SeguriPoint

https://drive.google.com/drive/folders/1ddtBnBCBCq17pQVRDnYvN3s4t381PQdC

Punto 1 - Modelo de Datos (35 puntos)

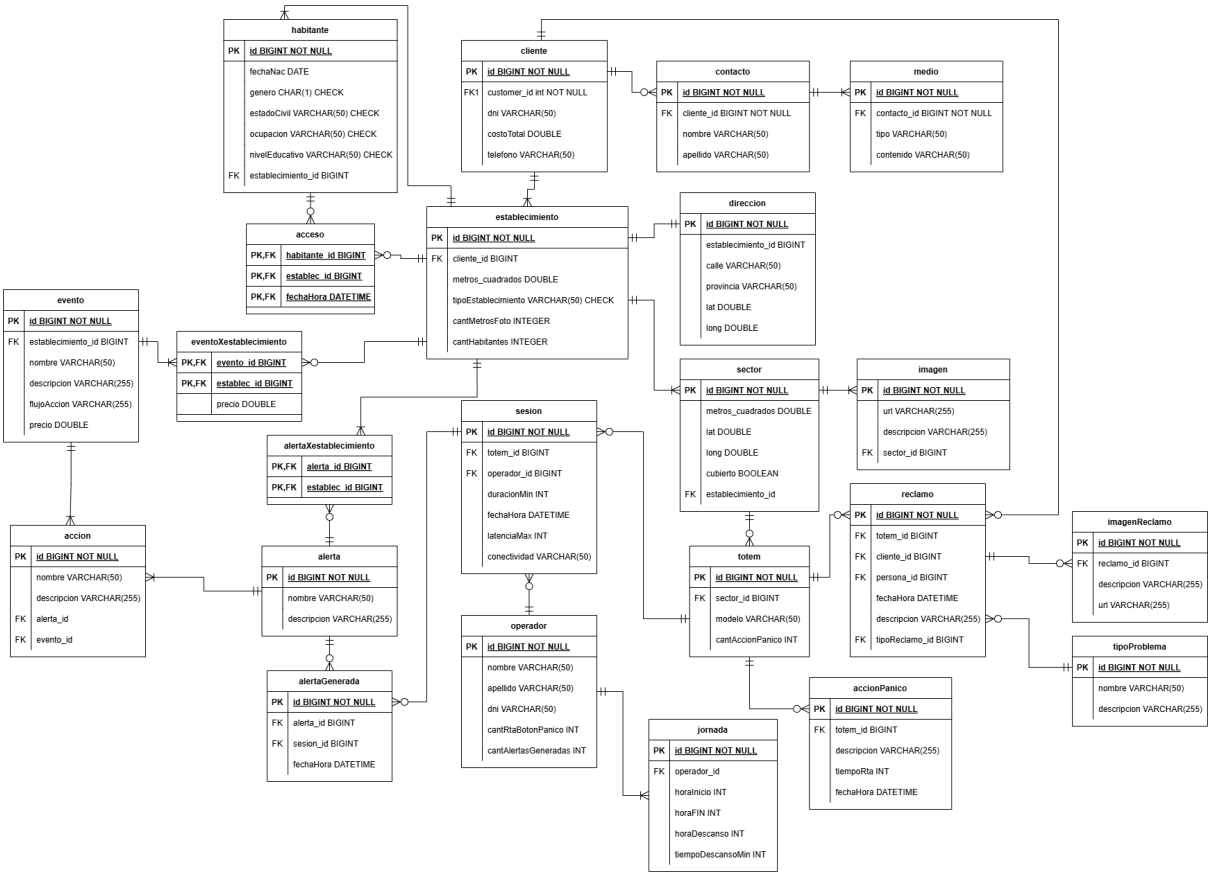
1) (10 puntos) Enumere las entidades identificadas y mencione, de forma concisa, qué representa cada una de ellas.

- Establecimientos: Lugares que serán vigilados. Los establecimientos pueden ser viviendas particulares (en cualquiera de sus formas), condominios familiares y/o barrios cerrados/privados.
    - Habitante: Viven en los establecimientos. Conocemos su información personal.
    - Sector: Los establecimientos se dividen en sectores. Sabemos si están cubiertos o no.
    - Imagen: Cada n metros se piden imágenes de los sectores.
- Cliente: Quienes contratan el servicio y registran establecimientos a vigilar.
    - Contacto: Poseen una lista de contactos para avisarles en caso de alerta
    - Medios de contacto: Direcciones para comunicarse con los contactos (mail, teléfono, etc.).
- Alerta: Predefinidas por el sistema. Se pueden elegir cuáles tener en el establecimiento.
    - Acción: Pasos a seguir tras una alerta. Una alerta puede tener una o más acciones.
- Evento: Los personaliza el cliente y tienen un costo aparte.
- Operador/Vigilador: Son quienes operan los tótems durante una sesión. Pueden activar alertas.
    - Jornada: Tiempo de inicio y fin de la jornada con descansos.
- Totem: Elementos que se instalan en los establecimientos para vigilancia.
    - Acción de pánico: Se registra cada vez que se presiona el botón y en qué momento sucede.
- Sesión: Tiempo que el tótem está activo y en uso por un vigilador.
    - AlertaGenerada: Todas las alertas generadas por el vigilador durante la sesión.

- Reclamo: Las personas pueden reclamar fallas en totems.
    - Tipo Problema: (Internet, personal, visualización, sonido, etc.).
    - Imágenes: Opcionales. Fotos del problema.

2) (25 puntos) Realice el modelo de datos del dominio presentado. Indique todos los supuestos que crea necesario considerar. Además, resulta de suma importancia que justifique todas las decisiones de diseño que tomó.

https://drive.google.com/file/d/15uOF7wDold4aExqq9U7_W4Tpk1E3p2az/view?usp=sharing



Punto 2 - Arquitectura (40 puntos)

1) (10 puntos) Considerando que la solución se apoya en un Estilo de Microservicios y que se quiere escalar a 3 instancias el Microservicio de Alertas.
a) ¿Qué componentes son necesarios que interactúen para soportar las nuevas instancias?

1. API Gateway
   - Actúa como punto de entrada único para los clientes.
   - Debe poder balancear la carga entre las 3 instancias del microservicio de Alertas.
   - Permite aplicar políticas de seguridad, rate limiting y enrutamiento dinámico.

2. Service Registry (Registro de Servicios)
   - Permite que cada instancia del Microservicio de Alertas se registre dinámicamente cuando arranca.
   - Mantiene la lista de instancias disponibles y sus ubicaciones (host/puerto).
   - El API Gateway lo utiliza para saber a qué instancias enviar las solicitudes.
   - Soporta descubrimiento de servicios y remueve instancias caídas automáticamente.

b) Genere el **Diagrama de Despliegue** dónde muestra cómo quedaría el Microservicios escalado.

https://drive.google.com/file/d/1zia8BxS1yToeZ8txhnPNYs2HGGyBIVVn/view?usp=sharing

- Service Registry y API Gateway.

- 3 instancias del microservicio Alertas.

- Microservicio Establecimientos para gestionar cada establecimiento, vigilancia, datos y clientes.

- Microservicio de Estadísticas.

- 2 Bases de datos relacionales. (Estadísticas y Establecimientos/Alertas)

- Se puede agregar un sistema de mensajería con Kafka y un microservicio de Notificaciones, para notificar a los contactos cuando ocurren alertas.

2) (10 puntos) Considerando el tipo de Sistema que estamos desarrollando.
a) ¿Qué tipo de cliente utilizará y por qué? Justifique utilizando 2 atributos de calidad.

Atributos más esperados:

- Fiabilidad (Disponibilidad): Nuestro sistema debe estar disponible durante las rondas de vigilancia. Debe estar siempre preparado ante posibles emergencias.

- Eficiencia en tiempos: Idealmente debe tener bajos tiempos de respuesta ante emergencias y permitir procesar grandes volúmenes de datos/llamados en tiempo real.

Ventaja del **Cliente Pesado**: Al ejecutar gran parte de la lógica de presentación y procesamiento en la máquina local (la PC de la oficina del vigilador), el cliente pesado reduce drásticamente la cantidad de intercambios de datos y la carga de procesamiento que debe manejar el servidor en tiempo real.

Ventajas principales
Funcionalidad sin conexión: Puede seguir funcionando y ejecutando aplicaciones sin depender de una conexión a internet constante a un servidor.

Mejor rendimiento: La mayor parte del procesamiento se realiza en la máquina local, lo que resulta en una ejecución más rápida de aplicaciones exigentes y gráficos intensivos.

Mayor capacidad del servidor: Como el cliente maneja gran parte del trabajo, el servidor recibe menos carga, lo que le permite atender a más usuarios simultáneamente.

Menor dependencia de la red: Almacena datos localmente, lo que reduce la carga de la red y los posibles problemas derivados de una conexión lenta o inestable.

Uso de infraestructura existente: Muchas organizaciones ya cuentan con computadoras potentes que pueden funcionar como clientes pesados, lo que reduce la necesidad de invertir en nueva infraestructura.

b) ¿Considera viable una Aplicación Móvil? ¿De qué tipo?

Creo que no es algo fundamental. Los vigiladores no lo usan desde el celular. Los clientes puede que sí. Como última instancia consideraría una app híbrida con cliente pesado. Ofrece mayor mantenibilidad, además de portabilidad en todos los SO móviles.

3) (10 puntos) Considerando que el cliente debe recibir el detalle de su reclamo por mail, posterior a la carga del mismo.
a) ¿Cómo haría este proceso?

Haría una cola de mensajes asincrónica (Ej. Kafka) de productor-consumidor. El cliente genera el reclamo y se encola. Luego el consumidor (Microservicio de Notificaciones) se encarga de despachar los mails.

Esto ofrece bajo acoplamiento. Cada servicio puede escalar o reiniciarse sin afectar al otro. Además, si el receptor está caído, los mensajes se encolan para enviar luego.

b) Considerando una Arquitectura de Microservicios ¿Qué servicios involucraría?

Se puede agregar un microservicio de Notificaciones (para notificar a los contactos cuando ocurren alertas) y otro de Reclamos (tomar datos del reclamo y persistir denuncia).

Cuando el cliente carga un reclamo, el Microservicio de Reclamos:

1. Persiste el reclamo.
2. Publica un evento en la cola: ReclamoCreado con el detalle (cliente, fecha, descripción, etc.).

Luego, el Microservicio de Notificaciones se suscribe a este evento y:

3. Consume el mensaje, formatea el correo y lo envía por email.
4. En caso de caída, los mensajes quedan en la cola y se procesan cuando vuelva a estar disponible.

c) ¿Qué mecanismo de integración utilizará entre ambos servicios?

Cola de mensajería asíncrona productor-consumidor (Ej. Kafka).

4) (10 puntos) Con el objetivo de trabajar en conjunto con los diferentes Ministerios de Seguridad de las jurisdicciones en la que están nuestros productos, los diferentes actores nos solicitaron acceso a las estadísticas. Cada actor requiere un conjunto de datos diferentes por lo que se requiere flexibilidad en el EndPoint. Se requiere minimizar las llamadas a los Endpoint disponibles.

a) ¿Qué tipo de integración propondría?

Haría una integración por API GraphQL, síncrona (petición-respuesta). GraphQL permite que cada Ministerio consulte **exactamente** los datos necesarios mediante un único endpoint flexible, evitando llamadas múltiples o endpoints específicos por actor. Esto cumple con el requerimiento de **flexibilidad** y **minimización de llamadas**.

b) ¿Qué atributos de calidad consideraría? Defina cada uno.

Performance/Escalabilidad: GraphQL reduce la cantidad de requests al servidor, ya que en una única consulta se pueden obtener múltiples entidades relacionadas. Minimiza el *over-fetching* (traer de más) y el *under-fetching* (traer de menos), optimizando el uso del ancho de banda y los tiempos de respuesta.

Flexibilidad: Los distintos Ministerios pueden pedir **conjuntos de datos personalizados** sin necesidad de crear endpoints específicos. El esquema GraphQL permite exponer un único endpoint adaptable a cada consulta.

Mantenibilidad: El backend puede evolucionar sin romper a los clientes. GraphQL permite agregar nuevos campos o tipos sin invalidar las consultas existentes y ofrece herramientas claras de tipado, documentación automática y validación, facilitando el mantenimiento del servicio de estadísticas.

c) ¿Qué cambios debería realizar sobre la Arquitectura del Sistema Actual?

1. Incorporar un Microservicio de Estadísticas
   - Será el responsable de exponer el endpoint GraphQL.
   - Centralizará la lógica de agregación y consulta de datos estadísticos.

2. Adaptar o duplicar la capa de datos
Dado que las estadísticas pueden requerir datos agregados o históricos, existen dos enfoques válidos:
   - Usar una base de datos separada optimizada para lecturas (por ejemplo, un Data Mart o una base desnormalizada).
   - Leer desde una réplica de la base de datos productiva, evitando sobrecargar los microservicios que manejan operaciones transaccionales.

Punto 3 - Persistencia (25 puntos)

```
@Entity @Table(name="persona")
class Persona {
  @Id @GeneratedValue(strategy = GenerationType.IDENTITY)
  private Integer id;

  @Column(name = "nombre")
  private String nombre;

  @Column(name = "apellido")
  private String apellido;

 @ManyToMany
 @JoinTable(
     name = "persona_vehiculo",
     joinColumns = @JoinColumn(name = "persona_id"),
     inverseJoinColumns = @JoinColumn(name = "vehiculo_id")
  )
  private List<Vehículo> flota;

  @Column(name = "nro_documento")
  private String nroDocumento;

 @Enumerated (EnumType.STRING)
 @Column(name="tipo_documento", nullable = false)
  private TipoDocumento tipoDocumento;
}
```

```
@Entity @Table(name="vehiculo")
class Vehiculo {
  @Id @GeneratedValue(strategy = GenerationType.IDENTITY)
  private Integer id;

  @Column(name = "patente",nullable = false, unique = true)
  private String patente;

  @ManyToMany(mappedBy = "flota")
  private List<Persona> propietarios;
}

@Entity @Table(name="ema")
class EMA {
  @Id @GeneratedValue(strategy = GenerationType.IDENTITY)
  private Integer id;

  @Column(name = "lugar_recomendado")
  private String lugarRecomendado;
}

@Entity @Table(name="autorizacion")
class Autorizacion {
  @Id @GeneratedValue
  private Integer id;

  @Column(name = "fecha_hora_inicio")
  private LocalDateTime fechaHoraInicio;

  @Column(name = "fecha_hora_fin")
  private LocalDateTime fechaHoraFin;

  @ManyToOne
  @JoinColumn(name="vehiculo_id", referencedColumnName="id")
  private Vehiculo vehiculo;

  @ManyToOne
  @JoinColumn(name="autorizante_id", referencedColumnName="id")
  private Persona autorizante;

  @ManyToOne
  @JoinColumn(name="autorizado_id", referencedColumnName="id")
  private Persona autorizado;
 }
```

```
@Entity @Table(name="busqueda_estacionamiento")
class BusquedaEstacionamiento {
  @Id @GeneratedValue(strategy = GenerationType.IDENTITY)
  private Integer id;

  @OneToOne
  @JoinColumn(name=”ema_id”, referencedColumnName=”id”)
  private EMA ema;

  @Column(name = "destino")
  private String destino;

  @Column(name = "distancia")
  private Double distancia;

  @Column(name = "finalizado")
  private Boolean finalizado;

  @Convert(converter = MetodoBusquedaConverter.class)
  @Column(name = "metodo_utilizado")
  private MetodoBusquedaEstacionamiento metodoUtilizado;

  @ManyToOne
  @JoinColumn(name=”vehiculo_id”, referencedColumnName=”id”)
  private Vehiculo vehiculo;
}

@Converter
public class MetodoBusquedaConverter implements
AttributeConverter<MetodoBusquedaEstacionamiento, String>{
  . . .
}
```

https://drive.google.com/file/d/130UFaYVaqhMp5o5A6_mSXDBYT8LCd2ZI/view?usp=sharing