Examen/Práctica: API de Noticias (Public / Secured) + Cliente de verificación (tests)

Este repositorio/proyecto contiene **un cliente Maven (Java 17)** con **tests JUnit 5** que validan que tu **API REST de Noticias** está bien implementada.

Tu tarea como alumnado es **implementar el backend** con **dos arquitecturas de acceso**:

- ✅ **API pública**: `/api/public/**` (sin token)
- 🔒 **API securizada**: `/api/secured/**` (requiere JWT)

Los tests consumen ambas APIs y calculan una **nota automática**:

- Parte **pública** → máximo **4** puntos (CRUD completo)
- Parte **securizada** → máximo **6** puntos (JWT + CRUD)
- Total → **10**

> Importante: este proyecto NO es el backend. El backend lo implementas tú en un proyecto Spring Boot aparte (o en el que os entregue el profesor).
> Este cliente se usa para comprobar automáticamente que tu backend cumple el contrato.

---

1) Arquitectura a implementar en el backend

1.1 Capas recomendadas (clean / por responsabilidades)

Se recomienda separar en capas (puedes adaptar nombres):

- `controller` (REST): expone endpoints HTTP
- `service` (negocio): reglas y validaciones (no nulos, etc.)
- `repository` (persistencia): en memoria (List/Map) o BD si se pidiera
- `model` (dominio/DTO): entidades/POJOs que viajan como JSON
- `security` (JWT): filtro, configuración y utilidades

Ejemplo de paquetes:

```
com.docencia.pgv.noticias
├─ controller
│  ├─ PublicNoticiaController
│  └─ SecuredNoticiaController
├─ service
│  └─ NoticiaService
├─ repository
│  └─ NoticiaRepository (in-memory)
├─ model
│  └─ Noticia
└─ security
   ├─ SecurityConfig
   ├─ JwtAuthFilter
```

```
    ├─ JwtService
    └─ AuthController
```

1.2 Modelo de datos (JSON)

Los tests esperan (mínimo) este JSON:

```
{
  "id": 1,
  "titulo": "Texto",
  "contenido": "Texto"
}
```

- `id`: number (Long o Integer)
- `titulo`: string (obligatorio, no vacío)
- `contenido`: string (obligatorio, no vacío)

> Puedes añadir más campos (fecha, autor, etc.) pero **no rompas** los mínimos anteriores.

---

2) Contrato de endpoints (lo que los tests van a llamar)

> Base URL por defecto: `http://localhost:8080`
> (El cliente permite cambiarla con `-Dapi.baseUrl=...`)

2.1 API pública (sin JWT) — prefijo `/api/public`

CRUD sobre `noticias`:

- `GET /api/public/noticias` → **200** y devuelve `[]`
- `GET /api/public/noticias/{id}` → **200** si existe / **404** si no
- `POST /api/public/noticias` → **201** y devuelve la noticia creada con `id`
- `PUT /api/public/noticias/{id}` → **200** y devuelve noticia actualizada
- `DELETE /api/public/noticias/{id}` → **204** (sin body)

Reglas mínimas

- Si `titulo` o `contenido` vienen vacíos/nulos en POST/PUT, devuelve **400** (Bad Request).

2.2 API securizada (con JWT) — prefijo `/api/secured`

Mismo CRUD, pero:

- Debe exigir cabecera: `Authorization: Bearer <TOKEN>`
- Sin token o token inválido → **401** o **403**

Endpoints:

- `GET /api/secured/noticias` → **200**

- `GET /api/secured/noticias/{id}` → **200** / **404**
- `POST /api/secured/noticias` → **201**
- `PUT /api/secured/noticias/{id}` → **200**
- `DELETE /api/secured/noticias/{id}` → **204**

---

3) Login (JWT)

Los tests obtienen el token así:

- `POST /auth/login`
- Body JSON:

```
{ "username": "admin", "password": "admin" }
```

- Respuesta **200** con:

```
{ "token": "xxxxx.yyyyy.zzzzz" }
```

> El nombre de la propiedad debe ser exactamente `token`.

Usuarios

Para el examen, basta con un usuario en memoria:

- username: `admin`
- password: `admin`

---

4) Swagger / OpenAPI (recomendado)

Configura Swagger/OpenAPI para documentar:

- `/api/public/noticias/**`
- `/api/secured/noticias/**`
- `/auth/login`

---

5) Cómo se evalúa (nota automática)

Pública (máx 4)

- `GET /api/public/noticias` → 0,5 punto
- `GET /api/public/noticias/{id}` → 0,5 punto
- `POST /api/public/noticias` → 1 punto
- `PUT /api/public/noticias/{id}` → 1 punto
- `DELETE /api/public/noticias/{id}` → 1 punto

Securizada (máx 6)

- `POST /auth/login` + `GET /api/secured/noticias` → 1 punto
- `POST /api/secured/noticias` → 2 puntos
- `GET /api/secured/noticias/{id}` → 1 punto
- `PUT /api/secured/noticias/{id}` → 1 punto
- `DELETE /api/secured/noticias/{id}` → 1 punto

---

6) Ejecutar el cliente (tests)

1. Arranca tu backend de noticias en `http://localhost:8080`
2. Desde este proyecto:

```
mvn -q test -Dapi.baseUrl=http://localhost:8080 -Dapi.username=user -
Dapi.password=user
```

Verás por consola:

- La URL consultada (método + ruta)
- OK/FAIL con el status
- Nota final

Cambiar URL/credenciales (si hiciera falta)

```
mvn -q test -Dapi.baseUrl=http://localhost:8080 -Dapi.username=user -
Dapi.password=user
```

---

7) Consejos para que pase los tests

- Asegura que `POST` devuelve **201** y un JSON con `id` numérico.
- `DELETE` debe devolver **204** sin contenido.
- El `GET` de lista debe devolver siempre un array JSON.
- Implementa bien `404` cuando no existe el id.
- JWT: el secured debe rechazar peticiones sin token.

```
mvn -q test -Dapi.baseUrl=http://localhost:8080 -Dapi.username=user -
Dapi.password=user
```

salida

```
== SECURED CRUD (max 6 puntos) ==
POST http://localhost:8080/auth/login -> OK (200)
GET http://localhost:8080/api/secured/noticias -> OK (200)
POST http://localhost:8080/api/secured/noticias -> OK (201)
GET http://localhost:8080/api/secured/noticias/10 -> OK (200)
PUT http://localhost:8080/api/secured/noticias/10 -> OK (200)
DELETE http://localhost:8080/api/secured/noticias/10 -> OK (204)
GET http://localhost:8080/api/secured/noticias/10 -> OK (404)
PUNTUACIÓN SECURED: 6/6
------------------------------------------------
NOTA FINAL: 6,0/10 (public: 0/4, secured: 6/6)
------------------------------------------------
== PUBLIC CRUD (max 4 puntos) ==
GET http://localhost:8080/api/public/noticias -> OK (200)
POST http://localhost:8080/api/public/noticias -> OK (201)
PUT http://localhost:8080/api/public/noticias/11 -> OK (200)
DELETE http://localhost:8080/api/public/noticias/11 -> OK (204)
GET http://localhost:8080/api/public/noticias/11 -> OK (404)
PUNTUACIÓN PUBLIC: 4/4
GET http://localhost:8080/api/secured/noticias -> OK (401)
```