# 

# 

**MODELO MVC (Model - View - Controller)**

**APP Comercio Electrónico**

# 

# 

# 

# 

# 

# 

# 

# 

[**Arquitectura MVC (Model - View - Controller) 3**](#_kwow4p7whp97)

[Seguridad: 3](#_l6zqporcggzy)

[Escalabilidad y Rendimiento: 3](#_knlqacttgw8h)

[Manejo de Errores y Logging: 3](#_7mw6mosouanx)

[Microservicios y Arquitectura Orientada a Servicios: 3](#_wopi7satgrh7)

[**Tecnologias que utilizaria: 4**](#_od2p1m2yntj5)

[**Estructura de Archivos: 5**](#_dt6we5v9xptv)

# 

# Arquitectura MVC (Model - View - Controller)

Cada arquitectura puede variar dependiendo la escalabilidad de la aplicación, el equipo de desarrollo, y las tecnologías con las que se esté familiarizado

El patrón MVC es muy adecuado para este tipo de aplicaciones, ya que separa claramente la lógica de negocio, la presentación y el control de las peticiones.

**Beneficios:**

* **Separación de responsabilidades:** Facilita el mantenimiento del código al separar la lógica de negocio, la presentación y el control de las peticiones.
* **Escalabilidad:** Permite añadir nuevas funcionalidades de forma más organizada y modular.
* **Reutilización de Código:** Los modelos y servicios pueden ser reutilizados en diferentes partes de la aplicación.
* **Test:** Facilita la realización de pruebas unitarias y de integración debido a la clara separación de responsabilidades.

## Seguridad:

* Autenticación y Autorización
* Validación de Entradas: Validar y sanitizar todas las entradas del usuario para evitar inyecciones SQL, XSS y otras vulnerabilidades
* Protección contra CSRF y CORS

## Escalabilidad y Rendimiento:

* Cache: Utilizar cache para mejorar el rendimiento, especialmente para datos que no cambian frecuentemente
* Carga y Desempeño: Elastic Load Balancing (ELB)
* Optimización de Consultas

## Manejo de Errores y Logging:

* Manejo centralizado de errores
* Logging y Monitoreo

## Microservicios y Arquitectura Orientada a Servicios:

* Microservicios: Considerar la arquitectura de microservicios para grandes aplicaciones, dividiendo funcionalidades en servicios independientes.
* APIs y Comunicación entre servicios: Diseñar APIs claras y utilizar mecanismos de comunicación eficientes entre servicios (RabbitMQ).

# Tecnologías que utilizaría:

1. **Lenguajes de Programación:]**
   1. JavaScript: Nodejs, Express
2. **Bases de Datos:**
   1. Relacional: MySQL, PostgreSQL
   2. NoSQL: Mongodb
3. **Autenticación y Autorización:**
   1. JWT (JSON web tokens)
   2. OAuth2
4. **Almacenamiento de archivos:**
   1. AWS s3
   2. Cloudinary
5. **Servicios de Mensajería/Colas:**
   1. RabbitMQ
   2. Apache Kafka
6. **Caché:**
   1. Redis
7. **Infraestructura:**
   1. Docker
   2. Kubernetes

# Estructura de Archivos:

El archivo original de la estructura se encuentra en la carpeta ejercicios del proyecto principal **exercise5.txt**

