

📞 Especificação de Exercício: API de Agenda Telefônica (Spring Boot + JWT)

1. 🎯 Objetivo

Desenvolver uma **API RESTful** robusta e segura utilizando **Spring Boot** para gerenciar uma agenda telefônica pessoal. A segurança deve ser implementada com **JSON Web Tokens (JWT)** para autenticação e autorização dos usuários.

2. 🛠️ Tecnologia

- **Linguagem:** Java 17+
- **Framework:** Spring Boot 3+
 - **Módulos Essenciais:** Spring Web, Spring Data JPA, Spring Security, PostgreSQL. Validadion
- **Segurança:** JWT (JSON Web Token)
- **Banco de Dados:** H2 (em memória para desenvolvimento) ou PostgreSQL/MySQL.
- **Ferramenta de Build:** Maven ou Gradle.
- **Documentação:** Swagger/OpenAPI.

3. 📄 Estrutura do Domínio (Modelo de Dados)

A. Entidades Principais:

Entidade	Propriedades	Relacionamentos
Usuario	id (PK), username (único), password (criptografada), roles (ex: USER, ADMIN).	Uma relação Um-para-Muitos com Contato.
Contato	id (PK), nome, sobrenome, email, dataCriacao.	Uma relação Muitos-para-Um com Usuario. Uma relação Um-para-Muitos com Telefone.

Telefone	id (PK), numero (string), categoria (Enum), principal (boolean).	Uma relação Muitos-para-Um com Contato.
-----------------	--	---

B. Enum CategoriaTelefone:

Deve ser um enum com as seguintes opções:

- PESSOAL
- PROFISSIONAL
- RESIDENCIAL
- CELULAR
- OUTROS

4. Módulo de Segurança (JWT)

A API deve ser protegida de forma que apenas usuários autenticados possam acessar os endpoints de gerenciamento de contatos.

Endpoint	Método	Descrição	Acesso Requerido
/auth/login	POST	Autentica o usuário e retorna o JWT .	Público
/auth/register	POST	Permite o cadastro de um novo Usuario .	Público
/contatos/**	*	Todos os endpoints de contato.	Autenticado (JWT)

Tarefas Chave:

1. Configurar o **Spring Security** para ser *stateless* (sem estado), usando **JWT** para autenticação em cada requisição.
2. Implementar o filtro JWT para validar o token e carregar o usuário no contexto de segurança.
3. Criptografar a senha do **Usuario** antes de salvar no banco de dados (ex: `BCryptPasswordEncoder`).

5. Endpoints da Agenda Telefônica

Todos os endpoints de contato devem ser acessíveis apenas com um **JWT válido** no *header Authorization*. O *payload* deve garantir que o usuário só possa acessar e manipular **seus próprios** contatos.

URI	Método	Descrição
/contatos	POST	Cria um novo Contato associado ao usuário autenticado. Deve aceitar uma lista de Telefones.
/contatos	GET	Lista todos os contatos do usuário autenticado.
/contatos/{contatoid}	GET	Retorna um contato específico por ID.
/contatos/{contatoid}	PUT	Atualiza as informações básicas de um contato.
/contatos/{contatoid}	DELETE	Remove um contato.
/contatos/{contatoid}/telefones	POST	Adiciona um ou mais Telefones a um contato existente.

/contatos/{contatoId}/telefones/{telefoneId}	PUT	Atualiza um Telefone específico (ex: mudar a categoria ou número).
/contatos/{contatoId}/telefones/{telefoneId}	DELETE	Remove um Telefone de um contato.

6. Requisitos Funcionais Adicionais

1. **Validação de Dados:** Implementar validação (ex: `@NotNull`, `@Email`, `@Size` do `jakarta.validation`) nos DTOs de entrada para garantir a integridade dos dados (ex: nome do contato não pode ser vazio, número de telefone no formato correto, etc.).
2. **Tratamento de Exceções:** Implementar um manipulador de exceções global (ex: `@ControllerAdvice`) para retornar códigos de erro HTTP apropriados (400 Bad Request, 401 Unauthorized, 403 Forbidden, 404 Not Found) em formato JSON.
3. **DTOs (Data Transfer Objects):** Usar DTOs para a entrada e saída de dados dos endpoints, separando a camada de API da camada de domínio/persistência.
4. **Associação de Usuário:** Garantir que, ao criar ou listar, o `Contato` seja automaticamente associado ao `Usuario` cujo `JWT` foi usado na requisição.
5. **Autorização de Recurso:** Ao tentar buscar, atualizar ou deletar um `Contato` por ID, verificar se ele realmente pertence ao `Usuario` autenticado (evitar que um usuário manipule dados de outro).

7. Sugestões de Estrutura de Projeto (MVC)

- `com.agenda.api.controller`: Para `UsuarioController`, `ContatoController`, `AuthController`.
 - `com.agenda.api.service`: Para a lógica de negócio (ex: `ContatoService`, `AuthService`).
 - `com.agenda.api.repository`: Para as interfaces `JpaRepository`.
 - `com.agenda.api.model` (ou `.entity`): Para as classes de entidade.
 - `com.agenda.api.dto`: Para os objetos de transferência de dados (input/output).
 - `com.agenda.api.security`: Para as classes de configuração do Spring Security e JWT.
-