Manual Técnico - Sistema de Gestión de Sesiones de Bienestar

Amada Noemi Cárcamo Renderos

Tabla de Contenidos

1. Arquitectura del Sistema

1.1 Diagrama de Capas

El sistema está organizado en una arquitectura de 3 capas principales:

Capa de Presentación

- **Frontend Web (React)**: Interfaz para administradores y recepcionistas
  - Puerto: 3000
  - Tecnología: React 18, Tailwind CSS, Lucide Icons
  - Comunicación: API REST con JWT
- **App Móvil (Android)**: Interfaz para clientes
  - SDK mínimo: 24 (Android 7.0)
  - Lenguaje: Java + Kotlin
  - Comunicación: Retrofit con OkHttp

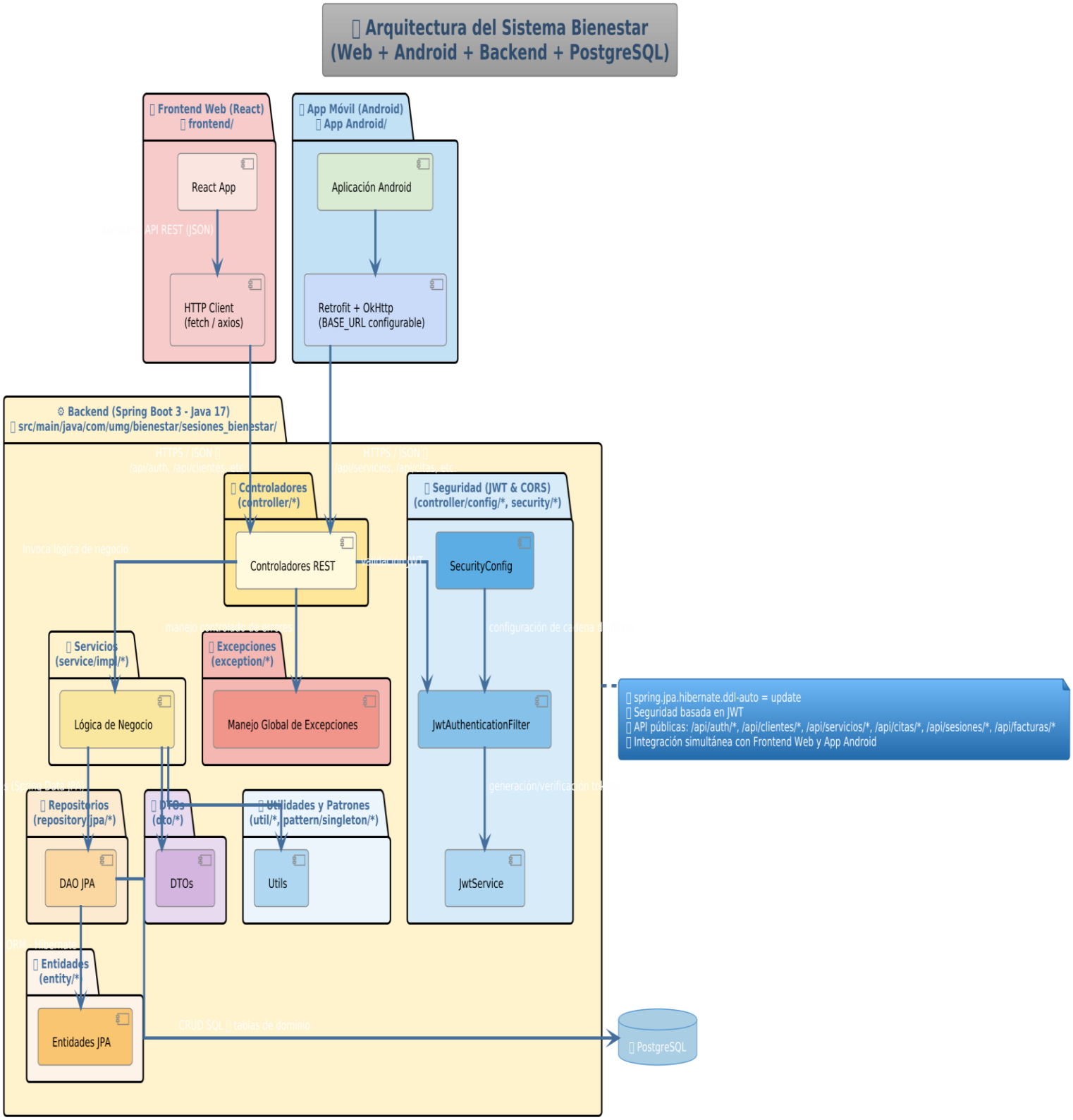
Capa de Lógica de Negocio (Backend)

- **Spring Boot 3.5.5**
  - Puerto: 8080
  - API RESTful
  - Seguridad: Spring Security + JWT
  - Validaciones: Jakarta Validation

Capa de Datos

- **PostgreSQL** (Producción - Railway)

1.2 Diagrama de Componentes



1.3 Comunicación entre Componentes

Frontend Web ↔ Backend

App.js

src > App.js > Clientes

```
137   const Clientes = ({ token }) => {
158
159     const fetchClientes = async () => {
160       setLoading(true);
161       try {
162         const response = await fetch(`${API_URL}/clientes`, {
163           headers: { 'Authorization': `Bearer ${token}` }
164         });
165         if (response.ok) {
166           const data = await response.json();
167           console.log('=== CLIENTES OBTENIDOS ===', data);
```

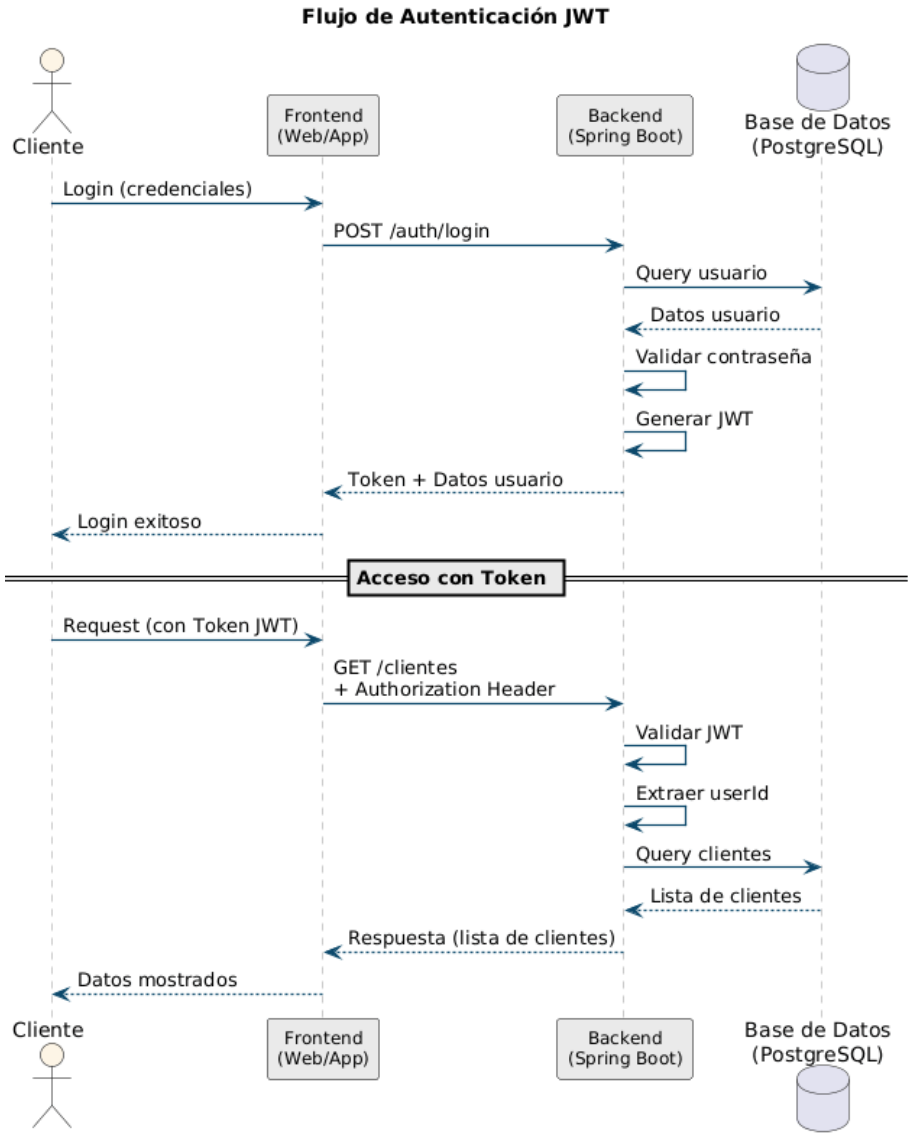
App Android ↔ Backend

```
public class ApiClient {
    public static Retrofit getClient(Context context) {
        if (retrofit == null) {
            // Logging interceptor para debug
            HttpLoggingInterceptor logging = new HttpLoggingInterceptor();
            logging.setLevel(HttpLoggingInterceptor.Level.BODY);

            // Cliente HTTP con interceptores
            OkHttpClient client = new OkHttpClient.Builder()
                    .addInterceptor(new AuthInterceptor(context))
                    .addInterceptor(logging)
                    .connectTimeout( timeout: 30, TimeUnit.SECONDS)
                    .readTimeout( timeout: 30, TimeUnit.SECONDS)
                    .writeTimeout( timeout: 30, TimeUnit.SECONDS)
                    .build();

            retrofit = new Retrofit.Builder()
                    .baseUrl(Constants.BASE_URL)
                    .client(client)
                    .addConverterFactory(GsonConverterFactory.create())
                    .build();
        }
        return retrofit;
    }
    7 usages
    public static ApiService getApiService(Context context) {
        return getClient(context).create(ApiService.class);
    }
```

Backend ↔ Base de Datos

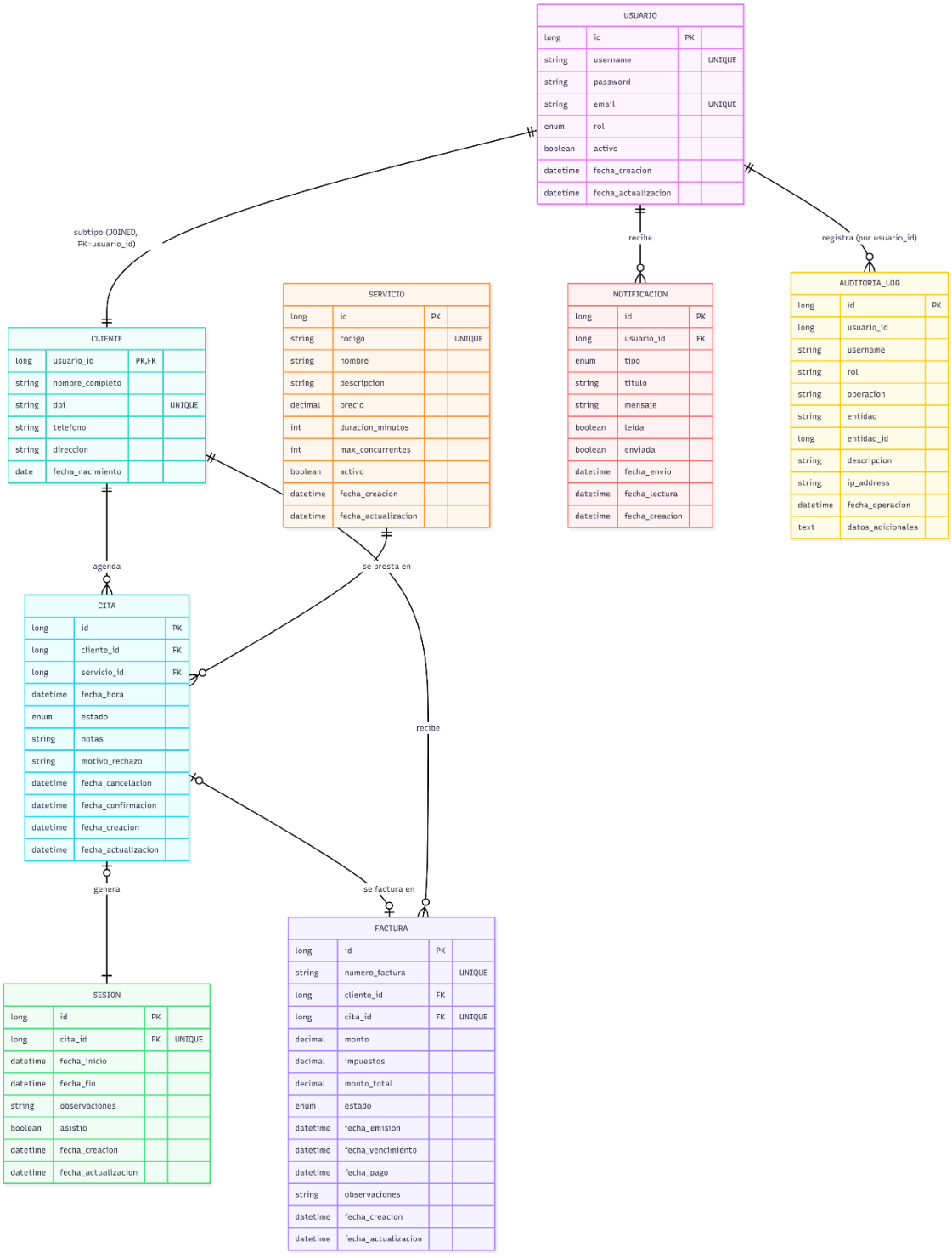
```java
@Repository
public interface ClienteRepository extends JpaRepository<Cliente, Long> {

    // Método agregado para CustomUserDetailsService
    Optional<Cliente> findByUsername(String username);

    boolean existsByUsername(String username);
    Optional<Cliente> findByDpi(String dpi);
    Optional<Cliente> findByEmail(String email);
    boolean existsByDpi(String dpi);
    boolean existsByEmail(String email);
    List<Cliente> findByActivoTrue();

    @Query("SELECT c FROM Cliente c WHERE c.nombreCompleto LIKE %:nombre%")
    List<Cliente> buscarPorNombre(@Param("nombre") String nombre);

    @Query("SELECT COUNT(ci) > 0 FROM Cita ci WHERE ci.cliente.id = :clienteId AND ci.estado IN ('PENDIENTE', 'CONFIRMADA')")
    boolean tieneCitasActivas(@Param("clienteId") Long clienteId);
}
```

1.4 Flujo de Autentificación JWT



2. Modelo Entidad-Relación

2. 1.Diagrama Entidad-Relación



- USUARIO–CLIENTE (1:1): Subtipo con herencia JOINED; CLIENTE.usuario_id es PK y FK a USUARIO.id.

- CLIENTE–CITA (1:N): Un cliente puede tener múltiples citas; cada cita pertenece a un cliente.

- SERVICIO–CITA (1:N): Un servicio puede estar asociado a múltiples citas.

- CITA–SESION (0..1:1): Una sesión se crea a partir de una cita; la cita puede no tener sesión todavía.

- CITA–FACTURA (0..1:0..1): Una cita puede tener como máximo una factura; la factura puede o no referir a una cita.

- CLIENTE–FACTURA (1:N): Un cliente puede acumular muchas facturas.

- USUARIO–NOTIFICACION (1:N): Un usuario recibe múltiples notificaciones.

- USUARIO–AUDITORIA_LOG (1:N lógico): El log almacena usuario_id; FK opcional en base de datos.

2. 2. Script de Creación de Base De Datos

Tabla de Usuarios

Query | Query History

```
-- Table: public.usuarios

-- DROP TABLE IF EXISTS public.usuarios;

CREATE TABLE IF NOT EXISTS public.usuarios
(
    activo boolean NOT NULL,
    fecha_actualizacion timestamp(6) without time zone,
    fecha_creacion timestamp(6) without time zone NOT NULL,
    id bigint NOT NULL GENERATED BY DEFAULT AS IDENTITY ( INCREMENT 1 START 1 MINVALUE 1 MAXVALUE 9223372036854775807 CACHE 1 )
    rol character varying(20) COLLATE pg_catalog."default" NOT NULL,
    username character varying(50) COLLATE pg_catalog."default" NOT NULL,
    email character varying(100) COLLATE pg_catalog."default" NOT NULL,
    password character varying(255) COLLATE pg_catalog."default" NOT NULL,
    CONSTRAINT usuarios_pkey PRIMARY KEY (id),
    CONSTRAINT usuarios_email_key UNIQUE (email),
    CONSTRAINT usuarios_username_key UNIQUE (username),
    CONSTRAINT usuarios_rol_check CHECK (rol::text = ANY (ARRAY['ADMINISTRADOR'::character varying, 'RECEPCIONISTA'::character
)

TABLESPACE pg_default;

ALTER TABLE IF EXISTS public.usuarios
    OWNER to postgres;
```

Tabla de Sesiones

```
-- DROP TABLE IF EXISTS public.sesiones;

CREATE TABLE IF NOT EXISTS public.sesiones
(
    asistio boolean,
    cita_id bigint NOT NULL,
    fecha_actualizacion timestamp(6) without time zone,
    fecha_creacion timestamp(6) without time zone NOT NULL,
    fecha_fin timestamp(6) without time zone,
    fecha_inicio timestamp(6) without time zone NOT NULL,
    id bigint NOT NULL GENERATED BY DEFAULT AS IDENTITY ( INCREMENT 1 START 1 MINVALUE 1 MAXVALUE 9223372036854775807 CACHE 1
    observaciones character varying(1000) COLLATE pg_catalog."default",
    CONSTRAINT sesiones_pkey PRIMARY KEY (id),
    CONSTRAINT sesiones_cita_id_key UNIQUE (cita_id),
    CONSTRAINT fkni83cn5t51g1rxwv5b699a162 FOREIGN KEY (cita_id)
        REFERENCES public.citas (id) MATCH SIMPLE
        ON UPDATE NO ACTION
        ON DELETE NO ACTION
)

TABLESPACE pg_default;

ALTER TABLE IF EXISTS public.sesiones
    OWNER to postgres;
```

Tabla de Servicios

```
-- DROP TABLE IF EXISTS public.servicios;

CREATE TABLE IF NOT EXISTS public.servicios
(
    activo boolean NOT NULL,
    duracion_minutos integer NOT NULL,
    max_concurrentes integer NOT NULL,
    precio numeric(10,2) NOT NULL,
    fecha_actualizacion timestamp(6) without time zone,
    fecha_creacion timestamp(6) without time zone NOT NULL,
    id bigint NOT NULL GENERATED BY DEFAULT AS IDENTITY ( INCREMENT 1 START 1 MINVALUE 1 MAXVALUE 9223372036854775807 CACHE 1
    codigo character varying(20) COLLATE pg_catalog."default" NOT NULL,
    nombre character varying(100) COLLATE pg_catalog."default" NOT NULL,
    descripcion character varying(500) COLLATE pg_catalog."default",
    CONSTRAINT servicios_pkey PRIMARY KEY (id),
    CONSTRAINT servicios_codigo_key UNIQUE (codigo),
    CONSTRAINT servicios_duracion_minutos_check CHECK (duracion_minutos >= 1),
    CONSTRAINT servicios_max_concurrentes_check CHECK (max_concurrentes <= 50 AND max_concurrentes >= 1)
)

TABLESPACE pg_default;

ALTER TABLE IF EXISTS public.servicios
    OWNER to postgres;
```

Tabla de Notificaciones

```
CREATE TABLE IF NOT EXISTS public.notificaciones
(
    enviada boolean NOT NULL,
    leida boolean NOT NULL,
    fecha_creacion timestamp(6) without time zone NOT NULL,
    fecha_envio timestamp(6) without time zone,
    fecha_lectura timestamp(6) without time zone,
    id bigint NOT NULL GENERATED BY DEFAULT AS IDENTITY ( INCREMENT 1 START 1 MINVALUE 1 MAXVALUE 9223372036854775807 CACHE 1
    usuario_id bigint NOT NULL,
    tipo character varying(20) COLLATE pg_catalog."default" NOT NULL,
    titulo character varying(200) COLLATE pg_catalog."default" NOT NULL,
    mensaje character varying(1000) COLLATE pg_catalog."default" NOT NULL,
    CONSTRAINT notificaciones_pkey PRIMARY KEY (id),
    CONSTRAINT fk1mxbjb81ft61gwlh0kabubndc FOREIGN KEY (usuario_id)
        REFERENCES public.usuarios (id) MATCH SIMPLE
        ON UPDATE NO ACTION
        ON DELETE NO ACTION,
    CONSTRAINT notificaciones_tipo_check CHECK (tipo::text = ANY (ARRAY['EMAIL'::character varying, 'SMS'::character varying,
)

TABLESPACE pg_default;

ALTER TABLE IF EXISTS public.notificaciones
    OWNER to postgres;
```

Tabla de Facturas

```
CREATE TABLE IF NOT EXISTS public.facturas
(
    impuestos numeric(10,2),
    monto numeric(10,2) NOT NULL,
    monto_total numeric(10,2),
    cita_id bigint,
    cliente_id bigint NOT NULL,
    fecha_actualizacion timestamp(6) without time zone,
    fecha_creacion timestamp(6) without time zone NOT NULL,
    fecha_emision timestamp(6) without time zone NOT NULL,
    fecha_pago timestamp(6) without time zone,
    fecha_vencimiento timestamp(6) without time zone,
    id bigint NOT NULL GENERATED BY DEFAULT AS IDENTITY ( INCREMENT 1 START 1 MINVALUE 1 MAXVALUE 9223372036854775807 CACHE 1
    estado character varying(20) COLLATE pg_catalog."default" NOT NULL,
    numero_factura character varying(50) COLLATE pg_catalog."default" NOT NULL,
    observaciones character varying(500) COLLATE pg_catalog."default",
    CONSTRAINT facturas_pkey PRIMARY KEY (id),
    CONSTRAINT facturas_cita_id_key UNIQUE (cita_id),
    CONSTRAINT facturas_numero_factura_key UNIQUE (numero_factura),
    CONSTRAINT fk1qiuk10rfkovhlfpsk7oic0v8 FOREIGN KEY (cliente_id)
        REFERENCES public.clientes (usuario_id) MATCH SIMPLE
        ON UPDATE NO ACTION
        ON DELETE NO ACTION,
    CONSTRAINT fkq9vi47c56fdqslypxnlj8l2f8 FOREIGN KEY (cita_id)
        REFERENCES public.citas (id) MATCH SIMPLE
        ON UPDATE NO ACTION
        ON DELETE NO ACTION,
    CONSTRAINT facturas_estado_check CHECK (estado::text = ANY (ARRAY['PENDIENTE'::character varying, 'PAGADA'::character vary
)

TABLESPACE pg_default;

ALTER TABLE IF EXISTS public.facturas
    OWNER to postgres;
```

Tabla de Clientes

```
-- Table: public.clientes

-- DROP TABLE IF EXISTS public.clientes;

CREATE TABLE IF NOT EXISTS public.clientes
(
    fecha_nacimiento date NOT NULL,
    usuario_id bigint NOT NULL,
    dpi character varying(13) COLLATE pg_catalog."default" NOT NULL,
    telefono character varying(15) COLLATE pg_catalog."default" NOT NULL,
    nombre_completo character varying(100) COLLATE pg_catalog."default" NOT NULL,
    direccion character varying(255) COLLATE pg_catalog."default" NOT NULL,
    CONSTRAINT clientes_pkey PRIMARY KEY (usuario_id),
    CONSTRAINT clientes_dpi_key UNIQUE (dpi),
    CONSTRAINT fkk6iwsq3kts1bblivkjy6epajx FOREIGN KEY (usuario_id)
        REFERENCES public.usuarios (id) MATCH SIMPLE
        ON UPDATE NO ACTION
        ON DELETE NO ACTION
)

TABLESPACE pg_default;

ALTER TABLE IF EXISTS public.clientes
    OWNER to postgres;
```

Tabla de Citas

```
CREATE TABLE IF NOT EXISTS public.citas
(
    cliente_id bigint NOT NULL,
    fecha_actualizacion timestamp(6) without time zone,
    fecha_cancelacion timestamp(6) without time zone,
    fecha_confirmacion timestamp(6) without time zone,
    fecha_creacion timestamp(6) without time zone NOT NULL,
    fecha_hora timestamp(6) without time zone NOT NULL,
    id bigint NOT NULL GENERATED BY DEFAULT AS IDENTITY ( INCREMENT 1 START 1 MINVALUE 1 MAXVALUE 9223372036854775807 CACHE 1
    servicio_id bigint NOT NULL,
    estado character varying(20) COLLATE pg_catalog."default" NOT NULL,
    motivo_rechazo character varying(500) COLLATE pg_catalog."default",
    notas character varying(500) COLLATE pg_catalog."default",
    CONSTRAINT citas_pkey PRIMARY KEY (id),
    CONSTRAINT fke3bl1mspam9t5c4rmg7feyn3o FOREIGN KEY (cliente_id)
        REFERENCES public.clientes (usuario_id) MATCH SIMPLE
        ON UPDATE NO ACTION
        ON DELETE NO ACTION,
    CONSTRAINT fkl89m4psdo9tdi6smrdq3n3p33 FOREIGN KEY (servicio_id)
        REFERENCES public.servicios (id) MATCH SIMPLE
        ON UPDATE NO ACTION
        ON DELETE NO ACTION,
    CONSTRAINT citas_estado_check CHECK (estado::text = ANY (ARRAY['PENDIENTE'::character varying, 'CONFIRMADA'::character va
)
```

```
TABLESPACE pg_default;

ALTER TABLE IF EXISTS public.citas
    OWNER to postgres;
```

Tabla de Auditoria_logs

```
-- Table: public.auditoria_logs

-- DROP TABLE IF EXISTS public.auditoria_logs;

CREATE TABLE IF NOT EXISTS public.auditoria_logs
(
    entidad_id bigint,
    fecha_operacion timestamp(6) without time zone NOT NULL,
    id bigint NOT NULL GENERATED BY DEFAULT AS IDENTITY ( INCREMENT 1 START 1 MINVALUE 1 MAXVALUE 9223372036854775807 CACHE 1 )
    usuario_id bigint,
    ip_address character varying(45) COLLATE pg_catalog."default",
    rol character varying(50) COLLATE pg_catalog."default",
    username character varying(50) COLLATE pg_catalog."default",
    entidad character varying(100) COLLATE pg_catalog."default" NOT NULL,
    operacion character varying(100) COLLATE pg_catalog."default" NOT NULL,
    descripcion character varying(500) COLLATE pg_catalog."default",
    datos_adicionales text COLLATE pg_catalog."default",
    CONSTRAINT auditoria_logs_pkey PRIMARY KEY (id)
)

TABLESPACE pg_default;

ALTER TABLE IF EXISTS public.auditoria_logs
    OWNER to postgres;
```

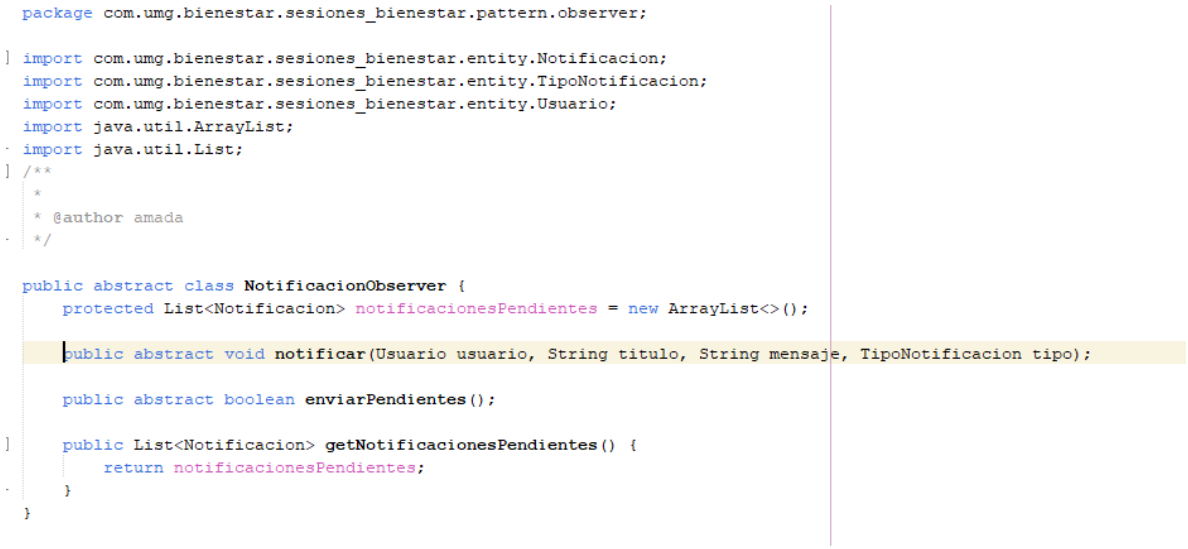
3. Descripción de Clases Principales

3.1 Aplicación de Programación Orientada a Objetos (POO)

3.1.1. Abstracción

- **Clase NotificacionObserver**

Aplicación: Define el contrato que todas las implementaciones de notificadores deben cumplir, ocultando detalles de implementación específicos.



```java
package com.umg.bienestar.sesiones_bienestar.pattern.observer;

import com.umg.bienestar.sesiones_bienestar.entity.Notificacion;
import com.umg.bienestar.sesiones_bienestar.entity.TipoNotificacion;
import com.umg.bienestar.sesiones_bienestar.entity.Usuario;
import java.util.ArrayList;
import java.util.List;
/**
 *
 * @author amada
 */

public abstract class NotificacionObserver {
    protected List<Notificacion> notificacionesPendientes = new ArrayList<>();

    public abstract void notificar(Usuario usuario, String titulo, String mensaje, TipoNotificacion tipo);

    public abstract boolean enviarPendientes();

    public List<Notificacion> getNotificacionesPendientes() {
        return notificacionesPendientes;
    }
}
```

3.1.2. Encapsulamiento

El encapsulamiento se aplica en todas las entidades mediante:

- **Atributos privados** con acceso controlado mediante getters/setters
- **Validaciones** en los métodos setters
- **Ocultamiento de la implementación** interna

Ejemplo: Clase Cliente

```java
@Entity
@Table(name = "clientes")
@PrimaryKeyJoinColumn(name = "usuario_id")
public class Cliente extends Usuario {

    @NotBlank(message = "El nombre completo es obligatorio")
    @Size(min = 3, max = 100, message = "El nombre debe tener entre 3 y 100 caracteres")
    @Column(name = "nombre_completo", nullable = false, length = 100)
    private String nombreCompleto;

    @NotBlank(message = "El DPI es obligatorio")
    @Pattern(regexp = "\\d{13}", message = "El DPI debe tener 13 dígitos")
    @Column(unique = true, nullable = false, length = 13)
    private String dpi;

    @NotBlank(message = "El teléfono es obligatorio")
    @Pattern(regexp = "\\d{8,15}", message = "El teléfono debe tener entre 8 y 15 dígitos")
    @Column(nullable = false, length = 15)
    private String telefono;

    @NotBlank(message = "La dirección es obligatoria")
    @Size(max = 255, message = "La dirección no puede exceder 255 caracteres")
    @Column(nullable = false)
    private String direccion;

    @NotNull(message = "La fecha de nacimiento es obligatoria")
    @Past(message = "La fecha de nacimiento debe ser en el pasado")
    @Column(name = "fecha_nacimiento", nullable = false)
    private LocalDate fechaNacimiento;
```

```java
    @OneToMany(mappedBy = "cliente", cascade = CascadeType.ALL, fetch = FetchType.LAZY)
    @JsonManagedReference("cliente-citas")
    private List<Cita> citas = new ArrayList<>();

    @OneToMany(mappedBy = "cliente", cascade = CascadeType.ALL, fetch = FetchType.LAZY)
    @JsonManagedReference("cliente-facturas")
    private List<Factura> facturas = new ArrayList<>();

    public Cliente() {
        super();
    }

    public Cliente(String username, String password, String email, String nombreCompleto,
                   String dpi, String telefono, String direccion, LocalDate fechaNacimiento) {
        super(username, password, email, RolUsuario.CLIENTE);
        this.nombreCompleto = nombreCompleto;
        this.dpi = dpi;
        this.telefono = telefono;
        this.direccion = direccion;
        this.fechaNacimiento = fechaNacimiento;
    }

    public String getNombreCompleto() {
        return nombreCompleto;
    }

    public void setNombreCompleto(String nombreCompleto) {
        this.nombreCompleto = nombreCompleto;
    }
```

```java
public void setDpi(String dpi) {
    this.dpi = dpi;
}

public String getTelefono() {
    return telefono;
}

public void setTelefono(String telefono) {
    this.telefono = telefono;
}

public String getDireccion() {
    return direccion;
}

public void setDireccion(String direccion) {
    this.direccion = direccion;
}

public LocalDate getFechaNacimiento() {
    return fechaNacimiento;
}

public void setFechaNacimiento(LocalDate fechaNacimiento) {
    this.fechaNacimiento = fechaNacimiento;
}

public List<Cita> getCitas() {
    return citas;
}
```

3.1.2 Herencia

El sistema implementa herencia en la jerarquía de usuarios: Usuario -> Cliente

Herencia: Clase pare Abstracta

```java
@Entity
@Table(name = "usuarios")
@Inheritance(strategy = InheritanceType.JOINED)
public class Usuario {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @NotBlank(message = "El nombre de usuario es obligatorio")
    @Size(min = 4, max = 50, message = "El nombre de usuario debe tener entre 4 y 50 caracteres")
    @Column(unique = true, nullable = false, length = 50)
    private String username;

    @NotBlank(message = "La contraseña es obligatoria")
    @Size(min = 8, message = "La contraseña debe tener al menos 8 caracteres")
    @Column(nullable = false)
    private String password;

    @NotBlank(message = "El email es obligatorio")
    @Email(message = "El email debe ser válido")
    @Column(unique = true, nullable = false, length = 100)
    private String email;

    @NotNull(message = "El rol es obligatorio")
    @Enumerated(EnumType.STRING)
    @Column(nullable = false, length = 20)
    private RolUsuario rol;
```

Herencia: Clase Hija

```java
@Entity
@Table(name = "clientes")
@PrimaryKeyJoinColumn(name = "usuario_id")
public class Cliente extends Usuario {

    @NotBlank(message = "El nombre completo es obligatorio")
    @Size(min = 3, max = 100, message = "El nombre debe tener entre 3 y 100 caracteres")
    @Column(name = "nombre_completo", nullable = false, length = 100)
    private String nombreCompleto;

    @NotBlank(message = "El DPI es obligatorio")
    @Pattern(regexp = "\\d{13}", message = "El DPI debe tener 13 dígitos")
    @Column(unique = true, nullable = false, length = 13)
    private String dpi;

    @NotBlank(message = "El teléfono es obligatorio")
    @Pattern(regexp = "\\d{8,15}", message = "El teléfono debe tener entre 8 y 15 dígitos")
    @Column(nullable = false, length = 15)
    private String telefono;

    @NotBlank(message = "La dirección es obligatoria")
    @Size(max = 255, message = "La dirección no puede exceder 255 caracteres")
    @Column(nullable = false)
    private String direccion;

    @NotNull(message = "La fecha de nacimiento es obligatoria")
    @Past(message = "La fecha de nacimiento debe ser en el pasado")
    @Column(name = "fecha_nacimiento", nullable = false)
    private LocalDate fechaNacimiento;
```

FK hacia la table padre

Ventajas de esta implementación:

- Reutilización de código (username, password, email en Usuario)
- Extensibilidad (fácil agregar Administrador, Recepcionista)
- Una sola tabla para autenticación
- Datos específicos separados por rol

3.1.3 Interfaces

El sistema define contratos mediante interfaces:

- **Interfaces: Contrato para entidades gestionables**

```java
/**
 *
 * @author amada
 */
public interface IGestionable {
    Long getId();
    void setId(Long id);
    boolean isActivo();
    void setActivo(boolean activo);
    LocalDateTime getFechaCreacion();
    LocalDateTime getFechaActualizacion();
    void activar();
    void desactivar();
}
```

```java
/
public interface IFacturable {
    Long getId();
    Long getClienteId();
    double calcularMonto();
    boolean esFacturable();
}
```

- **Implementación en cita**

```java
public class Cita implements IGestionable,IFacturable {
 private Long id;
    private LocalDateTime fechaHora;
    private EstadoCita estado;
    private String notas;
    private String motivoRechazo;
    private LocalDateTime fechaCreacion;
    private LocalDateTime fechaActualizacion;

    // Referencias por ID
    private Long clienteId;
    private Long servicioId;
    private Long usuarioModificacionId;

     // Constructores
    public Cita() {
        this.estado = EstadoCita.PENDIENTE;
        this.fechaCreacion = LocalDateTime.now();
        this.fechaActualizacion = LocalDateTime.now();
    }
```

Implementamos IGestionables, IFacturable

3.1.4 Polimorfismo

El polimorfismo se aplica en múltiples formas:

a) Polimorfismo De Herencia

Es cuando una subclase **sobrescribe** (override) un método de su clase padre.

Clase padre Usuario- Package.model

```
// Método que pueden sobrescribir las clases hijas
protected abstract boolean tienePermisoParaRecurso(String recurso);
```

Clase Hija Administrador

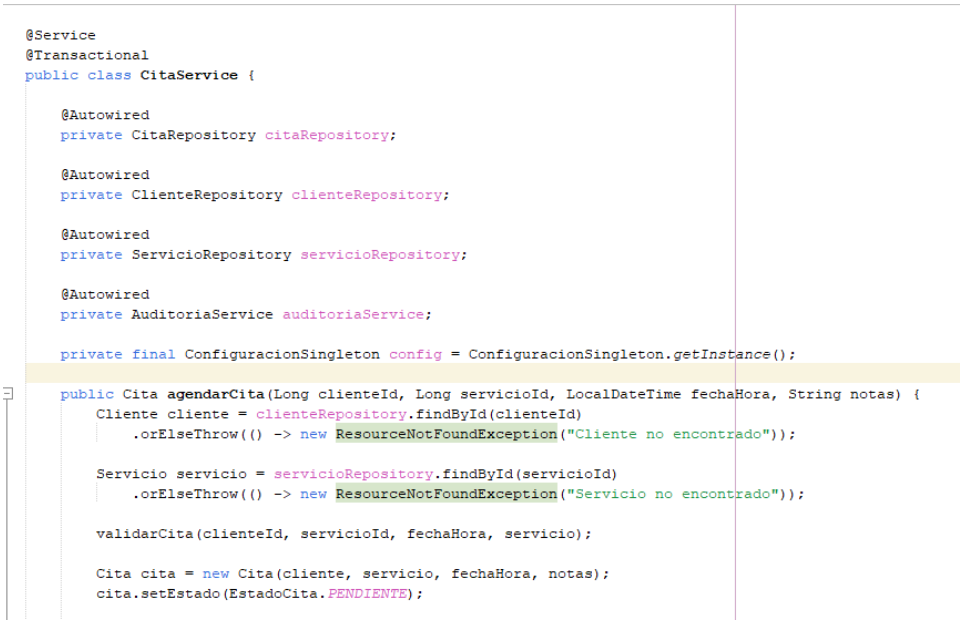
```
// Implementación del método abstracto
@Override
protected boolean tienePermisoParaRecurso(String recurso) {
    return permisos.contains("SUPER_ADMIN") || tienePermisoEspecifico(recurso);
}
```

3.2 Principios SOLID

S - Single Responsibility Principle (SRP)

Clase: CitaService.java

Esta clase tiene **una única responsabilidad**: gestionar la lógica de negocio relacionada con citas. Porque tiene responsabilidad para agendar citas, confirmar, rechazar, cancelar, etc



```
@Service
@Transactional
public class CitaService {

    @Autowired
    private CitaRepository citaRepository;

    @Autowired
    private ClienteRepository clienteRepository;

    @Autowired
    private ServicioRepository servicioRepository;

    @Autowired
    private AuditoriaService auditoriaService;

    private final ConfiguracionSingleton config = ConfiguracionSingleton.getInstance();

    public Cita agendarCita(Long clienteId, Long servicioId, LocalDateTime fechaHora, String notas) {
        Cliente cliente = clienteRepository.findById(clienteId)
            .orElseThrow(() -> new ResourceNotFoundException("Cliente no encontrado"));

        Servicio servicio = servicioRepository.findById(servicioId)
            .orElseThrow(() -> new ResourceNotFoundException("Servicio no encontrado"));

        validarCita(clienteId, servicioId, fechaHora, servicio);

        Cita cita = new Cita(cliente, servicio, fechaHora, notas);
        cita.setEstado(EstadoCita.PENDIENTE);
```

Clase: AuditoriaService.java

Responsabilidad única: Registrar eventos de auditoría del sistema.

```java
 * @author amada
 */
@Service
@Transactional
public class AuditoriaService {

    @Autowired
    private AuditoriaLogRepository auditoriaRepository;

    @Async
    public void registrar(Long usuarioId, String username, String rol,
                          String operacion, String entidad, Long entidadId,
                          String descripcion) {
        AuditoriaLog log = new AuditoriaLog(
            usuarioId, username, rol, operacion,
            entidad, entidadId, descripcion
        );

        try {
            ServletRequestAttributes attributes = (ServletRequestAttributes) RequestContextHolder.getReque
            if (attributes != null) {
                HttpServletRequest request = attributes.getRequest();
                String ipAddress = getClientIp(request);
                log.setIpAddress(ipAddress);
            }
        } catch (Exception e) {
        }

        auditoriaRepository.save(log);
    }
```

Beneficio: Cada clase tiene una razón única para cambiar, facilitando el mantenimiento y las pruebas.

O – Open/closed Principle (OCP)

Patrón Observer: NotificacionObserver.java

El sistema está **abierto a extensión** (nuevos tipos de notificadores) pero **cerrado a modificación** (no requiere cambios en código existente)

```java
package com.umg.bienestar.sesiones_bienestar.pattern.observer;

import com.umg.bienestar.sesiones_bienestar.entity.Notificacion;
import com.umg.bienestar.sesiones_bienestar.entity.TipoNotificacion;
import com.umg.bienestar.sesiones_bienestar.entity.Usuario;
import java.util.ArrayList;
import java.util.List;
/**
 *
 * @author amada
 */

public abstract class NotificacionObserver {
    protected List<Notificacion> notificacionesPendientes = new ArrayList<>();

    public abstract void notificar(Usuario usuario, String titulo, String mensaje, TipoNotificacion tipo);

    public abstract boolean enviarPendientes();

    public List<Notificacion> getNotificacionesPendientes() {
        return notificacionesPendientes;
    }
}
```

Clase base abstracta (cerrada a modificación)

Extensión sin modificar la base (abierta a extensión)

```java
 * @author amada
 */
@Component
public class EmailNotificacionObserver extends NotificacionObserver {

    @Autowired
    private NotificacionRepository notificacionRepository;

    @Override
    public void notificar(Usuario usuario, String titulo, String mensaje, TipoNotificacion tipo) {
        Notificacion notificacion = new Notificacion(usuario, tipo, titulo, mensaje);
        notificacionesPendientes.add(notificacion);
        System.out.println("Email notificación creada para: " + usuario.getEmail());
    }

    @Override
    public boolean enviarPendientes() {
        for (Notificacion notif : notificacionesPendientes) {
            System.out.println("Enviando email a: " + notif.getUsuario().getEmail());
            System.out.println("Título: " + notif.getTitulo());
            System.out.println("Mensaje: " + notif.getMensaje());

            notif.setEnviada(true);
            notif.setFechaEnvio(LocalDateTime.now());

            notificacionRepository.save(notif);
        }
        notificacionesPendientes.clear();
        return true;
    }
}
```

Beneficio: Agregar nuevos tipos de notificaciones no requiere modificar el código existente.

L – Liskov Subtitution Principle (LSP)

Herencia: Cliente extends Usuario

Las instancias de Cliente pueden sustituir a instancias de Usuario sin romper el comportamiento esperado.

```java
@Entity
@Table(name = "clientes")
@PrimaryKeyJoinColumn(name = "usuario_id")
public class Cliente extends Usuario {
```

Beneficio: La jerarquía de herencia respeta el contrato de la clase base.

I - Interface Segregation Principle (ISP)

Interfaces específicas: En lugar de una interfaz "grande", se usan interfaces pequeñas y específicas.

```java
public interface IFacturable {
    Long getId();
    Long getClienteId();
    double calcularMonto();
    boolean esFacturable();
}
```

```java
public interface IGestionable {
    Long getId();
    void setId(Long id);
    boolean isActivo();
    void setActivo(boolean activo);
    LocalDateTime getFechaCreacion();
    LocalDateTime getFechaActualizacion();
    void activar();
    void desactivar();
}
```

Interfaces Segregadas por responsabilidad en Clase Cita

```java
43
44      // Implementación de IGestionable
45      @Override
46      public void activar() {
47          if (estado == EstadoCita.CANCELADA) {
48              this.estado = EstadoCita.PENDIENTE;
49              actualizarFecha();
50          }
51      }
52
53      @Override
54      public void desactivar() {
55          if (puedeSerCancelada()) {
56              this.estado = EstadoCita.CANCELADA;
57              actualizarFecha();
58          }
59      }
60
61      // Implementación de IFacturable
62      @Override
63      public double calcularMonto() {
64          return 0.0;
65      }
66
67      @Override
68      public boolean esFacturable() {
69          return estado == EstadoCita.ATENDIDA;
70      }
```

D - Dependency Inversion Principle (DIP)

Inyección de Dependencias: Los servicios dependen de abstracciones (interfaces de repositorios), no de implementaciones concretas.

```java
@Transactional
public class CitaService {

    @Autowired
    private CitaRepository citaRepository;

    @Autowired
    private ClienteRepository clienteRepository;

    @Autowired
    private ServicioRepository servicioRepository;

    @Autowired
    private AuditoriaService auditoriaService;

    private final ConfiguracionSingleton config = ConfiguracionSingleton.getInstance();

    public Cita agendarCita(Long clienteId, Long servicioId, LocalDateTime fechaHora, String notas) {
        Cliente cliente = clienteRepository.findById(clienteId)
            .orElseThrow(() -> new ResourceNotFoundException("Cliente no encontrado"));

        Servicio servicio = servicioRepository.findById(servicioId)
            .orElseThrow(() -> new ResourceNotFoundException("Servicio no encontrado"));

        validarCita(clienteId, servicioId, fechaHora, servicio);

        Cita cita = new Cita(cliente, servicio, fechaHora, notas);
        cita.setEstado(EstadoCita.PENDIENTE);

        Cita guardada = citaRepository.save(cita);
```

Configuración de Spring: Clase Security Config

```java
@Configuration
@EnableWebSecurity
public class SecurityConfig {

    @Autowired
    private JwtAuthenticationFilter jwtAuthFilter;

    @Bean
    public PasswordEncoder passwordEncoder() {
        return new BCryptPasswordEncoder();
    }

    @Bean
    public SecurityFilterChain securityFilterChain(HttpSecurity http) throws Exception {
        http
            // Deshabilitar CSRF
            .csrf(csrf -> csrf.disable())

            // Configurar CORS
            .cors(cors -> cors.configurationSource(corsConfigurationSource()))

            // Sesiones stateless
            .sessionManagement(session -> session.sessionCreationPolicy(SessionCreationPolicy.STATELESS))

            // Autorización de endpoints
            .authorizeHttpRequests(auth -> auth
                // Endpoints públicos
                .requestMatchers(
                    "/api/auth/**",
```

4. Patrones de Diseño Utilizados

4.1 Patrón Singleton

Clase: ConfiguracionSingleton.java

Propósito: Garantizar una única instancia de configuración global del sistema.

```java
public class ConfiguracionSingleton {
  private static ConfiguracionSingleton instancia;
    private String nombreSistema;
    private String version;
    private int maxCitasPendientesPorCliente;
    private int horasMinAnticipacion;
    private int horasLimiteCancelacion;

    private ConfiguracionSingleton() {
        this.nombreSistema = "Sistema de Gestión de Sesiones de Bienestar";
        this.version = "1.0.0";
        this.maxCitasPendientesPorCliente = 3;
        this.horasMinAnticipacion = 2;
        this.horasLimiteCancelacion = 24;
    }

    public static synchronized ConfiguracionSingleton getInstance() {
        if (instancia == null) {
            instancia = new ConfiguracionSingleton();
        }
        return instancia;
    }
```

Uso en el código: Cita Service

```java
@Service
@Transactional
public class CitaService {

    @Autowired
    private CitaRepository citaRepository;

    @Autowired
    private ClienteRepository clienteRepository;

    @Autowired
    private ServicioRepository servicioRepository;

    @Autowired
    private AuditoriaService auditoriaService;

    private final ConfiguracionSingleton config = ConfiguracionSingleton.getInstance();
```

4.2 Patrón Observer

Clases: NotificacionObserver.java, EmailNotificacionObserver.java

Propósito: Notificar a múltiples observadores cuando ocurren eventos en el sistema.

Clase Abstracta (Subject):

```java
package com.umg.bienestar.sesiones_bienestar.pattern.observer;

import com.umg.bienestar.sesiones_bienestar.entity.Notificacion;
import com.umg.bienestar.sesiones_bienestar.entity.TipoNotificacion;
import com.umg.bienestar.sesiones_bienestar.entity.Usuario;
import java.util.ArrayList;
import java.util.List;
/**
 *
 * @author amada
 */

public abstract class NotificacionObserver {
    protected List<Notificacion> notificacionesPendientes = new ArrayList<>();

    public abstract void notificar(Usuario usuario, String titulo, String mensaje, TipoNotificacion tipo);

    public abstract boolean enviarPendientes();
```

Observador Concreto (Email): EmailNotificacionObserver

```java
    @Override
    public void notificar(Usuario usuario, String titulo, String mensaje, TipoNotificacion tipo) {
        Notificacion notificacion = new Notificacion(usuario, tipo, titulo, mensaje);
        notificacionesPendientes.add(notificacion);
        System.out.println("Email notificación creada para: " + usuario.getEmail());
    }

    @Override
    public boolean enviarPendientes() {
        for (Notificacion notif : notificacionesPendientes) {
            System.out.println("Enviando email a: " + notif.getUsuario().getEmail());
            System.out.println("Título: " + notif.getTitulo());
            System.out.println("Mensaje: " + notif.getMensaje());

            notif.setEnviada(true);
            notif.setFechaEnvio(LocalDateTime.now());

            notificacionRepository.save(notif);
        }
        notificacionesPendientes.clear();
        return true;
    }
}
```

Uso en el sistema: GestionCitasFacade

```java
@Component
public class GestionCitasFacade {

    @Autowired
    private CitaService citaService;

    @Autowired
    private SesionService sesionService;

    @Autowired
    private FacturaService facturaService;

    @Autowired
    private EmailNotificacionObserver emailObserver;

    public Cita agendarYNotificar(Long clienteId, Long servicioId, LocalDateTime fechaHora, String notas) {
        Cita cita = citaService.agendarCita(clienteId, servicioId, fechaHora, notas);

        emailObserver.notificar(
            cita.getCliente(),
            "Solicitud de cita recibida",
            "Su solicitud de cita para " + cita.getServicio().getNombre() +
            " el " + cita.getFechaHora() + " ha sido recibida.",
            TipoNotificacion.EMAIL
        );
        emailObserver.enviarPendientes();

        return cita;
    }
```

4.3 Patrón Facade

Clase: GestionCitasFacade.java

Propósito: Proporcionar una interfaz simplificada para operaciones complejas que involucran múltiples servicios.

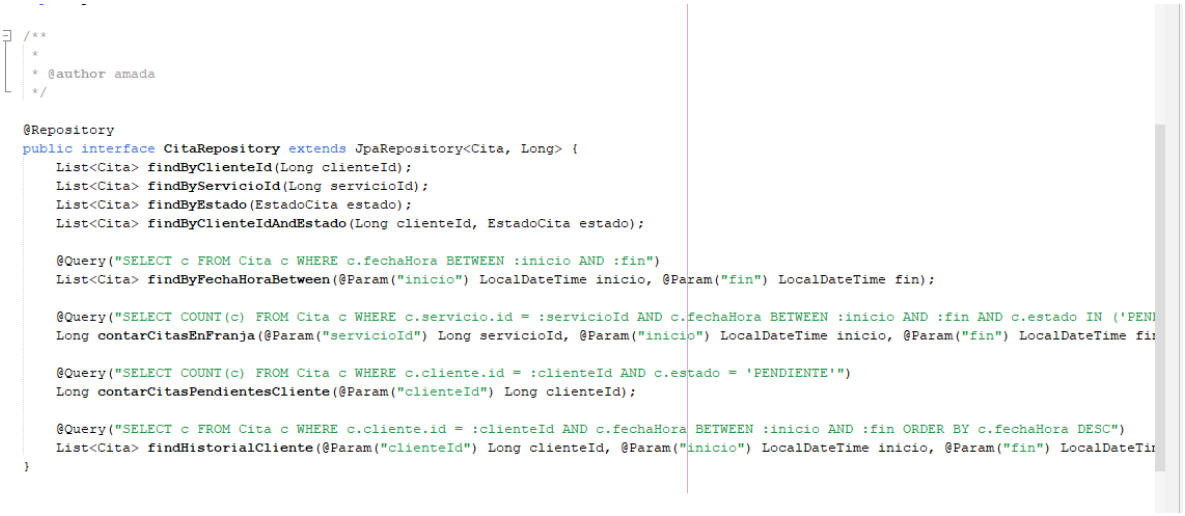

```java
@Component
public class GestionCitasFacade {

    @Autowired
    private CitaService citaService;

    @Autowired
    private SesionService sesionService;

    @Autowired
    private FacturaService facturaService;

    @Autowired
    private EmailNotificacionObserver emailObserver;

    public Cita agendarYNotificar(Long clienteId, Long servicioId, LocalDateTime fechaHora, String notas) {
        Cita cita = citaService.agendarCita(clienteId, servicioId, fechaHora, notas);

        emailObserver.notificar(
            cita.getCliente(),
            "Solicitud de cita recibida",
            "Su solicitud de cita para " + cita.getServicio().getNombre() +
            " el " + cita.getFechaHora() + " ha sido recibida.",
            TipoNotificacion.EMAIL
        );
        emailObserver.enviarPendientes();

        return cita;
    }
```

4.4 Patrón Repository

Interfaz: CitaRepository.java (Spring Data JPA)

Propósito: Abstraer el acceso a datos y proporcionar una capa de persistencia desacoplada.

CitaRepository



```java
/**
 *
 * @author amada
 */

@Repository
public interface CitaRepository extends JpaRepository<Cita, Long> {
    List<Cita> findByClienteId(Long clienteId);
    List<Cita> findByServicioId(Long servicioId);
    List<Cita> findByEstado(EstadoCita estado);
    List<Cita> findByClienteIdAndEstado(Long clienteId, EstadoCita estado);

    @Query("SELECT c FROM Cita c WHERE c.fechaHora BETWEEN :inicio AND :fin")
    List<Cita> findByFechaHoraBetween(@Param("inicio") LocalDateTime inicio, @Param("fin") LocalDateTime fin);

    @Query("SELECT COUNT(c) FROM Cita c WHERE c.servicio.id = :servicioId AND c.fechaHora BETWEEN :inicio AND :fin AND c.estado IN ('PENI
    Long contarCitasEnFranja(@Param("servicioId") Long servicioId, @Param("inicio") LocalDateTime inicio, @Param("fin") LocalDateTime fi

    @Query("SELECT COUNT(c) FROM Cita c WHERE c.cliente.id = :clienteId AND c.estado = 'PENDIENTE'")
    Long contarCitasPendientesCliente(@Param("clienteId") Long clienteId);

    @Query("SELECT c FROM Cita c WHERE c.cliente.id = :clienteId AND c.fechaHora BETWEEN :inicio AND :fin ORDER BY c.fechaHora DESC")
    List<Cita> findHistorialCliente(@Param("clienteId") Long clienteId, @Param("inicio") LocalDateTime inicio, @Param("fin") LocalDateTi
}
```

Uso en Servicios: Cita Service

```java
@Service
@Transactional
public class CitaService {

    @Autowired
    private CitaRepository citaRepository;
```

Beneficio:

- Separa la lógica de acceso a datos de la lógica de negocio
- Facilita el testing con repositorios mock

4.5 Patrón DTO (Data Transfer Object)

Clase: ClienteDTO.java

Propósito: Transferir datos entre capas sin exponer las entidades de dominio directamente.

```java
public class ClienteDTO {
    private Long id;

    @NotBlank(message = "El nombre de usuario es obligatorio")
    private String username;

    @NotBlank(message = "La contraseña es obligatoria")
    @Size(min = 8, message = "La contraseña debe tener al menos 8 caracteres
    private String password;

    @NotBlank(message = "El email es obligatorio")
    @Email(message = "El email debe ser válido")
    private String email;

    @NotBlank(message = "El nombre completo es obligatorio")
    private String nombreCompleto;

    @NotBlank(message = "El DPI es obligatorio")
    @Pattern(regexp = "\\d{13}", message = "El DPI debe tener 13 dígitos")
    private String dpi;

    @NotBlank(message = "El teléfono es obligatorio")
    private String telefono;

    @NotBlank(message = "La dirección es obligatoria")
    private String direccion;

    @NotBlank(message = "La fecha de nacimiento es obligatoria")
    private String fechaNacimiento;
    private Boolean activo;
```

Clase CitaDTO

```java
public class CitaDTO {
    private Long id;

    @NotNull(message = "El cliente es obligatorio")
    private Long clienteId;

    @NotNull(message = "El servicio es obligatorio")
    private Long servicioId;

    @NotNull(message = "La fecha y hora son obligatorias")
    @Future(message = "La cita debe ser en el futuro")
    private LocalDateTime fechaHora;

    private EstadoCita estado;
    private String notas;
    private String motivoRechazo;
```

Uso en Controladores:

Cliente Controller

```java
@RestController
@RequestMapping("/api/clientes")
@Tag(name = "Clientes", description = "API para gestión de clientes")
public class ClienteController {

    @Autowired
    private ClienteService clienteService;

    @PostMapping("/registro")
    @Operation(summary = "Registrar nuevo cliente", description = "UC-01: Permite registrar un nuevo cliente en el siste
    public ResponseEntity<Cliente> registrar(@Valid @RequestBody ClienteDTO clienteDTO) {
        Cliente cliente = clienteService.registrarCliente(clienteDTO);
        return ResponseEntity.status(HttpStatus.CREATED).body(cliente);
    }

    @GetMapping
    @Operation(summary = "Listar todos los clientes")
    public ResponseEntity<List<Cliente>> listarTodos() {
        return ResponseEntity.ok(clienteService.listarTodos());
    }

    @GetMapping("/activos")
    @Operation(summary = "Listar clientes activos")
    public ResponseEntity<List<Cliente>> listarActivos() {
        return ResponseEntity.ok(clienteService.listarActivos());
    }

    @GetMapping("/{id}")
    @Operation(summary = "Obtener cliente por ID")
    public ResponseEntity<Cliente> obtenerPorId(@PathVariable Long id) {
```

Cita Controller

```java
@RestController
@RequestMapping("/api/citas")
@Tag(name = "Citas", description = "API para gestión de citas")
public class CitaController {

    @Autowired
    private CitaService citaService;

      @GetMapping
    @Operation(summary = "Listar todas las citas o filtrar por estado")
    public ResponseEntity<List<CitaDTO>> listar(
        @RequestParam(required = false) EstadoCita estado
    ) {
        List<Cita> citas;

        if (estado != null) {
            citas = citaService.listarPorEstado(estado);
        } else {
            citas = citaService.listarTodas();
        }

        List<CitaDTO> citasDTO = citas.stream()
                .map(this::convertirADTO)
                .collect(Collectors.toList());

        return ResponseEntity.ok(citasDTO);
    }
```

Beneficio:

- Valida datos de entrada con anotaciones
- Protege las entidades de dominio de modificaciones externas
- Evita problemas de serialización circular con relaciones bidireccionales

4.6 Patrón Service Layer

Propósito: Centralizar la lógica de negocio en servicios transaccionales.

Ejemplo: CitaService.java

```java
@Service
@Transactional
public class CitaService {

    @Autowired
    private CitaRepository citaRepository;

    @Autowired
    private ClienteRepository clienteRepository;

    @Autowired
    private ServicioRepository servicioRepository;

    @Autowired
    private AuditoriaService auditoriaService;

    private final ConfiguracionSingleton config = ConfiguracionSingleton.getInstance();

    public Cita agendarCita(Long clienteId, Long servicioId, LocalDateTime fechaHora, String notas) {
        Cliente cliente = clienteRepository.findById(clienteId)
            .orElseThrow(() -> new ResourceNotFoundException("Cliente no encontrado"));

        Servicio servicio = servicioRepository.findById(servicioId)
            .orElseThrow(() -> new ResourceNotFoundException("Servicio no encontrado"));

        validarCita(clienteId, servicioId, fechaHora, servicio);

        Cita cita = new Cita(cliente, servicio, fechaHora, notas);
        cita.setEstado(EstadoCita.PENDIENTE);
```

Beneficio:

- Centraliza la lógica de negocio en un solo lugar
- Transacciones manejadas automáticamente con `@Transactional`
- Facilita el testing y el mantenimiento
- Separa claramente responsabilidades de controladores y repositorios

5. Configuración del Entorno

5.1 Backend (Spring Boot)

Requisitos:

- Java 17+
- Maven 3.8+
- PostgreSQL 14+
- IDE: IntelliJ IDEA / Eclipse

Configuración de Variables de Entorno:

Crear archivo .env en la raíz del proyecto:

Base de datos PostgreSQL

DB_URL=jdbc:postgresql://localhost:5432/bienestar_db

DB_USERNAME=postgres

DB_PASSWORD=tu_password

JWT Secret (mínimo 256 bits)

JWT_SECRET=404E635266556A586E3272357538782F413F4428472B4B6250645367566B5970

JWT_EXPIRATION=86400000

CORS

CORS_ORIGINS=http://localhost:3000,http://localhost:5173

Archivo application.properties:

```
spring.application.name=sesiones-bienestar

# Configuracion PostgreSQL Railway
# Las credenciales se cargan desde variables de entorno (ver .env)
spring.datasource.url=${DB_URL}
spring.datasource.username=${DB_USERNAME}
spring.datasource.password=${DB_PASSWORD}
spring.datasource.driver-class-name=org.postgresql.Driver

# Configuracion JPA/Hibernate
spring.jpa.database=postgresql
spring.jpa.database-platform=org.hibernate.dialect.PostgreSQLDialect
spring.jpa.hibernate.ddl-auto=update
spring.jpa.show-sql=true
spring.jpa.properties.hibernate.format_sql=true
spring.jpa.properties.hibernate.use_sql_comments=true
spring.jpa.properties.hibernate.jdbc.time_zone=UTC

# Pool de conexiones HikariCP
spring.datasource.hikari.maximum-pool-size=10
spring.datasource.hikari.minimum-idle=2
spring.datasource.hikari.idle-timeout=300000
spring.datasource.hikari.connection-timeout=20000

# JWT
jwt.secret=${JWT_SECRET}
jwt.expiration=${JWT_EXPIRATION:86400000}

# CORS Configuration
```

```
# CORS Configuration
spring.web.cors.allowed-origins=${CORS_ORIGINS:http://localhost:3000}
spring.web.cors.allowed-methods=GET,POST,PUT,DELETE,OPTIONS
spring.web.cors.allowed-headers=*
spring.web.cors.allow-credentials=true


# Configuracion de logging
logging.level.org.hibernate.SQL=DEBUG
logging.level.org.hibernate.type.descriptor.sql.BasicBinder=TRACE
logging.level.com.umg.bienestar=DEBUG

#Puerto 8080
server.port=8080
```

Pasos de Instalación:

 # 1. Clonar repositorio

git clone https://github.com/tu-usuario/sesiones-bienestar.git

cd sesiones-bienestar

 # 2. Crear base de datos

psql -U postgres

CREATE DATABASE bienestar_db;

\q

 # 3. Configurar variables de entorno

cp .env.example .env

Editar .env con tus credenciales

 # 4. Compilar proyecto

mvn clean install

 # 5. Ejecutar aplicación

mvn spring-boot:run

Verificación:

- API REST: http://localhost:8080
- Swagger UI: http://localhost:8080/swagger-ui/index.html
- Health Check: http://localhost:8080/actuator/health

5.2 Frontend Web (React)

Requisitos:

- Node.js 18+
- npm 9+ / yarn 1.22+

Configuración:

Crear archivo .env en frontend/:

properties

```
REACT_APP_API_URL=http://localhost:8080/api
```

Pasos de Instalación:

```
# 1. Navegar al directorio frontend
cd frontend

# 2. Instalar dependencias
npm install

# 3. Iniciar servidor de desarrollo
npm start
```

Verificación:

- Frontend: http://localhost:3000
- Login con: admin / admin123

Dependencias Principales:

```
 "dependencies": {
   "react": "^18.2.0",
   "react-dom": "^18.2.0",
   "lucide-react": "^0.263.1"}}
```

5.3 App Móvil Android

Requisitos:

- Android Studio Hedgehog (2023.1.1)+
- JDK 17+
- Android SDK API 24+ (Nougat)
- Emulador Android / Dispositivo físico

Configuración:

Archivo: App Android/app/src/main/java/com/umg/bienestar/cliente/utils/Constants.java

```
public class Constants {
    // OPCIÓN 1: Emulador de Android Studio
    public static final String BASE_URL = "http://10.0.2.2:8080/";
    // OPCIÓN 2: Dispositivo físico (reemplazar con tu IP local)
    // public static final String BASE_URL = "http://192.168.1.X:8080/";
}
```

Pasos de Instalación:

 # 1. Abrir proyecto en Android Studio

File → Open → Seleccionar carpeta "App Android"

 # 2. Sincronizar Gradle

Build → Make Project

 # 3. Configurar emulador

Tools → Device Manager → Create Device

- Phone: Pixel 6

- System Image: API 34 (Android 14)

 # 4. Ejecutar aplicación

Run → Run 'app'

Configuración de Red:

Para Emulador:

BASE_URL = "http://10.0.2.2:8080/";

Para Dispositivo Físico:

1. Obtener IP de tu PC:

 # Windows

ipconfig

 # Linux/Mac

ifconfig

2. **Configurar en Constants.java**
   BASE_URL = "http://192.168.1.105:8080/"; // Tu IP local

3. Asegurar que PC y dispositivo estén en la misma red WiFi

Verificación:

- Login con: admin / admin123
- Verificar logs en Logcat para errores de conexión