Clínica Odontológica “Sonrisa Feliz”

La clínica odontológica “Sonrisa Feliz” creció desde un pequeño consultorio de barrio hasta convertirse en un centro de atención con múltiples profesionales y un flujo constante de pacientes. Su identidad siempre fue la calidez humana: saludar por el nombre, recordar historias clínicas, recomendar turnos de seguimiento. Esa cultura fue suficiente durante años… hasta que los números en base al crecimiento aumentaron considerablemente, esto demandó nuestra atención.

La libreta de papel en recepción —la misma que alguna vez fue orgullo de orden— empezó a revelar su fragilidad: citas superpuestas, teléfonos mal anotados, confirmaciones que se perdían, reportes que nunca llegaban. Los odontólogos pedían agenda con urgencia; los pacientes reclamaban claridad; la administración necesitaba métricas para decidir. Y la libreta no daba más.

El director de la clínica nos convocó. La petición fue concreta: “Necesitamos un sistema que funcione y que nos acompañe a crecer. **No busquen el ideal imposible: queremos avances reales y visibles en cada etapa**”. Con esa premisa, y en mi doble rol de Product Owner (voz del negocio) y Arquitecto de Software (responsable del diseño técnico), definimos un camino iterativo, pedagógico y profesional.

Este proyecto no es solo una práctica de programación; es un viaje de diseño y entrega de valor. El equipo de estudiantes-desarrolladores vivirá la evolución de una arquitectura monolítica moderna en Spring Boot 3.5.2 y Java 17, transitando conscientemente por decisiones iniciales (como el **patrón DAO** acoplado) para luego refactorizar a MVC, incorporar ORM con Hibernate y Spring Data, diseñar DTOs, manejar errores de forma global y, finalmente, presentar y “lanzar” el producto en un entorno simulado de cliente. Cada sprint es una escena de esta historia. Cada decisión técnica tiene un porqué.

Ten presente que los sprint pueden ir cambiando, esto hace rica a la metodología ágil optada. Entonces es importante que puedas ir desarrollando sprint por sprint.

Visión y objetivos

Visión: Digitalizar la gestión de turnos de la clínica, mejorando la disponibilidad de información, la trazabilidad de operaciones y la experiencia de pacientes y profesionales.

Objetivos:

- Gestionar Pacientes y Odontólogos como entidades centrales.
- Gestionar Turnos vinculando paciente, odontólogo y fecha/hora.
- Evolucionar la arquitectura: de DAO acoplado a MVC y ORM con Hibernate/Spring Data.
- Exponer APIs REST y consumirlas desde vistas simples.
- Implementar logging consistente, validaciones y manejo global de excepciones.
- Entregar incrementos funcionales por sprint y culminar con una presentación/lanzamiento (defensa técnica).

Alcance funcional

- Pacientes: alta, consulta, listado, actualización y eliminación. Asociación a un Domicilio (clase independiente; en el sistema, domicilio no es protagonista).
- Odontólogos: alta, consulta, listado, actualización y eliminación.
- Turnos: alta, consulta, listado, actualización y eliminación; relación con Paciente y Odontólogo.
- Reportes básicos (a futuro): agenda semanal por odontólogo, pacientes nuevos por período.

4. Requerimientos no funcionales

- Lenguaje: Java 17.
- Framework: Spring Boot.
- Build: Maven.
- Arquitectura: Monolítica en capas; evolución progresiva DAO → MVC → ORM.
- Persistencia: H2 (inicial), luego JPA/Hibernate + Spring Data.
- Logging: Log4j.
- Pruebas: JUnit (mínimo por historia clave).
- UI: Vistas HTML con Bootstrap desde Sprint 3.

- Estilo: convenciones estándar Java; separación de responsabilidades clara desde Sprint 2.
- Observabilidad: logs de operaciones y errores; mensajes consistentes.

5. Backlog inicial (épicas y user stories)

Sprint 1 (EXAMEN): DAO, Gestión de Pacientes y Odontólogos

Objetivo del Sprint

Como Product Owner/Arquitecto, solicito la construcción de la base del sistema en una arquitectura monolítica con Spring Boot 3.5.6 (Java 17, Maven) aplicando patrón DAO sobre H2. Se debe gestionar Pacientes (con asociación a Domicilio) y Odontólogos. Este sprint es un examen práctico con entrega.

Alcance y reglas específicas del examen

- **Domicilio** es una clase con su propio DAO (DomicilioDAOH2). Se persiste como soporte, pero no es protagonista del sistema en este sprint.
- **Pacientes**: el docente implementa en vivo guardarPaciente y listarPacientes. El equipo de estudiantes completa el CRUD restante.
- **Odontólogos**: CRUD completo realizado por los estudiantes.

5. Backlog inicial (épicas y user stories)

- Épica Pacientes
    - US-P1: Como recepcionista, quiero registrar pacientes (con domicilio asociado) para poder agendar turnos.
    - US-P2: Como recepcionista, quiero consultar por ID y listar pacientes para verificar información.
    - US-P3: Como recepcionista, quiero actualizar/eliminar pacientes para mantener datos consistentes.
- Épica Odontólogos
    - US-O1: Como administrador, quiero registrar odontólogos con matrícula para habilitarlos en turnos.
    - US-O2: Como administrador, quiero consultar/actualizar/eliminar odontólogos.

- Épica Calidad y Presentación
    - US-Q1: Como sistema, quiero validar entradas para evitar datos inválidos.
    - US-Q2: Como operador, quiero ver mensajes claros ante errores.
    - US-Q3: Como dirección, quiero una demo final end-to-end.

6. Metodología, roles y Definition of Done (DoD)

- Marco: Scrum académico adaptado.
- Roles:
    - Docente (PO + Arquitecto): prioriza, define estándares, construye parte base en vivo, asegura coherencia técnica.
    - Estudiantes (equipo): completan historias, prueban, documentan, presentan y defienden.
- DoD por historia:
    - Código integrado y funcional.
    - Logs clave presentes (info/errores).
    - 1–2 tests cubriendo casos felices y algún borde.
    - Revisión por pares (mesa).
    - Demostración en la review del sprint.

7. Roadmap y deadlines

- Sprint 1 (EXAMEN; deadline urgente: mañana): DAO; Pacientes y Odontólogos; H2; Log4j; pruebas básicas. Parte docente + parte estudiantes; demo en clase.
- Sprint 2 (7 días): Log4j depurado; APIs REST; consumo desde una vista (al menos 1); Postman; MVC claro.
- Sprint 3 (7–10 días; Taller de Coding intermedio): ORM, Hibernate, Spring Data; Turnos; DTOs; vistas Bootstrap.
- Sprint 4 (7 días): Excepciones personalizadas; GlobalExceptionHandler; validaciones @Valid; mensajes en vistas/JSON.
- Sprint 5 (fecha de examen final): Presentación y lanzamiento; demo end-to-end; documentación mínima; defensa técnica.

8. Narrativa de decisiones técnicas

- **Sprint 1:** elegimos DAO crudo para experimentar acoplamiento consciente y justificar la migración a MVC.
- **Sprint 2**: MVC/REST separa capas y habilita consumo desde vistas; Postman como evidencia de API.
- **Sprint 3:** ORM/Repositories/DTOs para modelar relaciones reales y desacoplar transporte.
- Sprint 4: robustecemos UX y DX con validaciones y manejo global de errores.
- Sprint 5: defendemos y "lanzamos" el producto como un equipo profesional.

9. Cierre

Este proyecto enseña a pensar, diseñar, construir, probar y contar. La clínica es el contexto; la ingeniería, el camino; la presentación final, la validación del aprendizaje.

Sprint 1 – EXAMEN: DAO, Gestión de Pacientes y Odontólogos

Versión: 1.0
Rol redactor: Product Owner y Arquitecto de Software (Docente)
Deadline: Urgente, mañana. Parcial en clase al cierre del sprint.
Ámbito: Arquitectura monolítica con Spring Boot, Java 17, Maven, H2, Log4j, JUnit.

Durante el relevamiento, recepción, administración y dirección nos explicaron qué información mínima manejan a diario y cómo operan.

- **Paciente** según negocio
    - Identidad y contacto: nombre, apellido, cédula, email. El email permite confirmaciones y recordatorios.
    - Trazabilidad: fechaIngreso, para saber desde cuándo se atiende con nosotros.
    - Dirección: la clínica registra domicilio para notificaciones post-operatorias y cuestiones administrativas (calle, número, localidad, provincia).

- **Decisión del Arquitecto**
    - Paciente es entidad central y se asocia a Domicilio mediante un atributo de asociación: **private Domicilio domicilio.**
    - Domicilio es clase independiente con su propio DAO (DomicilioDaoH2). Se persiste como soporte, pero no es protagonista en este sprint (no tiene flujo de gestión autónomo).
    - **Odontólogo** es una entidad central con matrícula como dato clave de auditoría.
    - id, nombre, apellido, matrícula: identidad profesional y requisito para asignación de turnos.

Por qué hemos optado por un Patron DAO:

- **Lógica de Negocio vs. Lógica de Datos**: El sistema de la clínica tiene una lógica de negocio compleja (ej. agendar citas, calcular tarifas según el plan, gestionar el historial médico). El patrón DAO crea una capa que aísla esta lógica de los detalles técnicos de cómo se guardan o recuperan los datos (consultas SQL, conexiones JDBC/JPA, manejo de transacciones).
- **Beneficio para la Clínica:** Mantiene el código de las reglas clínicas limpio y claro. Si necesitas cambiar cómo calculas un precio, no tendrás que tocar el código de acceso a la base de datos, y viceversa. Esto reduce la probabilidad de errores en un sistema que maneja datos sensibles de salud.
- **Independencia de la Fuente de Datos:** En un futuro, la clínica podría necesitar migrar de una base de datos a otra (por ejemplo, de MySQL a PostgreSQL, o a una solución en la nube que cumpla con normativas de salud más estrictas).
- **Beneficio para la Clínica:** Con el patrón DAO, si cambias la base de datos, solo tendrás que modificar la implementación de las clases DAO. La capa de negocio de la aplicación (donde se agendan las citas y se guardan los diagnósticos) no se ve afectada, lo que minimiza el tiempo de inactividad y los costos de migración.
- **Centralización del Acceso:** Todas las operaciones de acceso a datos están centralizadas en la capa DAO (por ejemplo, `PacienteDAO`, `CitaDAO`).
- **Beneficio para la Clínica:** Si hay un problema de rendimiento o de conexión con la base de datos, o si necesitas optimizar una consulta de búsqueda de pacientes,

sabes exactamente dónde buscar y corregir: solo en la capa DAO. Esto hace que el sistema sea más robusto y fácil de actualizar con el tiempo.

- **Pruebas sin Base de Datos:** Para garantizar que la lógica de la clínica funcione correctamente (ej. el sistema solo permite que un dentista agende 8 citas al día), necesitas probar esa lógica.
- **Beneficio para la Clínica:** El patrón DAO permite a los desarrolladores simular (o *mockear*) el acceso a la base de datos. De esta manera, pueden probar la capa de negocio sin depender de una conexión activa a la base de datos, haciendo que las pruebas sean más rápidas y confiables. Esto es crucial para un sistema médico donde la exactitud y la fiabilidad son obligatorias.

La **base de datos H2** se usa en Spring Boot como una herramienta esencial para el aprendizaje, el desarrollo rápido y las pruebas, y nunca para almacenar los datos reales de los pacientes en producción.

Servicios:

- PacienteService y OdontologoService articulan la lógica funcional por entidad y servirán como puente natural cuando migremos a MVC en Sprint 2.
- En el examen: el docente implementa guardarPaciente y listarPacientes. El equipo de estudiantes completa el resto del CRUD de Paciente y construye todo Odontólogo (modelo, DAO, service).

Log4j:

- Desde ahora registramos operaciones CRUD y errores; es trazabilidad para la demo, para auditoría y para desarrollar criterio.

Pruebas:

- **JUnit con casos mínimos pero representativos (insertar y luego buscar; listar; actualizar; eliminar).**

2. Objetivo del Sprint (Sprint Goal)

Tener un proyecto inicial en Spring Boot (Java 17, Maven) que persista Pacientes (asociados a Domicilio) y Odontólogos en H2, usando DAOs, con trazabilidad por Log4j y

pruebas unitarias mínimas. Este sprint es un examen: parte guiada por el docente y parte autónoma por estudiantes.

3. Backlog del Sprint 1 (User Stories del PO)

1. Gestión de Pacientes – base (Docente)

- Como recepcionista, quiero registrar pacientes con su domicilio para iniciar su seguimiento.
- Como recepcionista, quiero listar pacientes para verificar altas recientes.

2. Gestión de Pacientes – completar CRUD (Estudiantes)

- Como recepcionista, quiero buscar paciente por ID, actualizar sus datos y eliminarlo si corresponde.

3. Gestión de Odontólogos – CRUD completo (Estudiantes)

- Como administrador, quiero registrar, consultar, actualizar y eliminar odontólogos para habilitarlos en turnos.

4. Alcance del examen y división de tareas

- Docente (PO/Arquitecto):
    - Estructura de paquetes del proyecto.
    - Clases: Paciente (con asociación a Domicilio) y Domicilio.
    - DomicilioDaoH2 (persistencia de soporte para Domicilio).
    - PacienteDAOH2 (implementa iDao con: guardar, buscarPorID, actualizar, eliminar, buscarTodos, buscarPorString) y PacienteService con:
        - guardarPaciente
        - listarPacientes
    - Clase BD con métodos getConnection (y utilidades de cierre).
    - Configuración de Log4j.
    - Demo en vivo: inserción y listado de pacientes con logs visibles.
- **Estudiantes (equipo en mesas – EXAMEN):**
    - **Paciente**: implementar en DAO/Service los métodos pendientes para:
        - buscarPacientePorId
        - actualizarPaciente
        - eliminarPaciente
        - buscarTodos (si no está expuesto desde Service, incorporarlo)

        - buscarPorString (exponerlo desde Service si se requiere en su demo)
    - **Odontólogo**:
        - Clase Odontologo: id, nombre, apellido, matricula.
        - OdontologoDAOH2: implementa iDao con CRUD completo.
        - OdontologoService: métodos CRUD equivalentes (guardar, buscar, actualizar, eliminar, listar).
    - **Pruebas** (JUnit):
        - Paciente: guardar y buscarPorID; listar; actualizar; eliminar; opcionalmente buscarPorString.
        - Odontólogo: guardar y listar; buscarPorID; eliminar; opcionalmente actualizar.

Notas:

- **Domicilio se persiste con su DAO pero no se gestiona como módulo. No perder foco: el protagonista es Paciente y Odontólogo.**
- **Mantener paquetes: model, dao, service, util (BD), config (si aplica), test.**

5. Formato de Entrega

- **Proyecto compilable y ejecutable.**
- **Demo en clase:**
    - **Guardar y listar pacientes (docente).**
    - **CRUD completo de odontólogos (estudiantes).**
    - **Mostrar logs de cada operación en consola.**
- **Pruebas unitarias ejecutadas en local; evidenciar resultados.**

6. Criterios de Aceptación (Definition of Done)

- **Proyecto corre y persiste en H2.**
- **Paciente:**
    - **guardarPaciente y listarPacientes (docente) funcionando y demostrados.**
    - **buscarPorID, actualizar, eliminar, buscarTodos y (si exponen) buscarPorString implementados por estudiantes y probados.**
- **Odontólogo:**

    - **CRUD completo implementado por estudiantes (DAO + Service) y probado.**
- **Clase BD presente con getConnection y utilidades.**
- **Log4j registra operaciones clave y errores manejados.**
- **Pruebas básicas pasando.**
- **Demo clara y ordenada en el deadline.**

7. Rúbrica de evaluación (parcial)

- **Funcionalidad (40%):**
    - **Paciente: métodos del docente operan; estudiantes completan CRUD restante.**
    - **Odontólogo: CRUD completo funcional.**
- **Calidad técnica (30%):**
    - **Organización por paquetes; nombres claros; convenciones Java.**
    - **Manejo básico de errores (sin crashs silenciosos).**
- **Pruebas (20%):**
    - **JUnit mínimos, reproducibles, con datos claros.**
- **Presentación (10%):**
    - **Demo breve, comprensible, con narrativa técnica mínima (qué hicieron, por qué, qué mejorarán en MVC).**

8. Deadline

Entrega 10/10/2025 - 12:00 en el siguiente enlace: enviar

9. Mensaje del Product Owner / Arquitecto

“Este sprint es la base. Quiero ver cómo transforman decisiones de negocio en clases concretas, y cómo esas clases se traducen en operaciones persistentes con trazabilidad. No busco fuegos artificiales: busco orden, criterio y dominio de lo esencial. Si en la demo pueden explicar ‘por qué’ eligieron un camino y qué plan tienen para el MVC del Sprint 2, van por buen camino.”

Este primer sprint es un cimiento. Ustedes completarán el CRUD de Paciente y desarrollarán Odontólogo completo. Es un examen con deadline urgente: mañana. La clínica (simulada) espera ver algo funcional, aunque simple, y nosotros queremos que aprendan a organizar un código que “respire” arquitectura. Mañana, al presentar, no solo muestren que funciona: cuenten por qué lo hicieron así y qué mejorarían en la migración a MVC del próximo sprint.