Arquitectura Recomendada para una Aplicación Web de Contenido

Infraestructura para hosting del sitio

Amazon Web Services (AWS):

- **EC2 Instances:** Utilizar instancias EC2 para alojar servidores web y backend. Configurar Auto Scaling Groups para manejar picos de tráfico y asegurar disponibilidad.
- **Amazon S3:** Almacenar contenido estático (imágenes, videos, CSS/JS) en Amazon S3 para una entrega rápida y eficiente.
- **Amazon CloudFront (CDN):** Implementar CloudFront para entregar contenido estático desde ubicaciones cercanas al usuario, mejorando los tiempos de carga.
- **Elastic Load Balancing (ELB):** Utilizar ELB para distribuir el tráfico entre múltiples instancias EC2, asegurando balanceo de carga y alta disponibilidad.
- **Amazon RDS:** Para datos relacionales que requieren transacciones ACID, como datos de usuarios registrados.
- **Amazon DynamoDB:** Para datos no estructurados y de alta velocidad, como sesiones de usuario y perfiles de usuario.
- **Amazon ElastiCache (Redis/Memcached):** Para reducir la carga de las bases de datos y mejorar los tiempos de respuesta mediante el almacenamiento en caché de datos frecuentemente consultados.
- **Amazon EKS (Elastic Kubernetes Service):** En caso de que los servidores estén gestionados con Kubernetes, utilizar EKS para ejecutar y gestionar aplicaciones contenerizadas en la nube de AWS y en centros de datos locales.

Arquitectura de componentes
Front-end:

- **Next.js:** Utilizar Next.js, un framework basado en React, que permite renderizado del lado del servidor (SSR) y generación estática (SSG). Esto mejora los tiempos de carga inicial y la optimización para motores de búsqueda (SEO), además de facilitar la implementación de aplicaciones web dinámicas y responsivas.
- **Bootstrap o Tailwind CSS:** Implementar diseño responsivo utilizando uno de estos frameworks CSS para asegurar accesibilidad en dispositivos móviles y computadoras

Back-end:

- **Microservicios en Java Spring Boot:** Utilizar Spring Boot para desarrollar microservicios, permitiendo una arquitectura modular y escalable. Cada microservicio puede encargarse de una funcionalidad específica (por ejemplo, autenticación, gestión de contenido, etc.).
- **API Gateway:** Usar Amazon API Gateway para gestionar las solicitudes API y facilitar la comunicación entre los microservicios.

- **Autenticación:** Utilizar Amazon Cognito para la autenticación y gestión de usuarios, proporcionando un inicio de sesión seguro y eficiente.
- **Caching:** Utilizar Amazon ElastiCache (Redis o Memcached) para almacenar en caché los resultados de consultas frecuentes, mejorando los tiempos de respuesta.

Bases de datos recomendadas

- **Amazon RDS (MySQL, PostgreSQL):** Para datos transaccionales y estructurados. Ideal para almacenar datos de usuarios, transacciones y cualquier otro dato que requiera consistencia y durabilidad.
- **Amazon DynamoDB:** Para datos no estructurados, perfiles de usuario, sesiones y cualquier otro dato que requiera alta velocidad de lectura/escritura y escalabilidad automática.
- **Amazon ElastiCache (Redis/Memcached):** Para almacenamiento en caché, reduciendo la carga de las bases de datos y mejorando los tiempos de respuesta.

Frameworks

- **Front-end:**
    - **React:** Para construir aplicaciones web dinámicas y responsivas. React es adecuado para aplicaciones que requieren actualizaciones frecuentes y fácil mantenimiento.
    - **Bootstrap o Tailwind CSS:** Para diseño responsivo y rápido desarrollo de interfaces de usuario.
- **Back-end:**
    - **Spring Boot:** Para desarrollar microservicios en Java, facilitando una arquitectura modular, escalable y fácil de mantener.
    - **Amazon Cognito:** Para implementar autenticación y autorización de manera segura y eficiente.
    - **Hibernate:** Para el manejo de bases de datos relacionales, permitiendo mapeo objeto-relacional (ORM) eficiente.

Resumen de la Arquitectura

1. **Usuario Anónimo (80% del tráfico):**
    - **CloudFront:** Entrega contenido estático desde S3.
    - **ELB:** Distribuye el tráfico a las instancias EC2.
    - **EC2 Instances:** Servidores web que entregan páginas HTML estáticas y redirigen a microservicios para contenido dinámico.
    - **Next.js:** Renderiza las páginas del lado del servidor para mejorar el tiempo de carga inicial y el SEO.
    - **ElastiCache:** Reduce las consultas a la base de datos mediante almacenamiento en caché.

- **EKS:** En caso de utilizar Kubernetes, servicio administrado por Kubernetes que permite ejecutar y gestionar aplicaciones contenedorizadas utilizando Kubernetes en la nube de AWS y en centros de datos locales.

2. **Usuario Registrado (20% del tráfico):**
   - **Amazon Cognito:** Gestiona el login y la autenticación de los usuarios, proporcionando un inicio de sesión seguro y eficiente.
   - **API Gateway:** Gestiona las solicitudes API.
   - **Microservicios Spring Boot en EKS:** Manejan la lógica de negocio y se despliegan en contenedores orquestados por EKS.
   - **RDS y DynamoDB:** Almacenan datos transaccionales y no estructurados, respectivamente.
   - **ElastiCache:** Mejora los tiempos de respuesta mediante caché de datos frecuentes.

Set de APIs para Almacenar y Actualizar Información de Clientes

Consideraciones Generales

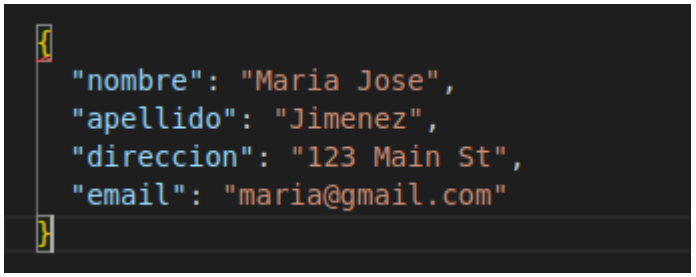
- **Identificador Único:** Cada cliente tendrá un código único asignado que será utilizado como identificador en las API.
- **Seguridad:** No se almacenará directamente información sensible como tarjetas de crédito o CVV en nuestra base de datos. Estas operaciones se delegarán a Stripe.
- **Tokenización:** El token generado por Stripe es lo único que se almacena en nuestro backend, representando la información de la tarjeta de manera segura.

API para Gestión de Clientes

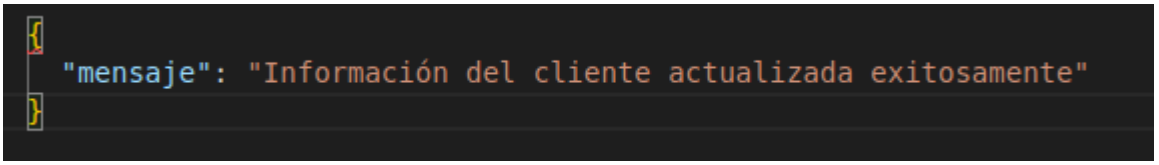
1. **Crear Cliente:**
   - **Endpoint:** `POST /api/clientes`
   - **Descripción:** Crear un nuevo cliente.
   - **Request Body**
   -
```
{
  "nombre": "Maria Jose",
  "apellido": "Jimenez",
  "direccion": "123 Main St",
  "email": "maria@gmail.com"
}
```
   - **Response**

```
{
  "mensaje": "Información del cliente actualizada exitosamente"
}
```

2. **Actualizar Cliente:**

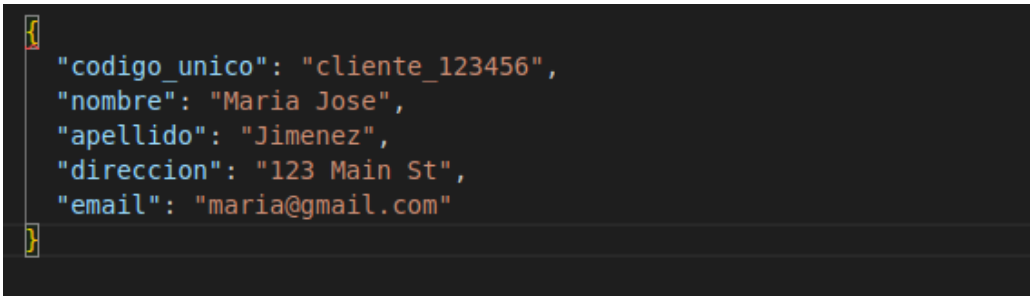
- **Endpoint:** `PUT /api/clientes/{codigo_unico}`
- **Descripción:** Actualizar información del cliente.
- **Request Body:**
  - 

- **Response**



3. **Obtener Cliente:**

- **Endpoint:** `GET /api/clientes/{codigo_unico}`
- **Descripción:** Obtener información de un cliente específico.
- **Request Parameters:**
  `codigo_unico`: El código único asignado al cliente.
- **Response:**



Eliminar Cliente

- **Endpoint:** `DELETE /api/clientes/{codigo_unico}`
- **Descripción:** Eliminar un cliente de la base de datos.
- **Request Parameters:**
  - `codigo_unico`: El código único asignado al cliente.
- **Response:**

- **Código de Respuesta HTTP:** `200 OK`

```
{
  "mensaje": "Cliente eliminado exitosamente"
}
```

Integración con Stripe para Información de Tarjetas de Crédito

Dado que no se almacenará directamente la información de tarjetas de crédito y CVV, se utilizará Stripe para manejar estos datos de manera segura. Las siguientes operaciones explican cómo integrar Stripe en el flujo de trabajo:

1. Capturar Información de Tarjeta en el Frontend

- **Stripe Elements:** Utilizar Stripe Elements en el frontend para capturar la información de la tarjeta de crédito de manera segura.
- **Tokenización:** Stripe Elements envía los datos de la tarjeta directamente a Stripe y obtiene un token seguro que representa la tarjeta.

2. Envío del Token al Backend

- **Formulario o AJAX:** El token generado por Stripe se envía al backend de tu aplicación.

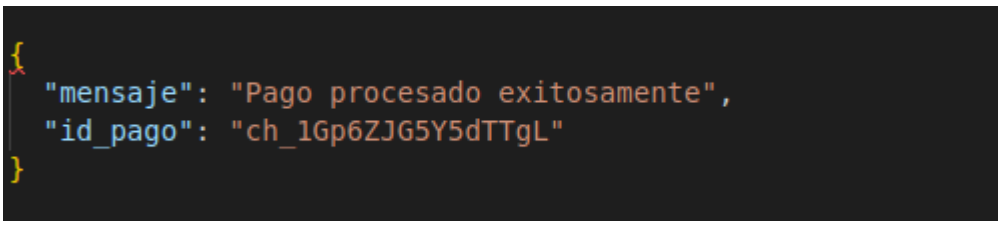
3. Asociar el Token al Cliente en el Backend

- **Endpoint:** `POST /api/pagos`
- **Descripción:** Asociar el token de Stripe al cliente y procesar el pago.
- **Request Body:**

```
{
  "codigo_unico_cliente": "cliente_123456",
  "monto": 5000,
  "moneda": "usd",
  "token_stripe": "tok_visa"
}
```

Response:

- **Código de Respuesta HTTP:** `200 OK`
- **Response Body:**



4. Actualizar Tarjeta de Crédito en Stripe

Captura de la Nueva Información de la Tarjeta en el Frontend:

- Utilizar Stripe Elements en el frontend para capturar de manera segura la nueva información de la tarjeta de crédito del usuario.
- Stripe.js envía esta información directamente a Stripe y devuelve un nuevo token que representa la tarjeta actualizada.

Envío del Token al Backend:

- El token generado por Stripe se envía al backend de tu aplicación a través de una solicitud PUT a la API.

Actualización de la Tarjeta en el Backend:

- En el backend, se recibe el token y se utiliza la API de Stripe para actualizar la tarjeta del cliente asociada al objeto `Customer`.

Códigos de Respuesta HTTP

- **200 OK:** La solicitud se completó con éxito.
- **400 Bad Request:** La solicitud no se pudo procesar debido a un error del cliente (por ejemplo, datos faltantes o incorrectos).
- **401 Unauthorized:** La solicitud requiere autenticación del usuario.
- **404 Not Found:** El recurso solicitado no se encontró.
- **500 Internal Server Error:** Se produjo un error en el servidor.

Conocimientos

1. En java, ¿que hace la palabra"synchronized"?

La palabra clave synchronized en Java se utiliza para controlar el acceso a bloques de código o métodos por múltiples threads. Cuando un método o bloque de código está marcado como synchronized, se asegura que sólo un thread a la vez pueda ejecutar ese código, mientras los demás threads que intenten acceder a la sección sincronizada deben esperar hasta que el thread actual termine.

2. ¿Qué es la inversión de control (IoC) y cómo se implementa en Spring?

La inversión de control (IoC) es un principio de diseño que invierte el control del flujo de un programa. En lugar de que el código de la aplicación controle la ejecución, se delega este control a un contenedor o framework, lo que facilita la gestión de las dependencias entre los componentes y mejora la modularidad y testabilidad del código.

En Spring, IoC se implementa a través del contenedor de Spring, también conocido como el contenedor de IoC. Este contenedor se encarga de instanciar, configurar y ensamblar los objetos de la aplicación, conocidos como beans en Spring.

3. Describa el propósito de las siguientes anotaciones:

@SpringBootApplication:

Es la anotación que aparece en la función main de todo proyecto que definamos con Spring Boot su propósito es

1. **Configurar la aplicación**: Permite que la clase sea usada como fuente de configuración para Spring Boot.
2. **Habilitar la configuración automática**: Configura automáticamente la aplicación basada en las dependencias presentes en el classpath.
3. **Habilitar el escaneo de componentes**: Activa el escaneo de componentes para encontrar y registrar beans con anotaciones como @Component,@Service,@Repository, y @Controller.

@RestController

El propósito de @RestController es combinar las anotaciones @Controller y @ResponseBody, lo que significa que los métodos de la clase devuelven directamente datos en formato JSON o XML en lugar de una vista. Es ideal para construir servicios web RESTful.

@GetMapping

El propósito de @GetMapping es manejar solicitudes HTTP GET a una URL específica. Simplifica la configuración de rutas para los métodos del controlador y es una forma más moderna y específica de @RequestMapping(method = RequestMethod.GET).

@Autowired

El propósito de @Autowired es inyectar automáticamente dependencias en Spring. Permite que Spring resuelva y asigne automáticamente los beans necesarios a una clase, eliminando la necesidad de instanciar manualmente las dependencias y facilitando la inyección de dependencias y la inversión de control (IoC).

4. En HTML, ¿cómo empieza un documento de HTML5?
comienza con la declaración del tipo de documento <!DOCTYPE html>
Esta declaración le dice al navegador que el documento está escrito en HTML5.

5. Explica la diferencia entre var, let y const. Da un ejemplo de cuándo usarías cada uno.

1. var

- **Alcance (Scope)**: Tiene alcance de función. Si se declara dentro de una función, no es accesible fuera de ella, pero si se declara fuera de una función, es accesible en todo el contexto de ejecución.
- **Hoisting**: Las variables declaradas con var se elevan al inicio de su contexto, pero su inicialización no. Esto significa que puedes usar la variable antes de su declaración, pero tendrá el valor undefined hasta que se inicialice.
- **Redefinición**: Puedes redeclarar y reasignar variables definidas con var.

```
function ejemploVar() {
  var mensaje = "Hola, mundo!";
  if (true) {
      var mensaje = "Hola, desde el bloque!";
      console.log(mensaje); // "Hola, desde el bloque!"
  }
  console.log(mensaje); // "Hola, desde el bloque!"
}
ejemploVar();
```

let

- **Alcance (Scope)**: Tiene alcance de bloque. Solo es accesible dentro del bloque donde se declara (por ejemplo, dentro de llaves {}).

- **Hoisting**: Las variables declaradas con let se elevan al inicio de su contexto, pero no se inicializan. No puedes usarlas antes de su declaración.
- **Redefinición**: No puedes redeclarar una variable con let dentro del mismo bloque, pero sí puedes reasignar su valor.

```
function ejemploLet() {
    let mensaje = "Hola, mundo!";
    if (true) {
        let mensaje = "Hola, desde el bloque!";
        console.log(mensaje); // "Hola, desde el bloque!"
    }
    console.log(mensaje); // "Hola, mundo!"
}
ejemploLet();
```

const

- **Alcance (Scope)**: Tiene alcance de bloque, igual que let.
- **Hoisting**: Las variables declaradas con const se elevan al inicio de su contexto, pero no se inicializan. No puedes usarlas antes de su declaración.
- **Redefinición y reasignación**: No puedes redeclarar ni reasignar una variable declarada con const. Sin embargo, si el valor es un objeto o un array, puedes cambiar sus propiedades o elementos.

```
function ejemploConst() {

  const mensaje = "Hola, mundo!";

  // mensaje = "Hola, de nuevo!"; // Esto causará un error



  const objeto = { saludo: "Hola" };

  objeto.saludo = "Hola, mundo!"; // Esto es válido

  console.log(objeto.saludo); // "Hola, mundo!"

}

ejemploConst();
```

6. Explica qué es JSX y por qué se utiliza en React

JSX es una sintaxis que combina JavaScript y XML/HTML. Aunque se parece mucho a HTML, JSX es esencialmente una forma de escribir estructuras de elementos y componentes de React de una manera que es fácil de leer y escribir. Se utiliza en React porque facilita la creación de interfaces de usuario dinámicas y eficientes.

7. ¿Qué evalúa la siguiente expresión regular? ^[0-9]+$
Evalúa si una cadena de texto contiene únicamente uno o más dígitos del 0 al 9

8. ¿Cuál es la diferencia entre POST/GET y PUT/PATCH?

- **GET vs. POST**:
  - **GET**: Recupera datos. Los parámetros se envían en la URL.
  - **POST**: Envía datos. Los parámetros se envían en el cuerpo de la solicitud.
- **PUT vs. PATCH**:
  - **PUT**: Actualiza completamente un recurso existente.
  - **PATCH**: Actualiza parcialmente un recurso existente.

9. En Node.js ¿qué hace la instrucción async?
La instrucción `async` se utiliza para declarar una función asíncrona. Esto permite utilizar la palabra clave `await` dentro de la función, que pausa la ejecución de la función asíncrona y espera a que se resuelva una promesa. Cuando se usa `async`, la función siempre devuelve una promesa.

10. ¿Qué es el event loop de node.js y en que ayuda?
Es un mecanismo que gestiona la ejecución de operaciones asíncronas. Ayuda a Node.js a manejar múltiples operaciones sin bloquear el hilo principal, permitiendo un manejo eficiente de E/S (entrada/salida) y otras tareas asíncronas.

11. ¿Es recomendable implementar bases de datos en contenedores? ¿Porqué?
Implementar bases de datos en contenedores es recomendable en entornos de desarrollo y pruebas debido a su portabilidad y consistencia de entorno, pero en producción puede ser menos ideal debido a desafíos con la persistencia de datos y el rendimiento.

12. Indiqué a que se refiere OWASP

OWASP se refiere a la **Open Web Application Security Project**, una organización sin fines de lucro que se dedica a mejorar la seguridad del software. OWASP proporciona recursos, herramientas, y documentación para ayudar a los desarrolladores y organizaciones a construir aplicaciones web seguras, siendo conocido especialmente por su lista OWASP Top Ten, que destaca las vulnerabilidades de seguridad más críticas en aplicaciones web.

Gestión de Proyectos

1. ¿En qué momento Inicia un sprint en metodología Scrum?
En la metodología Scrum, un sprint inicia inmediatamente después de la reunión de planificación del sprint. Durante esta reunión, el equipo define qué trabajo se realizará durante el sprint, establece un objetivo claro y crea un plan para completar ese trabajo. Una vez que la planificación está completa, comienza oficialmente el sprint y el equipo se enfoca en alcanzar los objetivos establecidos.

2. Mencione 3 roles involucrados en metodología Scrum
En la metodología Scrum, hay tres roles principales involucrados: el **Product Owner**, el **Scrum Master**, y el **Development Team**. El Product Owner se encarga de definir y priorizar el trabajo, el Scrum Master facilita el proceso y elimina impedimentos, y el Development Team realiza el trabajo de desarrollo y entrega del producto.

3. ¿Según el PMBOK qué significa SPI y qué representa su valor igual a 1?
Según el PMBOK, SPI significa **Índice de Desempeño del Cronograma** (Schedule Performance Index). Representa una medida de la eficiencia del tiempo de un proyecto. Un valor de SPI igual a 1 indica que el proyecto está perfectamente alineado con el cronograma planificado, es decir, el trabajo se está completando exactamente según lo programado.