mobydigital.

Evaluación N°1

Backend

Spring Web MVC

Fecha límite de entrega: *21/08/25*

Proyecto: Gestor de Turnos Médicos (API REST)

Una clínica necesita implementar un sistema web básico para gestionar turnos médicos utilizando el stack **Spring Boot + Spring MVC**. La aplicación será una API REST que permita gestionar pacientes, profesionales, turnos, y especialidades médicas.

Objetivos

Esta evaluación busca afianzar los conocimientos de:

- Arquitectura MVC en Spring Boot
- Controladores REST y endpoints HTTP
- Manejo de rutas y parámetros
- Inyección de dependencias con *@Autowired*
- Manejo de errores y validaciones
- Buenas prácticas de estructuración de proyectos

Consignas

1. Estructura del proyecto

Utilizar Spring Boot (mínimo versión 3.4.x)

Paquetes organizados en:

- `model` → clases entidad
- `controller` → controladores REST

- `service` → lógica de negocio
- `repository` → almacenamiento simulado en memoria (por ahora sin BD)
- `exception` → manejo de excepciones personalizadas

2. Modelado de Clases

Clase `Paciente`:

- `Long id`
- `String nombre`
- `String apellido`
- `String dni`
- `String email`

Clase `Profesional`:

- `Long id`
- `String nombreCompleto`
- `String especialidad`

Clase `Turno`:

- `Long id`
- `Paciente paciente`
- `Profesional profesional`

- `LocalDate fecha`

3. Controladores REST

Implementar endpoints RESTful para las siguientes operaciones:

📘 **`PacienteController`**

- `POST /pacientes` → Crear paciente
- `GET /pacientes/{id}` → Obtener paciente por ID
- `GET /pacientes` → Listar todos
- `DELETE /pacientes/{id}` → Eliminar paciente

🩺 **`ProfesionalController`**

- `POST /profesionales` → Crear profesional
- `GET /profesionales?especialidad=...` → Listar por especialidad

📅 **`TurnoController`**

- `POST /turnos` → Registrar turno (verifica que paciente y profesional existan)
- `GET /turnos` → Listar todos
- `GET /turnos/fecha/{fecha}` → Listar turnos por fecha (formato `yyyy-MM-dd`)
- `DELETE /turnos/{id}` → Eliminar turno

4. Lógica de Servicio

- Validar que no se creen turnos sin paciente o profesional válidos.
- Si el paciente o profesional no existen, lanzar una excepción personalizada.
- Evitar duplicados en la creación de turnos (mismo paciente, profesional y fecha).

5. Manejo de Excepciones

Crear excepciones personalizadas como:

```
class RecursoNoEncontradoException extends RuntimeException {}

class DatoInvalidoException extends RuntimeException {}
```

Y una clase global con `@ControllerAdvice` para manejar errores con respuestas HTTP coherentes.

6. Uso de Git + Git Flow

- Crear un repositorio con el nombre `segunda_evaluacion`
- Seguir el flujo de ramas (`feature`, `develop`, `main`)
- Commits atómicos y descriptivos

Ejecución final

Crear una clase `TestCargaInicial` (en el `controller` o en un `Runner`) que:

- Cree 2 pacientes
- Cree 2 profesionales (uno de cada especialidad: clínica y odontología)
- Registre al menos 3 turnos
- Pruebe endpoints con distintos filtros (fecha, especialidad, etc.)

Plus

1. **Validaciones con `@Valid` + Bean Validation (`@NotNull`, `@Email`)**
2. **Respuestas uniformes con DTOs personalizados**
3. **Filtrado de turnos por rango de fechas (`GET /turnos?desde=...&hasta=...`)**
4. **Implementar logs informativos y de errores usando Log4j**

Consideraciones

- No hace falta conexión a BD, se puede usar `Map<Long, Object>` como "repositorio", aunque si ya manejas bases de datos en memoria podes utilizarla.
- Podés usar `@PostConstruct` o un `CommandLineRunner` para precargar datos.
- El foco está en: arquitectura, buenas prácticas, Spring MVC.

Ante cualquier duda, contactate con el equipo de mentores, por el medio que consideres más cómodo..

¡Éxitos en la evaluación! 🚀