
📞 Specification Exercise: Phone Book API (Spring Boot + JWT)

1. Objective

Develop a **robust and secure RESTful API** using Spring Boot to manage a personal phone book[1]. Security must be implemented using **JSON Web Tokens (JWT)** for user authentication and authorization[2].

---

2. Technology

- **Language:** Java 17+ [3]
- **Framework:** Spring Boot 3+ [4]
- **Essential Modules:** Spring Web, Spring Data JPA, Spring Security, Validation [5]
- **Security:** JWT (JSON Web Token) [6]
- **Database:** H2 (in-memory for development) or PostgreSQL/MySQL [7]
- **Build Tool:** Maven or Gradle [8]
- **Documentation:** Swagger/OpenAPI [9]

---

3. Domain Structure (Data Model)

A. Core Entities

Entity	Properties	Relationships
User (Usuario)	id (PK), username (unique), password (encrypted), roles (e.g., USER, ADMIN) [10]	One-to-Many relationship with Contact [11].

Contact (contact)	id (PK), name, lastName, email, creationDate [12]	Many-to-One relationship with User[13]. One-to-Many relationship with Telephone[14].
Telephone (telephone)	id (PK), number, category, principal [15]	Many-to-One relationship with Contact[16].

B. Telephone Category Enum

The telephone category must be an **Enum** with the following options[17]:

- PERSONAL [18]
- PROFESSIONAL [19]
- RESIDENTIAL [20]
- CELLULAR [21]
- OTHERS [22]

---

4. Security Module (JWT)

The API must be protected so that only **authenticated users** can access the contact management endpoints[23].

Endpoint	Method	Description	Required Access
/auth/login	POST	Authenticates the user and returns the JWT[24].	Public [25]

/auth/register	POST	Allows the registration of a new User[26].	Public[27]
/contacts/**		All contact endpoints[28].	Authenticated (JWT)[29]

Key Tasks

1. Configure Spring Security to be **stateless** (sessionless), using JWT for authentication in every request[30].
2. Implement the **JWT filter** to validate the token and load the user into the security context[31].
3. **Encrypt the User's password** before saving it to the database (e.g., BCryptPasswordEncoder)[32].

---

5. Phone Book Endpoints

All contact endpoints must only be accessible with a **valid JWT** in the Authorization header[33].

The payload must ensure that the user can only access and manipulate **their own contacts**[34].

URI	Method	Description
/contacts	POST	Creates a **new Contact** associated with the authenticated user. Must accept a list of Telephones[35].
/contacts	GET	**Lists all contacts** belonging to the authenticated user[36].

/contacts/{contactId}	GET	**Returns a specific contact** by ID[37].
/contacts/{contactId}	PUT	**Updates the basic information** of a contact[38].
/contacts/{contactId}	DELETE	**Removes a contact**[39].
/contacts/{contactId}/telephones	POST	**Adds one or more Telephones** to an existing contact[40].
/contacts/{contactId}/telephones/{telephoneId}	PUT	**Updates a specific Telephone** (e.g., changing the category or number)[41].
/contacts/{contactId}/telephones/{telephoneId}	DELETE	**Removes a Telephone** from a contact[42].

---

6. Additional Functional Requirements

1. **Data Validation:** Implement validation (e.g., @NotNull, @Email, @Size from jakarta.validation) on input DTOs to ensure data integrity (e.g., contact name cannot be empty, phone number in the correct format, etc.)[43].
2. **Exception Handling:** Implement a **global exception handler** (e.g., @ControllerAdvice) to return appropriate HTTP error codes (400 Bad Request, 401 Unauthorized, 403 Forbidden, 404 Not Found) in JSON format[44].
3. **DTOs (Data Transfer Objects):** Use DTOs for data input and output from the endpoints, separating the API layer from the domain/persistence layer[45].
4. **User Association:** Ensure that when creating or listing, the Contact is **automatically associated** with the User whose JWT was used in the request[46].

5. **Resource Authorization:** When attempting to fetch, update, or delete a Contact by ID, **verify that it truly belongs to the authenticated User** (preventing one user from manipulating another's data)[47].
