Nintendo eShop Project

Diana Narváez - dianapa.narvaez@gmail.com

Copyright

Autora

Diana Paola Narváez Martínez, estudiante de Ingeniería en Sistemas Computacionales en la Universidad Autónoma de Aguascalientes en México, actualmente estudiante internacional en la Universidad Viña del Mar en Chile.

Prefacio

En la era digital actual, la distribución de videojuegos ha experimentado una transformación significativa con el auge del comercio electrónico y los servicios en línea. La Nintendo eShop ha emergido como una plataforma clave para que millones de jugadores en todo el mundo adquieran juegos digitales y otros contenidos relacionados.

Con un enfoque en la implementación tecnológica avanzada y el aprovechamiento de servicios de infraestructura en la nube, este proyecto busca proporcionar un esquema detallado de las soluciones técnicas que permitirán manejar un volumen de hasta 10 millones de transacciones diarias, garantizando al mismo tiempo la seguridad, disponibilidad y rendimiento de la plataforma.

Este informe abarca todas las fases de planificación y diseño, desde la selección de servidores hasta la elección de tecnologías de frontend y backend, la estimación de costos y plazos de implementación, y la identificación de los profesionales necesarios para llevar a cabo el proyecto.

La presente documentación tiene como fin servir de guía tanto para los desarrolladores como para los futuros interesados en la expansión y mejora de plataformas de distribución digital de videojuegos, respondiendo a las necesidades de un mercado en constante crecimiento y evolución.

Abstract

Este proyecto presenta el diseño e implementación de una plataforma Ecommerce destinada a la distribución de videojuegos digitales a través de la Nintendo eShop. Dada la creciente demanda de servicios digitales y la importancia del comercio electrónico en la industria de los videojuegos, el sistema debe ser capaz de gestionar hasta 10 millones de transacciones diarias, garantizando escalabilidad, rendimiento y seguridad.

El informe abarca la selección de servidores en la nube con capacidades de autoescalado, el diseño de una base de datos optimizada para el manejo de grandes volúmenes de datos y un conjunto de endpoints REST que aseguran la fluidez de las transacciones. Además, se examina la elección de tecnologías backend y frontend, considerando la opción entre un CMS o un desarrollo customizado. La estimación de costos operativos y los plazos de implementación, así como la identificación de los profesionales necesarios, se presentan para asegurar un desarrollo eficiente y efectivo del proyecto.

Este diseño de sistema busca proveer una solución moderna, flexible y de alto rendimiento, posicionando a la Nintendo eShop como una plataforma confiable y fácil de usar para millones de jugadores alrededor del mundo.

Table of Contents

Chapter 1. Sistema

Dado el alto nivel de transacciones esperadas (hasta 10 millones diarias), es crucial seleccionar una infraestructura de servidor que ofrezca alta disponibilidad, escalabilidad, rendimiento y seguridad.

1.1. ¿Qué es el sistema?

El sistema **Nintendo eShop** es una plataforma de comercio electrónico diseñada para permitir a los usuarios adquirir juegos digitales, contenido descargable (DLC), y otros productos relacionados con el ecosistema de Nintendo. La eShop proporciona una interfaz fácil de usar donde los jugadores pueden navegar, buscar y comprar títulos de videojuegos, así como gestionar sus suscripciones y cuentas de usuario. También incluye funcionalidades para reseñas de productos, recomendaciones personalizadas y opciones de pago seguras.

1.2. ¿Qué no es el sistema?

El sistema **Nintendo eShop** no es un servicio de distribución física de videojuegos, ni una plataforma de juego en línea. No proporciona juegos en formato físico (como cartuchos o discos) y no incluye la posibilidad de jugar directamente en la plataforma. Además, no es un sistema de redes sociales, aunque puede incluir interacciones básicas de comentarios y reseñas entre usuarios. Tampoco es un sistema de soporte técnico para problemas de hardware relacionados con las consolas de Nintendo.

1.3. ¿Qué hace el sistema?

- **Compra y descarga**: Permite a los usuarios comprar y descargar juegos digitales y contenido adicional directamente en su consola Nintendo.
- **Gestión de cuentas**: Proporciona a los usuarios la capacidad de gestionar su perfil, historial de compras, y detalles de suscripción.
- **Ofertas y promociones**: Ofrece descuentos y promociones en juegos, DLCs y suscripciones.
- **Reseñas y recomendaciones**: Permite a los usuarios dejar reseñas sobre los juegos que han adquirido y proporciona recomendaciones personalizadas basadas en el historial de compras y preferencias.
- **Métodos de pago**: Integra múltiples opciones de pago para facilitar la compra, incluyendo tarjetas de crédito, débito y monederos electrónicos.

1.4. ¿Qué no hace el sistema?

- **No proporciona soporte técnico**: El sistema no ofrece asistencia técnica directa para problemas de hardware o software de las consolas Nintendo.
- **No ofrece juegos físicos**: No vende juegos en formato físico, ni permite el canje de códigos de juegos físicos.
- **No incluye juegos de otras plataformas**: La eShop está limitada a juegos y contenido

desarrollado y distribuido por Nintendo, por lo que no incluye títulos de otras compañías o plataformas.

- **No permite la comunicación directa entre usuarios**: Aunque puede incluir reseñas y comentarios, no permite la mensajería o comunicación en tiempo real entre usuarios dentro de la plataforma.
- **No proporciona servicios de streaming de juegos**: La eShop no ofrece la opción de jugar juegos en streaming, ya que todos los juegos deben ser descargados en la consola para su uso.

1.5. Personas

1.5.1. Persona Positiva

Nombre: Alex Rodríguez - **Edad**: 25 años - **Ocupación**: Estudiante universitario de diseño gráfico - **Ubicación**: Ciudad de México, México - **Objetivos**: - Quiere comprar y jugar a los últimos lanzamientos de juegos de Nintendo para disfrutar en su tiempo libre y compartir experiencias con amigos. - Busca descuentos y promociones en juegos digitales. - Está interesado en explorar contenido adicional, como DLCs y complementos para los juegos que ya posee. - **Comportamiento**: - Activo en redes sociales, sigue a desarrolladores y comunidades de videojuegos. - Participa en foros y grupos de discusión sobre videojuegos. - Compara precios en diferentes plataformas antes de realizar una compra. - **Motivaciones**: - Valora la calidad y la innovación en los videojuegos. - Prefiere adquirir juegos digitales por la comodidad y la rapidez de acceso. - Busca una experiencia de usuario fluida y atractiva en la plataforma de compra.

1.5.2. Persona Negativa

Nombre: Laura Sánchez - **Edad**: 42 años - **Ocupación**: Profesora de educación primaria - **Ubicación**: Buenos Aires, Argentina - **Objetivos**: - Busca opciones de entretenimiento para sus hijos, pero no está interesada en juegos para sí misma. - Prefiere juegos gratuitos o de bajo costo. - **Comportamiento**: - No utiliza redes sociales y prefiere el contacto cara a cara para resolver problemas. - No tiene experiencia en compras en línea y se siente incómoda realizando transacciones digitales. - **Motivaciones**: - Le preocupa la seguridad de las compras en línea. - No está dispuesta a pagar por juegos digitales, prefiriendo los productos físicos. - Valora las recomendaciones de amigos y familiares, pero desconfía de las ofertas en línea.

1.6. Diagramas de Arquitectura

1.6.1. Diagrama de Contexto

Figure 1. C4

1.6.2. Diagrama de Contenedor

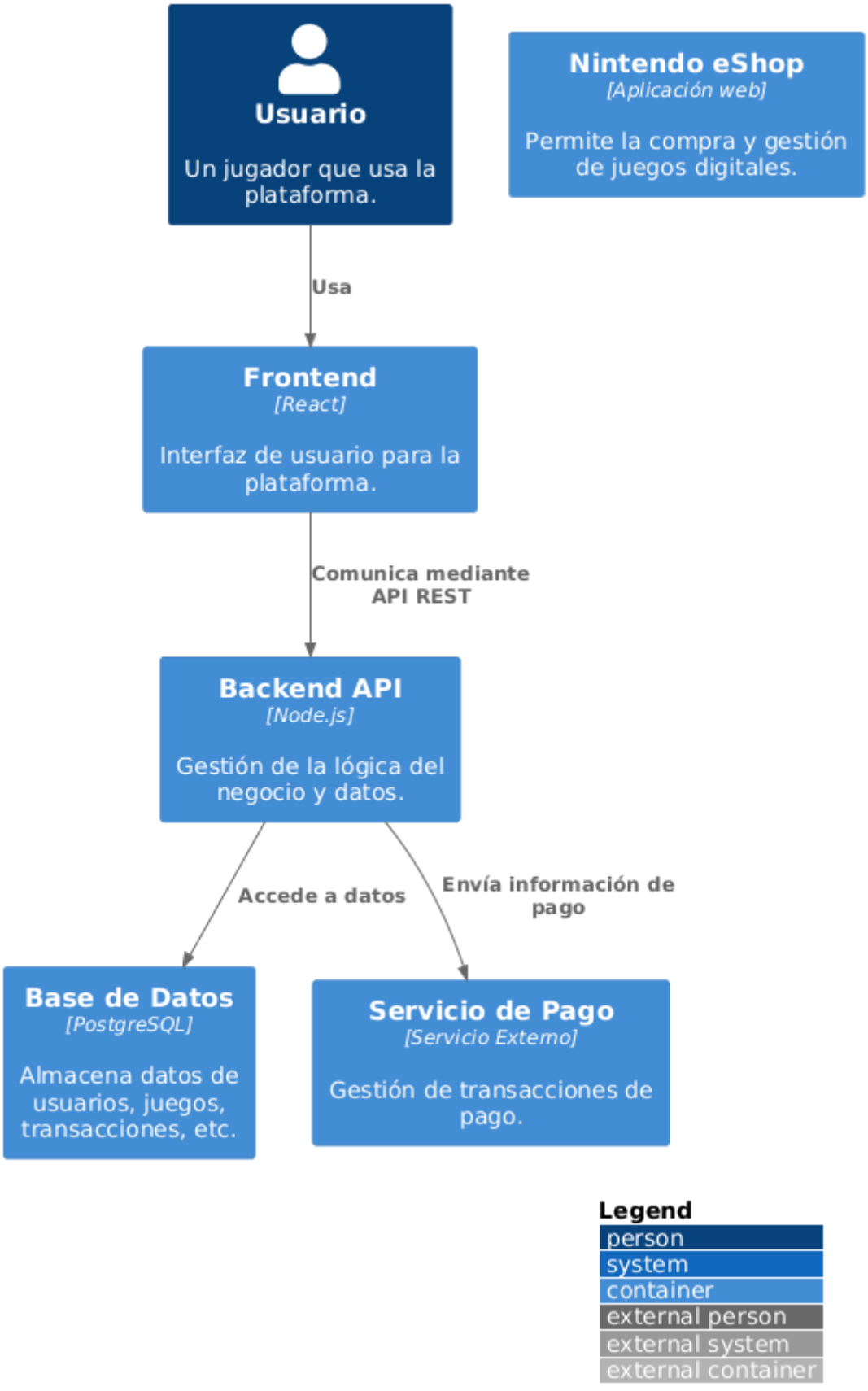

Figure 2. C4

1.6.3. Diagrama de Componente (Backend API)

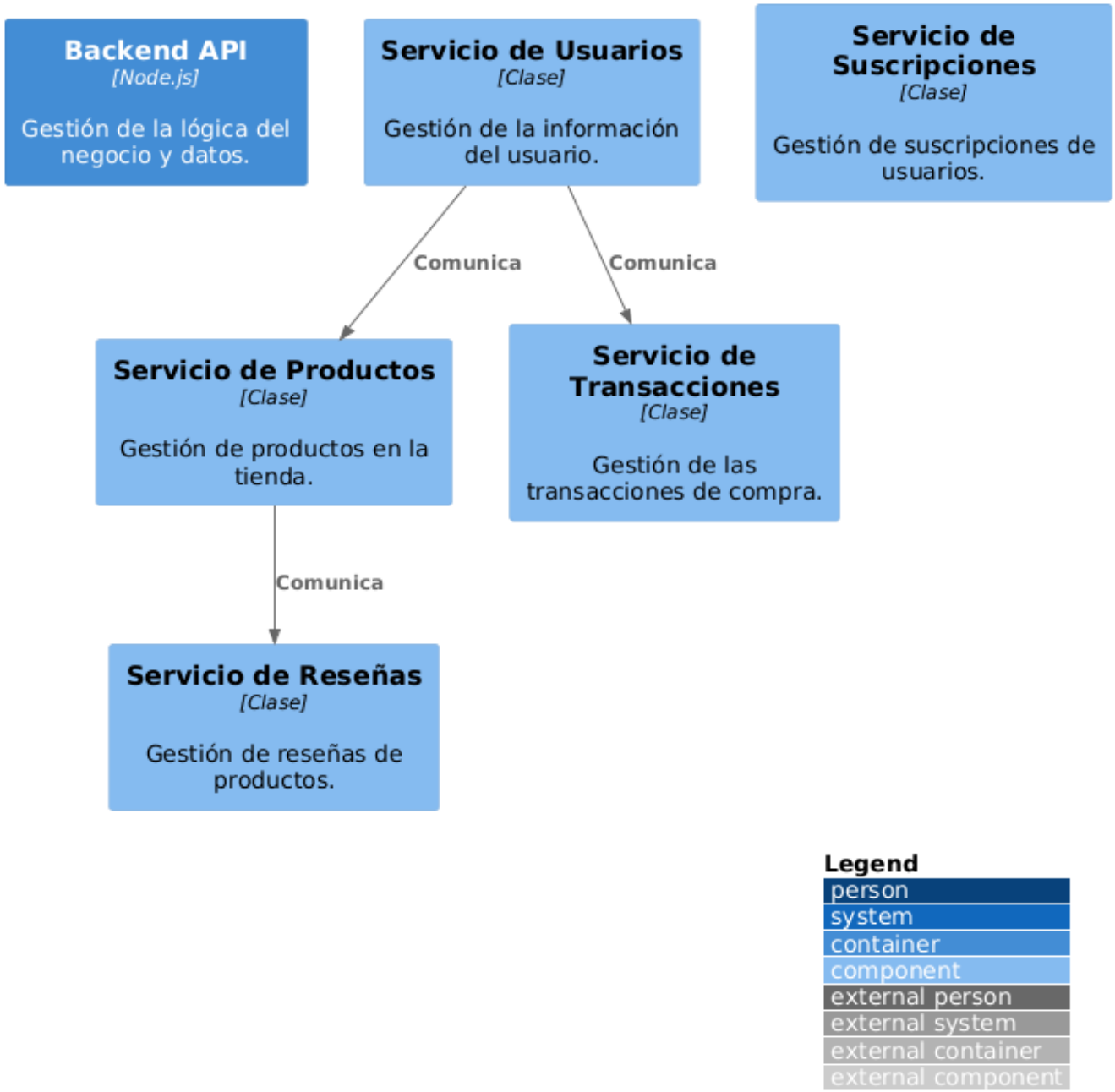

Figure 3. C4

1.7. Architecture Decision Record (ADR)

Título: Selección de Servidor para el Proyecto Nintendo eShop **Fecha**: 17 de octubre de 2024

1.7.1. Contexto

La plataforma **Nintendo eShop** requiere una infraestructura de servidor que soporte hasta 10 millones de transacciones diarias, garantizando una alta disponibilidad, rendimiento óptimo y escalabilidad. Además, debe ofrecer soporte para el despliegue de servicios backend, bases de datos, y microservicios relacionados con la gestión de usuarios, transacciones, catálogos de productos, y sistemas de pago.

El entorno incluye la ejecución de aplicaciones de frontend y backend, almacenamiento de grandes

cantidades de datos transaccionales y productos, así como la necesidad de contar con herramientas de monitorización, escalabilidad automática, y políticas de seguridad avanzadas.

1.7.2. Problema

El desafío es seleccionar un proveedor de servidor y una arquitectura que: - **Escale fácilmente** para manejar grandes volúmenes de usuarios y transacciones. - Ofrezca **alta disponibilidad** y recuperación rápida ante fallos. - Sea capaz de soportar tanto la **infraestructura backend** como el **frontend**. - Ofrezca un **coste eficiente** en diferentes escenarios de tráfico: bajo, normal, y crítico. - **Integre herramientas de DevOps** como CI/CD y permita automatización en los despliegues.

Se evaluaron varias opciones de infraestructura de servidor, considerando sus capacidades, costos, y la facilidad de integración con las tecnologías seleccionadas.

1.7.3. Opciones Evaluadas

1. **Amazon Web Services (AWS)**:
    - **Ventajas**:
    - **Alta escalabilidad** con servicios como EC2 para instancias de computación, RDS para bases de datos relacionales y S3 para almacenamiento de archivos.
    - Herramientas avanzadas como **Auto Scaling** y **Elastic Load Balancing** para manejar picos de tráfico de forma automática.
    - **Amplio ecosistema** de servicios complementarios como Lambda (para microservicios serverless), CloudFront (CDN), y Route 53 (gestión de DNS).
    - **Seguridad avanzada** y cumplimiento de estándares como PCI-DSS, esencial para sistemas de pago.
    - Gran **comunidad y soporte**, facilitando la resolución de problemas y la integración con herramientas de DevOps.
    - **Desventajas**:
    - Complejidad de uso para equipos con poca experiencia en AWS.
    - Costos que pueden aumentar rápidamente si no se gestionan eficientemente los recursos.
2. **Google Cloud Platform (GCP)**:
    - **Ventajas**:
    - Fuerte integración con **herramientas de análisis de datos** y **machine learning**, útiles para futuras expansiones en análisis de comportamiento de usuarios.
    - Servicios como **Compute Engine** y **Kubernetes Engine** para manejar aplicaciones en contenedores.
    - **Cloud Spanner** para bases de datos globales escalables, ideal para manejar grandes volúmenes de transacciones.
    - **Red global rápida** que puede reducir la latencia en aplicaciones de alto tráfico.
    - **Desventajas**:
    - Ecosistema algo menos maduro en comparación con AWS, especialmente en términos de

servicios secundarios.

- Costos ligeramente superiores en algunos servicios de alto rendimiento.

3. **Microsoft Azure**:
   - **Ventajas**:
   - Excelente opción para organizaciones que ya utilizan productos de Microsoft, como **Active Directory** o **Office 365**.
   - Herramientas robustas de integración como **Azure DevOps** para CI/CD.
   - Soporte para **Azure Functions** (serverless) y **Kubernetes Services**, similares a los ofrecidos por AWS y GCP.
   - **Opciones flexibles de pago** y soporte a largo plazo.
   - **Desventajas**:
   - Curva de aprendizaje elevada para equipos no familiarizados con el ecosistema de Microsoft.
   - Rendimiento de base de datos no tan optimizado como AWS en casos de tráfico extremadamente alto.

1.7.4. Decisión

Se ha decidido utilizar **Amazon Web Services (AWS)** como la plataforma de servidor para el proyecto Nintendo eShop. Esta decisión se basa en los siguientes criterios:

1. **Escalabilidad y Elasticidad**: AWS es altamente escalable, permitiendo gestionar grandes volúmenes de tráfico mediante **Auto Scaling** y **Elastic Load Balancing**. Esto es esencial dado que el sistema debe soportar hasta 10 millones de transacciones diarias, con la capacidad de incrementar o reducir los recursos en función de la demanda.
2. **Ecosistema Amplio**: AWS ofrece una **amplia gama de servicios** que no solo cubren necesidades básicas como el alojamiento y la computación, sino también otras áreas críticas como **seguridad**, **monitorización**, y **almacenamiento de datos**. El uso de **Amazon RDS** y **Amazon DynamoDB** como servicios de base de datos garantiza un rendimiento óptimo y gestión de datos eficiente, mientras que **S3** es ideal para almacenamiento de contenido multimedia como juegos y catálogos.
3. **Herramientas DevOps**: AWS proporciona una serie de servicios integrados para la **automatización de despliegues** y **gestión de infraestructura como código** a través de **AWS CodePipeline** y **CloudFormation**, lo que facilita la adopción de prácticas de DevOps y asegura un desarrollo ágil y controlado.
4. **Seguridad y Cumplimiento**: AWS tiene un historial comprobado de conformidad con regulaciones como **PCI-DSS**, lo que es crítico para la integración de sistemas de pago seguros y la protección de los datos de los usuarios.
5. **Soporte Global y Fiabilidad**: Con una **red de distribución global** a través de regiones y zonas de disponibilidad, AWS ofrece redundancia y capacidad de recuperación ante desastres, garantizando que el sistema esté siempre disponible para los usuarios a nivel global.

1.7.5. Consecuencias

- **Positivas**:
- La plataforma será fácilmente escalable para manejar picos de tráfico, reduciendo la posibilidad de tiempo de inactividad.
- La amplia gama de herramientas AWS permitirá una integración sencilla con otras tecnologías y una operación eficiente del sistema.
- La alta disponibilidad y la capacidad de recuperación ante fallos garantizan que los servicios estén disponibles de manera continua, minimizando el riesgo de interrupciones.
- **Negativas**:
- La gestión de costos en AWS puede ser compleja y podría generar sobrecostos si no se gestionan eficientemente los recursos asignados.
- AWS puede ser desafiante para equipos con poca experiencia en sus herramientas, lo que requerirá una capacitación adicional.

1.8. Amazon Web Services (AWS)

AWS es una de las soluciones más populares y robustas para grandes proyectos de Ecommerce debido a su capacidad de escalado, su gran variedad de servicios y su infraestructura global.

- **Ventajas**:
- **Escalabilidad automática**: Servicios como **EC2** y **Elastic Load Balancing** permiten que la plataforma escale automáticamente para manejar aumentos repentinos en el tráfico.
- **Disponibilidad global**: AWS ofrece data centers en todo el mundo, lo que permite implementar una arquitectura distribuida geográficamente, reduciendo la latencia y mejorando la disponibilidad.
- **Seguridad avanzada**: AWS incluye protección contra ataques DDoS, cifrado de datos, y políticas de seguridad estrictas.
- **Integraciones nativas**: Herramientas como **AWS Lambda** permiten construir microservicios serverless, y con **AWS RDS** puedes gestionar bases de datos de manera eficiente.
- **Desventajas**:
- **Costo**: Aunque ofrece flexibilidad en el pago por uso, los costos de AWS pueden aumentar rápidamente con grandes volúmenes de tráfico y transacciones.
- **Curva de aprendizaje**: La administración de AWS puede ser compleja, especialmente si no se cuenta con personal experimentado.
- **Conclusión**: AWS es ideal para proyectos con un presupuesto alto, donde la escalabilidad y la fiabilidad son críticas. Su capacidad para manejar grandes volúmenes de transacciones lo convierte en una opción sólida para la Nintendo eShop.

1.9. Google Cloud Platform (GCP)

GCP es otra opción líder en infraestructura en la nube, con un enfoque en big data, inteligencia

artificial y un ecosistema de servicios altamente integrados.

- **Ventajas**:
- **Rendimiento optimizado**: Google Cloud ofrece latencias muy bajas y una red global privada, lo que es ventajoso para ofrecer tiempos de respuesta rápidos.
- **Machine learning e IA**: GCP tiene servicios avanzados de inteligencia artificial que pueden integrarse para ofrecer recomendaciones personalizadas a los usuarios de la eShop.
- **Escalabilidad**: Al igual que AWS, GCP permite escalar automáticamente los servicios con **Google Kubernetes Engine (GKE)** y **Compute Engine**.
- **Precios competitivos**: Ofrece precios más bajos en algunos casos, con una política de descuentos por uso continuo que puede ser útil para operaciones a largo plazo.
- **Desventajas**:
- **Ecosistema limitado**: Aunque potente, el ecosistema de herramientas y servicios integrados es menos diverso en comparación con AWS.
- **Adopción corporativa más baja**: GCP es menos común entre empresas tradicionales, lo que podría dificultar la contratación de expertos con experiencia directa.
- **Conclusión**: GCP es una gran opción si se quiere aprovechar el análisis de datos y machine learning para mejorar la experiencia de los usuarios. También es ideal para proyectos que requieren alta disponibilidad global con costos competitivos.

1.10. Microsoft Azure

Azure es otra plataforma en la nube confiable, especialmente para empresas que ya tienen servicios basados en tecnologías Microsoft, como .NET o Azure Active Directory.

- **Ventajas**:
- **Integración empresarial**: Azure es una excelente opción si ya se utilizan productos de Microsoft, ofreciendo una integración fluida con servicios como **Active Directory** y herramientas empresariales de Microsoft.
- **Escalabilidad y flexibilidad**: Al igual que AWS y GCP, Azure proporciona autoescalado con **Azure VM Scale Sets** y **Azure Kubernetes Service (AKS)**.
- **Red global**: Con una infraestructura global sólida, Azure garantiza baja latencia para usuarios distribuidos en diferentes regiones.
- **Opciones híbridas**: Azure se destaca en ofrecer soluciones híbridas, lo que permite combinar servidores en la nube y on-premise para empresas que necesitan un enfoque mixto.
- **Desventajas**:
- **Complejidad en costos**: La estructura de precios de Azure puede ser difícil de prever, lo que requiere una planificación cuidadosa para evitar sobrecostos.
- **Menor popularidad**: Aunque Azure es popular, todavía está detrás de AWS en términos de cuota de mercado y soporte de comunidad.
- **Conclusión**: Azure es ideal si ya estás utilizando o planeas utilizar otros servicios de Microsoft, y si deseas flexibilidad en la integración con sistemas on-premise. Es una opción sólida para el

Ecommerce a gran escala.

1.11. Opciones de Infraestructura On-premise

Aunque el uso de soluciones en la nube es el enfoque más común hoy en día, también es posible considerar servidores on-premise (propios o alquilados) si se requiere un control total sobre la infraestructura.

- **Ventajas**:
- **Control completo**: Al gestionar directamente la infraestructura, la empresa tiene un control total sobre las configuraciones, políticas de seguridad y la optimización del rendimiento.
- **Costos fijos**: Una vez que se realiza la inversión inicial, los costos son más predecibles a largo plazo, aunque pueden aumentar en mantenimiento.
- **Personalización**: Puedes ajustar cada aspecto de la infraestructura a las necesidades específicas del proyecto sin depender de proveedores externos.
- **Desventajas**:
- **Altos costos iniciales**: La inversión en hardware, espacio físico, y recursos humanos es significativa, lo que hace que sea una opción más costosa a corto plazo.
- **Escalabilidad limitada**: A diferencia de la nube, escalar una infraestructura on-premise puede ser un proceso más lento y costoso.
- **Mantenimiento continuo**: Se necesita un equipo interno especializado para gestionar y mantener los servidores.
- **Conclusión**: Los servidores on-premise solo son recomendables si se necesita un control extremo sobre la infraestructura o si la empresa ya tiene una infraestructura física sólida. No es la opción más flexible ni escalable para un Ecommerce que espera grandes volúmenes de transacciones.

1.12. Soluciones Híbridas (Multi-cloud)

Una tendencia creciente es utilizar una estrategia **multi-cloud** o **híbrida**, combinando varios proveedores de nube o una mezcla de infraestructura on-premise con servicios en la nube.

- **Ventajas**:
- **Resiliencia**: Al distribuir las cargas de trabajo entre diferentes proveedores, se reduce el riesgo de una caída total del sistema.
- **Optimización de costos**: Se pueden aprovechar los mejores precios de cada proveedor y distribuir las cargas según sus fortalezas.
- **Flexibilidad**: La empresa no se "encierra" con un solo proveedor, lo que permite cambiar de estrategia si alguna plataforma ya no es adecuada.
- **Desventajas**:
- **Complejidad de gestión**: Manejar múltiples entornos puede aumentar la complejidad técnica y requerir mayores recursos humanos y de gestión.

- **Sinergias limitadas**: Aunque las soluciones multi-cloud ofrecen flexibilidad, no todas las herramientas y servicios están optimizados para funcionar juntos entre diferentes proveedores.

Chapter 2. Diseño de Base de Datos

El diseño de la base de datos para la Nintendo eShop debe ser capaz de manejar un gran volumen de transacciones, usuarios, productos y suscripciones. Se presenta una propuesta de modelo relacional que cubre las necesidades clave de la plataforma.

2.1. Requisitos de la Base de Datos

1. **Escalabilidad y rendimiento**: Dado que se estima un volumen de hasta 10 millones de transacciones diarias, es crucial que el diseño soporte un crecimiento continuo y consultas eficientes.
2. **Consistencia y seguridad**: Las transacciones financieras deben garantizar la integridad y consistencia de los datos para evitar duplicaciones o errores.
3. **Soporte para múltiples tipos de productos y servicios**: Juegos, DLCs, suscripciones, y productos relacionados.
4. **Optimización de búsquedas**: Búsquedas rápidas y filtros sobre el catálogo de productos (por género, precio, disponibilidad, etc.).
5. **Historial de transacciones y usuarios**: Almacenamiento del historial de compras, suscripciones, y detalles de los usuarios.

2.2. Esquema Relacional Propuesto

2.2.1. Tablas Principales

El diseño principal de la base de datos se basa en una arquitectura relacional con las siguientes entidades clave:

1. **Tabla** `Users` (Usuarios)
    - Esta tabla almacena la información de los usuarios registrados en la plataforma.
    - **Campos**:
    - `user_id` (PK, int, auto-increment): Identificador único del usuario.
    - `username` (varchar, único): Nombre de usuario.
    - `email` (varchar, único): Dirección de correo electrónico.
    - `password` (varchar): Hash de la contraseña del usuario.
    - `subscription_status` (enum: 'active', 'inactive', 'expired'): Estado de la suscripción.
    - `created_at` (timestamp): Fecha de creación del perfil.
    - `last_login` (timestamp): Último inicio de sesión.
    - `country` (varchar): País del usuario, importante para localizar servicios y precios regionales.
2. **Tabla** `Products` (Productos)
    - Almacena todos los juegos, DLCs y demás productos que la Nintendo eShop tiene

disponibles.

- **Campos**:
- `product_id` (PK, int, auto-increment): Identificador único del producto.
- `title` (varchar): Nombre del juego o producto.
- `description` (text): Descripción del producto.
- `price` (decimal): Precio del producto.
- `category_id` (FK, int): Categoría a la que pertenece el producto.
- `release_date` (date): Fecha de lanzamiento del juego.
- `stock` (int): Cantidad disponible (si aplica, en el caso de productos digitales generalmente ilimitado).
- `rating` (decimal): Clasificación promedio de los usuarios.

3. **Tabla `Categories`** (Categorías)
   - Almacena las categorías de productos (como géneros de juegos, tipos de productos, etc.).
   - **Campos**:
   - `category_id` (PK, int, auto-increment): Identificador único de la categoría.
   - `name` (varchar): Nombre de la categoría (ej. "Aventura", "Acción", "RPG").
   - `description` (text): Descripción de la categoría.
4. **Tabla `Transactions`** (Transacciones)
   - Almacena todas las transacciones realizadas por los usuarios.
   - **Campos**:
   - `transaction_id` (PK, int, auto-increment): Identificador único de la transacción.
   - `user_id` (FK, int): Referencia al usuario que realizó la compra.
   - `total_amount` (decimal): Monto total de la transacción.
   - `payment_method` (varchar): Método de pago utilizado.
   - `status` (enum: 'pending', 'completed', 'failed'): Estado de la transacción.
   - `transaction_date` (timestamp): Fecha y hora de la transacción.
5. **Tabla `Transaction_Details`** (Detalles de la Transacción)
   - Detalla los productos adquiridos en cada transacción.
   - **Campos**:
   - `transaction_detail_id` (PK, int, auto-increment): Identificador único del detalle de transacción.
   - `transaction_id` (FK, int): Identificador de la transacción principal.
   - `product_id` (FK, int): Producto adquirido.
   - `quantity` (int): Cantidad de productos adquiridos (para productos físicos o suscripciones).
6. **Tabla `Subscriptions`** (Suscripciones)

- Almacena la información sobre las suscripciones de los usuarios al servicio online.
- **Campos**:
- `subscription_id` (PK, int, auto-increment): Identificador único de la suscripción.
- `user_id` (FK, int): Identificador del usuario.
- `subscription_type` (varchar): Tipo de suscripción (por ejemplo, "mensual", "anual").
- `start_date` (date): Fecha de inicio de la suscripción.
- `end_date` (date): Fecha de finalización de la suscripción.
- `status` (enum: 'active', 'expired', 'cancelled'): Estado de la suscripción.

7. **Tabla `Reviews`** (Reseñas de Productos)
   - Permite a los usuarios dejar opiniones y calificaciones de los productos que han comprado.
   - **Campos**:
   - `review_id` (PK, int, auto-increment): Identificador único de la reseña.
   - `user_id` (FK, int): Referencia al usuario que realizó la reseña.
   - `product_id` (FK, int): Referencia al producto reseñado.
   - `rating` (int): Puntuación otorgada al producto (por ejemplo, entre 1 y 5 estrellas).
   - `review` (text): Texto de la reseña.
   - `created_at` (timestamp): Fecha en que se dejó la reseña.

2.2.2. Diagrama de Base de Datos

DBML

```
Failed to generate image: dbml-renderer failed: Could not parse input at line 12:28.
Expected "/*", "//", "[", "\n", "\r", "}", [a-zA-Z0-9_(),[\]], space, or whitespace
but "'" found.

Project Nintendo_eShop {
  database_type: "PostgreSQL"
}

// Tablas

Table Users {
  user_id int [pk, increment] // Identificador único del usuario
  username varchar [unique] // Nombre de usuario
  email varchar [unique] // Dirección de correo electrónico
  password varchar // Hash de la contraseña del usuario
  subscription_status enum('active', 'inactive', 'expired') // Estado de la
suscripción
  created_at timestamp // Fecha de creación del perfil
  last_login timestamp // Último inicio de sesión
  country varchar // País del usuario
}
```

```
Table Products {
  product_id int [pk, increment] // Identificador único del producto
  title varchar // Nombre del juego o producto
  description text // Descripción del producto
  price decimal // Precio del producto
  category_id int  // Categoría a la que pertenece el producto
  release_date date // Fecha de lanzamiento del juego
  stock int // Cantidad disponible
  rating decimal // Clasificación promedio de los usuarios
}

Table Categories {
  category_id int [pk, increment] // Identificador único de la categoría
  name varchar [unique] // Nombre de la categoría
  description text // Descripción de la categoría
}

Table Transactions {
  transaction_id int [pk, increment] // Identificador único de la transacción
  user_id int  // Referencia al usuario que realizó la compra
  total_amount decimal // Monto total de la transacción
  payment_method varchar // Método de pago utilizado
  status enum('pending', 'completed', 'failed') // Estado de la transacción
  transaction_date timestamp // Fecha y hora de la transacción
}

Table Transaction_Details {
  transaction_detail_id int [pk, increment] // Identificador único del detalle de
transacción
  transaction_id int  // Identificador de la transacción principal
  product_id int  // Producto adquirido
  quantity int // Cantidad de productos adquiridos
}

Table Subscriptions {
  subscription_id int [pk, increment] // Identificador único de la suscripción
  user_id int  // Identificador del usuario
  subscription_type varchar // Tipo de suscripción
  start_date date // Fecha de inicio de la suscripción
  end_date date // Fecha de finalización de la suscripción
  status enum('active', 'expired', 'cancelled') // Estado de la suscripción
}

Table Reviews {
  review_id int [pk, increment] // Identificador único de la reseña
  user_id int  // Referencia al usuario que realizó la reseña
  product_id int  // Referencia al producto reseñado
  rating int // Puntuación otorgada al producto
  review text // Texto de la reseña
  created_at timestamp // Fecha en que se dejó la reseña
```

```
}

// Relaciones

Ref: Users.user_id > Transactions.user_id
Ref: Products.product_id > Transaction_Details.product_id
Ref: Transactions.transaction_id > Transaction_Details.transaction_id
Ref: Users.user_id > Subscriptions.user_id
Ref: Products.category_id > Categories.category_id
Ref: Users.user_id > Reviews.user_id
Ref: Products.product_id > Reviews.product_id
```

2.3. Optimización y Escalabilidad

1. **Particionamiento**: Para mejorar el rendimiento, es recomendable particionar las tablas de transacciones y usuarios por regiones geográficas o por rango temporal. Esto permitirá distribuir la carga de trabajo y reducir la latencia.
2. **Caché**: Implementar un sistema de caché (con Redis o Memcached) para almacenar temporalmente consultas frecuentes, como búsquedas de productos populares o perfiles de usuarios, reduciendo la carga en la base de datos principal.
3. **Replicación de Base de Datos**: Utilizar bases de datos replicadas para manejar las consultas de lectura en regiones geográficamente distribuidas, mejorando la disponibilidad y reduciendo los tiempos de respuesta.
4. **Índices**: Añadir índices a los campos que serán utilizados frecuentemente en las búsquedas, como `user_id`, `product_id`, `transaction_date`, y `category_id`. Esto optimizará las consultas sobre las tablas.

2.4. Elección de Sistema de Base de Datos

Para una plataforma de la magnitud de la Nintendo eShop, un sistema de base de datos relacional robusto y escalable es esencial. Las siguientes opciones se consideran ideales:

- **PostgreSQL**: Ofrece características avanzadas como particionamiento nativo y soporte para JSON, lo que puede ser útil si es necesario almacenar datos semiestructurados. PostgreSQL es también altamente escalable y tiene una gran comunidad de soporte.
- **MySQL**: Es una opción ampliamente utilizada en Ecommerce, conocida por su velocidad en lecturas y su robustez. Con la configuración adecuada, MySQL puede manejar grandes volúmenes de datos.
- **Base de datos distribuida (Cassandra, CockroachDB)**: Si la necesidad de escalabilidad es extrema, una base de datos distribuida como Cassandra o CockroachDB puede manejar grandes volúmenes de datos replicados globalmente, pero requiere un mayor esfuerzo de mantenimiento y configuración. == Diseño de Endpoints REST

El diseño de los endpoints REST debe estar optimizado para manejar un gran volumen de solicitudes, permitir la integración con diferentes interfaces (web, móvil, consola) y garantizar la seguridad y el rendimiento de las transacciones. A continuación, se detallan los endpoints REST

fundamentales para el sistema de la Nintendo eShop, organizados por funcionalidad.

2.5. Consideraciones de Diseño

1. **Escalabilidad**: Dado el volumen estimado de 10 millones de transacciones diarias, es necesario implementar **pagos asíncronos**, **caché** y **limites de tasa** para evitar sobrecarga en el sistema.
2. **Seguridad**: Se debe utilizar **OAuth 2.0** o **JWT** para autenticar usuarios y proteger los datos de transacciones. Además, todos los endpoints deben comunicarse a través de **HTTPS**.
3. **Versionado**: Para mantener la compatibilidad hacia adelante, todos los endpoints deben estar versionados (ej. `/api/v1/`).

2.6. Endpoints para Usuarios

2.6.1. 1. Registro de Usuario

- **POST /api/v1/users/register**
- **Descripción**: Permite a los usuarios crear una nueva cuenta en la plataforma.

Cuerpo de la solicitud (JSON)

```
{
    "username": "string",
    "email": "string",
    "password": "string",
    "country": "string"
}
```

Respuesta (JSON)

```
{
    "user_id": 123,
    "username": "string",
    "email": "string",
    "created_at": "timestamp"
}
```

- **Consideraciones de seguridad**: Se deben realizar validaciones estrictas y almacenar la contraseña en formato hash.

2.6.2. 2. Autenticación de Usuario (Login)

- **POST /api/v1/users/login**
- **Descripción**: Autentica a los usuarios y devuelve un token de acceso (JWT) para las siguientes solicitudes.

Cuerpo de la solicitud (JSON)

```
{
    "email": "string",
    "password": "string"
}
```

Respuesta (JSON)

```
{
    "token": "JWT_token",
    "user_id": 123
}
```

- **Notas**: El token debe tener un tiempo de expiración configurable para evitar uso indebido.

2.6.3. 3. Perfil de Usuario

- **GET /api/v1/users/{user_id}**
- **Descripción**: Devuelve los detalles del perfil de usuario.
- **Cabecera**: El token JWT debe ser incluido en el encabezado.

Respuesta (JSON)

```
{
    "user_id": 123,
    "username": "string",
    "email": "string",
    "country": "string",
    "subscription_status": "active",
    "created_at": "timestamp",
    "last_login": "timestamp"
}
```

2.6.4. 4. Actualización de Perfil

- **PUT /api/v1/users/{user_id}**
- **Descripción**: Permite al usuario actualizar sus datos de perfil.

Cuerpo de la solicitud (JSON)

```
{
    "username": "string",
    "email": "string",
    "password": "string",
    "country": "string"
}
```

Respuesta (JSON)

```
{
    "message": "Profile updated successfully"
}
```

2.7. Endpoints para Productos

2.7.1. 1. Listado de Productos

- **GET /api/v1/products**
- **Descripción**: Devuelve un listado de productos disponibles en la eShop.
- **Parámetros de consulta** (opcional):
- `category`: Filtra por categoría (ej. "Acción", "Aventura").
- `sort_by`: Ordena por campos como `price`, `release_date`, `rating`.
- `page`, `limit`: Paginación.

Respuesta (JSON)

```
{
    "products": [
        {
        "product_id": 1,
        "title": "string",
        "description": "string",
        "price": 59.99,
        "category": "Acción",
        "release_date": "date",
        "rating": 4.5
        },
        {
        "product_id": 2,
        "title": "string",
        ...
        }
    ],
    "total_results": 100,
    "page": 1,
    "limit": 10
}
```

2.7.2. 2. Detalle de Producto

- **GET /api/v1/products/{product_id}**
- **Descripción**: Devuelve los detalles de un producto específico.

Respuesta (JSON)

```
{
    "product_id": 1,
    "title": "string",
    "description": "string",
    "price": 59.99,
    "category": "Acción",
    "release_date": "date",
    "rating": 4.5,
    "stock": 1000
}
```

2.8. Endpoints para Transacciones

2.8.1. 1. Realizar Compra

- **POST /api/v1/transactions**
- **Descripción**: Crea una nueva transacción de compra.

Cuerpo de la solicitud (JSON)

```
{
    "user_id": 123,
    "items": [
        {
        "product_id": 1,
        "quantity": 1
        },
        {
        "product_id": 2,
        "quantity": 2
        }
    ],
    "payment_method": "credit_card",
    "total_amount": 139.97
}
```

Respuesta (JSON)

```
{
    "transaction_id": 456,
    "status": "pending",
    "created_at": "timestamp"
}
```

- **Notas**: La transacción debe pasar por un procesamiento asíncrono para manejar el pago, lo que cambiará el estado de "pending" a "completed" o "failed".

2.8.2. 2. Ver Estado de Transacción

- **GET /api/v1/transactions/{transaction_id}**
- **Descripción**: Devuelve el estado de una transacción específica.

Respuesta (JSON)

```
{
    "transaction_id": 456,
    "user_id": 123,
    "total_amount": 139.97,
    "status": "completed",
    "transaction_date": "timestamp"
}
```

2.8.3. 3. Historial de Transacciones del Usuario

- **GET /api/v1/users/{user_id}/transactions**
- **Descripción**: Devuelve el historial de transacciones de un usuario.
- **Parámetros de consulta** (opcional):
- `status`: Filtrar por estado de la transacción (`pending`, `completed`, `failed`).
- `page`, `limit`: Paginación.

Respuesta (JSON)

```
{
    "transactions": [
        {
          "transaction_id": 456,
          "total_amount": 139.97,
          "status": "completed",
          "transaction_date": "timestamp"
        },
        {
          "transaction_id": 789,
          ...
        }
    ]
}
```

Este diseño de endpoints REST está pensado para ser modular, escalable y seguro, permitiendo que los usuarios interactúen con la Nintendo eShop de manera fluida. == Estimación de Costos

Para ofrecer una visión completa de los costos potenciales de la plataforma **Nintendo eShop** en diferentes entornos de operación, se pueden definir tres niveles de escalabilidad y complejidad: **Bajo**, **Normal** y **Crítico**. Estos entornos corresponden a diferentes escenarios de demanda, transacciones diarias y complejidad del sistema.

2.9. Entorno Bajo (Pequeña escala / Baja demanda)

En este entorno, la carga de trabajo es moderada y las necesidades de escalabilidad son mínimas. Se espera un volumen de transacciones significativamente menor que el máximo estimado de **10 millones de transacciones diarias** (probablemente entre 1 y 2 millones).

2.9.1. Infraestructura de Servidores

- **Instancias de servidores medianas** (AWS EC2, GCP o Azure): 2-4 instancias.
- **Costo de balanceadores de carga y CDN**: Menor uso de servicios de distribución.
- **Costo estimado**: $2,000 - $5,000 USD/mes.

2.9.2. Bases de Datos

- **DynamoDB para transacciones masivas**: Uso limitado.
- **PostgreSQL para transacciones críticas**: Instancias medianas.
- **Costo estimado**: $2,000 - $4,000 USD/mes.

2.9.3. Backend

- Backend en **Node.js** desplegado en pocas instancias.
- **Costo estimado**: $200 - $500 USD/mes.

2.9.4. Frontend

- **Despliegue en Vercel/Netlify** o un servicio similar con CDN básico.
- **Costo estimado**: $500 - $1,500 USD/mes.

2.9.5. Costos de Desarrollo y Personal

- Equipo reducido: 3-4 desarrolladores en total (backend, frontend, DevOps).
- **Costo estimado**: $30,000 - $50,000 USD/mes.

2.9.6. Operación Continua y Mantenimiento

- **Monitoreo** básico, menor esfuerzo en mantenimiento.
- **Costo estimado**: $5,000 - $8,000 USD/mes.

2.9.7. Costo Total Aproximado Mensual (Entorno Bajo):

- **$39,700 - $69,000 USD/mes**.

2.10. Entorno Normal (Escala intermedia / Demanda moderada)

Este escenario representa una operación más estándar para un sistema como la Nintendo eShop, con un volumen de transacciones entre **3 y 5 millones de transacciones diarias**.

2.10.1. Infraestructura de Servidores

- **Instancias más grandes** y escalables, con varias réplicas y redundancia.
- **Balanceadores de carga** y **CDN** más robustos.
- **Costo estimado**: $5,000 - $10,000 USD/mes.

2.10.2. Bases de Datos

- **DynamoDB** con mayor capacidad de lectura/escritura.
- **PostgreSQL** gestionado en RDS u otro servicio equivalente.
- **Costo estimado**: $5,000 - $8,000 USD/mes.

2.10.3. Backend

- Backend distribuido en varias instancias.
- **Costo estimado**: $500 - $1,000 USD/mes.

2.10.4. Frontend

- **Despliegue avanzado con CDN** para manejar tráfico global.
- **Costo estimado**: $1,500 - $3,000 USD/mes.

2.10.5. Costos de Desarrollo y Personal

- Equipo ampliado: 6-8 desarrolladores.
- **Costo estimado**: $50,000 - $80,000 USD/mes.

2.10.6. Operación Continua y Mantenimiento

- Mayor inversión en monitoreo, alertas y soporte continuo.
- **Costo estimado**: $8,000 - $15,000 USD/mes.

2.10.7. Costo Total Aproximado Mensual (Entorno Normal):

- **$70,000 - $117,000 USD/mes**.

2.11. Entorno Crítico (Alta Escala / Máxima demanda)

Este es el entorno más complejo y avanzado, diseñado para manejar el volumen máximo esperado de **10 millones de transacciones diarias**, con alta disponibilidad, redundancia global, y operaciones 24/7 sin interrupciones.

2.11.1. Infraestructura de Servidores

- **Instancias de alta capacidad** (dedicadas o muy grandes) y con múltiples réplicas.
- **Balanceadores de carga globales** y **CDN de alta gama** para garantizar bajas latencias.
- **Costo estimado**: $10,000 - $20,000 USD/mes.

2.11.2. Bases de Datos

- **DynamoDB** para grandes volúmenes de transacciones, con escalabilidad automática.
- **PostgreSQL** con instancias dedicadas de alto rendimiento.
- **Costo estimado**: $7,000 - $12,000 USD/mes.

2.11.3. Backend

- **Despliegue distribuido** en múltiples regiones, con microservicios y mayor capacidad.
- **Costo estimado**: $1,000 - $2,000 USD/mes.

2.11.4. Frontend

- **Despliegue en una red global** de CDN con baja latencia y gran capacidad para picos de tráfico.
- **Costo estimado**: $3,000 - $5,000 USD/mes.

2.11.5. Costos de Desarrollo y Personal

- Equipo completo: 10+ personas (backend, frontend, DevOps, QA, UX/UI).
- **Costo estimado**: $80,000 - $120,000 USD/mes.

2.11.6. Operación Continua y Mantenimiento

- Soporte continuo 24/7, con inversión en monitoreo avanzado y alertas.
- **Costo estimado**: $15,000 - $30,000 USD/mes.

2.11.7. Costo Total Aproximado Mensual (Entorno Crítico):

- **$116,000 - $189,000 USD/mes**.

2.12. Resumen de Costos

Entorno	Costo Estimado Mensual (USD)
Bajo	$39,700 - $69,000 USD/mes
Normal	$70,000 - $117,000 USD/mes
Crítico	$116,000 - $189,000 USD/mes

Chapter 3. Selección Tecnológica Backend

La elección de la tecnología backend es fundamental para garantizar que la plataforma Nintendo eShop pueda manejar el gran volumen de transacciones diarias (hasta 10 millones) y proporcione una experiencia eficiente y segura a los usuarios. A continuación se analiza si es más adecuado usar un **CMS (Content Management System)** o un backend **custom** desarrollado a medida, así como las tecnologías clave que se podrían utilizar en cada enfoque.

3.1. Architecture Decision Record (ADR)

Título: Selección Tecnológica del Backend para el Proyecto Nintendo eShop **Fecha**: 17 de octubre de 2024

3.1.1. Contexto

El proyecto Nintendo eShop es una plataforma Ecommerce destinada a manejar hasta 10 millones de transacciones diarias, permitiendo a los usuarios adquirir juegos digitales y otros servicios relacionados. El backend debe ser capaz de soportar un alto volumen de tráfico, garantizar la escalabilidad, y ofrecer baja latencia y alta disponibilidad para una experiencia fluida.

3.1.2. Problema

Se requiere tomar una decisión sobre la tecnología más adecuada para el desarrollo del backend, teniendo en cuenta: - **Alta concurrencia**: Manejo de millones de solicitudes diarias. - **Escalabilidad**: Capacidad de ajustarse al crecimiento del número de usuarios. - **Mantenibilidad**: Estructura modular que facilite la evolución del sistema. - **Integraciones**: Facilitar la conexión con servicios en la nube, bases de datos, sistemas de pago y API externas. - **Tiempo de desarrollo**: Equilibrar flexibilidad y productividad en el desarrollo.

Las opciones evaluadas fueron el uso de un **CMS** (Content Management System) de Ecommerce o un **desarrollo a medida (custom)** con tecnologías de microservicios y frameworks modernos.

3.1.3. Opciones Evaluadas

1. **CMS de Ecommerce**:
    - Herramientas como **Shopify Plus** o **Magento Enterprise** proporcionan una solución rápida de implementar con funcionalidades preconstruidas para manejar plataformas de Ecommerce.
    - **Ventajas**:
    - Rápida implementación.
    - Gestión simplificada de productos, usuarios, y pagos.
    - Escalabilidad integrada en servicios gestionados.
    - **Desventajas**:
    - Falta de flexibilidad para personalizaciones complejas.

- Limitaciones en el manejo de grandes volúmenes de transacciones simultáneas.
- Costos elevados de licencias para alto volumen.

2. **Desarrollo Custom con Node.js y Nest.js**:
   - Desarrollo a medida utilizando **Node.js** como entorno de ejecución y **Nest.js** como framework para la creación de APIs escalables.
   - **Ventajas**:
   - **Escalabilidad**: Node.js es altamente eficiente en el manejo de múltiples conexiones simultáneas, siendo ideal para una plataforma de Ecommerce con gran tráfico.
   - **Flexibilidad**: El enfoque modular de **Nest.js** permite construir un backend personalizable, optimizado y adaptado a las necesidades específicas del proyecto.
   - **Integración con servicios en la nube**: Amplia compatibilidad con plataformas como **AWS Lambda**, bases de datos como **DynamoDB** y PostgreSQL, y servicios de almacenamiento.
   - **Comunidad y soporte**: Gran ecosistema de herramientas y bibliotecas para acelerar el desarrollo.
   - **Desventajas**:
   - Mayor tiempo de desarrollo inicial en comparación con una solución CMS.
   - Requiere un equipo con conocimientos especializados en desarrollo y DevOps para manejar la infraestructura.

3.1.4. Decisión

Se ha decidido utilizar **Node.js** junto con **Nest.js** para el desarrollo del backend de la Nintendo eShop. Esta elección se justifica por las siguientes razones:

1. **Escalabilidad**: Node.js está diseñado para manejar grandes cantidades de conexiones simultáneas con eficiencia, lo que es crucial para soportar el volumen de tráfico esperado de hasta 10 millones de transacciones diarias.
2. **Flexibilidad**: La naturaleza modular de Nest.js permite desarrollar un backend a medida, optimizando la estructura de la aplicación para ofrecer servicios personalizados, como la gestión de usuarios, carrito de compras, sistemas de pago y recomendaciones personalizadas.
3. **Compatibilidad con microservicios**: Nest.js facilita la arquitectura basada en microservicios, lo que permitirá desplegar y escalar diferentes partes de la aplicación de forma independiente. Esto es especialmente útil para integrar diferentes funciones del sistema, como el procesamiento de pagos, la gestión de productos y las notificaciones.
4. **Soporte a tecnologías en la nube**: El ecosistema de Node.js y Nest.js ofrece una integración nativa con servicios de nube como **AWS Lambda**, **Google Cloud Functions**, y bases de datos distribuidas como **DynamoDB**. Esto facilitará la creación de una infraestructura en la nube altamente eficiente y escalable.
5. **Tiempos de respuesta**: Node.js, con su arquitectura basada en eventos no bloqueantes, es ideal para minimizar la latencia en aplicaciones que requieren procesamiento en tiempo real, como un Ecommerce donde los usuarios esperan respuestas inmediatas al interactuar con la plataforma.

6. **Mantenibilidad y productividad**: Nest.js sigue principios de desarrollo bien organizados y estandarizados, lo que facilita la mantenibilidad a largo plazo del código y permite que nuevos desarrolladores se integren al equipo sin dificultad.

3.1.5. Consecuencias

- **Positivas**:
- La plataforma será altamente escalable, con la capacidad de manejar millones de usuarios simultáneamente y adaptarse a futuros crecimientos.
- La arquitectura modular y flexible permitirá hacer mejoras o añadir nuevas funcionalidades sin afectar el funcionamiento general del sistema.
- La integración con servicios en la nube proporcionará un entorno robusto y altamente disponible, mejorando la experiencia del usuario.
- **Negativas**:
- Requiere un tiempo de desarrollo inicial mayor en comparación con el uso de un CMS, aunque esto se compensa a largo plazo con la flexibilidad y capacidad de personalización.
- El equipo de desarrollo necesitará experiencia avanzada en tecnologías de backend y gestión de infraestructura en la nube, lo que podría incrementar los costos iniciales de contratación y formación.

3.2. CMS (Content Management System)

Un **CMS** es una plataforma prediseñada que permite crear y gestionar sitios web con una funcionalidad estándar y extensible. Sin embargo, en el contexto de una tienda como la Nintendo eShop, los CMS convencionales pueden no ser la mejor opción, ya que tienen limitaciones en cuanto a la escalabilidad y la personalización para manejar transacciones tan masivas y complejas.

3.2.1. Ventajas

- **Implementación rápida**: Muchas funcionalidades, como la gestión de productos y usuarios, ya están preconstruidas.
- **Costos iniciales más bajos**: Los CMS son fáciles de implementar sin requerir tanto desarrollo desde cero.
- **Integraciones predefinidas**: Con módulos o plugins para integrar pasarelas de pago, seguridad, y otras funcionalidades comunes.

3.2.2. Desventajas de un CMS

- **Escalabilidad limitada**: CMS como Magento, Shopify o WooCommerce tienen problemas para manejar un alto número de transacciones simultáneas sin modificaciones costosas.
- **Rigidez**: Personalizar la lógica de negocio o adaptarse a requisitos específicos (como la gestión de suscripciones o la cantidad de usuarios simultáneos) puede ser complicado.
- **Costos de mantenimiento**: A medida que se introducen personalizaciones, el mantenimiento y la escalabilidad se vuelven más difíciles y costosos.

Por lo tanto, para un proyecto con el volumen y la complejidad de la Nintendo eShop, un CMS no sería la opción ideal si se busca optimizar a largo plazo.

3.3. Backend Custom

Un **backend custom** ofrece la flexibilidad y el control necesarios para construir una plataforma robusta y escalable desde cero. Aunque esto implica un mayor costo inicial y más tiempo de desarrollo, es la mejor opción para una plataforma que debe gestionar un alto tráfico, personalizaciones complejas, y ofrecer soporte en tiempo real a los usuarios.

3.3.1. Ventajas de un Backend Custom

- **Escalabilidad**: Se puede diseñar la infraestructura para escalar automáticamente en función de la demanda, utilizando tecnologías como microservicios y bases de datos distribuidas.
- **Control total**: La lógica de negocio puede ser completamente personalizada, lo que es esencial para funciones como la gestión de suscripciones, control de stock, o un sistema de recompensas.
- **Integración optimizada**: Se puede integrar con diferentes servicios de pago, analítica, y gestión de usuarios de manera eficiente y bajo demanda.

3.3.2. Desventajas de un Backend Custom

- **Mayor tiempo y costo inicial**: El desarrollo desde cero es más lento y costoso comparado con un CMS, aunque esto se amortiza a largo plazo por la flexibilidad.
- **Requiere un equipo especializado**: Necesitas desarrolladores con experiencia en tecnologías escalables y microservicios.

3.4. Tecnologías Sugeridas para un Backend Custom

Para un proyecto de la magnitud de la Nintendo eShop, las siguientes tecnologías fueron consideradas para el desarrollo del backend:

3.4.1. Lenguajes y Frameworks

- **Node.js** con **Express.js**:
- **Descripción**: Node.js es una opción muy popular para aplicaciones de alto rendimiento y en tiempo real. Permite manejar múltiples conexiones concurrentes gracias a su naturaleza asíncrona y orientada a eventos. Además, cuenta con una gran comunidad y ecosistema de paquetes.
- **Ventajas**:
- Excelente para manejar múltiples solicitudes simultáneas.
- Buen rendimiento para operaciones de I/O, lo que es útil para manejar transacciones.
- Amplia gama de bibliotecas disponibles para optimizar el desarrollo.
- **Java** con **Spring Boot**:
- **Descripción**: Spring Boot es un framework maduro y robusto para el desarrollo de aplicaciones

empresariales escalables. Ofrece excelentes herramientas para el manejo de transacciones, seguridad, y escalabilidad.

- **Ventajas**:
- Escalabilidad empresarial.
- Gran soporte para integraciones de bases de datos distribuidas y transacciones.
- Soporte sólido para microservicios y desarrollo modular.
- **Python** con **Django** o **Flask**:
- **Descripción**: Python es un lenguaje poderoso y versátil, especialmente útil para desarrollar APIs rápidas. Django, en particular, ofrece una arquitectura "batteries included", mientras que Flask es más ligero y flexible.
- **Ventajas**:
- Rápido desarrollo.
- Buen soporte para escalabilidad y seguridad.
- Ecosistema maduro para integraciones con bases de datos y otros servicios.

3.4.2. Bases de Datos

- **PostgreSQL**:
- **Descripción**: Es una base de datos relacional que ofrece gran flexibilidad y soporte para operaciones complejas, como las que requiere una tienda online con un gran catálogo y transacciones complejas.
- **Ventajas**:
- Soporte para replicación y particionamiento.
- Excelente manejo de datos transaccionales y consistencia ACID.
- Compatibilidad con JSON para almacenar datos semiestructurados.
- **MongoDB**:
- **Descripción**: Una base de datos NoSQL que permite almacenar datos en forma de documentos JSON. Es ideal para manejar catálogos grandes y sistemas que requieren flexibilidad de esquemas.
- **Ventajas**:
- Escalabilidad horizontal fácil.
- Flexibilidad para almacenar productos con diferentes atributos dinámicos (como descripciones, precios por región, etc.).
- **DynamoDB** (NoSQL Administrado en la Nube):
- **Descripción**: Es un servicio de base de datos noSQL ofrecido por Amazon como parte de Amazon Web Services.
- **Ventajas**:
- Escalabilidad automática: DynamoDB, como servicio administrado de AWS, está diseñado para escalar automáticamente según el tráfico, lo que lo hace ideal para cargas variables y grandes

volúmenes de transacciones.

- Alto rendimiento: DynamoDB es capaz de manejar millones de solicitudes por segundo, lo que lo convierte en una excelente opción para sistemas de Ecommerce de gran escala.
- Modelo de facturación por uso: Su modelo de pago por consumo lo hace atractivo, ya que se paga en función de las lecturas y escrituras realizadas, optimizando costos.
- Gestión simplificada: Como es un servicio totalmente administrado por AWS, no se necesita gestionar la infraestructura de la base de datos, lo que reduce la complejidad operativa.
- **Redis** (para caché y sesiones):
- **Descripción**: Una base de datos en memoria que se usa comúnmente para almacenar sesiones de usuarios, caché de productos o resultados de consultas frecuentes, lo que ayuda a reducir la carga en la base de datos principal.
- **Ventajas**:
- Tiempo de respuesta extremadamente rápido.
- Optimización de rendimiento para lecturas frecuentes.

3.4.3. Arquitectura de microservicios

- **Microservicios**:
- **Descripción**: En lugar de tener un solo servidor monolítico, dividir la aplicación en múltiples microservicios (por ejemplo, uno para usuarios, otro para transacciones, y otro para productos) permite una escalabilidad y un mantenimiento más eficientes.
- **Ventajas**:
- Cada componente puede ser desarrollado, escalado y desplegado de manera independiente.
- Mejor tolerancia a fallos y más fácil de mantener a medida que la aplicación crece.

3.4.4. Herramientas para Mensajería y Procesamiento Asíncrono

- **RabbitMQ** o **Apache Kafka**:
- **Descripción**: Sistemas de mensajería que permiten manejar de manera eficiente las transacciones asíncronas y las tareas en segundo plano, como la confirmación de pagos o el procesamiento de actualizaciones en tiempo real.
- **Ventajas**:
- Permiten procesar grandes volúmenes de eventos sin sobrecargar la aplicación principal.
- Integración sencilla con microservicios y arquitecturas distribuidas.

3.4.5. Seguridad

- **OAuth 2.0 / JWT (JSON Web Tokens)**:
- **Descripción**: Se recomienda utilizar OAuth 2.0 para manejar la autenticación y autorización de los usuarios, y JWT para emitir tokens de acceso seguros que los usuarios puedan utilizar en las sesiones.

- **Ventajas**:
- Tokens ligeros y seguros.
- Integración sencilla con APIs y microservicios.

3.4.6. Escalabilidad y Orquestación

- **Kubernetes**:
- **Descripción**: Es una herramienta de orquestación que ayuda a gestionar contenedores Docker, lo que facilita la implementación y escalado de microservicios.
- **Ventajas**:
- Facilita la implementación y el escalado automático de los microservicios.
- Buena integración con las principales nubes (AWS, Google Cloud, Azure).
- Optimización de recursos y mejor resiliencia.

3.4.7. Plataformas en la Nube

- **Amazon Web Services (AWS)**, **Google Cloud Platform (GCP)** o **Microsoft Azure**:
- **Descripción**: Proveedores de servicios en la nube que ofrecen servicios de bases de datos, almacenamiento, y escalabilidad automatizada. También proporcionan soluciones serverless como **AWS Lambda** o **Google Cloud Functions** para manejar partes específicas de la lógica de backend.
- **Ventajas**:
- Escalabilidad automática y alta disponibilidad.
- Amplias opciones de bases de datos, almacenamiento en caché, y sistemas de mensajería.
- Servicios de pago bajo demanda.

3.5. Selección

Para una plataforma como la Nintendo eShop, que requiere alta escalabilidad, control total sobre la personalización, y soporte para millones de transacciones diarias, la mejor opción es desarrollar un **backend custom**, junto con arquitecturas de **microservicios** y herramientas de orquestación como **Kubernetes**.

Node.js es ideal si se priorizan la velocidad de desarrollo, el manejo de un alto número de conexiones simultáneas y una arquitectura basada en microservicios. Es una opción sólida para el backend de la Nintendo eShop.

La combinación de DynamoDB para datos de alto volumen y consultas rápidas, con PostgreSQL para datos estructurados y transacciones críticas, es la mejor opción para la Nintendo eShop: - DynamoDB: Manejará las transacciones relacionadas con productos, usuarios, inventarios y todas las lecturas y escrituras de alto volumen. Es escalable, completamente administrado, y capaz de manejar el tráfico masivo esperado. - PostgreSQL: Se utilizará para manejar las transacciones críticas y las operaciones que requieren ACID y consistencia fuerte, como los pagos y registros de

facturación. Esta combinación ofrece la flexibilidad y escalabilidad de NoSQL, junto con la consistencia y transacciones seguras de una base de datos relacional, optimizando el rendimiento y la seguridad del proyecto.

Este enfoque proporcionará la flexibilidad, el rendimiento y la seguridad necesarios para construir una plataforma sólida y confiable a largo plazo. == Selección Tecnológica Frontend

El frontend de la Nintendo eShop debe ofrecer una experiencia de usuario rápida, fluida y visualmente atractiva, tanto en navegadores web como en dispositivos móviles y consolas. La elección de las tecnologías frontend debe enfocarse en la **experiencia de usuario (UX)**, **rendimiento**, **mantenibilidad** y **escalabilidad**. A continuación, se detallan las opciones tecnológicas más adecuadas para construir la interfaz de usuario de la eShop.

3.6. Architecture Decision Record (ADR)

Título: Selección Tecnológica del Frontend para el Proyecto Nintendo eShop **Fecha**: 17 de octubre de 2024

3.6.1. Contexto

El frontend del proyecto **Nintendo eShop** es una plataforma que permite a los usuarios interactuar con la tienda de manera eficiente, adquirir juegos digitales y gestionar sus cuentas. El frontend debe ser capaz de ofrecer una **experiencia de usuario fluida, rápida y atractiva**, considerando la posibilidad de altos volúmenes de usuarios simultáneos y la necesidad de compatibilidad entre dispositivos. La plataforma también debe ser escalable, accesible y fácilmente mantenible.

3.6.2. Problema

El principal desafío es seleccionar una tecnología que: - Proporcione una **experiencia de usuario rápida** y **optimizada**. - Garantice la **escalabilidad** para soportar millones de usuarios. - Sea fácil de mantener y permita la **adición de nuevas características** de forma ágil. - Permita una integración fluida con el backend basado en **Node.js y Nest.js**. - Proporcione **compatibilidad multiplataforma**, especialmente en dispositivos móviles y web.

Se evaluaron varias opciones para la tecnología frontend, considerando su capacidad para abordar estos desafíos.

3.6.3. Opciones Evaluadas

1. **React.js** con **Next.js**:
    - React es una biblioteca de JavaScript popular para construir interfaces de usuario dinámicas y Next.js es un framework para aplicaciones web estáticas y dinámicas.
    - **Ventajas**:
    - **SSR (Server-Side Rendering)**: Next.js proporciona SSR, lo que mejora la velocidad de carga y optimiza el SEO.
    - **Componentes reutilizables**: El uso de componentes en React facilita la modularidad y la reutilización de código.

- **Amplio ecosistema**: Gran cantidad de herramientas, bibliotecas y soporte de la comunidad.
- **Escalabilidad**: React y Next.js permiten manejar interfaces complejas y de alto tráfico.
- **Desventajas**:
- Mayor curva de aprendizaje para desarrolladores que no estén familiarizados con React.
- Requiere un poco más de configuración comparado con frameworks más simples.

2. **Vue.js** con **Nuxt.js**:
   - Vue es otro framework popular de JavaScript, y Nuxt.js ofrece beneficios similares a Next.js en el ecosistema Vue.
   - **Ventajas**:
   - Curva de aprendizaje más amigable para desarrolladores nuevos.
   - Soporta SSR, optimizando SEO y tiempos de carga.
   - **Flexibilidad** para construir interfaces reactivas y escalables.
   - **Desventajas**:
   - Comunidad y ecosistema algo más pequeños que los de React.
   - Menos herramientas avanzadas para grandes volúmenes de usuarios.
3. **Angular**:
   - Angular es un framework completo para construir aplicaciones web dinámicas con soporte empresarial.
   - **Ventajas**:
   - Gran capacidad para manejar aplicaciones empresariales y grandes.
   - Ofrece una arquitectura robusta con herramientas integradas como el enrutamiento y la gestión de estado.
   - Facilita el desarrollo de aplicaciones de gran escala.
   - **Desventajas**:
   - Curva de aprendizaje pronunciada debido a su complejidad.
   - Menor flexibilidad y rendimiento para pequeñas aplicaciones comparado con React y Vue.

3.6.4. Decisión

Se ha decidido utilizar **React.js** con **Next.js** para el desarrollo del frontend de la plataforma Nintendo eShop. Esta decisión se justifica por las siguientes razones:

1. **Escalabilidad y rendimiento**: React.js es ampliamente utilizado por grandes plataformas de Ecommerce y ha demostrado ser una tecnología confiable para manejar altos volúmenes de usuarios simultáneamente. La capacidad de **Next.js** para ofrecer **Server-Side Rendering (SSR)** y **Static Site Generation (SSG)** optimiza el rendimiento de la página y reduce los tiempos de carga, mejorando la experiencia del usuario.
2. **Modularidad y Mantenibilidad**: La naturaleza basada en componentes de React permite una fácil **modularidad** del código, lo que facilita la implementación y modificación de las distintas

secciones de la aplicación (catálogo de productos, carritos de compra, perfiles de usuario, etc.).

3. **Desempeño SEO**: La combinación de React.js con **Next.js** ofrece herramientas sólidas para mejorar el SEO, lo cual es importante para atraer usuarios y mejorar la visibilidad de la plataforma en motores de búsqueda. La capacidad de generar contenido dinámico con SSR es clave para mantener tiempos de respuesta rápidos y optimizar la indexación.
4. **Amplitud del Ecosistema**: React.js tiene una **amplia comunidad y un ecosistema de bibliotecas** que aceleran el desarrollo de nuevas funcionalidades. Esto es crucial para una plataforma como Nintendo eShop que requerirá actualizaciones continuas y mejoras en el frontend.
5. **Compatibilidad Multiplataforma**: React es altamente compatible con **dispositivos móviles** y otras plataformas, lo que permite una experiencia de usuario fluida en navegadores web y aplicaciones móviles.
6. **Facilidad de Integración con el Backend**: Dado que el backend se está desarrollando en **Node.js**, React/Next.js es una opción ideal debido a su compatibilidad con el stack de JavaScript completo, lo que facilita la integración con el backend sin necesidad de gestionar múltiples lenguajes de programación.

3.6.5. Consecuencias

- **Positivas**:
- La arquitectura basada en React y Next.js permitirá que el frontend sea fácilmente escalable, modular y flexible, garantizando un buen rendimiento incluso con millones de usuarios activos.
- La capacidad de SSR y la optimización para SEO proporcionarán una **experiencia de usuario rápida** y una mayor visibilidad en los motores de búsqueda.
- La integración con el backend basado en Node.js será fluida, reduciendo la complejidad de gestión del proyecto.
- **Negativas**:
- La curva de aprendizaje de React puede ser un desafío para los desarrolladores que no estén familiarizados con esta tecnología, lo que puede retrasar un poco el inicio del desarrollo.
- Requiere una configuración adecuada para manejar adecuadamente la gestión de estado en aplicaciones complejas, lo que puede incrementar ligeramente la complejidad del proyecto.

3.7. Requisitos

1. **Interfaz Moderna y Responsiva**: La interfaz debe ser adaptable a diferentes dispositivos y tamaños de pantalla (desktop, mobile, consola).
2. **Rendimiento**: Dado el alto tráfico esperado, el frontend debe cargar rápidamente y manejar múltiples usuarios concurrentes.
3. **Experiencia de Usuario**: Debe proporcionar una experiencia intuitiva y fluida, con animaciones, transiciones suaves y tiempos de carga mínimos.
4. **Mantenibilidad**: El código debe ser modular y fácil de mantener para futuras expansiones y actualizaciones.

5. **SEO**: La plataforma debe estar optimizada para motores de búsqueda (en caso de que se desee atraer tráfico orgánico).
6. **Accesibilidad**: Es importante que el sitio sea accesible para usuarios con diversas discapacidades.

3.8. Opciones de Frameworks y Bibliotecas

3.8.1. React.js

- **Descripción**: React es una biblioteca de JavaScript ampliamente utilizada para construir interfaces de usuario rápidas y dinámicas. Es mantenida por Facebook y tiene una gran comunidad y ecosistema.
- **Ventajas**:
- **Componentización**: Permite dividir la interfaz en componentes reutilizables, lo que mejora la mantenibilidad y modularidad.
- **Virtual DOM**: Optimiza las actualizaciones del DOM, mejorando el rendimiento en aplicaciones con gran cantidad de datos o interacciones frecuentes.
- **Ecosistema Extenso**: Tiene un amplio soporte para herramientas como Redux (para manejar el estado global), Next.js (para renderizado en servidor y optimización SEO), y una variedad de bibliotecas para gestionar formularios, enrutamiento, etc.
- **SSR (Server-Side Rendering)**: Con Next.js, React permite el renderizado del lado del servidor para mejorar la velocidad de carga inicial y optimizar SEO.
- **Flexibilidad**: Facilita la integración con otras tecnologías, permitiendo escalar la complejidad de la aplicación conforme crezca.

3.8.2. Vue.js

- **Descripción**: Vue es un framework progresivo de JavaScript que se enfoca en ser sencillo de integrar en proyectos nuevos o existentes, con un enfoque en la vista (frontend).
- **Ventajas**:
- **Simplicidad y Flexibilidad**: Es fácil de aprender y usar, permitiendo desarrollar rápidamente interfaces interactivas.
- **Componentización**: Al igual que React, Vue facilita la creación de componentes reutilizables.
- **Rendimiento Ligero**: Es más ligero que otros frameworks y tiene tiempos de carga rápidos.
- **SEO-Friendly**: Con herramientas como Nuxt.js, Vue soporta renderizado del lado del servidor para optimización SEO.
- **Excelente Documentación y Comunidad**: La curva de aprendizaje es menos pronunciada comparada con React o Angular.

3.8.3. Angular

- **Descripción**: Angular es un framework desarrollado y mantenido por Google, ideal para

aplicaciones empresariales robustas. Ofrece una estructura completa para el desarrollo de aplicaciones web escalables.

- **Ventajas**:
- **Framework Completo**: Angular viene con muchas funcionalidades preconstruidas como inyección de dependencias, validaciones de formularios y enrutamiento avanzado.
- **Typescript**: Angular utiliza Typescript, lo que añade tipado estático y mejora la mantenibilidad del código en proyectos grandes.
- **Soporte Corporativo**: Es una excelente opción para proyectos empresariales a gran escala.
- **Componentización y Modularidad**: Angular también se basa en componentes, lo que facilita la reutilización y modularidad del código.

3.8.4. Comparativa

Característica	React	Vue	Angular
Curva de Aprendizaje	Media	Baja	Alta
Rendimiento	Alto	Alto	Medio
Flexibilidad	Alta	Alta	Media
Ecosistema	Amplio (con Next.js)	Amplio (con Nuxt.js)	Completo, menos flexible
Tamaño del Framework	Ligero	Ligero	Pesado
Mantenido por	Facebook	Comunidad	Google
SEO y SSR	Next.js para SSR	Nuxt.js para SSR	Renderizado por el cliente, requiere configuración extra para SEO

3.9. Librerías Complementarias

3.9.1. Redux (o Context API)

- **Descripción**: Una librería para manejar el estado global de la aplicación, esencial para una tienda online donde múltiples componentes necesitan acceder a la misma información (ej. carrito de compras, detalles del usuario).
- **Uso**:
- **Carrito de Compras**: Gestiona el estado del carrito a lo largo de la navegación del usuario.
- **Autenticación**: Maneja el estado del usuario autenticado en diferentes partes de la aplicación.

3.9.2. Axios o Fetch API

- **Descripción**: Para realizar solicitudes HTTP al backend. **Axios** es una librería popular, pero

Fetch API es nativa de los navegadores modernos y es más ligera.

- **Uso**:
- **Comunicación con APIs REST**: Recupera datos de productos, usuarios, y realiza transacciones.
- **Manejo de errores y reintentos**: Gestión eficiente de errores en las solicitudes HTTP para mejorar la experiencia de usuario.

3.9.3. Tailwind CSS o Material-UI

- **Tailwind CSS**:
- **Descripción**: Un framework CSS utilitario que permite crear interfaces rápidas y personalizadas sin necesidad de escribir mucho CSS personalizado.
- **Ventajas**:
- Muy flexible para crear diseños responsivos y personalizados.
- Menos sobrecarga de CSS y más control sobre el diseño final.
- **Material-UI**:
- **Descripción**: Un conjunto de componentes UI preconstruidos basados en el diseño Material de Google. Facilita la creación de una interfaz coherente y atractiva sin mucho esfuerzo de diseño personalizado.
- **Ventajas**:
- Fácil de usar y bien documentado.
- Proporciona una experiencia visual moderna y profesional.

3.9.4. WebSockets

- **Descripción**: Para manejar la comunicación en tiempo real, como notificaciones en vivo de compras, actualizaciones de stock o mensajes promocionales.
- **Uso**:
- **Actualización en Tiempo Real de Inventario**: Permite que el frontend se actualice sin necesidad de refrescar la página cuando los productos se agoten o se agreguen nuevas ofertas.
- **Notificaciones Push**: Enviar notificaciones sobre nuevas ofertas o cambios en la cuenta de usuario.

3.10. Otras Herramientas Clave

3.10.1. Next.js (para React)

- **Descripción**: Un framework basado en React que soporta el renderizado del lado del servidor (SSR), lo que mejora el SEO y reduce los tiempos de carga inicial.
- **Ventajas**:
- **Mejora el SEO**: SSR permite que el contenido sea visible para los motores de búsqueda desde la primera carga.

- **Optimización de rendimiento**: Pre-renderizado de páginas y gestión eficiente de rutas.
- **Incremental Static Regeneration (ISR)**: Permite la regeneración estática incremental de páginas, ideal para catálogos de productos que se actualizan constantemente.

3.10.2. Webpack o Vite

- **Descripción**: Herramientas de construcción que optimizan el código JavaScript, CSS, y otros archivos del frontend, para mejorar los tiempos de carga y reducir el tamaño de los archivos.
- **Uso**:
- **Optimización de Rendimiento**: Minificación de archivos, agrupación de dependencias y carga diferida de scripts para mejorar los tiempos de carga.

3.11. Selección

Para una plataforma como Nintendo eShop, donde se espera un alto tráfico, rendimiento ágil, y una experiencia de usuario atractiva, **React.js** es la mejor opción, especialmente si se usa con **Next.js** para aprovechar las capacidades de renderizado en servidor, lo que mejora tanto el rendimiento como el SEO. Complementado con herramientas como **Tailwind CSS** para la interfaz visual, **Redux** para el manejo del estado, y **WebSockets** para actualizaciones en tiempo real, se puede construir una plataforma robusta y escalable. == Selección de Profesionales Necesarios

El éxito del desarrollo de la plataforma de Ecommerce de Nintendo eShop depende de un equipo multidisciplinario de profesionales altamente capacitados. Dado que el proyecto implica manejar un alto volumen de transacciones (hasta 10 millones diarias), es necesario contar con especialistas en diversas áreas, tanto para el desarrollo técnico como para la operación y mantenimiento continuo de la plataforma. A continuación, se detallan los perfiles clave requeridos para cada etapa del proyecto.

3.12. Gerente de Proyecto

- **Responsabilidades**:
- Planificar, coordinar y supervisar todas las fases del proyecto, desde la conceptualización hasta la implementación.
- Gestionar los tiempos, costos y calidad del desarrollo.
- Asegurar la comunicación efectiva entre los equipos de desarrollo, diseño, negocio y marketing.
- Controlar los riesgos y gestionar los recursos de manera eficiente.
- **Perfil**:
- Experiencia en la gestión de proyectos de desarrollo de software y Ecommerce.
- Habilidades de liderazgo y organización.
- Conocimiento de metodologías ágiles (Scrum, Kanban).

3.13. Arquitecto de Software

- **Responsabilidades**:
- Diseñar la arquitectura general de la plataforma, asegurando la escalabilidad, seguridad y rendimiento del sistema.
- Tomar decisiones tecnológicas clave, como la selección de frameworks, bases de datos y herramientas de integración.
- Supervisar la correcta implementación de microservicios y asegurar que todas las partes del sistema funcionen de manera integrada.
- **Perfil**:
- Experiencia en la arquitectura de sistemas distribuidos y en el diseño de aplicaciones escalables.
- Conocimiento profundo de plataformas en la nube (AWS, GCP o Azure).
- Dominio en arquitectura de microservicios y patrones de diseño de software.

3.14. Desarrolladores Backend

- **Responsabilidades**:
- Implementar la lógica del negocio, incluyendo la creación de APIs REST, gestión de bases de datos, y la integración con servicios externos (pasarelas de pago, sistemas de notificaciones, etc.).
- Desarrollar microservicios escalables que soporten un alto volumen de transacciones.
- Asegurar la seguridad de los datos de los usuarios y las transacciones.
- **Perfil**:
- Expertos en tecnologías como Node.js, Java (Spring Boot) o Python (Django, Flask).
- Conocimientos en bases de datos relacionales (PostgreSQL, MySQL) y NoSQL (MongoDB).
- Experiencia con integración de APIs, seguridad de datos (OAuth, JWT), y optimización de rendimiento.
- Capacidad para trabajar en entornos distribuidos y con arquitecturas de microservicios.

3.15. Desarrolladores Frontend

- **Responsabilidades**:
- Implementar la interfaz de usuario de la plataforma, asegurando una experiencia de usuario rápida, atractiva y responsiva.
- Trabajar en estrecha colaboración con los diseñadores UI/UX para implementar los diseños y funcionalidades planeadas.
- Optimizar el frontend para SEO y mejorar los tiempos de carga utilizando técnicas como el renderizado del lado del servidor (SSR).
- **Perfil**:

- Expertos en React.js (preferiblemente con Next.js), Vue.js o Angular.
- Experiencia en diseño responsivo y optimización de interfaces para múltiples dispositivos.
- Conocimientos en herramientas de optimización de rendimiento (Webpack, Vite) y buenas prácticas de desarrollo CSS (Tailwind CSS o Material-UI).

3.16. Ingenieros DevOps

- **Responsabilidades**:
- Diseñar, implementar y mantener la infraestructura en la nube para asegurar la escalabilidad y disponibilidad continua de la plataforma.
- Gestionar la automatización del despliegue de microservicios, pruebas, monitoreo y seguridad.
- Asegurar la escalabilidad automática para soportar los picos de tráfico, y el monitoreo continuo para evitar tiempos de inactividad.
- **Perfil**:
- Experiencia con Kubernetes y Docker para orquestación de contenedores.
- Conocimientos en plataformas en la nube como AWS, GCP o Azure.
- Familiaridad con herramientas de CI/CD (Continuous Integration/Continuous Deployment) como Jenkins o GitLab CI.
- Habilidades en el manejo de bases de datos distribuidas y escalabilidad de infraestructuras.

3.17. Diseñador UI/UX

- **Responsabilidades**:
- Diseñar la interfaz gráfica y la experiencia de usuario, asegurando que la plataforma sea intuitiva y atractiva.
- Realizar pruebas de usabilidad y ajustes basados en los resultados para mejorar la experiencia del usuario.
- Crear interfaces consistentes con la identidad visual de Nintendo, prestando especial atención a la accesibilidad y diseño responsivo.
- **Perfil**:
- Experiencia en diseño web y móvil, con un enfoque en tiendas Ecommerce.
- Conocimientos en herramientas de diseño como Figma, Sketch o Adobe XD.
- Familiaridad con las directrices de accesibilidad y buenas prácticas de diseño de interfaces responsivas.

3.18. Especialista en Seguridad Informática

- **Responsabilidades**:
- Implementar medidas de seguridad para proteger la plataforma contra ataques externos, robo de datos y fraudes.

- Realizar auditorías de seguridad regulares y pruebas de penetración.
- Asegurar que los datos sensibles (como información de usuarios y pagos) estén cifrados y protegidos adecuadamente.
- **Perfil**:
- Experiencia en la implementación de protocolos de seguridad (SSL, TLS, OAuth).
- Conocimientos en prevención de fraudes y protección contra ataques como inyecciones SQL, XSS, CSRF.
- Familiaridad con herramientas de monitoreo de seguridad y protección de datos.

3.19. Analista de Datos

- **Responsabilidades**:
- Analizar el comportamiento de los usuarios dentro de la plataforma para identificar patrones de compra, oportunidades de mejora y optimizar las campañas de marketing.
- Trabajar con grandes volúmenes de datos para ofrecer insights valiosos sobre el rendimiento de la plataforma.
- Asegurar que la toma de decisiones esté basada en datos, ofreciendo reportes periódicos.
- **Perfil**:
- Experiencia con herramientas de análisis de datos y visualización como Google Analytics, Power BI o Tableau.
- Conocimientos en análisis de datos para Ecommerce y optimización de conversiones.
- Habilidades en SQL, Python y R para extraer y analizar grandes volúmenes de datos.

3.20. Especialista en Marketing Digital

- **Responsabilidades**:
- Definir e implementar estrategias de marketing digital para atraer y retener usuarios, incluyendo campañas de publicidad, email marketing, y optimización de la conversión.
- Gestionar las campañas de adquisición de usuarios a través de redes sociales, SEO/SEM y plataformas de anuncios.
- Analizar el rendimiento de las campañas y ajustar las estrategias para maximizar el retorno de inversión (ROI).
- **Perfil**:
- Experiencia en marketing digital enfocado a Ecommerce.
- Conocimientos en plataformas publicitarias (Google Ads, Facebook Ads, etc.).
- Habilidades en análisis de datos y optimización de campañas (A/B testing, segmentación de audiencias).

3.21. Especialista en Atención al Cliente

- **Responsabilidades**:
- Proporcionar soporte a los usuarios de la Nintendo eShop, respondiendo a consultas, solucionando problemas y gestionando devoluciones.
- Coordinar con el equipo de desarrollo para resolver problemas técnicos reportados por los usuarios.
- Asegurar una experiencia positiva para los usuarios, aumentando la fidelización y satisfacción del cliente.
- **Perfil**:
- Experiencia en soporte al cliente en plataformas digitales.
- Habilidades de comunicación clara y efectiva.
- Conocimiento de herramientas de soporte como Zendesk o Freshdesk. == Estimación de Plazos pada Implementación

Dado que el proyecto de la Nintendo eShop implica la creación de una plataforma escalable con alta disponibilidad, seguridad, y una experiencia de usuario fluida, es fundamental desarrollar un cronograma detallado que divida el proceso en fases claras. El proyecto será desarrollado en un marco temporal basado en metodologías ágiles, particularmente Scrum, para permitir ajustes en función del feedback continuo y la evolución de los requisitos.

A continuación, se presenta una estimación de plazos desglosada por fases del proyecto, teniendo en cuenta las etapas clave de desarrollo, pruebas, implementación y lanzamiento. Cada sprint tendrá una duración de **2 semanas**, con iteraciones que permiten flexibilidad.

3.22. Fase 1: Planificación y Requisitos (3-4 semanas)

- **Objetivos**:
- Reunir todos los requisitos funcionales y no funcionales del proyecto.
- Definir el alcance completo del proyecto, incluyendo las funcionalidades principales (carrito de compras, catálogo de productos, sistemas de pagos, etc.).
- Establecer el diseño de alto nivel de la arquitectura y la infraestructura en la nube.
- **Entregables**:
- Documento de especificaciones de requisitos.
- Planificación del proyecto y cronograma detallado.
- Mockups iniciales de la interfaz de usuario.
- **Duración estimada**: 3 a 4 semanas.

3.23. Fase 2: Diseño de Arquitectura y Base de Datos (4 semanas)

- **Objetivos**:
- Definir la arquitectura técnica detallada, incluyendo la selección de tecnologías backend, frontend, y la infraestructura en la nube.
- Diseñar la estructura de la base de datos, asegurando que sea escalable y adecuada para el volumen de transacciones esperado.
- **Entregables**:
- Diagrama de la arquitectura de la plataforma.
- Diseño detallado de la base de datos.
- Definición de microservicios y endpoints REST.
- **Duración estimada**: 4 semanas.

3.24. Fase 3: Desarrollo del Backend (10-12 semanas)

- **Objetivos**:
- Implementar la lógica de negocio y los microservicios que gestionan las transacciones, usuarios, pagos, y la comunicación con los servicios externos.
- Asegurar la integración con pasarelas de pago, sistemas de inventario, y autenticación de usuarios.
- Crear los endpoints REST y garantizar la seguridad de los datos.
- **Entregables**:
- APIs REST completamente funcionales para la gestión de usuarios, productos, pagos y pedidos.
- Sistema de autenticación y autorización (OAuth, JWT).
- Integración con pasarelas de pago y servicios externos.
- **Duración estimada**: 10 a 12 semanas.

3.25. Fase 4: Desarrollo del Frontend (8-10 semanas, en paralelo con Backend)

- **Objetivos**:
- Implementar la interfaz de usuario utilizando React.js (con Next.js) o la tecnología seleccionada, asegurando una experiencia rápida y fluida.
- Integrar la interfaz con los endpoints del backend, permitiendo a los usuarios navegar por el catálogo, realizar compras, y gestionar su cuenta.
- Implementar características como carrito de compras, búsqueda de productos, y gestión de usuarios.

- **Entregables**:
- Prototipo del frontend con todas las funcionalidades principales.
- Interfaz optimizada para dispositivos móviles, tabletas y consolas.
- Pruebas de integración con el backend y simulación de cargas para garantizar el rendimiento.
- **Duración estimada**: 8 a 10 semanas (en paralelo con el backend).

3.26. Fase 5: Pruebas y Control de Calidad (6 semanas)

- **Objetivos**:
- Realizar pruebas unitarias, de integración y pruebas de carga para asegurar que el sistema funcione correctamente bajo escenarios de alto tráfico.
- Ejecutar pruebas de seguridad, asegurando que la plataforma esté protegida contra vulnerabilidades comunes (XSS, CSRF, inyecciones SQL, etc.).
- Llevar a cabo pruebas de usabilidad para garantizar que la experiencia de usuario sea fluida y satisfactoria.
- **Entregables**:
- Reporte de pruebas de carga y rendimiento.
- Pruebas de seguridad aprobadas.
- Feedback de pruebas de usabilidad con mejoras implementadas.
- **Duración estimada**: 6 semanas.

3.27. Fase 6: Despliegue y Lanzamiento Inicial (4 semanas)

- **Objetivos**:
- Desplegar la plataforma en un entorno de producción, asegurando la disponibilidad, escalabilidad y redundancia.
- Monitorear el desempeño inicial para resolver cualquier problema crítico que pueda surgir durante el uso en tiempo real.
- Ejecutar un lanzamiento controlado para un grupo limitado de usuarios antes del despliegue completo (fase beta).
- **Entregables**:
- Plataforma en producción con monitoreo activo.
- Informe de rendimiento post-lanzamiento.
- Ajustes finales tras el feedback inicial.
- **Duración estimada**: 4 semanas.

3.28. Cronograma General

Fase	Duración Estimada
Planificación y Requisitos	3 - 4 semanas
Diseño de Arquitectura y Base de Datos	4 semanas
Desarrollo del Backend	10 - 12 semanas
Desarrollo del Frontend	8 - 10 semanas (paralelo)
Pruebas y Control de Calidad	6 semanas
Despliegue y Lanzamiento Inicial	4 semanas
Total	35 - 40 semanas

Glossary

A glossary, also known as a vocabulary or clavis, is an alphabetical list of terms in a particular domain of knowledge with the definitions for those terms. Traditionally, a glossary appears at the end of a book and includes terms within that book that are either newly introduced, uncommon, or specialized.

API

Interfaz de Programación de Aplicaciones. Un conjunto de definiciones y protocolos que permiten la comunicación entre diferentes sistemas o aplicaciones.

Base de Datos

Sistema organizado para almacenar, gestionar y recuperar datos. En el contexto de la eShop, almacena información sobre usuarios, productos, transacciones, etc.

DLC

Contenido Descargable. Contenido adicional que se puede descargar para un juego existente, como nuevas misiones, personajes o expansiones.

E-commerce

Comercio Electrónico. La compra y venta de bienes o servicios a través de Internet. La Nintendo eShop es una plataforma de e-commerce en la industria de videojuegos.

Frontend

Parte del sistema que interactúa directamente con los usuarios, incluyendo la interfaz de usuario, el diseño y la experiencia del usuario (UX).

Backend

Parte del sistema que maneja la lógica del negocio, la base de datos y la comunicación con el frontend. No es visible para los usuarios finales.

Persona

Perfil ficticio que representa a un usuario típico de un producto o sistema, utilizado para guiar el diseño y la toma de decisiones en el desarrollo del producto.

Reseña

Comentario o crítica que un usuario deja sobre un producto, generalmente con una calificación, que puede influir en la decisión de compra de otros usuarios.

Suscripción

Modelo de pago en el que los usuarios pagan una tarifa periódica (mensual o anual) para acceder a un servicio o contenido exclusivo.

Transacción

Proceso de compra o intercambio de bienes o servicios. En la eShop, se refiere a la compra de juegos y contenido.

Usuario

Persona que interactúa con la plataforma eShop, ya sea comprando productos, dejando reseñas o gestionando su cuenta.

UI

Interfaz de Usuario. El espacio donde las interacciones del usuario tienen lugar, incluyendo todos los elementos visuales que los usuarios ven y utilizan.

UX

Experiencia del Usuario. El sentimiento y la experiencia general que un usuario tiene al interactuar con un sistema o producto, incluyendo facilidad de uso y satisfacción.

Método de Pago

Diferentes formas que los usuarios pueden utilizar para realizar pagos en la plataforma, como tarjetas de crédito, débito y monederos electrónicos.

Stock

Cantidad de productos disponibles para la venta. En el caso de productos digitales, generalmente se considera ilimitado.

Marketplace

Plataforma en línea donde múltiples vendedores pueden ofrecer sus productos o servicios a los compradores. La Nintendo eShop funciona como un marketplace para juegos y contenido digital de Nintendo.

References

Websites

- [dbml] DBML. 'DBML - Database Markup Language' https://dbml.dbdiagram.io/home/
- [elixirfullstack] Camilo A. Castro. 'Elixir FullStack' https://elixircl.github.io/elixir-fullstack/#_sobre_el_autor
- [aws] The Information Lab. 'Qué es AWS y para qué sirve' https://www.theinformationlab.es/blog/que-es-amazon-web-services-y-para-que-sirve/
- [gcp] Incentro. '¿Qué es Google Cloud y para qué sirve?' https://www.incentro.com/es-ES/blog/que-es-google-cloud-platform
- [microsoftazure] Conzultek. 'Microsoft Azure: qué es, cómo funciona y cómo ayuda en su empresa' https://blog.conzultek.com/microsoft-azure-que-es-como-funciona-como-ayuda-a-las-empresas
- [onpremise] UDS Enterprise. 'Infraestructura On-Premise: Control total en las instalaciones de tu empresa' https://udsenterprise.com/infraestructura-on-premise-control-instalaciones-empresa/#:~:text=La%20infraestructura%20on%20premise%2C%20o,servicios%20externos%20en%20la%20nube.
- [multicloud] Red Hat. '¿Qué es multicloud?' https://www.redhat.com/es/topics/cloud-computing/what-is-multicloud
- [databases] Pandorafms Tech Blog. 'Tipos de bases de datos y las mejores bases de datos' https://pandorafms.com/blog/es/tipos-de-bases-de-datos-y-las-mejores-bases-de-datos/#:~:text=Idioma-,Mejores%20bases%20de%20datos%20comerciales,suele%20ser%20la%20mejor%20opci%C3%B3n.
- [backend] daily.dev. 'Top 10 Backend Frameworks 2024' https://daily.dev/es/blog/top-10-backend-frameworks-2024
- [frontend] Richestsoft. '10 Most popular frontend frameworks 2024' https://richestsoft.com/es/blog/10-most-popular-front-end-frameworks/#:~:text=Los%2010%20frameworks%20front%2Dend%20m%C3%A1s%20populares%20en%202024%20incluyen,js%2C%20Ember.