SOFTWARE PARA GERENCIAMENTO DE MORADORES EM UM RESIDENCIAL

TÓPICOS DE ABORDAGEM

- Resumo;
- Materiais e métodos;
- Resultados;
- Discussão;
- Conclusão;

INTRODUÇÃO

- Sistemas de gerenciamento;
- Permite a manipulação de informações dos moradores
- Java na criação de servidores;
- Spring Boot como facilitador na criação de APIs RESTfull;
- O back-end utiliza um stateless
- JavaScript e React na criação de Interfaces;

MATERIAIS E MÉTODOS

- Java, Spring Boot, Eclipse, JavaScript, React JS, CSS, Yarn, VS Code;
- Back-end cria novo residente;
- Front-end cria a interface que o usuário faz as interações.

RESULTADOS

A figura ao lado representa a classe “Resident”, que pertence ao lado do servidor, código escrito em Java, que representa um residente no sistema de gerenciamento de condomínios, armazenando suas informações pessoais.

```java
package condominium.apicond.methods;
public class Resident {
        private int id;
        private static int count = 0;
        private String name;
        private String phone;
        private Integer apartment;
        private String parkingNumber;
        private String emergencyContact;

        public Resident() {
                this.setId(++count);
        }

   public void setData(String name, String phone, int apartment, String parkingNumber, String
emergencyContact){
       this.name = name;
       this.phone = phone;
       this.apartment = apartment;
       this.parkingNumber = parkingNumber;
       this.emergencyContact = emergencyContact;
   }

        public String[] getData() {
                String[] data = new String[5];

                data[0] = this.name;
                data[1] = this.phone;
                data[2] = Integer.toString(this.apartment);
                data[3] = this.parkingNumber;
                data[4] = this.emergencyContact;

                return data;
        }
        public int getId() {
                return this.id;
        }
        public void setId(int id) {
                this.id = id;
        }

        public String getName() {
                return name;
        }
        public void setName(String name) {
                this.name = name;
        }
        public String getPhone() {
                return this.phone;
        }
        public void setPhone(String phone) {
                this.phone = phone;
        }
        public Integer getAppartament() {
                return this.apartment;
        }
        public void setAppartament(int apartment) {
                this.apartment = apartment;
        }
        public String getParkingNumber() {
                return this.parkingNumber;
        }
        public void setParkingNumber(String parkingNumber) {
                this.parkingNumber = parkingNumber;
        }
        public String getEmergencyContact() {
                return this.emergencyContact;
        }
        public void setEmergencyContact(String emergencyContact) {
                this.emergencyContact = emergencyContact;
        }
}
```

RESULTADOS

A classe ao lado representa a classe “DynamicArray”, a qual é responsável por criar uma lista dinâmica de objetos da classe “Resident”, com métodos para adicionar, remover, atualizar e filtrar elementos da lista.

```
        private Resident[] list;
        private int maxSize;
        private int length;
        public DynamicArray() {
                this.maxSize = 1;
                this.list = new Resident[maxSize];
                this.length = 0;
        }
        public void add(Resident item) {
                if (isFull()) {
                        maxSize = 2 * maxSize;
                        Resident[] tempList = new Resident[2 * maxSize];
                        for (int i = 0; i < length; i++) {
                                tempList[i] = list[i];
                        }
                        list = tempList;
                }
                list[length] = item;
                length++;
        }
        public void removeItem(int id) {
                Utils Utils = new Utils();
                int indexToRemove = Utils.findIndex(list, id);

                if (indexToRemove != -1) {
                        for (int i = indexToRemove; i < this.length - 1; i++) {
                                list[i] = list[i + 1];
                        }
                        list[this.length - 1] = null;
                        this.length--;
                }

        }
        public void updateItem(int id,  Resident updatedResident) {
                Utils Utils = new Utils();
                System.out.println(this.printList());
                int indexToUpdate = Utils.findIndex(this.printList(), id);

      if (updatedResident.getName() != null) {
         list[indexToUpdate].setName(updatedResident.getName());
      }
      if (updatedResident.getPhone() != null) {
        list[indexToUpdate].setPhone(updatedResident.getPhone());
      }
      if (updatedResident.getAppartament() != null) {
        list[indexToUpdate].setAppartament(updatedResident.getAppartament());
      }
      if (updatedResident.getParkingNumber() != null) {
        list[indexToUpdate].setParkingNumber(updatedResident.getParkingNumber());
      }
      if (updatedResident.getEmergencyContact() != null) {
        list[indexToUpdate].setEmergencyContact(updatedResident.getEmergencyContact());
      }
        }
        public Resident filterItem(int id) {
                Utils Utils = new Utils();
                int indexToSearch =  Utils.findIndex(list, id);
                if(indexToSearch == -1) {
                        return null;
                }
                return list[indexToSearch];
        };
        public Resident[] printList() {
                return list;
        }
        boolean isFull() {
                return length == maxSize;
        }
}
```

RESULTADOS

- Representa a classe 'ResidentController'
- Define os endpoints
- Define a lógica utilizada nas operações CRUD dos objetos 'Resident' em API RESTful
- Objeto'DynamicArray' armazena e manipula os objetos 'Resident'

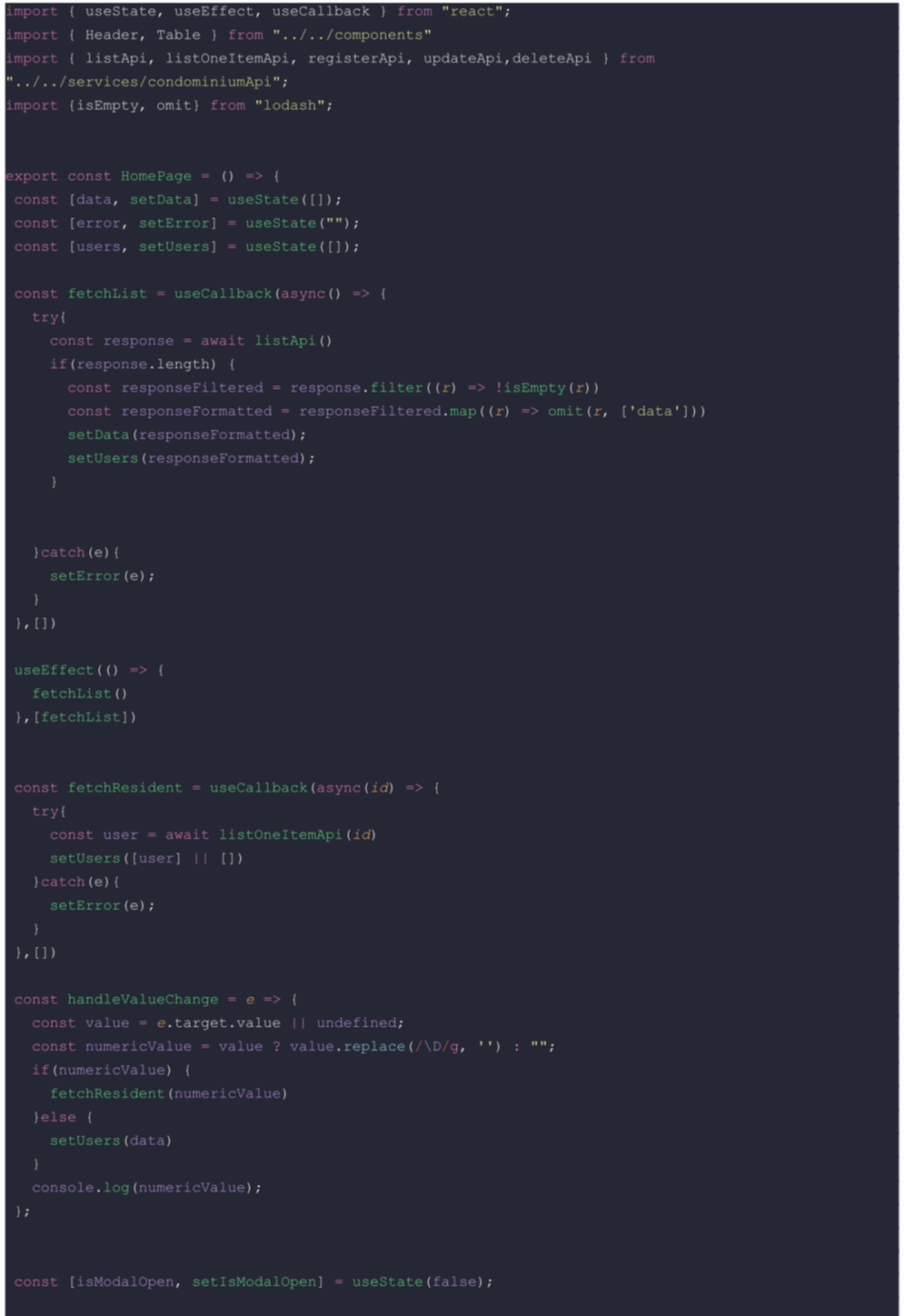
- Register
- List
- Item
- Update
- Delete

```java
package condominium.apicond.controller;

import org.springframework.http.HttpStatus;

@RestController
@RequestMapping

public class ResidentController {
    DynamicArray list = new DynamicArray();

    @CrossOrigin(origins = "*")
    @PostMapping("/resident")
    @ResponseStatus(code = HttpStatus.CREATED)
    public void register(@RequestBody ResidentModel data) {
        Resident resident = new Resident();
        resident.setData(data.name(), data.phone(), data.appartament(), data.parkingNumber(), data.emergencyContact());
        list.add(resident);
    }

    @GetMapping("/resident")
    @ResponseStatus(code = HttpStatus.OK)
    public Object list() {
        return list.printList();
    }

    @GetMapping("/resident/{id}")
    @ResponseStatus(code = HttpStatus.OK)
    public Object item(@PathVariable int id) {
        return list.filterItem(id);
    }

    @CrossOrigin(origins = "*")
    @PutMapping("/resident/{id}")
    @ResponseStatus(code = HttpStatus.ACCEPTED)
    public String update(@RequestBody Resident data, @PathVariable int id) {
        list.updateItem(id,data);
        return "Alterado com sucesso";
    }

    @CrossOrigin(origins = "*")
    @DeleteMapping("/resident/{id}")
    @ResponseStatus(code = HttpStatus.OK)
    public void delete(@PathVariable int id) {
        list.removeItem(id);
    }
}
```

RESULTADOS

A imagem ao lado, representa a implementação da página inicial que busca, exibe e interage com dados de residentes usando chamadas de API e componentes reutilizáveis, foi criada utilizando Javascript e a biblioteca React,

.



```
import { useState, useEffect, useCallback } from "react";
import { Header, Table } from "../../components"
import { listApi, listOneItemApi, registerApi, updateApi,deleteApi } from
"../../services/condominiumApi";
import {isEmpty, omit} from "lodash";


export const HomePage = () => {
 const [data, setData] = useState([]);
 const [error, setError] = useState("");
 const [users, setUsers] = useState([]);

 const fetchList = useCallback(async() => {
   try{
     const response = await listApi()
     if(response.length) {
       const responseFiltered = response.filter((r) => !isEmpty(r))
       const responseFormatted = responseFiltered.map((r) => omit(r, ['data']))
       setData(responseFormatted);
       setUsers(responseFormatted);
     }


   }catch(e){
     setError(e);
   }
 },[])

 useEffect(() => {
   fetchList()
 },[fetchList])


 const fetchResident = useCallback(async(id) => {
   try{
     const user = await listOneItemApi(id)
     setUsers([user] || [])
   }catch(e){
     setError(e);
   }
 },[])

 const handleValueChange = e => {
   const value = e.target.value || undefined;
   const numericValue = value ? value.replace(/\D/g, '') : "";
   if(numericValue) {
     fetchResident(numericValue)
   }else {
     setUsers(data)
   }
   console.log(numericValue);
 };


 const [isModalOpen, setIsModalOpen] = useState(false);
```

RESULTADOS

```
  setIsModalOpen(true);
};

const closeModal = () => {
  setIsModalOpen(false);
};

const addNewRow = async(data) => {
  try {
    data.appartament = Number(data.appartament)
    await registerApi(data);


    fetchList()
  } catch (e) {
    setError(e);
  }
  setIsModalOpen(false);
};

const columns =
[{Header:'Nome',accessorKey:'name'},{Header:'Telefone',accessorKey:'phone'},{Header:'Apartamento',
accessorKey:'appartament'},{Header:'Número da Vaga',accessorKey:'parkingNumber'},{Header:'Contato d
Emergência', accessorKey:'emergencyContact'}]


const handleDelete = async(data) => {
  try {
    console.log(data)
    await deleteApi(data.original.id)
    fetchList()
  }catch(e) {
    setError(e)
  }
}

const handleEdit = async (data) => {
  try {
    const payload = data.values
    await updateApi(data.row.original.id, payload)
    data.exitEditingMode()
    fetchList()
  }catch(e) {
    setError(e)
  }
}
```

```
return(
  <>
  <Header />
    <Table
    handleValueChange={handleValueChange}
    columns={columns}
    data={users}
    isOpen={isModalOpen}
    onRequestClose={closeModal}
    openModal={openModal}
    addNewResident={addNewRow}
    onSubmitForm={addNewRow}
    handleDelete={handleDelete}
    handleEdit={handleEdit}
  />
</>


)
}
```

CONCLUSÃO

A combinação do Spring Boot e ReactJS oferece uma solução moderna e eficaz para o gerenciamento de apartamentos. A abordagem orientada a objetos e a estrutura modular garantem a eficiência e facilidade de manutenção do sistema. O Spring Boot proporciona facilidade de desenvolvimento, suporte avançado e criação de APIs RESTful escaláveis. O ReactJS oferece uma interface do usuário centrada no navegador, atualizações suaves de página e um ecossistema colaborativo. Essas tecnologias fornecem um ambiente flexível para criar interfaces dinâmicas e intuitivas.

REACTJS X SWING

Optou-se em utilizar ReactJS à Swing, pelos seguintes motivos:

- ReactJs é uma biblioteca voltada ao desenvolvimento web, podendo ser acessado via despositivos móveis;
- Componentização e reutilzação de componentes;
- ReactJs é altamente utilizado na comunidade de desenvolvimento;
- Maior possibilidade de estilização de componentes;

DEMONSTRAÇÃO

Sistema de Residentes

Busca pelo ID

Adicionar Novo Residente

Actions	Nome	Telefone	Apartamento	Número da Vaga	Contato de Emergência
	Amanda	1198705948	61	a76	121298984

Linhas por página 10 1-1 of 1

DEMONSTRAÇÃO

Aplicação