten la dirección IP, la IPC, el nombre del host y otros recursos.

1.6.2. Docker

Docker es un popular entorno en tiempo de ejecución que se usa para crear y construir software dentro de contenedores. Usa imágenes de Docker (instantáneas de copia en escritura) para poner en marcha aplicaciones o software en contenedores en varios entornos, desde el desarrollo hasta las pruebas y la producción. Docker se basa en estándares abiertos y funciona en la mayoría de los entornos operativos más comunes, incluidos Linux, Microsoft Windows y otras infraestructuras locales o basadas en la nube.

-Ventajas

- **Despliegue rápido:** La implementación de un entorno de desarrollo o producción con Docker, reduce el tiempo a segundos ya que se crea un contenedor para cada proceso y no arranca un sistema operativo completo, lo que lo hace mucho mas manejable para eliminarlo y volverlo a crear si arriesgar todos los datos.
- **Inversión y costos:** Docker ayuda drásticamente a reducir los costos en implementación de recursos de infraestructura ya que la naturaleza de Docker es de utilizar recursos específicos que requiere una aplicación, y esto puede ahorrar

en todo, desde requerimientos de un servidor hasta personal necesario para su mantenimiento.

- **Aislamiento de recursos:** Docker facilita a que las aplicaciones cuenten con sus recursos necesarios dentro de sus aplicaciones y que este de manera aislada de otros contenedores. Y también los facilita que solo utilicen sus recursos asignados.
- **Estandarización:** Debido al uso de imágenes, esto nos ayuda a tener una estandarización de los recursos en los cuales se esta trabajando ya que se utilizara la misma imagen de Docker en el desarrollo de una aplicación. Esto ahorra mucho tiempo en el momento de implementar un entorno de desarrollo y producción, ya que se tiene lo necesario en una imagen de Docker.
- **Simplicidad y configuraciones rápidas:** Uno de los grandes beneficios de utiliza Docker es si simplicidad ya que con solo 3 comandos puedes tener levantado un servidor de paginas web sin depender del sistema operativo a comparación de una instalación tradicional ya que implica ver las versiones y la compatibilidad al sistema operativo.

Existe muchas mas razones por las cuales es muy eficiente utilizar Docker en nuestros entornos de trabajo.

-Desventajas

Docker presenta algunas desventajas en cuanto a su implementación de su entorno:

- Se requiere mínimo la versión de Kernel 3.8.
- Algunas versiones de Docker dan error debido a que se encuentran en constante desarrollo.
- Solo soporta a sistemas operativos Linux de arquitectura de 64 bits.
- Para Windows aún se encuentra en fase de desarrollo.

1.7. Reconocimiento Facial

2. Proceso de desarrollo de software

2.1. Scrum

2.1.1. Roles

PERSONAS	CARACTERÍSTICAS	ROL
Andres Montaña Anivarro	Encargado de optimizar y maximizar el valor del producto, siendo la persona encargada de gestionar el flujo de valor del producto a través del Product Backlog.	Product Owner
Jose Daniel Soliz Supayabe	Lidera los equipos en la gestión ágil de proyectos. Su misión es que los equipos de trabajo alcancen sus objetivos hasta llegar a la fase de sprint final eliminando cualquier dificultad que puedan encontrar en el camino.	Scrum Master
Andres Montaña Anivarro	El equipo de desarrollo se encargará de crear un incremento terminado a partir de los elementos del Product Backlog seleccionados (Sprint Backlog) durante el Sprint Planning.	Development Team
Jose Daniel Soliz Supayabe		

2.1.2. Product Backlog

Product Backlog						
	id	tarea	estimacion	tiempo	tipo	prioridad
sprint 0	T01	Planificar reunion de organización	1	1hr	analisis	Alta
	T02	Asignación de roles a los miembros del equipo	3	10 minutos	analisis	Alta
	T03	Definir los objetivos	4	1 hora	analisis	Alta
	T04	Definir y describir los problemas	4	30 minutos	analisis	Alta
	T05	Determinar el alcance	5	3 horas	analisis	Alta
	T06	Documentar el perfil del proyecto	5	4hr	Diseño	Alta
	T07	Determinar las herramientas de uso en el proyecto	1	45min	analisis	Media
	T08	Instalar y preparar las herramientas de uso	2	1 hr	Preparacion	baja
	T09	Listar casos de uso	3	1hr 30min	analisis	Media
	T10	Analizar y diseñar el modelo de dominio en base al al	5	5 horas	Diseño	Alta
sprint 1	T11	Planificar nueva reunion de organización	1	1hr	analisis	Alta
	T12	Realizar el respectivo mapeo	3	2 horas	diseño	Alta
	T13	Realizar el Script	3	2 horas	desarrollo	Alta
	T14	Generar las migraciones pertinentes según el modelo	5	3 horas	desarrollo	Alta
	T15	Preparar el entorno y alistar vistas basicas en react	1	2 horas	diseño	Alta
	T16	Implementar el crud de usuario	5	1 hora	desarrollo	Alta
	T17	implementar el crud de roles	5	1 hora	desarrollo	Ata
	T18	Implementar login	3	2 horas	desarrollo	Alta
	T19	Validar al usuario	2	30 minutos	desarrollo	Alta
	T20	visualizar el perfil de cada usuario	5	1 hora	desarrollo	Alta
sprint 2	T21	Implementar el log out	3	1 hora	desarrollo	Alta
	T22	Implementar pasarela de pago	3	3 hora	desarrollo	Alta
	T23	Implementar servidor de archivos	5	10 hora	desarrollo	Alta
	T24	Implementar servidor de autentificacion	5	10 horas	desarrollo	Alta
	T25	Ralizar posts	5	2horas	desarrollo	Alta
	T26	Realizar comentarios	1	1 hora	analisis	alta
	T27	Eliminar post	2	3 hora	desarrollo	alta
	T28	desarrollar crud transporte	4	3 hora	desarrollo	alta
	T29	Implementar tipos de subscripcion	3	4 hora	desarrollo	alta
	T30	Realizar subscripcion	4	3 hora	desarrollo	alta
	T31	ver mis pagos	3	3 hora	desarrollo	alta

2.1.3. Sprint 1

SPRINT 1							
ID	TAREA	CU	ESTIMACION	TIEMPO	TIPO	PRIORIDAD	Estado
T13	Planificar nueva reunion de organización		1	1hr	analisis	Alta	completado
T14	Realizar el respectivo mapeo		3	2 horas	diseño	Alta	completado
T15	Realizar el Script		3	2 horas	desarrollo	Alta	completado
T16	Generar las migraciones pertinentes según el modelo con		5	3 horas	desarrollo	Media	completado
T17	Preparar el entorno y alistar vistas basicas en react		3	1 hora	desarrollo	Media	En proceso
T18	Implementar el crud de usuario		5	3 horas	desarrollo	Alta	completado
HU1	implementar el crud de roles		1	2 horas	diseño	Alta	completado
HU2	Implementar login	CU1	5	1 hora	desarrollo	Alta	completado
HU3	Validar al usuario	CU2	5	1 hora	desarrollo	Ata	completado
HU4	visualizar el perfil de cada usuario	CU3	3	2 horas	desarrollo	Alta	completado
HU5	Implementar el log out	CU4	3	30min	desarrollo	Media	completado

Historia De Usuario	
Numero : HU1	Usuario: Administrador
Nombre de Historia: Implementar el Crud de Usuario	
Prioridad en negocio: A Riesgo en desarrollo: Media	
Programador responsable: Daniel	
Descripción: Esta funcionalidad permitirá al administrador registrar el nombre, edad, telefono	
Condición de Satisfacción:	
<ul style="list-style-type: none"> El Administrador podrá registrar, editar y dar de baja a un Usuario 	

Historia De Usuario	
Numero : HU2	Usuario: Administrador
Nombre de Historia: Implementar el Crud de roles	
Prioridad en negocio: A Riesgo en desarrollo: Media	
Programador responsable: Daniel	
Descripción: Esta funcionalidad permitirá al administrador añadir un rol dando el id y nombre	
Condición de Satisfacción:	
<ul style="list-style-type: none"> El Administrador podrá registrar, editar y dar de baja a un Rol 	

Historia De Usuario	
Numero : HU3	Usuario: Administrador y cliente
Nombre de Historia: Programar El login	
Prioridad en negocio: A Riesgo en desarrollo: Media	
Programador responsable: Daniel	
Descripción: Esta funcionalidad permitirá tanto al administrador como al cliente poder logearse	
Condición de Satisfacción:	
<ul style="list-style-type: none"> El Administrador podrá registrarse 	

Historia De Usuario	
Numero : HU4	Usuario: Administrador y cliente
Nombre de Historia: Visualizar el perfil de cada usuario	
Prioridad en negocio: M	Riesgo en desarrollo: Media
Programador responsable: Daniel	
Descripción: Esta funcionalidad permitirá al cliente y administrador visualizar su perfil	
Condición de Satisfacción:	
<ul style="list-style-type: none"> El Administrador podrá ver su perfil tambien el cliente podra hacerlo 	

Historia De Usuario	
Numero : HU5	Usuario: Administrador y cliente
Nombre de Historia: Implementar el logout	
Prioridad en negocio: A	Riesgo en desarrollo: Media
Programador responsable: Daniel	
Descripción: Esta funcionalidad permitira tanto al administrador o al cliente poder cerrar se	
Condición de Satisfacción:	
<ul style="list-style-type: none"> Cliente o Administrador ceraran session 	

2.1.4. Sprint 2

SPRINT 2							
ID	TAREA	NRO. CU	ESTIMACION	TIEMPO	TIPO	PRIORIDAD	Estado
HU6	Implementar pasarela de pago	CU5	1	1 hora	analisis	alta	completado
HU7	Implementar servidor de archivos	CU6	2	3 hora	desarrollo	alta	completado
HU8	Implementar servidor de autentificacion	CU7	4	3 hora	desarrollo	alta	completado
HU9	Ralizar posts	CU8	4	3 hora	desarrollo	alta	completado
HU10	Realizar comentarios	CU9	3	4 hora	desarrollo	alta	completado
HU11	Eliminar post	CU10	4	5 hora	desarrollo	alta	completado
HU12	Implementar tipos de subscripcion	CU11	5	3 hora	desarrollo	alta	completado
HU13	Realizar subscripcion	CU12	5	4 hora	desarrollo	alta	completado
HU14	ver mis pagos	CU13	4	5 hora	desarrollo	alta	completado

Historia De Usuario	
Numero : HU6	Usuario:
Nombre de Historia: Pasarela de pago	
Prioridad en negocio: A	Riesgo en desarrollo: Media
Programador responsable: Daniel	
Descripción: Esta funcionalidad permite poder escoger los tipos de pagos	
Condición de Satisfacción:	
<ul style="list-style-type: none"> Permitira al administrador poder registrar, modificar y eliminar una ciudad 	

Historia De Usuario	
Numero : HU7	Usuario: Administrador
Nombre de Historia: Implementar servidor de archivos	
Prioridad en negocio: A	Riesgo en desarrollo: Media
Programador responsable: Daniel	
Descripción: Permite poder guardar los archivos de los usuarios	
Condición de Satisfacción:	
<ul style="list-style-type: none"> ● Permiten que el equipo modifique y comparta el esquema de base de datos 	

Historia De Usuario	
Numero : HU8	Usuario: publico
Nombre de Historia: Perparar el Servidor para autenticacion	
Prioridad en negocio: M	Riesgo en desarrollo: Media
Programador responsable: Daniel	
Descripción: Permitira poder autenticarse con una captura del rostro del usaurio	
Condición de Satisfacción:	
<ul style="list-style-type: none"> ● Permitira visualizar el sistema 	

Historia De Usuario	
Numero : HU9	Usuario: publico
Nombre de Historia: realizar post	
Prioridad en negocio: M	Riesgo en desarrollo: Media
Programador responsable: Daniel	
Descripción: El usuario podra publicar libros	
Condición de Satisfacción:	
<ul style="list-style-type: none"> ● Usuario debera estar suscrito 	

Historia De Usuario	
Numero : HU10	Usuario: Administrador
Nombre de Historia: Realizar comentarios	
Prioridad en negocio: M	Riesgo en desarrollo: Media
Programador responsable: Daniel	
Descripción: El usuario podra realizar comentarios a los post publicados	
Condición de Satisfacción:	
<ul style="list-style-type: none"> ● registrado 	

Historia De Usuario	
Numero : HU11	Usuario: Administrador
Nombre de Historia: Eliminar post	
Prioridad en negocio: M	Riesgo en desarrollo: Media
Programador responsable: Daniel	
Descripción: El usuario podra eliminar los post publicados	
Condición de Satisfacción:	
● registrado	

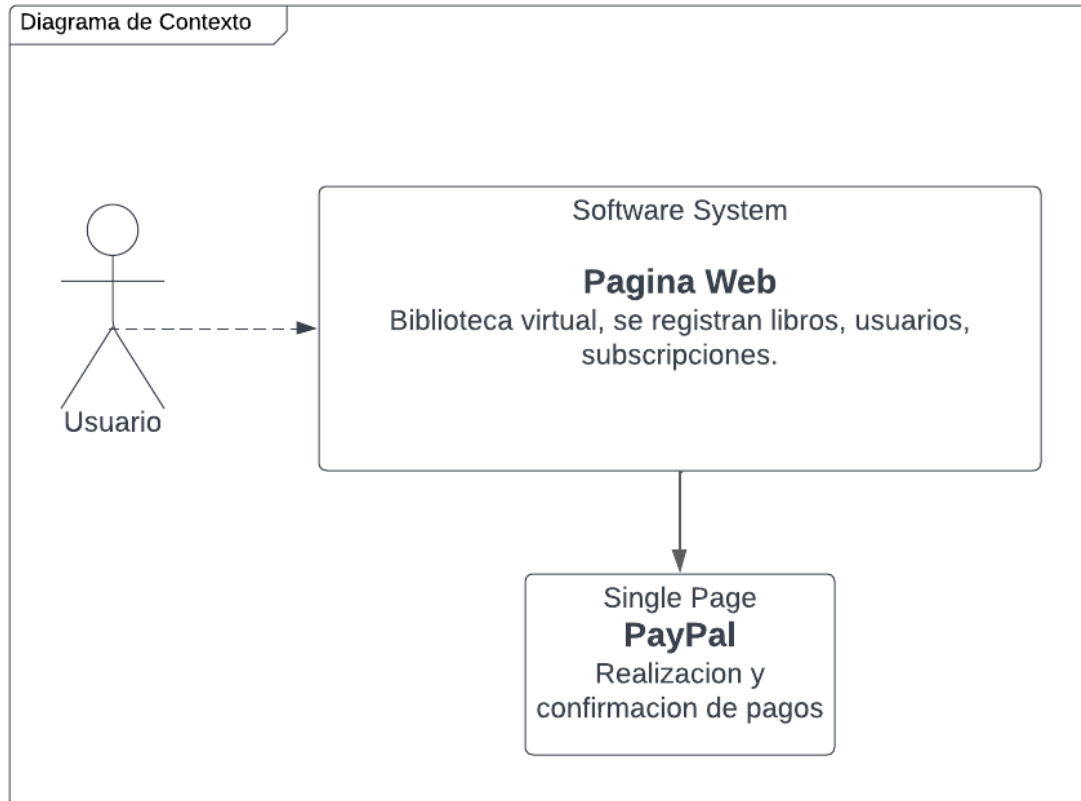
Historia De Usuario	
Numero : HU12	Usuario: Administrador
Nombre de Historia: Tipos de suscripciones	
Prioridad en negocio: M	Riesgo en desarrollo: Media
Programador responsable: Daniel	
Descripción: El administrador registrara los tipos de suscripciones q habran	
Condición de Satisfacción:	
● administrador	

Historia De Usuario	
Numero : HU13	Usuario: public
Nombre de Historia: Realizar suscripcion	
Prioridad en negocio: M	Riesgo en desarrollo: Media
Programador responsable: Daniel	
Descripción: El usuario podra pagar su tipo de suscripcion que desee	
Condición de Satisfacción:	
● confirmacion de pago paypal	

Historia De Usuario	
Numero : HU14	Usuario: public
Nombre de Historia: Ver mis pagos	
Prioridad en negocio: M	Riesgo en desarrollo: Media
Programador responsable: Daniel	
Descripción: El usuario podra visualizar sus pagos de suscripciones	
Condición de Satisfacción:	
● registrado	

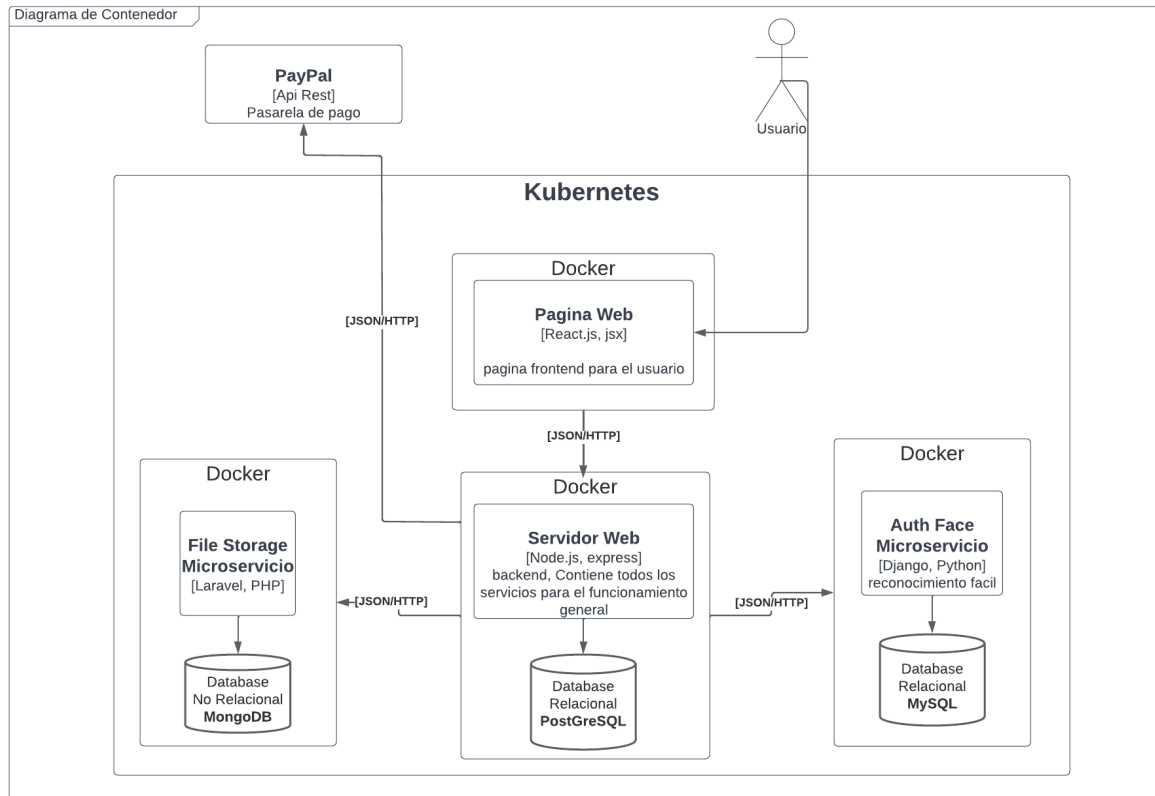
3. Modelado C4

3.1. Contexto

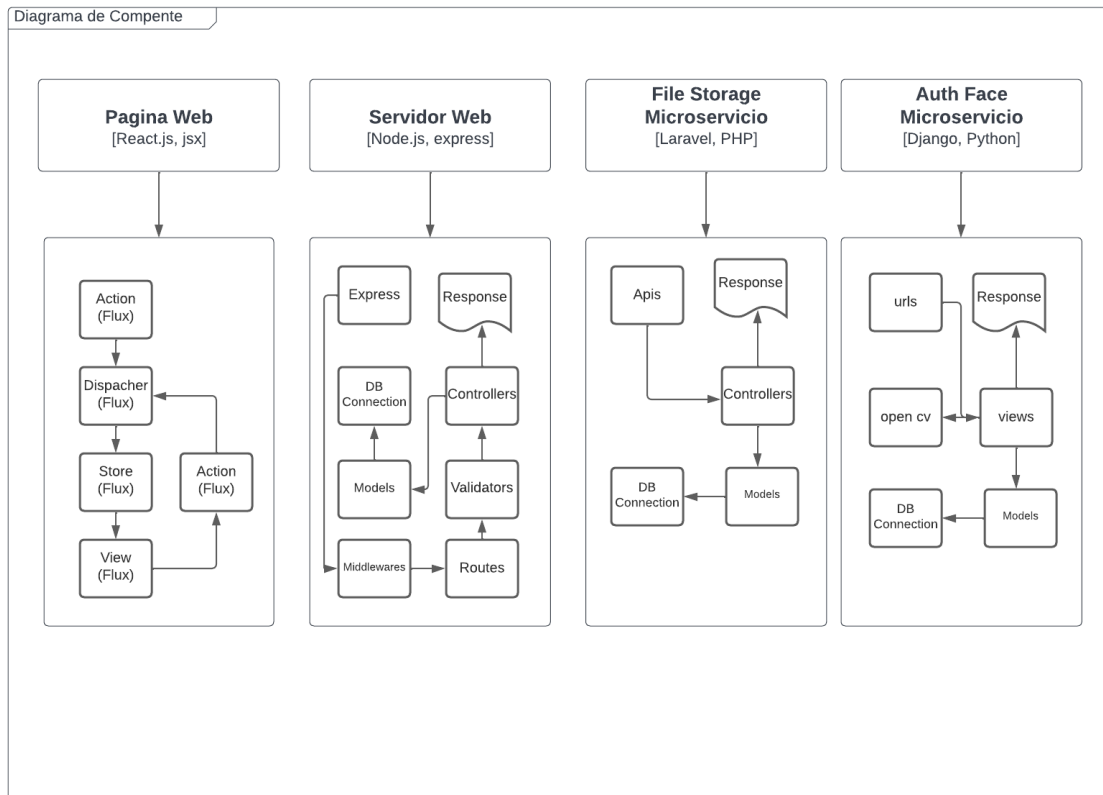


3.2. Modelo de Arquitectura

3.2.1. Diagrama de contenedores

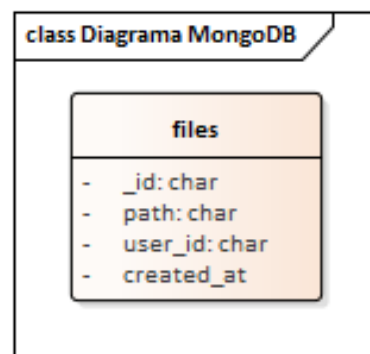
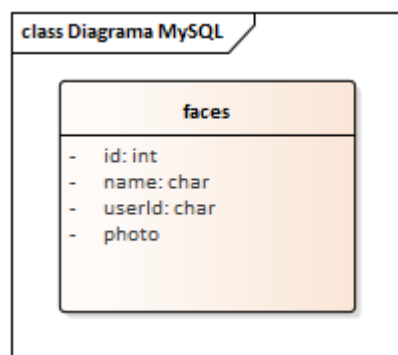
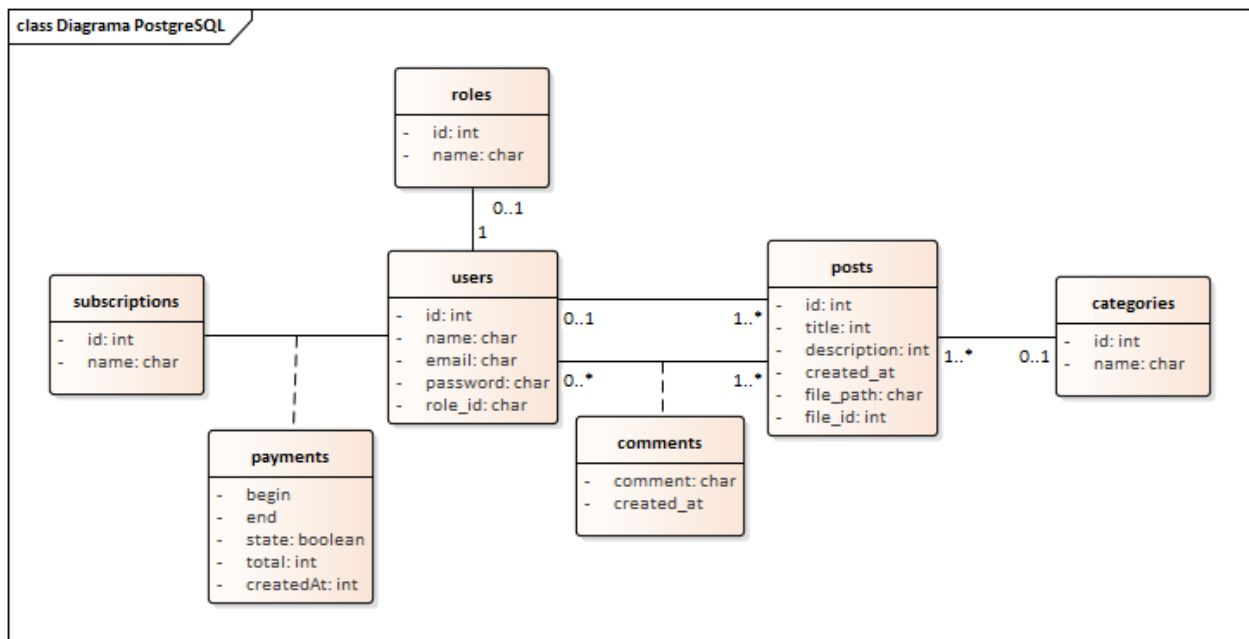


3.2.2. Diagrama de componentes



3.3. Modelo de datos

3.3.1. Diseño conceptual



3.3.2. Diseño lógico

PostgreSQL

subscriptions	
pk	
id	name

roles	
pk	
id	name

categories	
------------	--

pk	
id	name

users				
pk				fk
id	name	email	password	role_id

posts							
pk						fk	fk
id	title	description	created_at	file_path	file_id	user_id	categorie_id

comments				
pk			fk	fk
id	comment	created_at	user_id	post_id

payments							
pk						fk	fk
id	begin	end	state	total	created_at	subscription_id	user_id

MySQL

faces			
pk			
id	name	userId	photo

MongoDB

files			
pk			
id	path	user_id	created_at

3.3.3. Diseño físico

PostgreSQL

```
CREATE TABLE IF NOT EXISTS public.categories
(
    id integer NOT NULL DEFAULT nextval('categories_id_seq'::regclass),
    name character varying(255) COLLATE pg_catalog."default",
    "createdAt" timestamp with time zone NOT NULL,
    "updatedAt" timestamp with time zone NOT NULL,
    CONSTRAINT categories_pkey PRIMARY KEY (id)
);
```

```
CREATE TABLE IF NOT EXISTS public.comments
(
  id integer NOT NULL DEFAULT nextval('comments_id_seq'::regclass),
  comment text COLLATE pg_catalog."default",
  user_id integer NOT NULL,
  post_id integer NOT NULL,
  "createdAt" timestamp with time zone NOT NULL,
  "updatedAt" timestamp with time zone NOT NULL,
  CONSTRAINT comments_pkey PRIMARY KEY (id)
);
```

```
CREATE TABLE IF NOT EXISTS public.payments
(
  id integer NOT NULL DEFAULT nextval('payments_id_seq'::regclass),
  begin timestamp with time zone,
  "end" timestamp with time zone,
  total numeric,
  subscription_id integer NOT NULL,
  user_id integer NOT NULL,
  state boolean,
  "createdAt" timestamp with time zone NOT NULL,
  "updatedAt" timestamp with time zone NOT NULL,
  CONSTRAINT payments_pkey PRIMARY KEY (id)
);
```

```
CREATE TABLE IF NOT EXISTS public.posts
(
  id integer NOT NULL DEFAULT nextval('posts_id_seq'::regclass),
  title character varying(255) COLLATE pg_catalog."default",
  description character varying(255) COLLATE pg_catalog."default",
  file_path character varying(255) COLLATE pg_catalog."default",
  file_id character varying(255) COLLATE pg_catalog."default",
  category_id integer,
  user_id integer,
  "createdAt" timestamp with time zone NOT NULL,
  "updatedAt" timestamp with time zone NOT NULL,
  CONSTRAINT posts_pkey PRIMARY KEY (id)
);
```

```
CREATE TABLE IF NOT EXISTS public.roles
(
  id integer NOT NULL DEFAULT nextval('roles_id_seq'::regclass),
  name character varying(255) COLLATE pg_catalog."default",
  "createdAt" timestamp with time zone NOT NULL,
  "updatedAt" timestamp with time zone NOT NULL,
```

```

        CONSTRAINT roles_pkey PRIMARY KEY (id)
    );

CREATE TABLE IF NOT EXISTS public.subscriptions
(
    id integer NOT NULL DEFAULT nextval('subscriptions_id_seq'::regclass),
    name character varying(255) COLLATE pg_catalog."default",
    price numeric,
    "createdAt" timestamp with time zone NOT NULL,
    "updatedAt" timestamp with time zone NOT NULL,
    CONSTRAINT subscriptions_pkey PRIMARY KEY (id)
);

CREATE TABLE IF NOT EXISTS public.users
(
    id integer NOT NULL DEFAULT nextval('users_id_seq'::regclass),
    name character varying(255) COLLATE pg_catalog."default",
    email character varying(255) COLLATE pg_catalog."default",
    password character varying(255) COLLATE pg_catalog."default",
    role_id integer,
    "createdAt" timestamp with time zone NOT NULL,
    "updatedAt" timestamp with time zone NOT NULL,
    CONSTRAINT users_pkey PRIMARY KEY (id)
);

ALTER TABLE IF EXISTS public.comments
    ADD CONSTRAINT comments_post_id_fkey FOREIGN KEY (post_id)
    REFERENCES public.posts (id) MATCH SIMPLE
    ON UPDATE CASCADE
    ON DELETE CASCADE;

ALTER TABLE IF EXISTS public.comments
    ADD CONSTRAINT comments_user_id_fkey FOREIGN KEY (user_id)
    REFERENCES public.users (id) MATCH SIMPLE
    ON UPDATE CASCADE
    ON DELETE CASCADE;

ALTER TABLE IF EXISTS public.payments
    ADD CONSTRAINT payments_subscription_id_fkey FOREIGN KEY (subscription_id)
    REFERENCES public.subscriptions (id) MATCH SIMPLE
    ON UPDATE CASCADE
    ON DELETE CASCADE;

```



```
ALTER TABLE IF EXISTS public.payments
  ADD CONSTRAINT payments_user_id_fkey FOREIGN KEY (user_id)
  REFERENCES public.users (id) MATCH SIMPLE
  ON UPDATE CASCADE
  ON DELETE CASCADE;
```

```
ALTER TABLE IF EXISTS public.posts
  ADD CONSTRAINT posts_category_id_fkey FOREIGN KEY (category_id)
  REFERENCES public.categories (id) MATCH SIMPLE
  ON UPDATE CASCADE
  ON DELETE SET NULL;
```

```
ALTER TABLE IF EXISTS public.posts
  ADD CONSTRAINT posts_user_id_fkey FOREIGN KEY (user_id)
  REFERENCES public.users (id) MATCH SIMPLE
  ON UPDATE CASCADE
  ON DELETE SET NULL;
```

```
ALTER TABLE IF EXISTS public.users
  ADD CONSTRAINT users_role_id_fkey FOREIGN KEY (role_id)
  REFERENCES public.roles (id) MATCH SIMPLE
  ON UPDATE CASCADE
  ON DELETE CASCADE;
```

END;

MySQL

```
CREATE TABLE `faces` (
  `id` int(11) NOT NULL,
  `name` varchar(255) NOT NULL,
  `userId` int(11) NOT NULL,
  `photo` longblob DEFAULT NULL
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_general_ci;
```

```
ALTER TABLE `faces`
  ADD PRIMARY KEY (`id`);
ALTER TABLE `faces`
  MODIFY `id` int(11) NOT NULL AUTO_INCREMENT, AUTO_INCREMENT=17;
COMMIT;
```

MongoDB

```
{
  "_id": {
    "$oid": "637d4e9c283d248ef90ccd92"
  },
  "path": "/storage/documents/file637d4e9b2062e.jpg",
  "user_id": "1",
  "updated_at": {
    "$date": {
      "$numberLong": "1669156507663"
    }
  },
  "created_at": {
    "$date": {
      "$numberLong": "1669156507663"
    }
  }
}
```

4. Manual de Usuario

Ingreso a la Biblioteca Virtual

144.22.174.111:5173/auth/login

No seguro

12

12

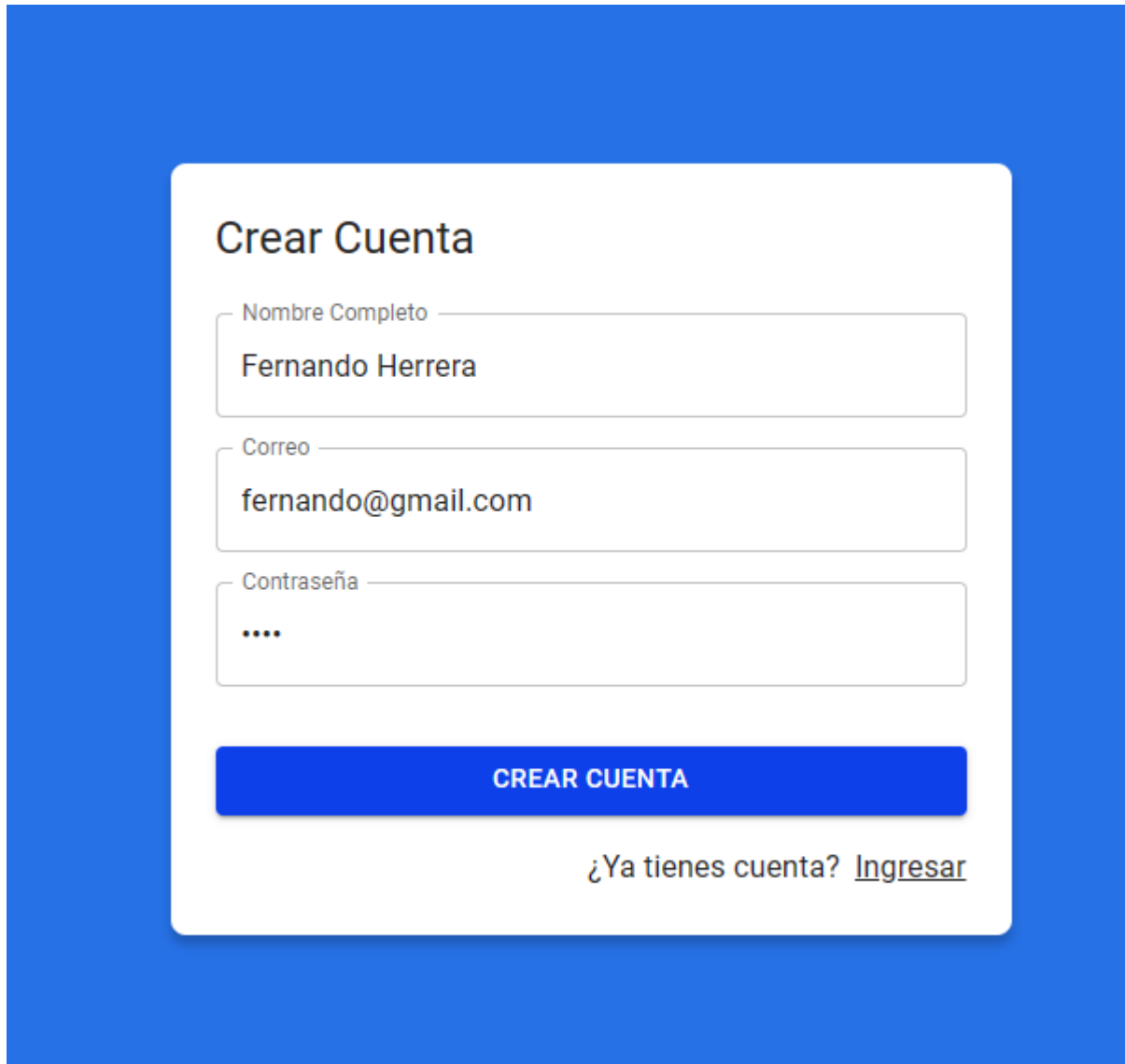
Paso 1

Debe ingresar su correo y contraseña.

Paso 2

Si no cuenta con una cuenta debe registrarse.

Crear una Cuenta

El formulario de creación de cuenta está centrado en un fondo azul sólido. El formulario mismo es un rectángulo blanco con una sombra sutil. En la parte superior del formulario, el título "Crear Cuenta" está en un tamaño de fuente grande y negro. Debajo del título, hay tres campos de entrada de texto, cada uno con un label grisado a la izquierda: "Nombre Completo", "Correo" y "Contraseña". El primer campo contiene el texto "Fernando Herrera", el segundo "fernando@gmail.com" y el tercero muestra cuatro puntos negros para ocultar la contraseña. Debajo de estos campos, hay un botón rectangular azul con el texto "CREAR CUENTA" en blanco y mayúsculas. En la parte inferior derecha del formulario, hay un enlace que dice "¿Ya tienes cuenta? Ingresar".

Crear Cuenta

Nombre Completo —
Fernando Herrera

Correo —
fernando@gmail.com

Contraseña —
....

CREAR CUENTA

¿Ya tienes cuenta? [Ingresar](#)

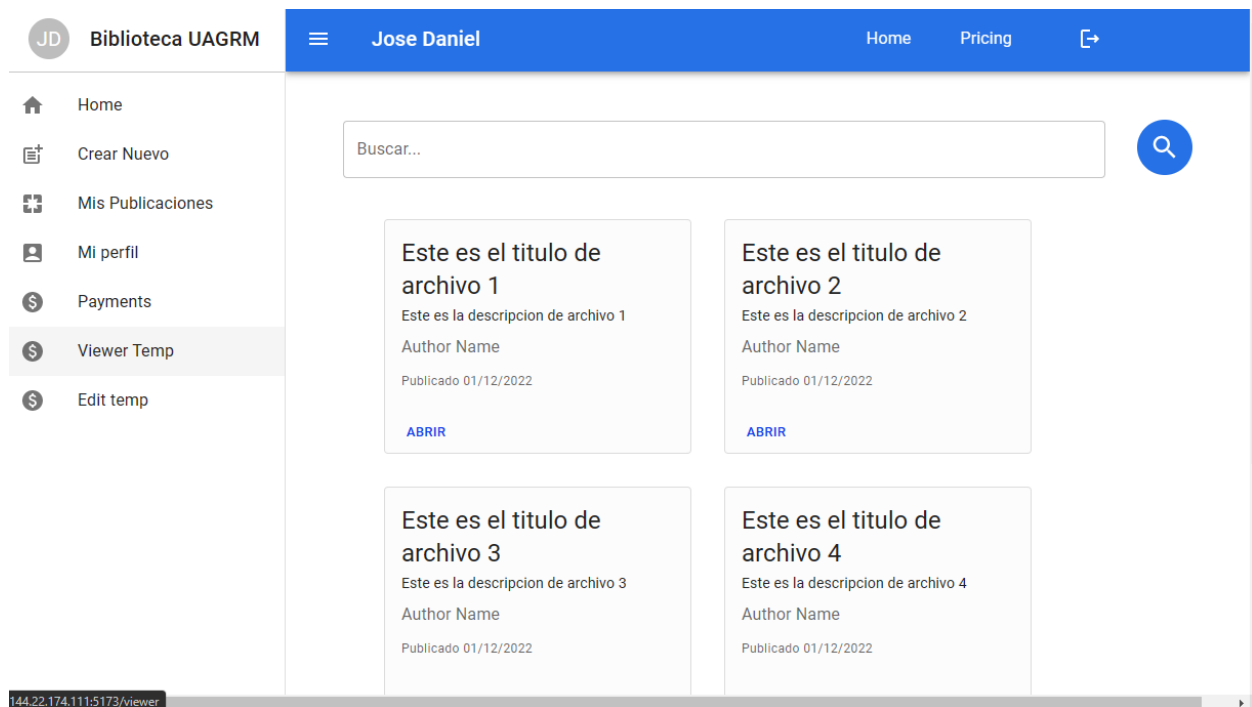
Paso 1

Ingresa sus datos en cada campo correspondiente.

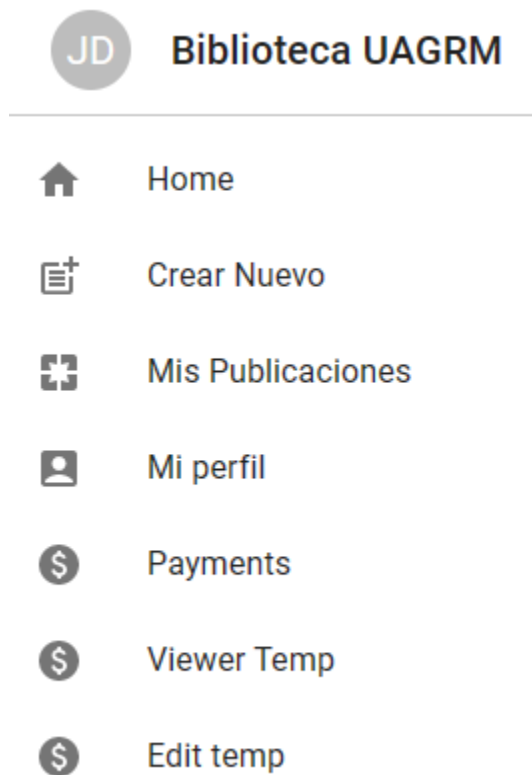
Paso 2

Presione el Botón CREAR CUENTA.

Menú Principal



Barra de Menú



Vista general de los libros publicados

Buscar...



Este es el titulo de archivo 1

Este es la descripcion de archivo 1

Author Name

Publicado 01/12/2022

[ABRIR](#)

Este es el titulo de archivo 2

Este es la descripcion de archivo 2

Author Name

Publicado 01/12/2022

[ABRIR](#)

Este es el titulo de archivo 3

Este es la descripcion de archivo 3

Author Name

Publicado 01/12/2022

Este es el titulo de archivo 4

Este es la descripcion de archivo 4

Author Name

Publicado 01/12/2022

Subir nuevo libro

Crear Publicacion

 PUBLICAR

Titulo


El titulo es obligatorio.

Descripcion

La descripcion es obligatoria

Categoria

La categoria es obligatoria

UPLOAD 

Debes subir un archivo pdf

Ver mis publicaciones

Mis Publicaciones

CREAR NUEVO

Este es el titulo de
archivo 1

Este es la descripcion de archivo 1

Publicado 01/12/2022

ELIMINAR

EDITAR

ABRIR

Este es el titulo de
archivo 2

Este es la descripcion de archivo 2

Publicado 01/12/2022

ELIMINAR

EDITAR

ABRIR

Este es el titulo de
archivo 3

Este es la descripcion de archivo 3

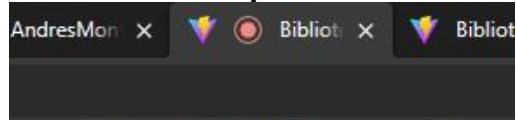
Publicado 01/12/2022

ELIMINAR

EDITAR

ABRIR


Autenticación por rostro



CAPTURE

ENVIAR

Realizar Pago de Suscripción a la biblioteca virtual media PayPal



Pagar con PayPal

Con una cuenta de PayPal, reúne los requisitos para recompensas y la Protección al Comprador.

Correo electrónico o número de celular
sb-43czsp22549309@personal.example.com

Contraseña
••••••••

Mostrar

[¿Ha olvidado su contraseña?](#)


Iniciar sesión

Abrir una cuenta

[Cancelar y volver a Subscription.com](#)

Confirmar Pago

CO





\$1.00 USD


Ship to Cliente one
1 Main St, San Jose, CA 95131


Change

Pay with

☒  PayPal balance
PREFERRED \$1.00 USD

☐  CREDIT UNION 1
Checking ****2779

☐  Visa
Credit ****2747

☐  PayPal Credit
Apply for PayPal Credit Pay over time for your purchase of \$1.00 with PayPal Credit. Subject to credit approval. [See terms](#)

[+ Add debit or credit card](#)

Complete Purchase

Payment method rights

Visualizar mis pagos realizados

☰	Jose Daniel	Home	Pricing	↔
---	-------------	------	---------	---

Mis Pagos

#ID	Tipo	Pagado	Desde	Hasta	Estado
4	Diario	03/12/2022	03/12/2022	04/12/2022	Active
3	Diario	02/12/2022	02/12/2022	03/12/2022	Expired
2	Mensual	01/11/2022	01/11/2022	01/12/2022	Expired
1	Anual	15/10/2021	15/10/2021	15/10/2022	Expired

5. Conclusiones

La arquitectura de microservicios define una forma de trabajar con grandes aplicaciones más desacoplada y surgen para atender unas demandas muy exigentes en cuanto a alta disponibilidad y capacidad de adaptación y cambio. No tiene por qué ser la mejor solución para nuestro caso.

6. Bibliografía

[ChatGPT \(openai.com\)](https://openai.com)

[Contenedores de Docker | ¿Qué es Docker? | AWS \(amazon.com\)](#)

[¿Qué es Kubernetes? | Kubernetes](#)

[¿Qué son y para qué sirven los microservicios? \(redhat.com\)](#)