Reporte de Implementación

---

1. Decisiones Técnicas Implementadas.

Las tecnologías utilizadas en este proyecto fueron elegidas por ser frameworks que usan el modelo de MVC (Model-View-Controller), son herramientas modernas y ampliamente utilizadas en el desarrollo de aplicaciones empresariales. La elección principal de estas tecnologías se basó en su robustez, estandarización y utilidad para proyectos de arquitectura por capas.

- **Spring Boot**: Se utilizó como framework principal para la creación del backend, permitiendo desarrollar servicios web de forma estructurada, reduciendo la configuración manual gracias a su naturaleza auto configurada que permite que el desarrollador se enfoque más en la lógica de negocio de la aplicación..

- **Java**: Es el lenguaje base usado para el backend, debido a su compatibilidad con Spring y su uso en entornos empresariales y académicos basado en objetos que guardan una relación estrecha con el modelo MVC.

- **Maven:** Se utilizó como gestor de dependencias y compilación del proyecto, facilitando la organización de librerías.

- **MySQL**: Base de datos elegida, por su simplicidad y porque se integra fácilmente con Spring mediante JPA (Java Persistence API) y además porque viene integrada con XAMPP para el desarrollo de forma local.

- **Docker**: Para permitir empaquetar la aplicación y la base de datos en contenedores, facilitando la portabilidad, despliegue y ejecución del sistema sin necesidad de instalar manualmente todas las dependencias en cada equipo.

2. Problemas Encontrados y Soluciones Implementadas.

2.1 Problemas de Configuración de Entorno

Se encontraron problemas relacionados con la configuración del entorno, la base de datos y la integración entre componentes.

---

1. **Problema: Archivos corruptos que impedían iniciar el servidor MySQL**

En un momento del desarrollo, el servidor MySQL dejó de funcionar debido a la corrupción de archivos críticos. El servicio intentaba arrancar, pero se cerraba automáticamente.

Solución:
Se optó por reinstalar completamente **XAMPP**, lo cual permitió restaurar el servicio MySQL a su estado funcional.

---

2. **Problema: MySQL no iniciaba a pesar de estar libre el puerto 3306**

En otra ocasión, el servidor MySQL tampoco iniciaba. Tras verificar que el puerto no estaba siendo utilizado por otro proceso, el error persistía.

Solución:
Se identificó que ciertos archivos en la carpeta mysql/data (como mysql-relay-log.info, mysql-log, master.info, etc.) estaban impidiendo el inicio correcto. Se **eliminaron manualmente estos archivos**, lo cual permitió que MySQL funcionara nuevamente.

2.2 Problemas de Autenticación y Roles Implementando Spring Security

La implementación de la autenticación con Spring Security y la gestión de roles para usuarios y administradores presentó problemas, esto debido a que por defecto spring Security espera ciertos parámetros o métodos específicos.

---

1. **Problema: Spring Security no encontraba los roles especificados.**

Spring Security no encontraba los roles especificados en la base de datos debido a que por defecto spring espera que los roles estén configurados con el prefijo ROLE_**, si bien es posible completar esto en el backend, para seguir la estructura base de Spring se opto mejor por cambiar la configuración en la base de datos.

Solución:
Se configuraron los roles correctamente en la base de datos con el prefijo `ROLE_`, se usó `BCryptPasswordEncoder` para encriptar las contraseñas y se aplicaron anotaciones de seguridad como `@PreAuthorize` en los controladores correspondientes.

---

2. **Problema: Spring Security no validaba el parámetro de autenticación "correo".**

Spring usa por defecto el parámetro "username" (además de la contraseña) para identificar al usuario que se loguea, inicialmente usábamos "correo" como parámetro y por ello Spring rechazaba al usuario porque no encontraba el parámetro correspondiente.

Solución:
Se configuraron "username" como parámetro en lugar de "correo", si bien era posible hacer que Spring Security use "correo" como parámetro se optó por seguir la línea base de desarrollo en Spring.

---

3. **Problema: Spring Security rechazaba el formulario de registro.**

Spring usa por defecto un token en cada solicitud POST para temas de seguridad contra ataques de tipo **CSRF (Cross-Site Request Forgery)** el cual es un tipo de ataque malicioso en el que un atacante engaña a un usuario para que realice acciones no deseadas en una aplicación web en la que ya está autenticado.

Solución:

Primeramente se verificó que el formulario se esté enviando, para ello se usó el "inspeccionar" del navegador para visualizar todas las solicitudes realizadas en la página al llenar el formulario y enviarlo (se activó Preserve Log para lograr visualizarlo debido a que se volvia a redirigir a registro cuando se enviaba el formulario y esto no permitia ver si el formulario se había enviado o no), una vez se confirmó que el formulario no era el problema se vio la configuración de spring security y se deshabilitó este token con `csrf().disable()` para seguir desarrollando la aplicación con planes a futuro de implementación.

---

4. **Problema: Spring Security no validaba el parámetro de autenticación "correo".**

Spring usa por defecto el parámetro "username" (además de la contraseña) para identificar al usuario que se loguea, inicialmente usábamos "correo" como parámetro y por ello Spring rechazaba al usuario porque no encontraba el parámetro correspondiente.

Solución:
Se configuraron "username" como parámetro en lugar de "correo", si bien era posible hacer

que Spring Security use “correo” como parámetro se optó por seguir la línea base de desarrollo en Spring.

2.3 Problemas de Desarrollo.

Durante la codificación de las funcionalidades del sistema, se presentaron algunos problemas asociados a diversas causas como: Dependencias, Compatibilidad, Lógica de Negocio ambigua, etc.

---

1. **Problema: Compatibilidad de versiones entre Spring y Java.**

Al generar el proyecto con Spring Initializr la versión de Spring Boot (V 3.4.4) generaba conflicto con la versión de Java usada (Java 21 Corretto).

Solución:
Se optó por una versión anterior (V 3.4.2) evitando problemas de compatibilidad.

---

2. **Problema: Ciclo de dependencias en la configuración de @Bean**

Durante el desarrollo, se presentó un error relacionado con un ciclo de dependencias entre los componentes de la aplicación, específicamente entre `@Controller` controladores, servicios `@Service` y la clase de configuración `@Configuration` de Spring, el problema ocurría porque algunos beans se estaban inyectando mutuamente de forma indirecta, generando un ciclo que Spring no podía resolver automáticamente al arrancar la aplicación.

Solución:
Se revisaron las dependencias entre componentes y se reorganizó la arquitectura para eliminar referencias innecesarias.

- Se evitó inyectar directamente controladores en servicios.
- Se analizaron cuidadosamente las clases con `@Autowired` para romper el ciclo.
- En algunos casos, se optó por usar interfaces en lugar de clases concretas, facilitando una mejor separación de responsabilidades y evitando referencias cruzadas.

---

3. **Problema: La aplicación no pudo conectarse a la base de datos MySQL porque el usuario `'admin'` no tiene permisos para acceder a la base de datos.**

La aplicación no podía conectarse a la base de datos MySQL porque el usuario `'admin'` no tiene permisos para acceder a nuestra base de datos llamada `'sistemamedico'`.

Solución:
Verificamos las credenciales de acceso en `application.properties`, y se otorgo permisos al usuario `'admin'` con `GRANT ALL PRIVILEGES ON sistemamedico.* TO 'admin'@'localhost' IDENTIFIED BY 'password'; FLUSH PRIVILEGES;`

---

4. **Problema: Sesión cerrada de Hibernate cuando se interna acceder a una colección @ManyToMany**

La aplicación al momento de acceder a los roles de usuario especificada por una tabla generada `@ManyToMany` entre “usuario” y “roles”, ya no puede acceder a los roles debido que para ese momento la sesión de hibernate cerro, esto debido a que por defecto está configurada como “**LAZY”** la cual busca mejorar el rendimiento si la relación tiene muchos datos y no siempre necesitas cargarlos.

Solución:
Se cambió la configuración por defecto a `@ManyToMany(fetch = FetchType.EAGER)` asi Hibernate cargará la colección de inmediato junto con el usuario para su correcta autenticación.

---

5. **Problema: Sesión cerrada de Hibernate cuando se interna acceder a una colección @ManyToMany**

La aplicación al momento de acceder a los roles de usuario especificada por una tabla generada `@ManyToMany` entre “usuario” y “roles”, ya no puede acceder a los roles debido que para ese momento la sesión de hibernate cerro, esto debido a que por defecto está configurada como “**LAZY”** la cual busca mejorar el rendimiento si la relación tiene muchos datos y no siempre necesitas cargarlos.

Solución:
Se cambió la configuración por defecto a `@ManyToMany(fetch = FetchType.EAGER)` asi Hibernate cargará la colección de inmediato junto con el usuario para su correcta autenticación.

---

6. **Problema: Ciclo infinito al serializar respuesta JSON**

Cuando se intenta retornar un objeto `Usuario` desde un `@RestController` debido a que contiene una relación `@ManyToMany` con `Rol`, ocurre un ciclo infinito al serializar la respuesta JSON. Esto se debe a que Jackson (la librería que convierte objetos Java en JSON) sigue la relación bidireccional `Usuario → Rol → Usuario → Rol`... y así sucesivamente.

Solución:
Anotar las relaciones con `@JsonManagedReference` y `@JsonBackReference` en las entidades `@Entity` indicando a Jackson que solo debe serializar la relación en una dirección (`Usuario → Rol`) y omitir la inversa, previniendo la recursión infinita.

---

3. Tecnologías y Patrones Utilizados.

Categoría	Herramienta / Patrón	Descripción
Frontend	Thymeleaf, HTML, CSS, JS	Plantillas para renderizado dinámico del frontend, junto con tecnologías web estándar
Lenguaje de Programación	Java (21) Corretto	Versión LTS de Java, utilizada para el desarrollo principal del backend.
Build Tool	Maven 4.0.0	Herramienta de gestión de dependencias y automatización de builds.
Framework Principal	Spring Boot 3.4.2	Framework para construir aplicaciones Java modernas con configuración mínima.
ORM y persistencia	Spring Data JPA	Abstracción para el acceso a base de datos basada en repositorios.
Base de Datos	MySQL / MariaDB 10.4.32	Sistema de gestión de base de datos relacional.
Contenedores	Docker	Contenerización de la aplicación para facilitar despliegue y portabilidad.
Arquitectura	Modelo MVC (Model-View-Controller)	Patrón de arquitectura para separar la lógica de presentación, control y datos.

---

4. Posibles Mejoras y Trabajo a Futuro.

Es posible implementar diversos modelos de diseño para mejorar algún aspecto específico en nuestra aplicación.

4.1 Seguridad.

- **Implementación de JWT (JSON Web Token)**

Reemplazar la autenticación basada en sesión de Spring por una alternativa basada en JWT, esto hará que no se conserven sesiones optimizando el acceso al servidor, será escalable para APIs REST lo cual es mejor para aplicaciones móviles.

- **Autenticación 2FA (Two-Factor Authentication)**

Implementar autenticación en dos pasos 2FA para dar una capa extra de seguridad al sistema modificando el proceso de login usando algún servicio como Twilio para SMS o Google Authenticator para TOTP.

- **Implementar patrón de diseño Decorator**

Decorator es como "envolver" un objeto con otro para agregar funcionalidad extra sin modificar su estructura original. El uso puede darse para validar citas médicas con CitaController para validar:

1. Si el usuario está autenticado.
2. Si es doctor/paciente según la acción.
3. Si la cita le pertenece.

- **Habilitar CSRF (Cross-Site Request Forgery)**

Usar este token contra este tipo de ataque en los formularios HTML.**4.2 Escalabilidad.**

- **Implementar patrón de diseño Factory**

Patrón de diseño creacional que sirve para centralizar la creación de objetos en un solo lugar, así podemos tener flexibilidad para añadir nuevos tipos de usuario sin modificar el código existente muy útil par diferenciar entre tipo de pacientes (si se busca tener una diferenciación más clara de estos)

- **Implementar DTOs (Data Transfer Objects)**

Usar DTOs para controlar qué datos se exponen y evitar enviar información sensible o innecesaria que el cliente no necesita simplificando las entidades devueltas con solo los datos necesarios, en práctica reemplazar los controladores @ModelAttribute Usuario por UsuarioRegistroDTO.

.