Serviço Web de autenticação

DIM0614 - Programação Distribuída

Luís Andrade Araújo e Giovanni Rosário

1.0 Interface de Servico

A interface do Serviço possui as seguintes operações:

Cadastrar

public String cadastrar(String username, String password, String ip);

A operação cadastrar possui três parâmetros de entrada: username (String), password(String) e ip (String). O retorno desta operação é uma string de mensagem sobre o resultado da tentativa de cadastro. As possíveis mensagens são:

"Username já cadastrado"

"Sua senha deve ter no mínimo 8 digitos"

"Nome de usuário inválido"

Exemplo de URI para chamada do serviço:
http://<servidor>:<porta>/auth/cadastro?username=user&password=senha123&ip=1311311311

Login

public String login(String username, String password, String ip);

A operação de login possui três parâmetros de entrada: username (String), password(String) e ip (String). O retorno desta operação é uma string de mensagem sobre o resultado da tentativa de cadastro. As possíveis mensagens são:

"Login aceito";

"Usuário ou senha inocorretos"

Exemplo de URI para chamada do serviço:
http://<servidor>:<porta>/auth/login?username=user&password=senha123&ip=1311311311

Checar Usuário

public String checarUsuario(String username);

A operação de checar usuário recebe um parâmetro de entrada: username(String) e retorna uma string de mensagem da operação de checar se o usuário está cadastrado ou não. As possíveis mensagens são:

"Usuário <username> já existe"

"Usuário não existe"

URI para chamada do serviço: *http://<servidor>:<porta>/auth/users/username=user*

Mostrar Usuários

public List<String> mostrarUsuarios();

A operação de mostrar usuários não recebe parâmetro de entrada e retorna uma lista de strings com todos os usuários cadastrados.

URI para chamada do serviço: *http://<servidor>:<porta>/auth/users/*

Arquitetura

Partindo da necessidade da aplicação, verificamos que as requisições do cliente ao servidor teriam no máximo três argumentos e poderiam ser manipuladas através da URI. As requisições são sempre do tipo criação e consulta. E além do mais, as requisições são independentes e não necessitam de controle de estado do cliente. Devido a essas características, optamos por usar a arquitetura REST.

Projetamos nossa aplicação com 3 módulos. Model, Controller e Repository. O módulo de Model contém a classe que define o objeto User, que encapsula uma String username e uma String password no objeto. O módulo Repository, contém as operações de escrita e consulta em arquivo (simulando um banco de dados), para registrar os cadastros e o log das operações. O módulo Controller é o qual mapeia as rotas da API e implementa os métodos que tratam as requisições.

Instruções

A aplicação foi desenvolvida a partir do framework Spring. Para compilar a aplicação, deve-se configurar um projeto com Gradle em sua IDE. Deve-se então adicionar o framework org.springframework.boot:spring-boot-gradle-plugin:2.0.3.RELEASE utilizado como dependência do projeto e executar o arquivo Server.java como Java Application.

Mudanças em relação ao RMI

Apesar da lógica da aplicação permanecer a mesma, a natureza stateless da arquitetura REST nos obrigou a implementar um módulo DAO para simular um banco de dados. Além disso, pela natureza da comunicação https, foi preciso definir as rotas em que os métodos seriam acessados, explicitar o tipo da operação (POST ou GET) e escolher uma estratégia para passar os argumentos pela URI (QueryParam ou PathParam).