DOCUMENTO FINAL DE INGENIERÍA DE SOFTWARE

CheTurno

Agustin Palma, Luis Córdoba - Facultad de Ingeniería UNPSJB

Índice General

3.4. Reglas de negocio críticas

4. Calidad del software

4.1. Estrategia de testing

4.2. Qué se probó y qué quedó pendiente

4.3. Manejo de errores y seguridad

4.4. Auditoría

5. Roadmap futuro (continuidad del producto)

5.1. Pendientes del backlog ordenados

1. Visión y alcance del producto

1.1. Visión del sistema

CheTurno se concibe como una plataforma integral de gestión de turnos médicos basada en una arquitectura **Software as a Service (SaaS)**. Su visión es democratizar la tecnología de gestión clínica, permitiendo que múltiples centros de atención (tenants) operen de manera segura, aislada y escalable sobre una única infraestructura compartida.

El sistema busca transformar la experiencia de acceso a la salud, evolucionando de modelos tradicionales de atención presencial o telefónica hacia un ecosistema digital autogestionable. El objetivo final es proveer una herramienta que garantice la **optimización de recursos físicos y humanos** (consultorios y médicos) mediante algoritmos de distribución inteligente, asegurando al mismo tiempo la trazabilidad total de las operaciones mediante un sistema de auditoría inmutable.

1.2. Problema resuelto

El desarrollo del sistema aborda y soluciona las siguientes problemáticas críticas en la gestión ambulatoria:

- **Ineficiencia operativa:** Elimina la dependencia de agendas en papel o sistemas legados desconectados que provocan solapamiento de turnos y subutilización de consultorios.
- **Cuellos de botella en recepción:** Reduce drásticamente la carga laboral del personal administrativo al delegar la reserva, reprogramación y cancelación de citas al paciente.
- **Ausentismo:** Mitiga el ausentismo mediante recordatorios automáticos y facilidades para la cancelación temprana, liberando slots para otros pacientes (lista de espera).
- **Gestión multi-sede fragmentada:** Resuelve la complejidad de administrar múltiples centros médicos con una lógica unificada,

permitiendo a los administradores gestionar recursos distribuidos desde un único dashboard.

- **Falta de Trazabilidad:** Soluciona la carencia de datos históricos sobre quién modificó un turno o configuración, proporcionando seguridad jurídica y operativa mediante auditoría detallada.

1.3. Público objetivo

El sistema está diseñado para interactuar con cuatro perfiles de actores principales, jerarquizados mediante un modelo de seguridad RBAC (Role-Based Access Control):

1. **Pacientes (usuario final):** Buscan autonomía para reservar, visualizar y cancelar turnos desde cualquier dispositivo sin restricciones horarias. Valoran la simplicidad y la confirmación inmediata vía correo electrónico.
2. **Profesionales de la salud (médicos):** Requieren visibilidad clara de su agenda diaria y semanal, así como la garantía de que sus horarios y consultorios están correctamente asignados sin conflictos.
3. **Personal de recepción (operadores):** Necesitan herramientas ágiles para la gestión de turnos en nombre de terceros (pacientes presenciales o telefónicos), gestión de listas de espera y validación de asistencia.
4. **Administradores del centro:** Responsables de la configuración estratégica del *tenant*: alta de profesionales, definición de especialidades, gestión de consultorios físicos y análisis de métricas de rendimiento (KPIs).

1.4. Alcance funcional

El sistema abarca los siguientes módulos funcionales, soportados por una arquitectura full stack:

- **Gestión del ciclo de vida del turno:**
    - Motor de reservas con validación de disponibilidad en tiempo real.

    - Máquina de estados estricta (Programado, Confirmado, Cancelado, Completado, Ausente).
    - Lógica de "Deep Linking" para confirmación y cancelación rápida vía email sin necesidad de inicio de sesión previo.
    - Gestión automatizada de listas de espera ante liberaciones de cupos.
- **Administración de infraestructura médica (SaaS):**
    - Gestión Multi-tenant de centros de atención.
    - Administración de consultorios y asignación de recursos físicos.
    - Catálogo global de especialidades médicas.
- **Gestión de agenda y disponibilidad:**
    - Configuración de disponibilidad médica semanal y esquemas de turnos.
    - Manejo de excepciones temporales (feriados, licencias, bloqueos por mantenimiento).
    - Detección y prevención automática de conflictos de horarios (solapamientos).
- **Seguridad y usuarios:**
    - Autenticación híbrida: Credenciales propias (JWT) y OAuth2 (Google).
    - Flujo de registro de pacientes con validación de identidad vía email.
    - Recuperación segura de credenciales.
- **Calidad y auditoría:**
    - Registro inmutable (AuditLog) de todas las transacciones críticas.
    - Sistema de encuestas de satisfacción post-atención.
    - Dashboards de gestión con métricas de ocupación y desempeño.

1.5. Exclusiones (qué NO hace)

Para garantizar la viabilidad del proyecto dentro de los plazos de ingeniería establecidos, se definen explícitamente los siguientes límites del alcance:

- **Gestión financiera:** El sistema no procesa pagos, no emite facturas electrónicas, ni gestiona cajas diarias o liquidaciones de honorarios médicos.
- **Historia clínica electrónica (HCE):** No se almacena información médica sensible del paciente (diagnósticos, tratamientos, recetas o evolución clínica). El sistema se limita estrictamente a la gestión administrativa del turno.
- **Mensajería instantánea:** No se realizan integraciones con servicios de SMS ni WhatsApp Business API; todas las notificaciones se canalizan exclusivamente vía correo electrónico (SMTP).
- **Aplicación móvil nativa:** El alcance se limita a una aplicación web *Responsive* (PWA) accesible desde navegadores móviles y de escritorio, sin desarrollo de apps nativas para iOS o Android.

2. Arquitectura final del sistema

2.1. Diagrama general de arquitectura

El sistema **CheTurno** implementa una arquitectura **Cliente-Servidor de tres capas** (presentación, lógica de negocio y persistencia), desacoplada y comunicada mediante servicios RESTful. Este diseño garantiza la modularidad, facilita el mantenimiento independiente de cada capa y soporta la escalabilidad requerida por el modelo SaaS.

La infraestructura se basa en contenedores docker orquestados, asegurando que cada componente (frontend, backend, base de datos) se ejecute en un entorno aislado y reproducible.

Figura 1: Arquitectura de alto nivel del sistema Turnero Web.

https://i.imgur.com/zHVCyaI.png

2.2. Decisiones arquitectónicas justificadas

Para satisfacer los requisitos funcionales y no funcionales, se tomaron las siguientes decisiones estructurales clave:

1. Modelo SaaS Multi-Tenant (tenencia lógica)
    - **Decisión:** Se optó por una arquitectura *Multi-tenant* basada en discriminación por columna (columna `centro_atencion_id` en tablas críticas) sobre una base de datos compartida.
    - **Justificación:** Esta estrategia maximiza la eficiencia de recursos y simplifica el mantenimiento en comparación con tener una base de datos por cliente. Permite gestionar múltiples clínicas desde una única instancia de la aplicación, reduciendo drásticamente los costos operativos y de infraestructura.

2. Separacion frontend-backend (SPA + API REST)
    - **Decisión:** El Frontend es una *Single Page Application* (SPA) construida en **Angular 19**, mientras que el Backend es una API REST en **Spring Boot 3**.
    - **Justificación:**
        - **Angular:** Proporciona una experiencia de usuario fluida y reactiva, minimizando la carga en el servidor al transferir la lógica de presentación al cliente.
        - **Spring Boot:** Ofrece un ecosistema robusto para la lógica empresarial compleja (gestión de concurrencia en turnos, seguridad), tipado estático fuerte con Java 17 y una integración excelente con herramientas de seguridad y acceso a datos.

3. Seguridad stateless con JWT
    - **Decisión:** Implementación de autenticación sin estado (*stateless*) utilizando JSON Web Tokens (JWT).
    - **Justificación:** Al no almacenar sesiones en el servidor, la arquitectura es fácilmente escalable horizontalmente. El token auto-contenido permite validar la identidad y los roles del usuario en cada petición sin necesidad de consultas constantes a la base de datos de sesiones.

4. Motor de base de datos relacional
    - **Decisión:** Uso de **PostgreSQL**.
    - **Justificación:** La naturaleza de los datos médicos y de turnos requiere una integridad referencial estricta y soporte transaccional ACID (atomicidad, consistencia, aislamiento, durabilidad) para evitar inconsistencias como la doble reserva de turnos (sobreturnos no deseados).

2.3. Diagrama Entidad–Relación

El modelo de datos está diseñado para soportar la integridad referencial y la segregación de datos por centro de atención. Las entidades principales incluyen `Turno`, `Paciente`, `Medico`, `CentroAtencion` y `AuditLog`.

Se destaca la entidad `StaffMedico` como tabla de unión que permite a un mismo profesional trabajar en múltiples centros con distintas disponibilidades, rompiendo la rigidez de modelos tradicionales.

Figura 2: Diagrama Entidad-Relación (DER) del sistema.

https://i.imgur.com/TL4dgqK.png

2.4. Flujo de autenticación

La seguridad del sistema se gestiona a través de **Spring Security**, implementando un esquema de control de acceso basado en roles (RBAC) jerárquico.

El flujo de autenticación sigue los siguientes pasos:

1. **Solicitud de acceso:** El cliente (angular) envía las credenciales (email/password) o un token de OAuth2 (google) al controlador de autenticación (`AuthController`).
2. **Validación:**
    - Para credenciales nativas: El `AuthenticationManager` verifica el hash de la contraseña contra la base de datos.
    - Para Google: El `GoogleAuthService` valida la integridad del token con los proveedores de identidad de Google.
3. **Generación de token:** Si la identidad es válida, el `JwtTokenProvider` genera un **JWT firmado** que encapsula:
    - La identidad del usuario (Subject).
    - El Rol (`ROLE_ADMIN`, `ROLE_MEDICO`, etc.).
    - El `centro_atencion_id` (contexto del tenant).
4. **Intercepción de peticiones:** El `JwtAuthenticationFilter` intercepta cada solicitud HTTP subsiguiente, extrae el token del encabezado `Authorization: Bearer`, valida su firma y caducidad, y establece el contexto de seguridad en el hilo de ejecución actual.

Este mecanismo garantiza que ninguna operación de negocio (como crear un turno) pueda ejecutarse sin un contexto de seguridad válido y autorizado.

2.5. Decisiones sobre operación y despliegue

Para garantizar la disponibilidad y facilitar el mantenimiento en un entorno productivo, se definieron las siguientes estrategias de operación sobre un servidor privado virtual (VPS):

1. Contenedorización total (Docker)

- **Decisión:** Todos los componentes del sistema (frontend, backend y base de datos) se despliegan como contenedores aislados utilizando **Docker**.
- **Justificación:** Esto elimina el problema de "funciona en mi máquina", garantizando que el entorno de producción sea idéntico al de desarrollo. Además, facilita la actualización independiente de servicios sin conflictos de dependencias.

2. Orquestación de servicios

- **Decisión:** Se utiliza **Docker Compose** para definir y ejecutar la arquitectura multi-contenedor.
- **Justificación:** Permite levantar toda la infraestructura con un solo comando, gestionando automáticamente la red interna (bridge network) para que el backend pueda comunicarse con la base de datos de forma segura sin exponer puertos innecesarios a internet.

3. Servidor web y proxy inverso (Nginx)

- **Decisión:** El contenedor del frontend utiliza **Nginx** como servidor web de alto rendimiento.
- **Justificación:** Nginx se encarga de servir los archivos estáticos de la aplicación Angular (SPA) y actúa como **reverse proxy**, redirigiendo las peticiones que van a `/api` hacia el contenedor del backend. Esto permite exponer la aplicación en un único puerto (80/443) y simplifica la gestión de certificados SSL.

4. Persistencia de datos

- **Decisión:** Uso de **volúmenes de docker** (docker volumes) para la base de datos PostgreSQL.

- **Justificación:** Asegura que la información de los turnos y pacientes sobreviva al reinicio o actualización de los contenedores, desacoplando los datos del ciclo de vida de la aplicación.

3. Modelos de requisitos

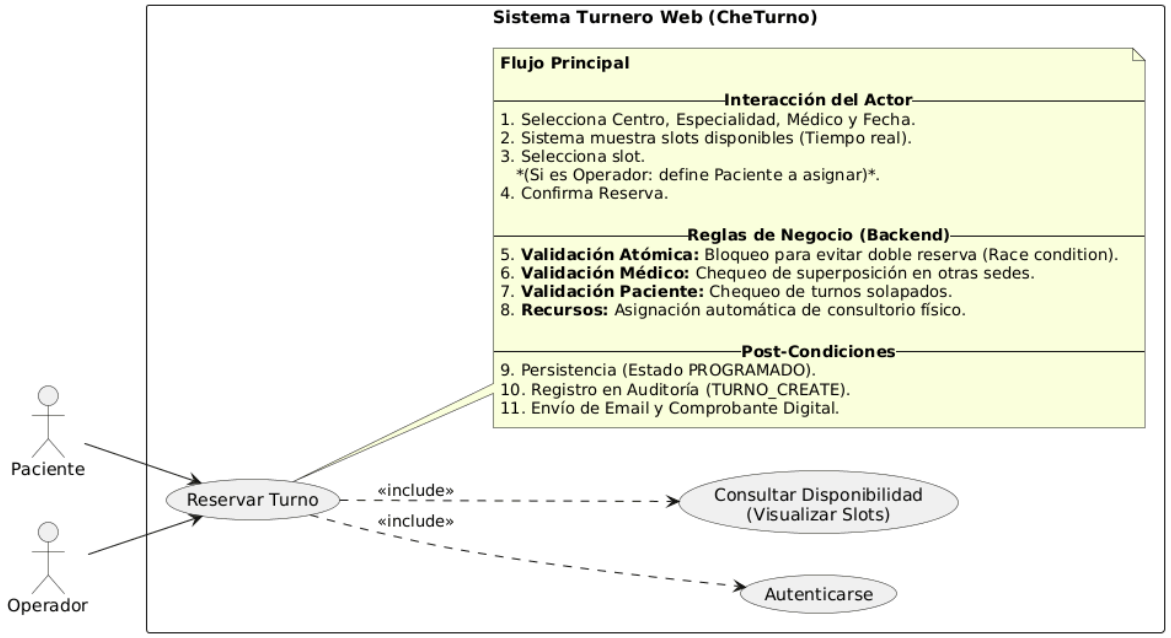
3.1. Diagrama de casos de uso

Para ilustrar la interacción funcional más crítica del sistema, se presenta el diagrama de caso de uso correspondiente a la **reserva de turno** por parte del **paciente**. Este proceso es el núcleo del autoservicio que ofrece la plataforma SaaS.

Descripción del diagrama

El diagrama modela el comportamiento del sistema desde una perspectiva arquitectónica, evitando la descomposición funcional (pasos de algoritmo) y enfocándose en las metas del usuario.

- **Actor:** Paciente / Operador.
- **Casos de uso principales:** Reservar turno.
- **Precondición:** El actor está autenticado y el médico/consultorio seleccionado tiene slots disponibles generados por el `AgendaService`.



Interpretación del flujo

- **Actor Principal:** Paciente / Operador.
- **Precondición:** El actor está autenticado y el médico/consultorio seleccionado tiene slots disponibles generados por el `AgendaService`.
- **Flujo Principal:**
    1. El actor selecciona el centro de atención, la especialidad, el médico y la fecha deseada.
    2. El sistema muestra los slots de tiempo disponibles (cálculo en tiempo real por `AgendaService`).
    3. El actor selecciona el slot de tiempo y, si es operador, al paciente a ser asignado.
    4. El actor confirma la reserva.
    5. **Sistema (backend - servicio transaccional):**
        - **Validación crítica (concurrencia):** Verifica de manera atómica que el slot de tiempo no haya sido tomado por otro usuario al momento de la confirmación, previniendo el *race condition*.

    - **Validación cross-tenant (médico):** Asegura que el médico no posea un turno superpuesto en ninguna otra sede de la plataforma.
    - **Validación de paciente:** Asegura que el paciente no tenga otro turno activo que se solape con el horario solicitado.
    - **Asignación de recursos:** Ejecuta la lógica de `ConsultorioDistribucionService` para asignar un consultorio disponible.
    - Persiste la entidad `turno` con estado inicial `PROGRAMADO`.
    - Genera un registro en `AuditLog` con la acción `TURNO_CREATE`.
    - Envía un email de recordatorio (con deep link para confirmación).
6. El sistema muestra un comprobante digital del turno reservado.

3.2. Casos de uso

A continuación, se describen los flujos transaccionales más relevantes del sistema, detallando la interacción entre los actores y la lógica interna del backend.

CU-01: Iniciar sesión (OAuth2)

- **Actor principal:** Usuario (paciente, médico, operador, administrador).
- **Precondición:** El usuario posee una cuenta activa de Google.
- **Flujo principal:**
    1. El usuario selecciona la opción "Ingresar con Google" en el frontend.
    2. El sistema redirige al proveedor de identidad de Google para la autenticación.
    3. Google retorna un *idToken* firmado al frontend.
    4. El frontend envía este token al endpoint `/api/auth/google`.
    5. **Sistema (backend):**

        - Verifica la integridad del token con Google.
        - Busca si el email ya existe en la base de datos `users`.
        - **Regla de negocio:** Si no existe, registra automáticamente un nuevo usuario con rol `ROLE_PACIENTE` y estado `enabled=true`.
        - Genera un **JWT** propio incluyendo el `centro_atencion_id` y el rol.
    6. El sistema retorna el JWT y el perfil del usuario.
    7. El frontend almacena el JWT en *LocalStorage* y redirige al dashboard correspondiente.

CU-02: Cancelar turno (paciente)

- **Actor principal:** Paciente.
- **Precondición:** Existe un turno en estado `PROGRAMADO` o `CONFIRMADO` y la fecha actual es anterior al turno.
- **Flujo principal:**
    1. El Paciente accede a "Mis Turnos" y selecciona la opción de cancelar.
    2. El sistema solicita confirmación.
    3. **Sistema (Backend):**
        - Valida que el turno pertenezca al usuario autenticado.
        - Cambia el estado de la entidad `Turno` a `CANCELADO`.
        - Libera el *slot* en la agenda del médico.
        - Dispara el proceso de **lista de espera**: busca si hay interesados en ese horario y envía notificaciones de disponibilidad.
        - Genera un registro en `AuditLog` con la acción `TURNO_CANCEL`.
    4. El sistema envía un email de confirmación de cancelación al paciente.

CU-03: Asignar paciente a lista de espera

- **Actor principal:** Operador.
- **Precondición:** No existen turnos disponibles para una Especialidad y Médico específicos en el rango deseado.
- **Flujo principal:**
    1. El operador selecciona el centro de atención, la especialidad y el médico (opcional).
    2. El sistema muestra la agenda completa.
    3. El operador busca al Paciente por dni o email en el sistema.
    4. El operador confirma la inscripción en la lista de espera.
    5. **Sistema:**
        - Valida que el paciente no esté ya inscrito en la lista para esos mismos criterios (`ListaEsperaRepository.exists...`).
        - Crea un registro en la tabla `lista_espera` con `fecha_alta` actual.
    6. El sistema confirma la operación con un mensaje de éxito.

CU-04: Registrar asistencia/ausencia

- **Actor principal:** Médico.
- **Precondición:** El turno está en estado `CONFIRMADO` y la fecha del turno es igual a la fecha actual (`LocalDate.now()`).
- **Flujo principal:**
    1. El médico visualiza su "agenda del día".
    2. Selecciona un turno específico de la grilla.
    3. El médico marca la opción "presente" o "ausente".
    4. **Sistema:**
        - **Validación temporal:** Verifica estrictamente que la operación se realice el mismo día del turno. Si es una fecha distinta, lanza una `BusinessException`.

        - Actualiza el campo `asistio` (boolean) en la entidad `Turno`.
        - Transiciona el estado del turno a `COMPLETADO`.
    5. La tarjeta del turno en la interfaz cambia de color para reflejar el estado final.

CU-05: Gestionar horarios médicos (disponibilidad)

- **Actor principal:** Administrador del centro / médico.
- **Precondición:** El médico debe existir en el `StaffMedico` del centro actual.
- **Flujo principal:**
    1. El actor selecciona "configurar disponibilidad".
    2. Define los días de la semana, hora de inicio, hora de fin, duración del turno (ej. 20 min) y consultorio asignado.
    3. **Sistema:**
        - **Validación de superposición:** Utiliza `AgendaService` para verificar que los nuevos horarios no se solapen con turnos ya otorgados.
        - **Validación cross-tenant:** Verifica que el médico no tenga disponibilidad horaria asignada en *otro* centro de atención al mismo tiempo.
        - Guarda la configuración en `DisponibilidadMedico` y regenera los esquemas de turnos virtuales.

CU-06: Consultar métricas y auditoría

- **Actor principal:** Administrador del centro.
- **Precondición:** Usuario con `ROLE_ADMIN` vinculado al tenant activo.
- **Flujo principal:**
    1. El administrador accede al módulo de "reportes y auditoría".
    2. El sistema presenta un dashboard con gráficos de torta (KPIs de estados de turnos, ausentismo, ocupación).

3. El administrador puede filtrar la tabla de auditoría por rango de fechas, usuario responsable o tipo de entidad (Turno, Paciente, Configuración).
4. **Sistema:**
    - Ejecuta consultas agregadas sobre `TurnoRepository` filtrando por `centro_atencion_id`.
    - Recupera registros inmutables de `AuditLogRepository` para garantizar la trazabilidad.

3.3. User Stories del producto final

El alcance funcional del sistema se definió mediante un conjunto de historias de usuario (User Stories) priorizadas para maximizar el valor entregado a pacientes, operadores y administradores de centro. A continuación, se describen las principales funcionalidades implementadas agrupadas por módulos.

Gestión de identidad y acceso seguro

Para garantizar la integridad de la información médica, se implementaron historias centradas en la seguridad del acceso.

- **Inicio de sesión y OAuth2:** Los usuarios pueden autenticarse mediante credenciales propias (con validación segura JWT) o utilizar su cuenta de **Google** para un acceso rápido sin recordar nuevas contraseñas.
- **Onboarding flexible:** Se desarrollaron dos flujos de registro: el **auto-registro** web para pacientes con validación de correo electrónico único y contraseñas robustas, y el **registro asistido** por operadores para dar de alta presencialmente a pacientes que no utilizan tecnología.
- **Recuperación de Acceso:** Se incluyó un mecanismo seguro de recuperación de contraseña mediante enlaces temporales de un solo uso enviados por email, protegiendo la cuenta ante olvidos.

Experiencia del paciente y gestión de turnos

El núcleo del sistema busca simplificar la interacción del paciente con la agenda médica.

- **Reserva guiada:** El paciente visualiza la disponibilidad en tiempo real y puede reservar turnos siguiendo un flujo validado que impide conflictos de horarios.
- **Preferencias horarias:** Para optimizar la búsqueda, se implementó la capacidad de filtrar turnos basándose en las **franjas horarias preferidas** del paciente (ej. "solo mañanas"), personalizando la experiencia de reserva.
- **Gestión de cancelaciones:** Ante la cancelación de un turno (por parte del médico o sistema), el paciente recibe una notificación inmediata con la razón y opciones para reprogramar, asegurando que esté informado a tiempo.

Eficiencia operativa y lista de espera inteligente

Estas historias se enfocan en maximizar la ocupación de los consultorios y reducir los tiempos muertos.

- **Gestión de demanda insatisfecha:** Se desarrolló una **lista de espera inteligente** que permite a los operadores registrar pacientes que no encontraron cupo. Cuando se libera un turno por cancelación, el sistema identifica automáticamente a los candidatos compatibles en espera, permitiendo reasignar el recurso rápidamente y notificar al beneficiado.
- **Calidad de atención:** Para medir la satisfacción sin generar ruido ("spam") al usuario, el disparo de **encuestas de satisfacción** se configuró estrictamente para ocurrir sólo después de turnos con estado "completado", excluyendo ausencias o cancelaciones.

Sistema de notificaciones transversales

Como soporte a todas las operaciones, se implementó un módulo de comunicaciones escalable.

- El sistema notifica proactivamente sobre eventos críticos (confirmación de reserva, recordatorios, cambios de estado) a través de **correo electrónico**, manteniendo un registro histórico de las comunicaciones enviadas para auditoría y transparencia.

3.4. Reglas de negocio críticas

Las siguientes reglas gobiernan la integridad transaccional y lógica del sistema, siendo de cumplimiento obligatorio para cualquier operación realizada a través de la API o la interfaz de usuario.

RN-01: Unicidad temporal del paciente (prevención de solapamiento)

- **Descripción:** El sistema prohíbe que un mismo paciente posea dos turnos activos (estados `PROGRAMADO` o `CONFIRMADO`) que se solapen en fecha y hora, independientemente de la especialidad o el centro de atención.
- **Justificación:** Prevenir el acaparamiento de turnos y garantizar la viabilidad física de la asistencia del paciente.
- **Implementación:** Validación atómica en `TurnoService` antes de la persistencia (`existsByPacienteAndFecha...`).

RN-02: Disponibilidad global del profesional (cross-tenant)

- **Descripción:** Dado que la entidad `Medico` es global en la plataforma SaaS, el sistema valida la disponibilidad del profesional en **todos** los Centros de Atención simultáneamente. Un médico no puede ser reservado en el "Centro A" si ya tiene un turno ocupado en el "Centro B" en la misma franja horaria.
- **Justificación:** Evitar la ubicuidad imposible del recurso humano en un entorno multi-sede.

RN-03: Jerarquía de excepciones de agenda

- **Descripción:** Al calcular la disponibilidad, las reglas de excepción tienen precedencia absoluta sobre la disponibilidad base contractual.
    1. **Feriado/bloqueo global:** Cancela cualquier disponibilidad en el día.
    2. **Mantenimiento de consultorio:** Inhabilita turnos en el consultorio específico afectado.
    3. **Disponibilidad base:** Se aplica solo si no existen las anteriores.
- **Justificación:** Permitir la gestión flexible de incidencias sin modificar los contratos horarios base de los médicos.

RN-04: Validación temporal de asistencia

- **Descripción:** El cambio de estado de un turno a `COMPLETADO` (marcar asistencia) o `AUSENTE` solo está permitido si la fecha del sistema (`LocalDate.now()`) coincide exactamente con la fecha programada del turno.
- **Justificación:** Garantizar la veracidad de los registros médicos y evitar manipulaciones históricas o futuras de la asistencia para fines de facturación o auditoría.

RN-05: Caducidad automática de reservas

- **Descripción:** Todo turno en estado `PROGRAMADO` que no haya transicionado a `CONFIRMADO` dentro del tiempo límite configurado por el *tenant* (por defecto 48 horas antes de la cita), debe ser cancelado automáticamente por el sistema.
- **Justificación:** Maximizar la eficiencia de la agenda liberando *slots* no ocupados para que puedan ser aprovechados por la lista de espera u otros pacientes.

RN-06: Aislamiento de datos multi-tenant

- **Descripción:** Ningún usuario con rol `ROLE_ADMIN` u `ROLE_OPERADOR` puede visualizar, asignar o modificar recursos (médicos, pacientes, turnos) que no estén vinculados explícitamente a su `centro_atencion_id`.
- **Justificación:** Cumplimiento estricto de la privacidad y competencia comercial entre las distintas clínicas que utilizan la plataforma SaaS.

RN-07: Restricción de auto-asignación

- **Descripción:** Un usuario con rol `ROLE_MEDICO` no puede crear turnos para sí mismo (donde `turno.medico.id` coincida con `usuario.persona.id`).
- **Justificación:** Prevención de fraudes operativos o bloqueos de agenda no autorizados por parte del staff médico.

4. Calidad del software

4.1. Estrategia de testing

Para garantizar la fiabilidad de un sistema crítico como *CheTurno*, se adoptó una estrategia de **Desarrollo Guiado por Comportamiento (BDD)**. Esta metodología permite validar no solo la corrección del código, sino el cumplimiento de las reglas de negocio descritas en lenguaje natural, sirviendo como documentación viva del sistema.

La arquitectura de pruebas se estructura de la siguiente manera:

1. **Especificación (gherkin):** Los requisitos se redactaron en archivos `.feature` utilizando sintaxis gherkin (given/when/then), cubriendo escenarios positivos y negativos.
2. **Automatización (cucumber js):** Se implementó un framework de automatización que traduce los pasos de gherkin a ejecuciones reales contra la API y la lógica de negocio.

3. **Tipos de pruebas ejecutadas:**
    - **Pruebas generales:** Pruebas de sanidad (`MT_00_SmokeTests`) que verifican la disponibilidad básica de los servicios críticos (salud del sistema, login básico) antes de ejecutar suites más complejas.
    - **Pruebas de aislamiento (SaaS):** Scripts específicos (`MT_05_Aislamiento_Datos`) diseñados para intentar violar la seguridad multi-tenant, garantizando que un usuario del "Centro A" jamás acceda a datos del "Centro B".
    - **Pruebas de flujo de negocio:** Validación *end-to-end* de los procesos complejos como la gestión de agendas y listas de espera.

4.2. Qué se probó y qué quedó pendiente

Cobertura alcanzada (lo probado)

La suite de pruebas actual valida exhaustivamente los flujos críticos y las excepciones de seguridad principales de los siguientes módulos:

- **Seguridad y acceso:**
    - Autenticación vía credenciales y manejo de Tokens JWT.
    - Validación de roles (RBAC) para impedir acceso no autorizado a endpoints administrativos.
- **Arquitectura multi-tenant:**
    - Verificación de que los médicos que forman parte del staff de un centro de atención son invisibles para otros centros.
- **Core de turnos y agenda:**
    - Creación, visualización y cancelación de turnos.
    - Generación automática de esquemas de turnos basados en la disponibilidad médica.
- **Lista de espera inteligente:**

    - Flujo completo de inscripción de pacientes en demanda insatisfecha.
    - Validación de duplicidad (evitar doble inscripción en la misma lista).
- **Gestión administrativa:**
    - Alta, baja y modificación (CRUD) de centros de atención, especialidades y médicos.

Deuda de testing (lo pendiente)

Debido a la complejidad temporal del dominio y restricciones de tiempo, los siguientes aspectos quedaron con cobertura parcial o manual:

- **Lógica de días excepcionales:** No se automatizaron escenarios complejos de superposición entre feriados, bloqueos por mantenimiento y agendas regulares. La precedencia de estas reglas se validó manualmente.
- **Auditoría profunda:** Si bien se prueba que los eventos generan registros en la tabla `audit_log`, falta validar automáticamente el contenido del JSON (`old_value` vs `new_value`) para asegurar que se están capturando *todos* los campos modificados en operaciones de edición compleja.
- **Pruebas de concurrencia (race conditions):** No se implementaron pruebas de carga automatizadas para simular dos pacientes intentando reservar el mismo milisegundo, confiando por ahora en la integridad transaccional de la base de datos.
- **Notificaciones:** La recepción real de correos electrónicos se valida mediante logs del servidor SMTP simulado, pero no mediante una integración E2E con bandejas de entrada reales.

4.3. Manejo de errores y seguridad

Arquitectura de respuestas y excepciones

El sistema implementa una estrategia de manejo de errores centralizada y uniforme a través de la clase utilitaria `Response`. Esta clase actúa como un *wrapper* sobre `ResponseEntity` de Spring, estandarizando la estructura JSON devuelta al cliente tanto para operaciones exitosas como para fallos.

El formato de respuesta sigue el patrón:

```
{
  "status_code": 200,
  "status_text": "OK",
  "data": { ... }
}
```

Esta abstracción permite:

1. **Ocultamiento de stacktrace:** En caso de errores de servidor (`serverError`), el sistema nunca expone la pila de llamadas interna ni detalles de la infraestructura, mitigando riesgos de seguridad por fuga de información.
2. **Mapeo semántico:** Se traducen excepciones de negocio complejas a códigos HTTP estándar (`409 CONFLICT` para errores de base de datos, `403 FORBIDDEN` para violaciones de acceso, `404 NOT FOUND` para recursos inexistentes), facilitando la interpretación por parte del cliente Angular.

Seguridad en capas:

La seguridad no es un módulo aislado, sino un aspecto transversal implementado en múltiples niveles:

- **Nivel de aplicación (spring security):** Se utiliza `JwtAuthenticationFilter` para validar tokens en cada *request*.
- **Nivel de Datos (JPA):** Se implementan cláusulas `WHERE` automáticas en las consultas para garantizar el aislamiento multi-tenant, impidiendo que

un usuario acceda a registros de otro centro de atención mediante manipulación de IDs en la URL.

4.4. Auditoría

Para cumplir con los requisitos de responsabilidad y seguridad operativa, el sistema incorpora un módulo de auditoría robusto basado en el patrón *Interceptor*.

La clase `AuditInterceptor` intercepta automáticamente las operaciones críticas (creación, modificación, eliminación) y delega en `AuditLogService` la persistencia de un registro inmutable en la tabla `audit_log`.

Estructura del registro de auditoría.

Cada evento de auditoría captura la siguiente información contextual, garantizando la reconstrucción forense de las acciones:

- **Timestamp:** Fecha y hora exacta del evento.
- **Actor:** ID del usuario responsable de la acción (obtenido del contexto de seguridad).
- **Acción:** Tipo de operación realizada (ej. `TURNO_CREATE`, `TURNO_CANCEL`, `PACIENTE_UPDATE`).
- **Entidad Afectada:** ID y tipo del recurso modificado.
- **Centro de atención:** Contexto del tenant donde ocurrió el evento.
- **Detalles (snapshot):** En operaciones de modificación, se almacena una representación JSON del estado de la entidad, permitiendo comparar el "antes" y el "después".

5. Roadmap futuro (continuidad del producto)

El desarrollo de *CheTurno* no concluye con esta entrega. Se ha definido un plan de evolución estratégica que busca incrementar el valor del producto mediante la incorporación de inteligencia artificial, nuevos canales de comunicación y herramientas avanzadas de gestión de calidad.

5.1. Pendientes del backlog ordenados

A continuación, se presentan las funcionalidades pendientes de desarrollo, priorizadas cronológicamente según su impacto en la experiencia del usuario y la complejidad técnica.

Q1: Expansión de canales y soporte (corto plazo)

1. **Notificaciones vía WhatsApp y SMS (integración Twilio/Meta API)**
    - **Justificación:** El correo electrónico, aunque formal, no tiene la inmediatez necesaria para recordatorios de último minuto. La integración con WhatsApp Business API aumentará drásticamente la tasa de asistencia y la confirmación de turnos.
    - **Implicancia técnica:** Implementación de un nuevo servicio en el backend (`NotificationProvider`) que abstraiga el envío de mensajes, permitiendo configurar proveedores externos como Twilio.
2. **Formulario de reporte de bugs integrado**
    - **Justificación:** Facilitar el feedback técnico directo desde la aplicación es crucial para la estabilidad en producción.
    - **Implicancia técnica:** Desarrollo de un componente flotante global en angular que capture automáticamente el contexto del error (URL, usuario, navegador) y envíe un ticket al sistema de gestión de incidencias (equipo de desarrollo).
3. **Encuestas de calidad: anonimato y consentimiento**
    - **Justificación:** Para obtener feedback honesto y cumplir con regulaciones de privacidad, es vital permitir que los pacientes respondan encuestas de forma anónima tras dar su consentimiento explícito.
    - **Implicancia técnica:** Refactorización del modelo `EncuestaRespuesta` para permitir valores nulos en la relación con `Paciente` y adición de *check-boxes* legales en el flujo de respuesta.

Q2: Inteligencia y automatización (mediano plazo)

4. **Chatbot/Asistente virtual con IA**
    - **Justificación:** Un asistente conversacional disponible 24/7 reducirá la carga operativa de los recepcionistas, resolviendo consultas frecuentes ("¿atienden hoy?", "¿cómo llego?") y asistiendo en la reserva de turnos mediante lenguaje natural.
    - **Implicancia técnica:** Integración de un motor de NLP (como Dialogflow o la API de OpenAI) conectado a los servicios de `AgendaService` y `DisponibilidadService` mediante webhooks seguros.
5. **Dashboard de calidad para directivos**
    - **Justificación:** Los directores médicos necesitan ir más allá de las métricas operativas y visualizar la calidad percibida (NPS, CSAT). La capacidad de hacer *drill-down* (ir de lo general a lo particular) permitirá identificar qué servicio específico está fallando.
    - **Implicancia técnica:** Implementación de consultas analíticas complejas (OLAP) sobre las respuestas de encuestas y visualización avanzada con librerías de gráficos interactivos en el frontend.
6. **Sistema de alertas tempranas de calidad**
    - **Justificación:** La reacción rápida ante una caída en la calidad del servicio es crítica. El sistema debe alertar proactivamente a los responsables cuando los KPIs (ej. Satisfacción < 3/5) caigan por debajo de un umbral crítico.
    - **Implicancia técnica:** Tarea programada (*cron job*) nocturna que analice las métricas del día y dispare correos de alerta a los usuarios con rol `ROLE_ADMIN` si se detectan anomalías.

Q3: Integración clínica y escalabilidad (largo plazo)

7. **Middleware de interoperabilidad (HL7/FHIR)**

- **Justificación**: Para que *CheTurno* se integre en ecosistemas hospitalarios grandes, debe ser capaz de comunicarse con sistemas de Historia Clínica Electrónica (HCE) existentes.
- **Implicancia técnica**: Desarrollo de un microservicio "middleware" que traduzca los eventos de turnos (llegada del paciente) a estándares internacionales de salud (HL7 FHIR) para notificar al HCE del médico.

8. **Vista de mensual de alta densidad**
    - **Justificación:** Para centros de alto volumen, los coordinadores necesitan una visualización compacta de la ocupación mensual (ej. 700 turnos en una pantalla) para detectar patrones de saturación a simple vista.
    - **Implicancia técnica:** Optimización extrema del renderizado en el frontend (virtual scrolling) y endpoints de backend optimizados para devolver solo metadatos ligeros de miles de turnos en una sola consulta.