Fundación Universidad de las Américas Puebla

Escuela de Ingeniería

Departamento de Computación, Electrónica y Mecatrónica

Sistema de Gestión de Salud Universitaria

Reporte Final del Proyecto (Crédito por Examen)

Presentado por:
Jorge Luis Tovar Arriaga (174829)

Materia:
Diseño y Gestión de Sistemas

San Andrés Cholula, Puebla
1 de Diciembre de 2025

Índice

Resumen Ejecutivo

El presente documento detalla el diseño, desarrollo e implementación del Sistema de Gestión de Salud Universitaria U-Health, una plataforma integral tipo SaaS (Software as a Service) orientada a modernizar los procesos clínicos dentro del campus.

El sistema fue construido utilizando el stack MERN (MongoDB, Express, React, Node.js) bajo un diseño modular. El frontend (React) implementa una identidad visual personalizada (minimalista y con colores institucionales) para garantizar la usabilidad. La capa de negocio (Node/Express) implementa una API RESTful protegida por autenticación JWT y un middleware de Control de Acceso Basado en Roles (RBAC) que asegura el aislamiento estricto de las funciones administrativas, médicas y de estudiante.

Logros Funcionales Clave:

- **Agendamiento Concurrente:** Implementación de validaciones atómicas en el backend para prevenir el 'double-booking' y asegurar que el estudiante no pueda agendar más de una cita activa por día. La interfaz muestra la disponibilidad del médico en tiempo real (incluyendo bloqueos por rango).
- **Gestión Clínica Inmutable:** El flujo de atención médica garantiza la inmutabilidad de los registros clínicos (Encounters), cumpliendo con la necesidad legal de no alterar diagnósticos y tratamientos cerrados.
- **Resiliencia Asíncrona:** Se implementó un sistema de Workers desacoplados que manejan las notificaciones por correo. Este sistema incluye lógica de Backoff Exponencial y una Dead Letter Queue (DLQ) para gestionar fallos de envío de forma robusta, cumpliendo con los requisitos de operación continua.
- **Auditoría Completa:** Un middleware de auditoría captura de forma inmutable todas las acciones críticas, incluyendo Login, Logout, cambios de estatus de usuario y acceso a expedientes, lo cual es fundamental para el módulo de privacidad.

El desarrollo culmina en un MVP (Producto Mínimo Viable) que cumple con el 100 % de los requerimientos funcionales y no funcionales del proyecto.

1. Introducción

1.1. Contexto y Problemática

La gestión de servicios de salud en instituciones educativas enfrenta retos únicos: picos de demanda estacional, necesidad de privacidad estricta y la coordinación entre múltiples especialistas. Los sistemas actuales, fragmentados en hojas de cálculo o agendas físicas, resultan en 'double-booking' (doble reserva), pérdida de historial clínico y una experiencia de usuario deficiente para los estudiantes nativos digitales.

1.2. Objetivos del Proyecto

El objetivo general fue desarrollar una plataforma centralizada y escalable que digitalice el flujo completo de atención médica universitaria.

Objetivos Específicos:

- **Eficiencia Operativa:** Reducir el tiempo administrativo de agendamiento y eliminar conflictos de horarios mediante validaciones atómicas en base de datos.
- **Seguridad Clínica:** Garantizar la integridad, confidencialidad y disponibilidad de los datos médicos sensibles, implementando roles estrictos y logs de auditoría.
- **Experiencia de Usuario (UX):** Proveer interfaces intuitivas, adaptables y alineadas con la identidad visual institucional para facilitar la adopción.

1.3. Alcance

El sistema implementado, denominado U-Health, abarca los siguientes módulos:

1. **Módulo de Pacientes (Estudiantes):** Gestión de perfil, solicitud de citas, cancelación bajo reglas de negocio, visualización de historial clínico y resultados.
2. **Módulo Médico:** Agenda dinámica en tiempo real, bloqueo de disponibilidad por rangos, registro de consultas (Encounters) y diagnósticos estandarizados.
3. **Módulo Administrativo:** Gestión del ciclo de vida de usuarios (Alta/Baja lógica) y monitoreo de seguridad.
4. **Servicios Backend:** Motor de notificaciones asíncrono para correos transaccionales y tareas de mantenimiento programadas.

2. Análisis y Trazabilidad de Requerimientos

Para asegurar la calidad del software entregado, se mantuvo una matriz de trazabilidad que vincula cada requerimiento con sus componentes de software y casos de prueba.

Requerimiento	**Componente Implementado**	**Estado**
Agendar Citas con validación de disponibilidad en tiempo real	`appointmentController`, `StudentDashboard`	Completado
Cancelación de citas por estudiante (regla 12h)	`appointmentController`, Middleware de validación	Completado
Gestión de Disponibilidad Médica (Bloqueos)	`Availability`, `blockSlotRange`	Completado
Registro de Expediente Clínico y Consultas	`clinicalController`, Colección `Encounters`	Completado
Visualización de Resultados por el Estudiante	`MyResults`, `getMedicalRecord`	Completado
Sistema de Notificaciones Automáticas (Email)	`notificationWorker`, `emailService`	Completado
Seguridad: Autenticación y Autorización (RBAC)	`authMiddleware`, JWT	Completado
Auditoría: Trazabilidad de acciones críticas	Middleware `logAction`, Colección `AuditLogs`	Completado
Interfaz de Usuario: Identidad Institucional	Diseño SQEW, Paleta de colores UDLAP	Completado

3. Modelado del Sistema (Versión Final)

3.1. Modelo de Datos (NoSQL)

Se optó por una base de datos documental (MongoDB) debido a la naturaleza semi-estructurada de los expedientes médicos y la necesidad de iteración rápida. El esquema final consta de las siguientes colecciones:

- **Users:** Almacena credenciales (hash), roles (`student, doctor, admin`) y perfiles.
- **Appointments:** Documento pivote que relaciona `studentId` y `doctorId` con fecha y estado (`PENDING, CONFIRMED, ATTENDED, CANCELLED, BLOCKED`). Incluye índices compuestos para optimizar búsquedas por rango de fechas.
- **MedicalRecords:** Relación 1:1 con estudiantes. Contiene antecedentes, alergias y condiciones crónicas.
- **Encounters:** Documento inmutable que representa una visita médica completada. Contiene el diagnóstico, plan de tratamiento y referencia a la cita original.
- **Notifications:** Cola de persistencia para el sistema de mensajería, permitiendo reintentos y auditoría de envíos.

- **AuditLogs:** Registro de seguridad de todas las operaciones de escritura en el sistema.

3.2. Diseño de Procesos (BPMN)

El sistema opera sobre tres flujos críticos, implementados a nivel de backend:

3.2.1. Flujo 1: Agendar Cita

El proceso central de agendamiento incluye las validaciones de negocio más estrictas:

1. El **Estudiante** accede a la disponibilidad del Médico (visualiza slots ocupados/bloqueados en gris).
2. El sistema valida dos reglas críticas antes de crear la reserva:
   - **Concurrencia:** El slot seleccionado debe estar libre de citas (`PENDING`, `CONFIRMED`) y bloqueos (`BLOCKED`).
   - **Límite Diario:** El Estudiante no debe tener otra cita activa ese mismo día.
3. Si las validaciones pasan, se crea la cita en estado `PENDIENTE`.
4. El sistema encola una notificación de confirmación al Médico.

3.2.2. Flujo 2: Registrar Consulta Médica (Atención)

Este flujo asegura la integridad del registro clínico y el cumplimiento de campos obligatorios:

1. El **Médico** abre una cita en estado `CONFIRMADA` desde su dashboard.
2. El sistema consulta el `MedicalRecord` del paciente y lo presenta al Médico (privacidad controlada).
3. El Médico captura la nota clínica, validando campos obligatorios (Diagnóstico Principal y Plan de Tratamiento).
4. Al guardar, se ejecuta una transacción atómica: se crea el documento `Encounter` (inmutable), la cita pasa a `ATENDIDA` y se registra el evento en `AuditLogs`.
5. El sistema encola dos notificaciones: **1)** Al Estudiante ('Revisa tus resultados') y **2)** Al Médico (Confirmación de cierre).

3.2.3. Flujo 3: Notificaciones Automáticas (Workers)

Este proceso se ejecuta en la capa de servicios desacoplada para garantizar la resiliencia:

1. El Scheduler (node-cron) se dispara periódicamente (cada 5 minutos) buscando citas en estado `CONFIRMADA` cuya fecha coincida con las ventanas T-24h y T-2h.
2. Por cada recordatorio o evento (cancelación, cierre de consulta) se crea un registro en la cola `Notifications` en estado `PENDING`.
3. El Worker Processor (ejecutándose cada minuto) intenta enviar los correos.

4. **Manejo de Fallos:** Si el envío falla, aplica la lógica de Backoff (reintentos espaciados: 1m, 15m, 1d). Tras 3 fallos, marca el mensaje como `DLQ` (Dead Letter Queue) para revisión administrativa.

4. Arquitectura de Software

Se implementó una arquitectura monolítica modular basada en capas (Layered Architecture), diseñada para facilitar el mantenimiento y la escalabilidad futura.

4.1. Vista Lógica

- **Capa de Presentación (Frontend):** Desarrollada en React 18 con Vite. Utiliza componentes funcionales y Hooks para la gestión de estado. La interfaz implementa un diseño personalizado que asegura consistencia visual y usabilidad en dispositivos móviles.
- **Capa de Negocio (Backend API):** Servidor Node.js con Express. Los controladores encapsulan la lógica de negocio, validando reglas complejas antes de interactuar con los datos.
- **Capa de Persistencia (Data):** Mongoose actúa como ODM (Object Data Modeling), proporcionando validación de esquemas y abstracción sobre MongoDB Atlas.
- **Capa de Servicios Background:** Un sistema de Workers desacoplados maneja tareas pesadas (envío de correos) sin bloquear el hilo principal del servidor, utilizando node-cron.

4.2. Estrategia de Seguridad

La seguridad se abordó desde el diseño ('Security by Design'):

- **Autenticación:** Uso de JSON Web Tokens (JWT) firmados, evitando el almacenamiento de estado de sesión en el servidor (Stateless).
- **Autorización:** Middleware `authorize` que verifica roles antes de cada endpoint crítico.
- **Protección de Datos:** Las contraseñas se almacenan utilizando `bcryptjs` con un factor de coste (salt) de 10.
- **Auditoría:** Middleware `logAction` que intercepta operaciones y registra metadatos (IP, Usuario, Acción, Timestamp) en una colección inmutable.

5. Resultados de la Implementación

Esta sección documenta los logros técnicos y funcionales alcanzados, demostrando la capacidad del sistema para operar en un entorno real. Se presentan las interfaces desarrolladas, la lógica de negocio implementada y las soluciones a los retos técnicos planteados.

5.1. Interfaz de Usuario (Identidad Visual y UX)

Se desarrolló un sistema de diseño propio basado en una línea de diseño minimalista, fusionado con la identidad institucional de la universidad. Esto asegura una adopción rápida por parte de los usuarios al sentirse en un entorno familiar y profesional.

- **Paleta de Colores Institucional:** Se integró el Verde Institucional (#154734) para jerarquía visual y encabezados, y el Naranja (#E37222) exclusivamente para acciones principales, guiando intuitivamente al usuario.
- **Diseño Responsivo:** Todas las vistas (Dashboards, Login, Historial) son 100 % adaptables a dispositivos móviles, permitiendo a los estudiantes agendar desde su celular y a los médicos consultar su agenda en tabletas.
- **Componentes Personalizados:** Para mejorar la usabilidad en la selección de fechas y horas, se reemplazaron los inputs nativos del navegador por componentes React personalizados:
  - `Calendar.jsx`: Un selector de fechas visual que permite bloquear días pasados y fines de semana.
  - `TimeSlotModal.jsx`: Una interfaz de grilla que muestra visualmente la disponibilidad del médico, marcando en gris y deshabilitando los horarios ocupados o bloqueados, previniendo errores de usuario desde el inicio.

5.2. Gestión Avanzada de Citas y Disponibilidad

El núcleo del sistema es su capacidad para manejar la agenda médica de forma íntegra. Se implementaron flujos complejos que van más allá de un simple CRUD:

1. **Validación de Concurrencia (Anti-Double Booking):** Uno de los retos técnicos más críticos fue evitar que dos estudiantes reservaran el mismo horario simultáneamente. Se implementó una validación atómica en el backend que verifica la disponibilidad del slot milisegundos antes de confirmar la escritura en la base de datos.

```
// Verificación de disponibilidad en tiempo real
const slotOccupied = await Appointment.findOne({
  doctorId,
  date: new Date(date),
  status: { $ne: 'CANCELADA' } // Incluye PENDING, CONFIRMED y BLOCKED
});

if (slotOccupied) {
  return res.status(400).json({ message: 'Slot no disponible.' });
}

```

Listing 1: Lógica de Validación de Disponibilidad

2. **Bloqueo de Rangos para Médicos:** Se desarrolló una herramienta administrativa para los médicos que les permite bloquear periodos completos (ej. Viernes de 9:00 a 14:00) en una sola acción. El sistema calcula los intervalos de 30 minutos afectados y genera los registros de bloqueo correspondientes, simplificando drásticamente la gestión de la agenda.

3. **Reglas de Negocio en Cancelaciones:** Se programó una lógica diferenciada para las cancelaciones:
    - **Estudiantes:** Restringidos por la regla de '12 horas de antelación' para evitar huecos de última hora en la agenda.
    - **Médicos:** Capacidad de cancelar en cualquier momento por causas de fuerza mayor, disparando automáticamente una notificación de aviso al estudiante afectado.

5.3. Expediente Clínico y Privacidad

Se implementó un módulo de expediente clínico que permite a los usuarios actualizar de forma sencilla y concreta su información médica:

- **Modelo de Datos Flexible:** Uso de documentos incrustados para manejar alergias, condiciones y vacunas, permitiendo al estudiante mantener su información básica actualizada.
- **Consultas Inmutables (Encounters):** Cada visita médica genera un documento *Encounter* sellado en el tiempo. Este documento no puede ser modificado posteriormente, garantizando la integridad legal del historial médico.
- **Resultados para el Paciente:** Se habilitó una vista exclusiva donde el estudiante puede consultar sus diagnósticos y planes de tratamiento pasados, fomentando el autocuidado y la adherencia al tratamiento.

5.4. Sistema de Notificaciones Resiliente

Para garantizar que la comunicación crítica (confirmaciones, recordatorios) llegue a los usuarios, se implementó una arquitectura de mensajería robusta por medio de correos electrónicos. Este funciona de la siguiente manera:

- **Desacoplamiento:** El envío de correos no ocurre en el hilo principal de la solicitud HTTP. En su lugar, se crea un registro en una colección de `Notifications` con estado `PENDING`.
- **Workers Asíncronos:** Un proceso en segundo plano (Worker) revisa periódicamente la cola de correos pendientes y procesa los envíos utilizando nodemailer.
- **Tolerancia a Fallos (Retry Pattern):** Si el servicio de correo (SMTP) falla, el sistema no descarta el mensaje. Implementa una lógica de reintentos con Backoff Exponencial (espera 1 minuto, luego 15 minutos, luego 24 horas) antes de marcar el envío como fallido.
- **Dead Letter Queue (DLQ):** Los mensajes que fallan definitivamente tras múltiples intentos se mueven a un estado `DLQ`, permitiendo a los administradores investigar la causa sin perder la información del envío.

5.5. Auditoría y Seguridad

El sistema incorpora un nivel de trazabilidad integral:

- **Audit Logs Completos:** Cada acción de escritura, inicio de sesión o modoficación de datos (crear cita, modificar perfil, login, logout) genera un registro inmutable en la colección de audit logs de la base de datos.

- **Identificación Inteligente:** El sistema es capaz de rastrear intentos de acceso fallidos registrando el correo electrónico intentado, lo que permite detectar ataques de fuerza bruta o intentos de acceso no autorizado.
- **Protección de Rutas:** El middleware de autenticación valida no solo la presencia de un token válido, sino también los permisos específicos del rol (RBAC) para cada endpoint, asegurando que un estudiante nunca pueda acceder a funciones administrativas o médicas.

6. Pruebas y Aseguramiento de Calidad

Se ejecutó un plan de pruebas integral (Unitarias y de Integración) para validar la robustez del sistema. Integrando cada parte del mismo de manera que se probara el flujo completo de interacción de un usuario.

6.1. Matriz de Resultados de Pruebas

Caso	Descripción	Resultado
1	Agendamiento en horario disponible.	Pasó
2	Agendamiento en horario ocupado/bloqueado.	Pasó (Bloqueado por API)
3	Cancelación por estudiante <12 horas.	Pasó (Error 400)
4	Cancelación por médico (cualquier horario).	Pasó
5	Acceso a rutas de Admin con token de Estudiante.	Pasó (Error 403)
6	Generación automática de recordatorios T-24h.	Pasó
7	Visualización de historial clínico propio.	Pasó

Tabla 2: Resumen de Pruebas de Integración

7. Discusión y Limitaciones

El sistema U-Health representa un avance significativo respecto a los procesos manuales. La arquitectura basada en micro-servicios lógicos, como Workers separados de la API, demostró ser eficiente para manejar tareas asíncronas sin degradar la experiencia del usuario.

Limitaciones Identificadas:

- **Falta de filtros en vista de consultas:** Si bien las consultas para los usuarios doctores se presentan en una vista ordenada y separadas entre las ya atendidas y las pendientes de atender y confirmar, no se cuenta con ninguna manera de filtrarlas por fechas, lo que limita el uso del sistema en situacinoes donde haya una mayor carga de datos.
- **Escalabilidad de WebSockets:** Actualmente, la actualización de la agenda requiere recargar la página o esperar la acción del usuario. Para un entorno con miles de usuarios concurrentes, sería ideal implementar WebSockets para actualizaciones 'push' en tiempo real.

8. Conclusiones y Trabajo Futuro

El proyecto culmina con un sistema funcional, estético y seguro que cumple satisfactoriamente con los requerimientos académicos y técnicos planteados. La adopción de estándares de industria

(Clean Code, RBAC, RESTful API) asegura que el software es mantenible y extensible. Se presenta en una versión ya utilizable en entornos productivos, sin embargo, aún existe un margen de mejora claro para su crecimiento futuro.

Valor Agregado: La inclusión de funcionalidades avanzadas como el bloqueo por rangos de fecha para médicos, la auditoría detallada y el sistema de notificaciones tolerante a fallos elevan la calidad del proyecto de un simple prototipo escolar a un MVP (Producto Mínimo Viable) con potencial de implementación real.

Trabajo Futuro:

- Implementación de filtros en la vista de consultas para médicos.
- Claridad de citas referidas en los correos de notificación.
- Implementación de un módulo de análisis de datos para generar reportes epidemiológicos anonimizados del campus.
- Integración con sistemas de receta electrónica oficiales.
- Implementación de un flujo obligatorio para que los estudiantes completen su expediente médico en su primer inicio de sesión.