Este é um exemplo de um sistema de cadastro com as tecnologias solicitadas, utilizando React para a interface web, Java 8 com recursos como streams, Wildfly como servidor de aplicação e um banco de dados relacional.

1. Estrutura do Projeto:

* Backend (Java 8 com Wildfly):
  + Entidades:
    - User: Possui atributos como nome, email, senha.
    - Address: Possui atributos como rua, cidade, estado.
  + Repositórios:
    - UserRepository: Responsável pelas operações com a tabela User.
    - AddressRepository: Responsável pelas operações com a tabela Address.
  + Serviços:
    - UserService: Fornece lógica de negócios para usuários, como criação, atualização, busca.
    - AddressService: Fornece lógica de negócios para endereços, como criação, atualização, busca.
  + Controladores REST:
    - UserController: Executa operações com usuários através de endpoints REST.
    - AddressController: Executa operações com endereços através de endpoints REST.
* Frontend (React):
  + Componentes:
    - UserForm: Formulário para cadastro e edição de usuários.
    - AddressForm: Formulário para cadastro e edição de endereços.
    - UserList: Lista de usuários.
    - AddressList: Lista de endereços.
  + Serviços:
    - UserService: Interage com os endpoints REST do backend para operações com usuários.
    - AddressService: Interage com os endpoints REST do backend para operações com endereços.

2. Código de Exemplo (Backend):

// User.java

package com.example.entities;

import javax.persistence.\*;

@Entity

public class User {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long id;

private String name;

private String email;

private String password;

@OneToOne(cascade = CascadeType.ALL)

@JoinColumn(name = "address\_id")

private Address address;

// Getters and setters

}

// Address.java

package com.example.entities;

import javax.persistence.\*;

@Entity

public class Address {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long id;

private String street;

private String city;

private String state;

// Getters and setters

}

// UserService.java

package com.example.services;

import com.example.entities.User;

import com.example.repositories.UserRepository;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.stereotype.Service;

import java.util.List;

import java.util.Optional;

@Service

public class UserService {

@Autowired

private UserRepository userRepository;

public User createUser(User user) {

return userRepository.save(user);

}

public List<User> getAllUsers() {

return userRepository.findAll();

}

public Optional<User> getUserById(Long id) {

return userRepository.findById(id);

}

// ... other methods for updating and deleting users

}

// UserController.java

package com.example.controllers;

import com.example.entities.User;

import com.example.services.UserService;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.http.HttpStatus;

import org.springframework.http.ResponseEntity;

import org.springframework.web.bind.annotation.\*;

import java.util.List;

import java.util.Optional;

@RestController

@RequestMapping("/users")

public class UserController {

@Autowired

private UserService userService;

@PostMapping

public ResponseEntity<User> createUser(@RequestBody User user) {

User createdUser = userService.createUser(user);

return new ResponseEntity<>(createdUser, HttpStatus.CREATED);

}

@GetMapping

public ResponseEntity<List<User>> getAllUsers() {

List<User> users = userService.getAllUsers();

return new ResponseEntity<>(users, HttpStatus.OK);

}

@GetMapping("/{id}")

public ResponseEntity<User> getUserById(@PathVariable Long id) {

Optional<User> user = userService.getUserById(id);

if (user.isPresent()) {

return new ResponseEntity<>(user.get(), HttpStatus.OK);

} else {

return new ResponseEntity<>(HttpStatus.NOT\_FOUND);

}

}

// ... other methods for updating and deleting users

}

3. Recursos do Java 8:

* Streams: Utilizados para realizar operações com coleções de forma mais eficiente e concisa, como buscar todos os usuários com um determinado nome ou filtrar endereços por cidade.

4. Publicação em GIT:

* Utilize um serviço de hospedagem de repositórios GIT como GitHub ou GitLab para armazenar o código fonte do projeto.
* Crie um repositório e adicione os arquivos do projeto.
* Faça commit das alterações e envie o código para o repositório.

5. Frontend (React) - Exemplo:

// UserForm.js

import React, { useState } from 'react';

import UserService from '../services/UserService';

const UserForm = () => {

const [name, setName] = useState('');

const [email, setEmail] = useState('');

const [password, setPassword] = useState('');

const [street, setStreet] = useState('');

const [city, setCity] = useState('');

const [state, setState] = useState('');

const handleSubmit = async (event) => {

event.preventDefault();

const user = { name, email, password, address: { street, city, state } };

try {

await UserService.createUser(user);

// Handle success

} catch (error) {

// Handle error

}

};

return (

<form onSubmit={handleSubmit}>

{/\* Form fields for name, email, password, street, city, and state \*/}

<button type="submit">Criar Usuário</button>

</form>

);

};

export default UserForm;

// UserService.js

import axios from 'axios';

const API\_URL = 'http://localhost:8080/users'; // Replace with your actual API URL

class UserService {

static createUser(user) {

return axios.post(API\_URL, user);

}

// ... other methods for getting, updating, and deleting users

}

export default UserService;

Observações:

* Este é um exemplo simplificado. Um projeto real deve incluir tratamento de erros, validação de dados, testes, documentação, entre outros aspectos.
* A implementação do frontend em React é um exemplo básico, e existem diversas bibliotecas e frameworks que podem ser utilizados para criar uma interface web mais completa.
* O backend pode ser configurado para utilizar outras tecnologias de banco de dados, como PostgreSQL ou MySQL.

Benefícios:

* Tecnologia moderna: Utiliza tecnologias populares e de fácil manutenção.
* Facilidade de desenvolvimento: Permite um desenvolvimento rápido e eficiente.
* Escalabilidade: Pode ser facilmente escalado para atender a um grande número de usuários.
* Integração: Permite integração com outros sistemas através de APIs REST.

Este exemplo serve como ponto de partida para a criação de um sistema de cadastro completo e funcional. Você pode adaptá-lo de acordo com as suas necessidades e requisitos específicos.