Desenvolva essa aplicação Full Stack, que usa como Back-End java fazendo com conexão com Banco de Dados SQL o driver Postgre e Consumindo recurso a aplicação Front-End em Angular 11.

Spring + Angular

API Sala de Reuniões

Jefferson Rodrigues Da Silva

Sumário

Introdução

Este projeto tem por objetivo ser um tutorial escrito e autoexplicativo da solução proposta pela Kamila Santos, autora e desenvolvedora de toda a lógica desse código.

O objetivo é que esse manual escrito possa ajudar de alguma forma e a ter um documento por escrito.

Requisitos

Para desenvolver a aplicação:

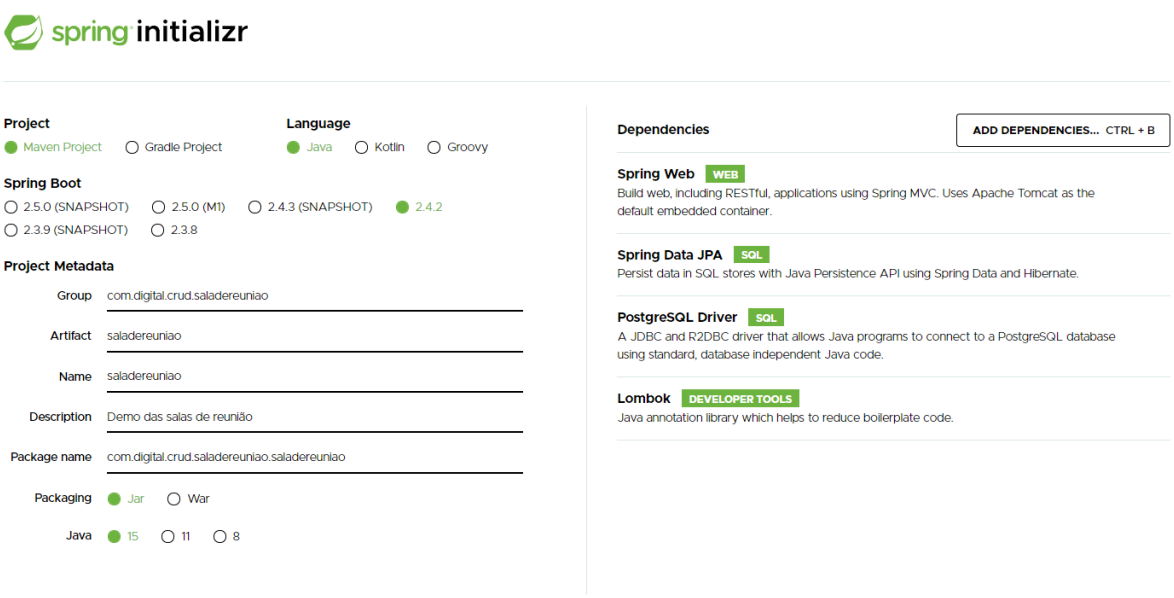
- Node
- Npm
- Eclipse
- PostgreSQL
- Visual Studio Code
- Conhecimentos em Angular
- Conhecimentos em Java
- Conhecimentos em aplicações MVC
- Conhecimentos em consumo de End Point.

Jefferson Rodrigues rs.mensur@gmail.com + 55 11 9 7973-0321

Inicialização do Projeto

Utilizando o https://start.spring.io/

Configuração



Dependências

Spring Web: Utilizado para desenvolver aplicações WEB.

Spring Data JPA: Persistência e comunicação com o Banco de Dados.

PostgreSQL Driver: Driver de conexão com o Banco de Dados.

Lombok: Biblioteca para reduzir códigos necessários e repetíveis na aplicação Java.

Configurações do Projeto Maven

pom.xml

No pom.xml acrescentamos o jakarta para cuidar das nossas validações de dados no Banco de Dados.

```
35        <dependency>
36            <groupId>org.projectlombok</groupId>
37            <artifactId>lombok</artifactId>
38            <optional>true</optional>
39        </dependency>
40
41        <!-- https://mvnrepository.com/artifact/jakarta.validation/jakarta.validation-api -->
42        <!-- Cuida das validações dos dado no Banco -->
43        <dependency>
44            <groupId>jakarta.validation</groupId>
45            <artifactId>jakarta.validation-api</artifactId>
46        </dependency>
```

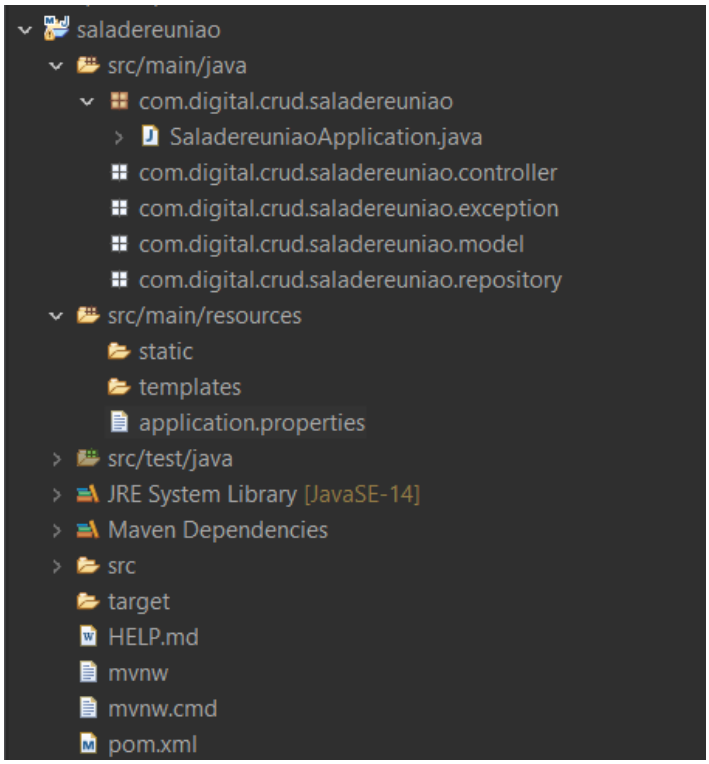
application.properties

Configurar a Conexão com o Banco de Dados.

```
1 spring.datasource.url=jdbc:postgresql://localhost:5432/db_saladereuniao
2 spring.datasource.driver-class-name=org.postgresql.Driver
3 spring.datasource.username=postgres
4 spring.datasource.password=admin
5 spring.jpa.database=postgresql
6 spring.datasource.platform=postgres
7 spring.jpa.database-platform=org.hibernate.dialect.PostgreSQL10Dialect
8 spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.PostgreSQL10Dialect
9 spring.jpa.properties.hibernate.format_sql=true
10 spring.jpa.hibernate.ddl-auto=update
11 spring.jpa.hibernate.show-sql=true
```

Estrutura do Projeto - Maven Java Spring Boot

Criar a Estrutura do Projeto antes de popular - Pacotes/Pastas principais



Controller – onde teremos o end-point da aplicação

Exception – mensagens de tratamento de exceção

Model – estrutura de dados da aplicação

Repository – comunicação com o BD em si

Observação: Seguimos essa ordem porque como um componente fará uso de outro pode incorrer em erros/alertas na IDE o que pode nos gerar a sensação de erro de criação do projeto.

 # Pacotes / Packages:

Model, Repository, Exception e por fim Controller.

 # Arquivos / Classes de Exceptions:

ErrorDetails, ResourceNotFoundException e GlobalExceptionHandler.

Controller

RoomController

```java
package com.digital.crud.saladereuniao.controller;

import java.util.HashMap;
import java.util.List;
import java.util.Map;

import javax.validation.Valid;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.CrossOrigin;
import org.springframework.web.bind.annotation.DeleteMapping;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.PutMapping;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

import com.digital.crud.saladereuniao.exception.ResourceNotFoundException;
import com.digital.crud.saladereuniao.model.Room;
import com.digital.crud.saladereuniao.repository.RoomRepository;

/* Informar que é um RestController */
@RestController
/* Habilitar o CrossOrigin para o front poder consumir
 * Pode habilidar de qualquer lugar, ou no caso, informar
 * o endpoint no qual a aplicação angular irá subir.
 * */
@CrossOrigin(origins = "http://localhost:4200")
@RequestMapping("/api/v1")
public class RoomController {

    @Autowired
    private RoomRepository roomRepository;

    @GetMapping("/rooms")
    public List<Room> getAllRooms() {
        return roomRepository.findAll();
    }
```

```java

    @GetMapping("/rooms/{id}")
    public ResponseEntity<Room> getRoomById(@PathVariable(value = "id") long roomId)
            throws ResourceNotFoundException {

        Room room = roomRepository
                .findById(roomId)
                .orElseThrow(() -> new ResourceNotFoundException("Room not found for id:: " + roomId))
        ;

        return ResponseEntity.ok().body(room);
    }

    /* @Valid -> Validar se o json que mandamos pra criarn a sala é válido  */
    @PostMapping("/rooms")
    public Room createRoom(@Valid @RequestBody Room room) {
        return roomRepository.save(room);
    }

    @PutMapping("/rooms/{id}")
    public ResponseEntity<Room> updateRoom(
            @PathVariable (value = "id") Long roomId,
            @Valid @RequestBody Room roomDetails
            ) throws ResourceNotFoundException {

        Room room = roomRepository
                .findById(roomId)
                .orElseThrow(() -> new ResourceNotFoundException("Room not found for this id: " + roomId))
        ;

        room.setName(roomDetails.getName());
        room.setDate(roomDetails.getDate());
        room.setStartHour(roomDetails.getStartHour());
        room.setEndHour(roomDetails.getEndHour());

        final Room updateRoom = roomRepository.save(room);

        return ResponseEntity.ok(updateRoom);
    }
```

```java

    @DeleteMapping("/rooms/{id}")
    public Map<String, Boolean> deleteRoom(
            @PathVariable (value = "id") Long roomId
            )
            throws ResourceNotFoundException {

        Room room = roomRepository
                .findById(roomId)
                .orElseThrow(() -> new ResourceNotFoundException("Room not found for this id: " + roomId))
        ;

        roomRepository.delete(room);
        Map<String, Boolean> response = new HashMap<>();
        response.put("deleted", Boolean.TRUE);

        return response;
    }

}
```

Exception

ErrorDetails

```java
package com.digital.crud.saladereuniao.exception;

import java.util.Date;

public class ErrorDetails {

    // Atributos
    private Date timestamp;
    private String message;
    private String details;


    // Construtor
    public ErrorDetails(Date timestamp, String message, String details) {
        super();
        this.timestamp = timestamp;
        this.message = message;
        this.details = details;
    }

    // Getters
    public Date getTimestamp() {
        return timestamp;
    }

    public String getMessage() {
        return message;
    }

    public String getDetails() {
        return details;
    }


}

```

GlobalExceptionHandler

```java
package com.digital.crud.saladereuniao.exception;

import java.util.Date;

import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.ControllerAdvice;
import org.springframework.web.bind.annotation.ExceptionHandler;
import org.springframework.web.context.request.WebRequest;

@ControllerAdvice // Sinalizar q é um advice do controlller q vamos criar
public class GlobalExceptionHandler {

    // ResponseEntity -> Resposta da Entidade
    // Qual a classe que esse global é responsável por lançar a exceção
    @ExceptionHandler (ResourceNotFoundException.class)
    public ResponseEntity<?> resourceNotFoundException(
            ResourceNotFoundException ex,
            WebRequest request
            ) {
        ErrorDetails errorDetails = new ErrorDetails(
                    new Date(),
                    ex.getMessage(),
                    request.getDescription(false)
                );
        return new ResponseEntity<>(errorDetails, HttpStatus.NOT_FOUND);
    }

    @ExceptionHandler(Exception.class)
    public ResponseEntity<?> globalExceptionHandler(
            Exception ex,
            WebRequest request
            ) {
        ErrorDetails errorDetails = new ErrorDetails(
                    new Date(),
                    ex.getMessage(),
                    request.getDescription(false)
                );
        return new ResponseEntity<>(errorDetails, HttpStatus.INTERNAL_SERVER_ERROR);
    }
}
```

ResourceNotFoundException

```java
package com.digital.crud.saladereuniao.exception;

import org.springframework.http.HttpStatus;
import org.springframework.web.bind.annotation.ResponseStatus;

@ResponseStatus(value = HttpStatus.NOT_FOUND)
public class ResourceNotFoundException extends Exception {

    private static final long serialVersionUID = 1L;

    public ResourceNotFoundException(String message) {
        super(message);
    }

}
```

Model

Room

```java
package com.digital.crud.saladereuniao.model;

import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.Table;

@Entity // Sinalizar Hibernate/Spring que é classe de configuração da Entidade
@Table( name = "meetingroom" ) // Informações da tabela
public class Room {
    // Atributos
    private long id;
    private String name;
    private String date; // Converter pro formato correto.
    private String startHour; // Converter pro formato correto.
    private String endHour; // Converter pro formato correto.

    // Construtores
    public Room() {}
    public Room(long id, String name, String date, String startHour, String endHour) {
        this.id = id;
        this.name = name;
        this.date = date;
        this.startHour = startHour;
        this.endHour = endHour;
    }

    // Getters e Setters
    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    public long getId() {
        return id;
    }
    public void setId(long id) {
        this.id = id;
    }

    @Column(name = "name", nullable = false)
    public String getName() {
        return name;
    }
    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }


    @Column(name = "date", nullable = false)
    public String getDate() {
        return date;
    }
    public void setDate(String date) {
        this.date = date;
    }


    @Column(name = "startHour", nullable = false)
    public String getStartHour() {
        return startHour;
    }
    public void setStartHour(String startHour) {
        this.startHour = startHour;
    }


    @Column(name = "endHour", nullable = false)
    public String getEndHour() {
        return endHour;
    }
    public void setEndHour(String endHour) {
        this.endHour = endHour;
    }

    @Override
    public String toString() {
        return "Room [id=" + id + ", name=" + name + ", date=" + date + ", startHour=" + startHour + ", endHour="
                + endHour + "]";
    }


}
```

Repository

RoomRepository

```java
package com.digital.crud.saladereuniao.repository;

import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.stereotype.Repository;

import com.digital.crud.saladereuniao.model.Room;

@Repository
public interface RoomRepository extends JpaRepository<Room, Long> {

}
```

Front End

O front-end que irá consumir serviços de acesso da API será uma aplicação SPA desenvolvida em Angular.

Requisitos

- Node
- NPM
- Angular

Node

Para ver a versão digitar no terminal:

node –v

v14.15.5

NPM

Para ver a versão digitar no terminal:

npm –v

6.14.11

Angular

Para ver a versão digitar no terminal:

ng v

Ivy Workspace - 11.2.0

Angular

Framework para construção da Interface de aplicações usando HTML, CSS e, principalmente, JavaScript, criada pelos desenvolvedores da Google.

Baseado em componentes.

Desenvolvers SPA's: uma aplicação web que roda em uma única página e atualiza só o que você desejar.

Angular tem um universo de possibilidades dentro de suas aplicações:

Angular: Framework

Protractor: Para testes End-To-End

Forms: Desenvolver formulários

PWA: Aplicações estilo mobile, sem efetivamente fazer o download do app, só instalar o manifest.

Augury: Debug do Angular pelo Crome.

Language Services: .

Router: Parte de roteamento da aplicação.

Elements: Criação de elementos.

CDK: Para parte de componentes.

Universal: Para parte de mobile.

Karma: Para testes.

Labs: Para testes.

Compiler: Parte de compilação do Angular.

i18n: Para serviços de linguagem.

Http: Protocolo.

Material: Responsável pelo design de nossa aplicação.

Animations: Parte de animação.

CLI: Linha de comando.

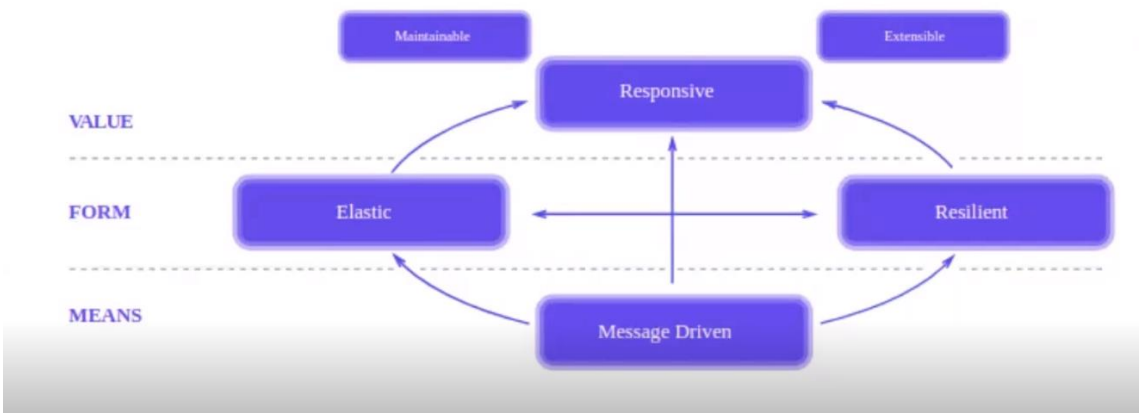
RXJS

Biblioteca para desenvolver aplicações assíncronas baseadas em eventos usando sequências de observables.

Observables

Tem um tipo "core" que é o Observable e os chamados tipos "secundários" (Schedulers, Subscriptions, Subjects).

O RXJS gira em torno do manifesto Reativo:



Responsivo

O sistema responde em tempo hábil, se possível.

Resiliente

O sistema permanece responsivo diante de falhas.

Elástivo
O sistema permanece responsivo diante de uma carga de trabalho variável.

Message Driven
Aplicações reativas dependem da passagem de mensagens assíncronas para estabelecer um limite entre os componentes, garantindo um acoplamento flexível, isolamento e transparência.

Lifecicle Hooks
Gerenciadores do ciclo de vida dos componentes.

Todo componente angular tem um ciclo de vida: passa pelo processo de criação, execução e possível destruição.

É realizado pelo Angular ao criar o componente, renderizar, criar e renderizar seus filhos, verificar modificações nas propriedades e no DOM, realizar alterações, destruir e remover do DOM.

ngOnChanges()
É o primeiro lifeCycle Hook, é chamado logo após a inicialização da classe e o componente é criado.

Também é chamado quando há alteração de propriedade dentro do seu componente.

ngOnInit()
É chamado depois do **ngOnChanges()**, só é chamado uma única vez, inicializa o componente, define e exibe suas propriedades de entrada.

É o lifeCycle mais importante do Angular, pois “alerta” que um componente foi criado.

ngDoCheck()
É chamado durante todas verificações de mudança durante a execução, geralmente depois que o **ngOnInit()** é chamado.

ngAfterContentInit()
Só é executado uma vez depois que o primeiro **ngDoCheck()** é chamado, semelhando ao ndDoCheck(), mas projetado para visualização do componente.

ngAfterContentChecked()
Este hook é chamado depois que o conteúdo projetado para a visualização do componente é inicializado, após o **ngAfterContentInit()** e **ngDoCheck()** forem chamados.

ngAfterViewInit()
Chamado após as visualizações dos componentes e seus subsequentes filhos, após a execução do hook **ngAfterContentChecked()**.

ngAfterViewChecked()
Nosso penúltimo lifecycle hook, executados após todos os demais.

ngOnDestoy()
O último lifeCycle, é chamado antes do componente ser removido do DOM.

É feita a limpeza do componente, desde a desanexação de event handlers até a desinscrição de observables.

Event Binding

Utilizamos o eventBinding para vincular o código da aplicação a um determinado evento disparado pelo navegador, como **“apertar botão”** ou a **“submissão de formulário”**.

Como usar o Evend Binding ?

1. Qual elemento emitirá o evento?
   1.1. **(click)** -> Quando o mouse clica em algum elemento ele é disparado.

```
@Component({
  selector: 'app-clique-aqui',
  template: `
    <button (click)="AoClicar()">Me clique!</button>
    {{clickMessage}}`
})
```

   1.2. **(keyup.enter)** -> Ouve o pressionamento da tecla Enter, pois sinaliza que o usuário terminou de digitar.

```
@Component({
  selector: 'app-key-up-example',
  template: `
    <input #box (keyup.enter)="onEnter(box.value)">
    <p>{{value}}</p>
  `
})
```

```
export class KeyUpComponentExample {
  value = '';
  onEnter(value: string) { this.value = value; }
}
```

   1.3. E vários outros eventos:
      1.3.1. **(drag)** = ”myDragFunction()”
      1.3.2. **(drop)** = ”myDropFunction()”
      1.3.3. **(dragover)** = ”myDragOverFunction()”
      1.3.4. **(blur)** = “myBlurFunction()”
      1.3.5. **(focus)** = “myFocusFunction()”
      1.3.6. **(scroll) =** “myScrollFunction()”
      1.3.7. **(submit) =** “mySubmitFunction()”
      1.3.8. **(click) =** “myClickFunction()”
      1.3.9. **(dbclick) =** “myDbClickFunction()”
      1.3.10. **(cut) =** “myCutFunction()”
      1.3.11. **(copy) =** “myCopyFunction()”
      1.3.12. **(paste) =** “myPasteFunction()”
      1.3.13. **(keyup) =** “myKeyUpFunction()”
      1.3.14. **(keypress) =** “myKeyPressFunction()”
      1.3.15. **(keydown) =** “myKeyDownFunction()”

1.3.16. **(mouseup) =** “myMouseUpFunction()”
1.3.17. **(mousedown) =** “myMouseDownFunction()”
1.3.18. **(mouseenter) =** “myMouseEnterFunction()”

2. Qual o nome do evento que será emitido?
3. Qual o nome do método do componente que será executado em resposta ao evento?

Inicialização do Projeto

Com o terminal **CDM** no caminho da pasta que deseja criar a sua aplicação front-end, a que irá consumir a aplicação Java.

AÇÃO	COMANDO VIA LINHA DE COMANDO (CDM) / NAVEGADOR
Criar Projeto	C:\...\Desktop\saladereuniao_webapp> ng new client-room
Entrar no Projeto	C:\ ...\Desktop\saladereuniao_webapp> cd .\client-room\
Instalar pacotes de desenvolvimento	C:\ ...\Desktop\saladereuniao_webapp\client-room> npm i --only=dev
Instalar demais pacotes	C:\ ...\Desktop\saladereuniao_webapp\client-room> npm i
Instalar JQuery	C:\ ...\Desktop\saladereuniao_webapp\client-room> npm i bootstrap jquery --save
Rodar a aplicação	C:\ ...\Desktop\saladereuniao_webapp\client-room> ng serve
Verificar no Browser	No navegador de sua preferência entrar no endereço: http://localhost:4200/

Estrutura da Aplicação - Angular

A estrutura é criada de forma automática por causa do Angular CLI.

Numa apresentação geral temos as pastas de: e2e, node_modules, src.

e2e

Parte de testes da aplicação.

node_modules

Onde as dependências de funcionalidade e de desenvolvimento são instaladas.

src

Local onde implementos e desenvolvemos a aplicação.

src/app

fd

src/assets

Armazenamos imagens

src/environments

Configuração de deploy

Package.json

Configurações com scripts para serviços da aplicação, como build deploy.

Local onde está informado quais as depências da nossa aplicação.

Pacotes / Bibliotecas / Frameworks - Externos

Nesse projeto vamos baixar os pacotes para utilizá-los em nossa aplicação. Logo as configurações serão feitas no arquivo do angular.json. E também já fizemos o download dos arquivos conforme Configuração Inicial o Projeto.

Bootstrap

Usado para estilizar a aplicação.

Instalar e Configurar.

Existe 3 formatos de configurar o Bootstrap no Angular:

1. **angular.json**
   1.1. Informamos ao Angular para usar códigos de estilização do Bootstrap e não os defaults do Angular.

   De

   Para

2. **styles.scss**
3. **index.html**

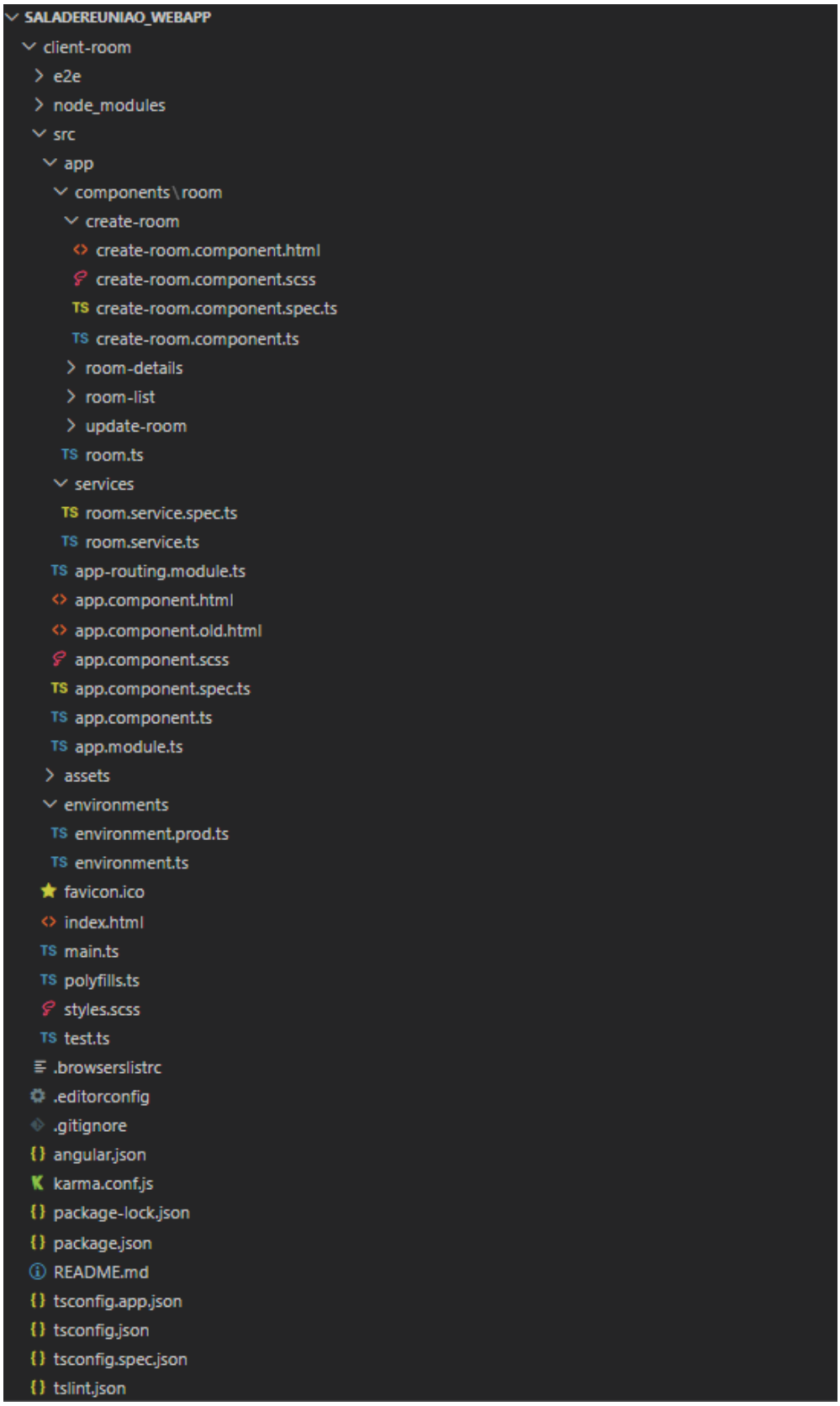
JQuery

Usado para manipular e tratar eventos relacionados aos botões do nosso CRUD.

Instalar e Configurar Jquery.

Estrutura do Projeto - Angular

Estrutura para renderizar a aplicação ao usuário e consumir os dados.



Services

A função do service é o de ser o controller da aplicação Front-End. Ela irá receber as solicitações da Model que será gerado pelo usuário, fazer solicitações a aplicação Back-End, no caso nossa aplicação Java, receber a resposta e passa-la ao componente que solicitou, seja a resposta a solicitação desejada ou informação de erro.

Ao gerar o service pelo Angular CLI é gerado tanto o service como também o arquivo para realizar testes do service (cuja extensão é: ***.spec.ts**).

Criar Service

C:\Users\re041598\Desktop\saladereuniao_webapp\client-room> ng g s services/room

client-room > src > app > services > TS room.service.ts > ...

```typescript
import { Injectable } from '@angular/core';
import { HttpClient } from '@angular/common/http';
import { Observable } from 'rxjs';

@Injectable({
  providedIn: 'root'
})
export class RoomService {

  private baseUrl = 'http://localhost:8080/api/v1/rooms';

  constructor(
    private http: HttpClient
  ) { }

  getRoom(id: number): Observable<any> {
    return this.http.get(`${this.baseUrl}/${id}`);
  }

  createRoom(room: Object): Observable<Object> {
    return this.http.post(`${this.baseUrl}`, room);
  }

  updateRoom(id: number, value: any): Observable<Object> {
    return this.http.put(`${this.baseUrl}/${id}`, value);
  }

  deleteRoom(id: number): Observable<Object> {
    return this.http.delete(`${this.baseUrl}/${id}`, { responseType: 'text'});
  }

  getRoomList(): Observable<any> {
    return this.http.get(`${this.baseUrl}`);
  }
}

```

Components

Terá por finalidade, numa aplicação MVC ser o Model. Renderizar informações ao usuário e também solicitar informações ao Back-End.

Ao gerar o componente pelo angular CLI, geralmente, é gerado 4 arquivos:

1. ***.component.html**
   1.1. É a parte que é renderizada ao usuário, e conforme suas interações, chama métodos no arquivo "***.component.ts**" e depois informa as respostas ao usuário.
2. ***.component.scss**
   2.1. Definimos alguma lógica de estilo caso necessário.
3. ***.component.spec.ts**
   3.1. Para desenvolver lógica de testes do componente.
4. ***.component.ts**
   4.1. Implementamos a lógica que responderá às interações do usuário, ou seja, é nesse arquivo que realizamos as chamadas ao **service**, e mapeamos as respostas para que seja renderizada no arquivo "***.component.html**".

Criar Componentes

C:\Users\...\saladereuniao_webapp\client-room> ng g c components/room /create-room

C:\Users\...\saladereuniao_webapp\client-room> ng g c components/room /room-details

C:\Users\...\saladereuniao_webapp\client-room> ng g c componentes/room/room-list

C:\Users\...\saladereuniao_webapp\client-room> ng g c components/room /update-room

Observação: Não iremos abordar testes e como, no quesito de estilos, iremos usar o Bootstrap então não iremos usar os arquivos: "***.component.spec.ts**" e "***.component.scss**".

CreateRoomComponent

client-room > src > app > components > room > create-room > create-room.component.html > div

```html
<h3>Create Room</h3>
<div class="largura" [hidden]="submitted">
    <form (ngSubmit)="onSubmit()">
        <div class="form-group">
            <label for="name">name</label>
            <input type="text" class="form-control" id="name" required [(ngModel)]="room.name" name="name">
        </div>
        <div class="form-group">
            <label for="date">date</label>
            <input type="text" class="form-control" id="date" required [(ngModel)]="room.date" name="date">
        </div>
        <div class="form-group">
            <label for="startHour">startHour</label>
            <input type="text" class="form-control" id="startHour" required [(ngModel)]="room.startHour"
                name="startHour">
        </div>
        <div class="form-group">
            <label for="endHour">endHour</label>
            <input type="text" class="form-control" id="endHour" required [(ngModel)]="room.endHour" name="endHour">
        </div>
        <button type="submit" class="btn btn-success">Submit</button>
    </form>
</div>

<div [hidden]="!submitted">
    <button class="btn btn-success">Add</button> -->
</div>
```

client-room > src > app > components > room > create-room > create-room.component.ts > ...

```ts
import { Component, OnInit } from '@angular/core';
import { Router } from '@angular/router';
import { RoomService } from 'src/app/services/room.service';
import { Room } from '../room';
@Component({
  selector: 'app-create-room',
  templateUrl: './create-room.component.html',
  styleUrls: ['./create-room.component.scss']
})
export class CreateRoomComponent implements OnInit {
  room: Room = new Room();
  submitted = false;
  constructor(
    private roomService: RoomService,
    private router: Router
  ) { }
  ngOnInit(): void {
  }
  newRoom(): void {
    this.submitted = false;
    this.room = new Room();
  }
  save() {
    this.roomService
      .createRoom(this.room)
      .subscribe(
        (data: any) => console.log(data),
        (error: any) => console.log(error)
      );
    this.room = new Room();
    this.goToList();
  }
  onSubmit() {
    this.submitted = true;
    this.save();
  }
  goToList() {
    this.router.navigate(['/rooms']);
  }
}
```

RoomDetailsComponent

client-room > src > app > components > room > room-details > room-details.component.html > button.btn.btn-primary

```html
<h2>Room Details</h2>
<hr>
<div *ngIf="room">
    <div>
        <label>Name</label> {{ room.name }}
    </div>
    <div>
        <label>date</label> {{ room.date }}
    </div>
    <div>
        <label>startHour</label> {{ room.startHour }}
    </div>
    <div>
        <label>endHour</label> {{ room.endHour }}
    </div>
</div>
<br>
<br>
<button (click)="list()" class="btn btn-primary" >Back to Room List</button>
```

client-room > src > app > components > room > room-details > room-details.component.ts > ...

```ts
import { Component, OnInit } from '@angular/core';
import { ActivatedRoute, Router } from '@angular/router';
import { RoomService } from 'src/app/services/room.service';
import { Room } from '../room';

@Component({
  selector: 'app-room-details',
  templateUrl: './room-details.component.html',
  styleUrls: ['./room-details.component.scss']
})
export class RoomDetailsComponent implements OnInit {

  id!: number;
  room!: Room;

  constructor(
    private route: ActivatedRoute,
    private router: Router,
    private roomService: RoomService
  ) { }

  ngOnInit(): void {
    this.room = new Room();
    this.id = this.route.snapshot.params['id'];

    this.roomService
      .getRoom(this.id)
      .subscribe((data: any) => {
        console.log(data);
        this.room = data;
      },
      (error: any) => console.log(error))
    ;
  }

  list() {
    this.router.navigate(['rooms']);
  }
}
```

RoomListComponent

client-room > src > app > components > room > room-list > room-list.component.html > div.panel.panel-primary

```html
<div class="panel panel-primary">
    <div class="panel-heading">
        <h2>Room List</h2>
    </div>
    <div class="panel-body">
        <table class="table table-striped">
            <thead>
                <tr>
                    <th>name</th>
                    <th>date</th>
                    <th>startHour</th>
                    <th>endHour</th>
                    <th>Actions</th>
                </tr>
            </thead>
            <tbody>
                <tr *ngFor="let room of rooms | async">
                    <td>{{ room.name }}</td>
                    <td>{{ room.date }}</td>
                    <td>{{ room.startHour }}</td>
                    <td>{{ room.endHour }}</td>
                    <td>
                        <button (click)="deleteRoom(room.id)" class="btn btn-danger" >Delete</button>
                        <button (click)="updateRoom(room.id)" class="btn btn-info" >Update</button>
                        <button (click)="roomDetails(room.id)" class="btn btn-info" >Details</button>
                    </td>
                </tr>
            </tbody>
        </table>
    </div>
</div>
```

client-room > src > app > components > room > room-list > TS room-list.component.ts > ...

```typescript
import { Component, OnInit } from '@angular/core';
import { Router } from '@angular/router';
import { Observable } from 'rxjs';
import { RoomService } from 'src/app/services/room.service';
import { Room } from '../room';
@Component({
  selector: 'app-room-list',
  templateUrl: './room-list.component.html',
  styleUrls: ['./room-list.component.scss']
})
export class RoomListComponent implements OnInit {
  rooms!: Observable<Room[]>;
  constructor(
    private roomService: RoomService,
    private router: Router
  ) { }
  ngOnInit(): void {
    this.reloadData();
  }
  reloadData() {
    this.rooms = this.roomService.getRoomList();
  }
  deleteRoom(id: number) {
    this.roomService
      .deleteRoom(id)
      .subscribe(
        (data: any) => {
          console.log(data)
          this.reloadData();
        },
        (error: any) => console.log(error)
      );
  }
  roomDetails(id: number) {
    this.router.navigate(['details', id]);
  }
  updateRoom(id: number) {
    this.router.navigate(['update', id]);
  }
}
```

UpdateRoomComponent

client-room > src > app > components > room > update-room > update-room.component.html > div

```html
<h3>Update Room</h3>
<div class="largura" [hidden]="submitted">
    <form (ngSubmit)="onSubmit()">
        <div class="form-group">
            <label for="name">name</label>
            <input type="text" class="form-control" id="name" required [(ngModel)]="room.name" name="name">
        </div>
        <div class="form-group">
            <label for="date">date</label>
            <input type="text" class="form-control" id="date" required [(ngModel)]="room.date" name="date">
        </div>
        <div class="form-group">
            <label for="startHour">startHour</label>
            <input type="text" class="form-control" id="startHour" required [(ngModel)]="room.startHour"
                name="startHour">
        </div>
        <div class="form-group">
            <label for="endHour">endHour</label>
            <input type="text" class="form-control" id="endHour" required [(ngModel)]="room.endHour" name="endHour">
        </div>
        <button type="submit" class="btn btn-success">Submit</button>
    </form>
</div>
<div [hidden]="!submitted">
    <button class="btn btn-success">Add</button> -->
</div>
```

client-room > src > app > components > room > update-room > update-room.component.ts > ...

```ts
import { Component, OnInit } from '@angular/core';
import { ActivatedRoute, Router } from '@angular/router';
import { RoomService } from 'src/app/services/room.service';
import { Room } from '../room';
@Component({
  selector: 'app-update-room',
  templateUrl: './update-room.component.html',
  styleUrls: ['./update-room.component.scss']
})
export class UpdateRoomComponent implements OnInit {
  id!: number;
  room!: Room;
  submitted = false;
  constructor( private route: ActivatedRoute, private router: Router, private roomService: RoomService
  ) { }
  ngOnInit(): void {
    this.room = new Room();
    this.id = this.route.snapshot.params['id'];
    this.roomService.getRoom(this.id).subscribe((data: any) => {
          console.log(data);
          this.room = data;
        }, (error: any) => console.log(error));
  }
  updateRoom() {
    this.roomService.updateRoom(this.id, this.room).subscribe(
        (data: any) => console.log(data),
        (error: any) => console.log(error)
      ) ;
    this.room = new Room();
    this.goToList();
  }
  onSubmit() {
    this.updateRoom();
  }
  goToList() {
    this.router.navigate(['/rooms']);
  }
}
```

Configurações de Outros Componentes

Para renderizar e para que possamos consumir as rotas, e também fazer o binding das propriedades dos componentes entre seus arquivos *.html e *.ts precisa-se alterar mais 3 arquivos: **app-routing.module.ts**, **app.component.html** e **app.module.ts**.

app-routing.module.ts

client-room > src > app > app-routing.module.ts > ...

```ts
import { RoomListComponent } from './components/room/room-list/room-list.component';
import { UpdateRoomComponent } from './components/room/update-room/update-room.component';

const routes: Routes = [
  { path: '', redirectTo: 'rooms', pathMatch: 'full' },
  { path: 'rooms', component: RoomListComponent },
  { path: 'add', component: CreateRoomComponent },
  { path: 'update/:id', component: UpdateRoomComponent },
  { path: 'details/:id', component: RoomDetailsComponent },
];

@NgModule({
  imports: [RouterModule.forRoot(routes)],
  exports: [RouterModule]
})
export class AppRoutingModule { }
```

app.component.html

client-room > src > app > app.component.html > footer.footer

```html
<nav class="navbar navbar-expand-sm bg-primary navbar-dark">
  <!-- Links -->
  <ul class="navbar-nav">
    <li class="nav-item">
      <a routerLink="/" class="nav-link" routerLinkActive="active">Room List</a>
    </li>
    <li class="nav-item">
      <a routerLink="add" class="nav-link" routerLinkActive="active">Add Room</a>
    </li>
  </ul>
</nav>
<div class="container">
  <br>
  <h2 style="text-align: center;">{{title}}</h2>
  <hr>
  <div class="card">
    <div class="card-body">
  <router-outlet></router-outlet>
    </div>
  </div>
</div>

<footer class="footer">
  <div class="container">
      <span>Live Coding Digital Innovation One - From Kamila Santos Oliveira</span>
      <span>Readapted to Angular 11 by Jefferson Rodrigues da Silva - on 13 - Fev - 2021</span>
  </div>
</footer>
```

app.module.ts

client-room > src > app > app.module.ts > ...

```ts
import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';

import { AppRoutingModule } from './app-routing.module';
import { FormsModule } from '@angular/forms';
import { HttpClientModule } from '@angular/common/http';
import { AppComponent } from './app.component';
import { CreateRoomComponent } from './components/room/create-room/create-room.component';
import { RoomDetailsComponent } from './components/room/room-details/room-details.component';
import { RoomListComponent } from './components/room/room-list/room-list.component';
import { UpdateRoomComponent } from './components/room/update-room/update-room.component';

@NgModule({
  declarations: [
    AppComponent,
    CreateRoomComponent,
    RoomDetailsComponent,
    RoomListComponent,
    UpdateRoomComponent
  ],
  imports: [
    BrowserModule,
    AppRoutingModule,
    FormsModule,
    HttpClientModule
  ],
  providers: [],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

Testar Projeto

Back End

Na aplicação eclipse:

Dar o play, para que a aplicação realize conexão com o Banco de Dados e disponibilize a porta http://localhost:8080/ para que a aplicação Front End consiga se conectar.

Front End

Na terminal cdm:

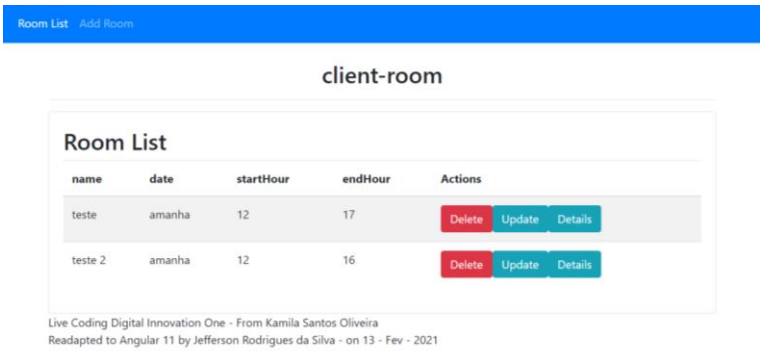
C:\Users\...\saladereuniao_webapp\client-room> ng s

A aplicação ficará disponível na porta padrão de aplicações angular na porta http://localhost:4200/ .

Layouts

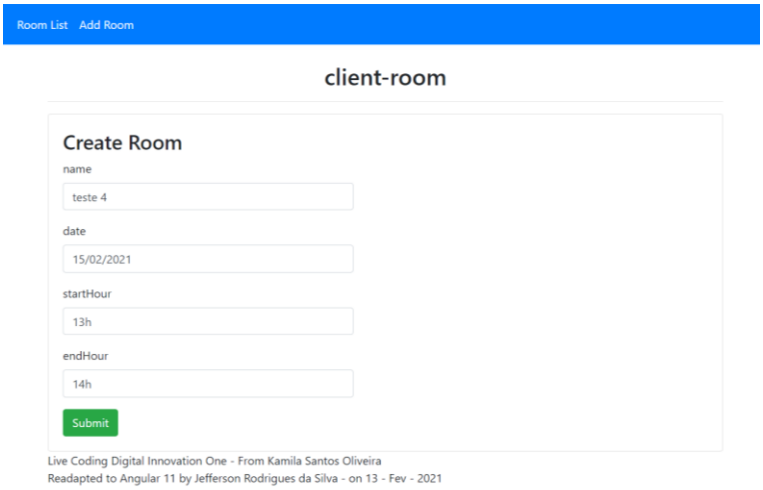
http://localhost:4200/rooms

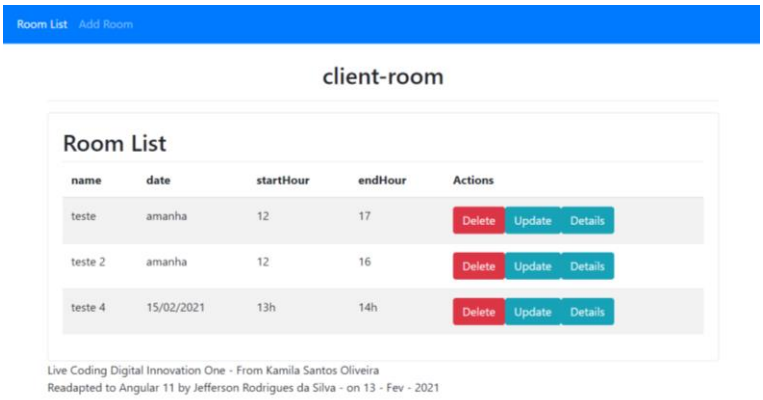
Como foi configurado no **app.routing** mesmo que apenas entremos no **http://localhost:4200/** ele será direcionado para o end point **/rooms**, onde renderiza o componente **RoomListComponent**.



http://localhost:4200/add

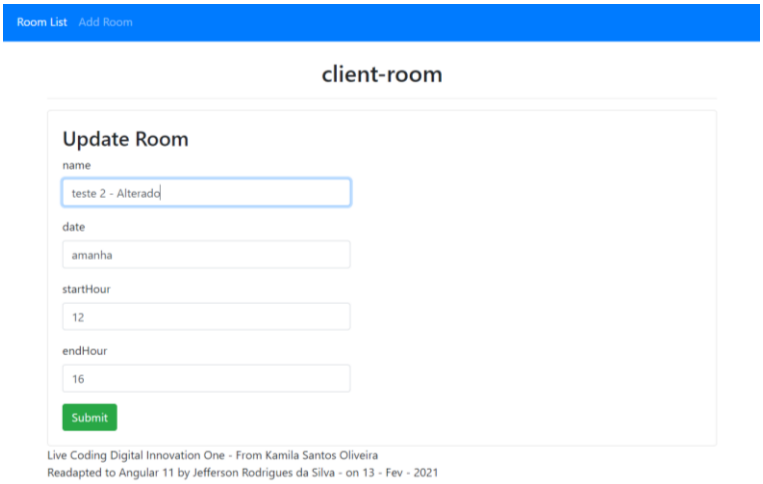
Na rota */add, conforme o **app.routing**, renderizamos o componente **CreateRoomComponent**, e após criado é retornado para a lista de rooms.

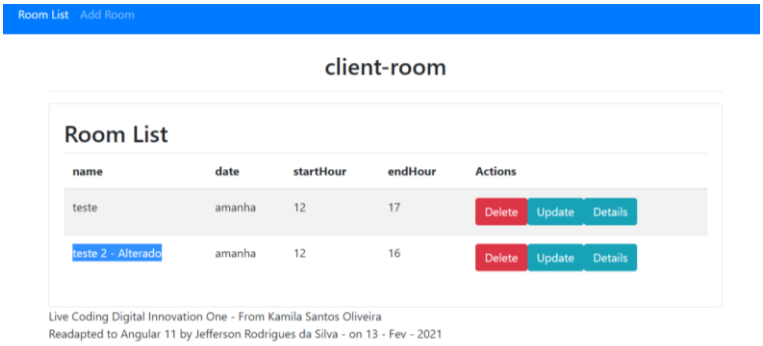




http://localhost:4200/update/2

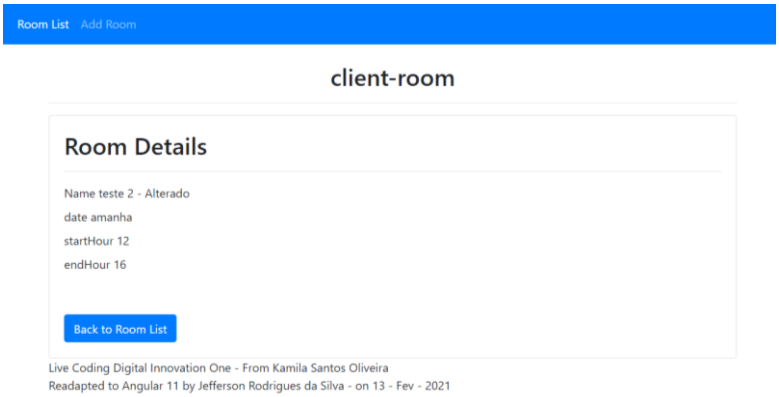
Na rota */update/2, conforme o **app.routing**, renderizamos o componente **UpdateRoomComponent**, e após **alterado** é retornado para a lista de rooms.





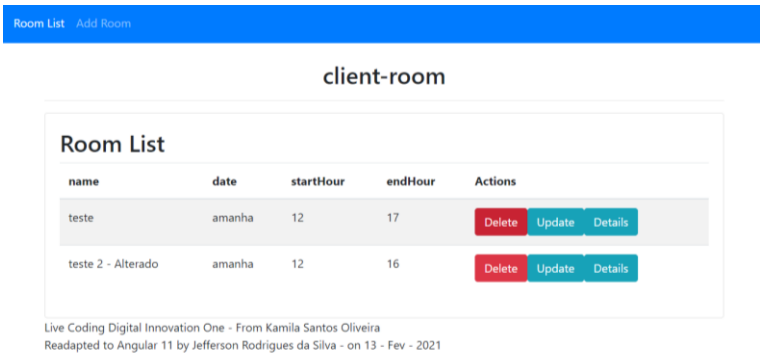
http://localhost:4200/details/2

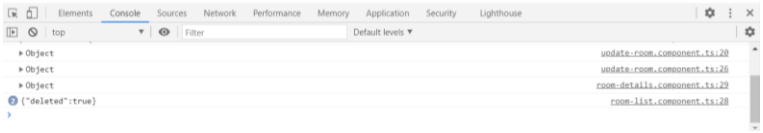
Na rota */details/2, conforme o **app.routing**, renderizamos o componente **RoomDetailsComponent**.



Delete/{id}

O end point do programa back-end java que deleta o objeto é chamado pelo front-end a partir do botão [Delete] no RoomListComponent, que ao ser clicado, o arquivo room-list.component.ts chama o seu service e que faz solicitação http ao http://localhost:8080/api/v1/rooms/1, usamos o número número **1** como exemplo, ele representa o parâmetro de idRoom, com o método delete que será enviado a aplicação Java, e recebe como resposta texto que informa se tal elemento foi deletado ou não.





Observação: Na aplicação iremos tirar essas notificações do console, e podemos usá-las como forma de mapear as respostas que chega a nós e exibir algum pop-up de notificação ao usuário.

Bibliografia

Curso Criar Gerenciador de Salas de Reuniões com Java e Angualr - Kamila Santos

Documentação Angular

Agradecimentos

Obrigado a Kamila Santos pela aula e a DioInnovation pela iniciativa.