ANGULAR

Installing Chocolatey Package Manager

Prerequisites
- Windows 7+ / Windows Server 2003+
- PowerShell v2+
- .NET Framework 4+
- Administrator access

Installation Steps

1. *Open PowerShell as Administrator*
   - Right-click on PowerShell
   - Select "Run as Administrator"

2. *Check Execution Policy*
   powershell
   Get-ExecutionPolicy

   If it returns "Restricted", run:
   powershell
   Set-ExecutionPolicy AllSigned

   or
   powershell
   Set-ExecutionPolicy Bypass -Scope Process

3. *Run Installation Command*
   Copy and paste this entire command:
   powershell
   Set-ExecutionPolicy Bypass -Scope Process -Force;
[System.Net.ServicePointManager]::SecurityProtocol =
[System.Net.ServicePointManager]::SecurityProtocol -bor 3072; iex ((New-Object
System.Net.WebClient).DownloadString('https://community.chocolatey.org/install.ps1'))

4. *Verify Installation*
   powershell
   choco --version

Basic Usage

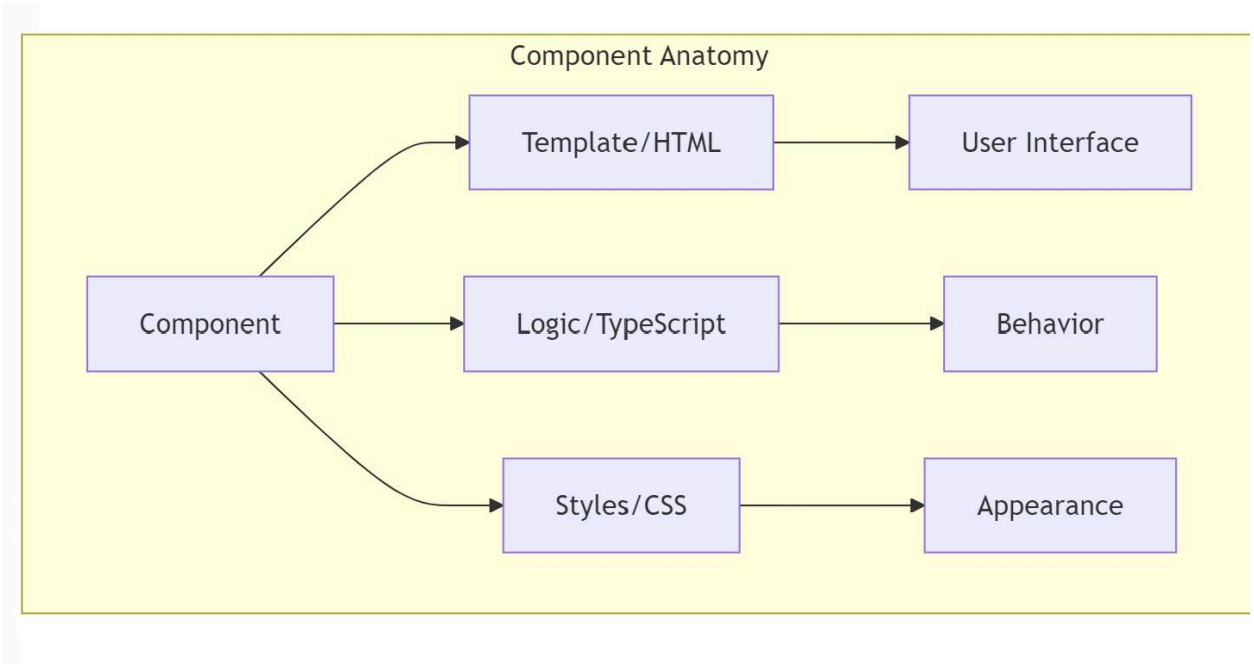
- Install a package: choco install packagename
- Update a package: choco upgrade packagename
- List installed packages: choco list --local-only
- Uninstall a package: choco uninstall packagename

Troubleshooting
- If you receive SSL/TLS errors, ensure you're using an updated version of PowerShell
- If you get access denied errors, make sure you're running PowerShell as Administrator
- For any errors, check the logs at C:\ProgramData\chocolatey\logs\chocolatey.log

Id CommandLine
-- -----------
1 tcs -v
2 npm install -g typescript
3 tsc -v
4 Set-ExecutionPolicy AllSigned
5 Set-ExecutionPolicy Bypass -Scope Process
6 tsc -v
7 npm install -g @angular/cli
8 npm fund
9 ng --version
10 cd D:\CodePython\
11 mkdir angular
12 cdangular
13 cd angular
14 ng new myfirstproject
15 cd D:\CodePython\angular\myfirstproject
16 ng serve -o

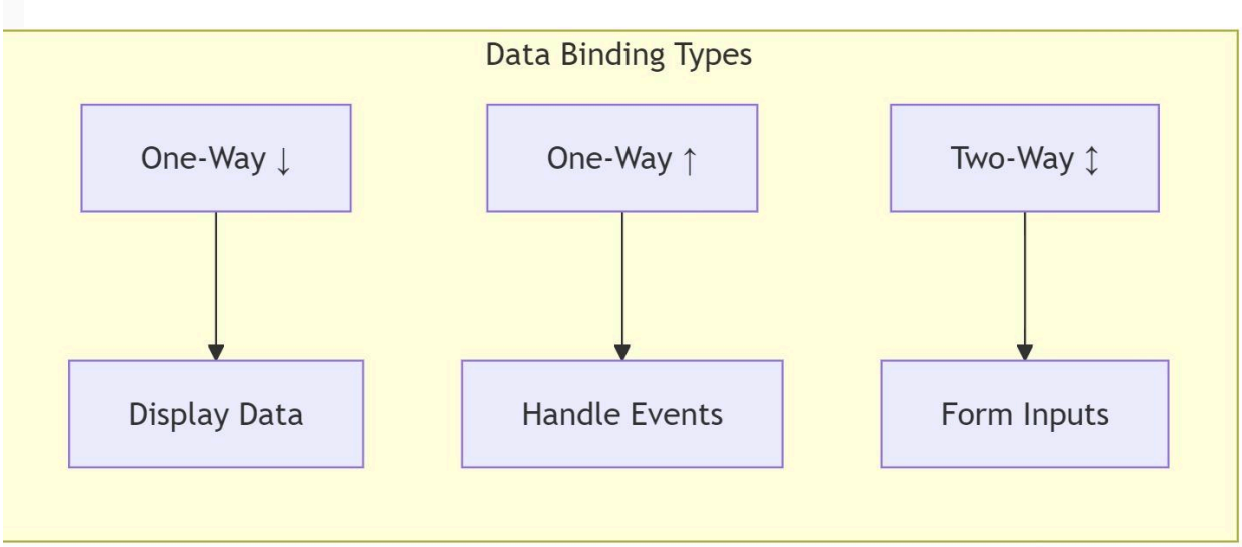
Key Project Files:

angular.json: Project configuration
package.json: Dependencies
tsconfig.json: TypeScript settings
src/: Source code directory
src/app/: Application code



Detailed Explanation:
Think of a Component like a TV set:

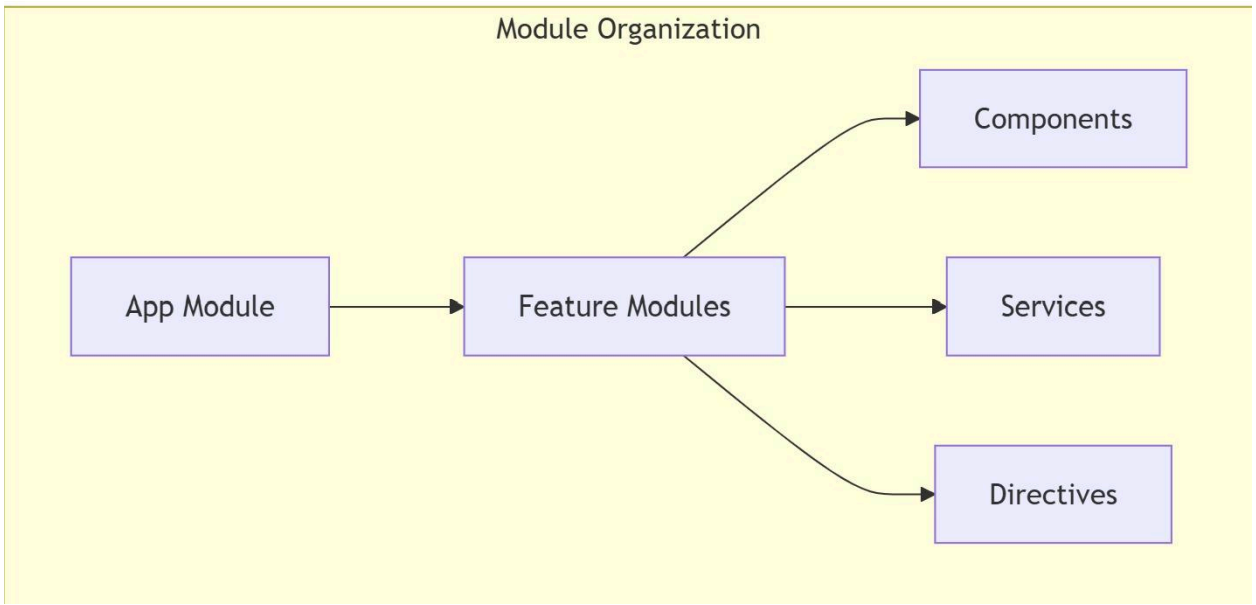
The screen (Template) is what users see
The circuit board (Logic) controls how it works
The outer case (Styles) determines how it looks




```
// Like a restaurant menu board showing prices
@Component({
  template: `
    <h1>Today's Special: {{dishName}}</h1>
    <p>Price: ${{price}}</p>
```

```
  `
})
class MenuComponent {
  dishName = "Pizza";
  price = 10;
}

// Like pressing a button to call waiter
@Component({
  template: `
    <button (click)="callWaiter()">Need Help</button>
  `
})
class TableComponent {
  callWaiter() {
    console.log("Waiter called!");
  }
}

// Hotel Service Example
@Injectable({
  providedIn: 'root'
})
class HotelService {
  // Shared resources (like cleaning supplies)
  private rooms = [];

  // Shared functions (like cleaning procedures)
  cleanRoom(roomNumber: number) {
    console.log(Cleaning room ${roomNumber});
  }

  // Data management (like room status)
  getRoomStatus(roomNumber: number) {
    return this.rooms[roomNumber];
  }
}

// Using the service in a component (like a floor manager)
@Component({})
class FloorComponent {
  constructor(private hotelService: HotelService) {
    // Can now use cleaning service
    this.hotelService.cleanRoom(101);
```

```
  }
}

// Like instructions: "If box is heavy, get help"
@Component({
  template: `
    <div *ngIf="isHeavy">
      Please get assistance!
    </div>

    <!-- Like "Repeat for each screw" -->
    <div *ngFor="let item of parts">
      Install {{item}}
    </div>
  `
})

// Like "Paint this part red if it's important"
@Component({
  template: `
    <div [ngStyle]="{'color': isImportant ? 'red' : 'black'}">
      Important Note
    </div>
  `
})
```



6 Set-ExecutionPolicy AllSigned

```
   7 ng new taskmanagement
   8 Set-ExecutionPolicy Bypass -Scope Process
   9 Set-ExecutionPolicy Bypass -Scope Process
  10 ng new taskmanagement
  11 ls -lrt
  12 ls
  13 cd .\taskmanagement\
  14 ng serve -o
  15 ls
  16 ng generate component components/task-list
  17 ng generate component components/task-form
  18 ng generate component components/task-item
  19 ng generate service services/task
```

```
export interface Task {
    id: number;
    title: string;
    description: string;
    completed: boolean;
    createdAt: Date;
}
```

```
import { Injectable } from '@angular/core';
import { Task } from '../models/task.interface';
import { BehaviorSubject,Observable } from 'rxjs';
@Injectable({
  providedIn: 'root'
})
export class TaskService {
  private tasks: Task[] = [];
  private tasksSubject: BehaviorSubject<Task[]> = new BehaviorSubject<Task[]>(this.tasks);
  // What is observable and why we use it here?
  // Observable is a stream of data that can be observed by the component.
  // We use it here to observe the changes in the tasks array.
  // BehaviorSubject is a type of observable that emits the current value of the tasks array.
  getTasks(): Observable<Task[]> {
    return this.tasksSubject.asObservable();
  }
  constructor() { }
}
addTask(task: Omit<Task, 'id' | 'createdAt'>): void {
    const newTask: Task = {  // Create a new task object
      ...task,  // Spread operator to copy the task object
      id: Date.now(),  // Generate a unique id using the current timestamp
```

```
    createdAt: new Date(),  // Set the createdAt date to the current date and time
   };
   this.tasks = [...this.tasks, newTask]; // Add the new task to the tasks array
   this.tasksSubject.next(this.tasks); // Emit the new tasks array to the observers
  }

import { Component } from '@angular/core';
import { TaskService } from '../../services/task.service';
import { FormsModule } from '@angular/forms';
import { NgModel } from '@angular/forms';
import { CommonModule } from '@angular/common';
@Component({
  selector: 'app-task-form',
  standalone: true,
  imports: [FormsModule, CommonModule],
  template: `<div class="task-form">
   <h2>Add Task</h2>
   <form (ngSubmit)="onSubmit()"  #taskForm="ngForm">
    <div class="form-group">
    <input type="text" [(ngModel)]="title" name="title" placeholder="Task Title" required
class="form-control">

    </div>
    <div class="form-group">
     <textarea [(ngModel)]="description" name="description" placeholder="Task Description"
required class="form-control">
     </textarea>
    </div>

    <button type="submit" [disabled]="!taskForm.invalid">Add Task</button>

   </form>
  </div>`,
  styles: [`
   .task-form {
    max-width: 500px;
    margin: 20px auto;
    padding: 20px;
   }
   .form-group {
    margin-bottom: 15px;
   }
   .form-control {
    width: 100%;
```

```
      padding: 10px;
      font-size: 16px;
    }
    button {
      background-color: #007bff;
      color: white;
      padding: 10px 20px;
      border: none;
      cursor: pointer;
    }
    button:disabled {
      background-color: #ccc;
      cursor: not-allowed;
    }
  `]
})
//Detail comment about html code written in the template section
//The template section contains a form with two input fields for the task title and description, and a submit button.
//The form is bound to the component's title and description properties using Angular's two-way data binding syntax.
//The form also uses Angular's reactive forms syntax to create a form object that can be used to validate the form data.
//The form is given a name of "taskForm" using the #taskForm syntax, which allows us to access the form object in the component's code.
//The submit button is disabled if the form is invalid, which is determined by the !taskForm.invalid expression.

export class TaskFormComponent {
  title: string = '';
  description: string = '';

  constructor(private taskService: TaskService) {}
  onSubmit():void {
   if (this.title.trim()){
     this.taskService
     .addTask({title: this.title, description: this.description,completed: false});
     this.title = '';
     this.description = '';
     //Detail comment about the class taskform component
     //The taskform component is a simple form that allows the user to add a new task to the task list.
     //The form has two input fields for the task title and description, and a submit button.
```

```
    //The form is bound to the component's title and description properties using Angular's two-way data binding syntax.
    //The form also uses Angular's reactive forms syntax to create a form object that can be used to validate the form data.
    //The form is given a name of "taskForm" using the #taskForm syntax, which allows us to access the form object in the component's code.
    //The submit button is disabled if the form is invalid, which is determined by the !taskForm.invalid expression.
   }
  }
}


import { Component, Input } from '@angular/core';
import { Task } from '../../models/task.interface';
import { TaskService } from '../../services/task.service';
import { FormsModule } from '@angular/forms';
import { DatePipe } from '@angular/common';
@Component({
  selector: 'app-task-item',
  standalone: true,
  imports: [FormsModule,DatePipe],
  template: `<div class="task-item" [class.completed]="task.completed">
    <div class="task-content">
      <input type="checkbox" [(ngModel)]="task.completed" (change)="onToggle(task)">
      <div class="task-text">
        <h3>{{ task.title }}</h3>
        <p>{{ task.description }}</p>
        <small>{{ task.createdAt | date:'short' }}</small>
      </div>
    </div>
    <button (click)="onDelete(task)" class="delete-btn">Delete</button>
  </div>`,
  styles: [`
    .task-item {
      display: flex;
      justify-content: space-between;
      align-items: center;
      padding: 10px;
      border-bottom: 1px solid #ccc;
      background-color: #f0f0f0;
    }
    .task-content {
      display: flex;
```

```
      align-items: center;
      gap: 10px;
    }
    .completed {
      background-color: #e0e0e0;
      .task-text {
        opacity: 0.5;
        text-decoration: line-through;
        color: #888;
      }
    }
    .delete-btn {
      background-color: #ff4500;
      color: #fff;
      border: none;
      padding: 5px 10px;
      cursor: pointer;
    }
  `]
})
export class TaskItemComponent {
  @Input() task!: Task;

  constructor(private taskService: TaskService) {}

  onToggle(task: Task) {
    this.taskService.toggleTask(this.task.id);
  }

  onDelete(task: Task) {
    this.taskService.deleteTask(this.task.id);
  }
}


import { Component, Input } from '@angular/core';
import { Task } from '../../models/task.interface';
import { TaskService } from '../../services/task.service';
import { FormsModule } from '@angular/forms';
import { DatePipe } from '@angular/common';
// Write detail line by line comment about the class task-item component
//The task-item component is a simple component that displays a task item in the task list.
//The component has an input property called task, which is used to pass the task data to the
component.
```

```
//The component also has two methods: onToggle and onDelete, which are used to toggle the task completion status and delete the task respectively.
//The component uses Angular's built-in date pipe to format the createdAt date.
//The component also uses Angular's built-in ngModel directive to bind the task completion status to the input checkbox.
//The component also uses Angular's built-in ngClass directive to conditionally apply the completed class to the task item.
@Component({
  selector: 'app-task-item',
  standalone: true,
  imports: [FormsModule,DatePipe],
  template: `<div class="task-item" [class.completed]="task.completed">
    <div class="task-content">
      <input type="checkbox" [(ngModel)]="task.completed" (change)="onToggle(task)">
      <div class="task-text">
        <h3>{{ task.title }}</h3>
        <p>{{ task.description }}</p>
        <small>{{ task.createdAt | date:'short' }}</small>
      </div>
    </div>
    <button (click)="onDelete(task)" class="delete-btn">Delete</button>
  </div>`,
  styles: [`
    .task-item {
      display: flex;
      justify-content: space-between;
      align-items: center;
      padding: 10px;
      border-bottom: 1px solid #ccc;
      background-color: #f0f0f0;
    }
    .task-content {
      display: flex;
      align-items: center;
      gap: 10px;
    }
    .completed {
      background-color: #e0e0e0;
      .task-text {
        opacity: 0.5;
        text-decoration: line-through;
        color: #888;
      }
    }
```

```
    .delete-btn {
      background-color: #ff4500;
      color: #fff;
      border: none;
      padding: 5px 10px;
      cursor: pointer;
    }
  `]
})
export class TaskItemComponent {
  // Detail comment about the @Input() task property
  //The @Input() task property is used to receive the task data from the parent component.
  //The task property is decorated with the @Input() decorator, which makes it an input property.
  //The task property is used to receive the task data from the parent component.
  @Input() task!: Task;

  constructor(private taskService: TaskService) {}

  onToggle(task: Task) {
    this.taskService.toggleTask(this.task.id);
  }

  onDelete(task: Task) {
    this.taskService.deleteTask(this.task.id);
  }
}

// Importing required Angular core modules and interfaces
import { Component, OnInit } from '@angular/core';
// Observable for handling asynchronous data streams
import { Observable } from 'rxjs';
// Task interface that defines the structure of a task object
import { Task } from '../../models/task.interface';
// Service that handles task-related operations
import { TaskService } from '../../services/task.service';

@Component({
  // Component's selector used in other templates as <app-task-list>
  selector: 'app-task-list',

  // Template definition using template literal syntax
  template: `
    <!-- Main container for the task list -->
    <div class="task-list">
```

```
    <!-- Form component for adding new tasks -->
    <app-task-form></app-task-form>

    <!-- Container for displaying tasks -->
    <div class="tasks">
      <h3>My Tasks</h3>

      <!-- Using ng-container with async pipe to handle Observable -->
      <!-- 'tasks$ | async as tasks' subscribes to the Observable and assigns the value to 'tasks'
-->
      <ng-container *ngIf="tasks$ | async as tasks">
        <!-- Iterate over tasks array using ngFor -->
        <!-- Create a task-item component for each task -->
        <app-task-item
          *ngFor="let task of tasks"
          [task]="task"  <!-- Pass task data to child component -->
        ></app-task-item>

        <!-- Show message when no tasks exist -->
        <p *ngIf="tasks.length === 0" class="no-tasks">
          No tasks yet! Add some tasks above.
        </p>
      </ng-container>
    </div>
  </div>
  `,

  // Component-specific styles
  styles: [`
    /* Center the task list and set maximum width */
    .task-list {
      max-width: 800px;
      margin: 0 auto;
      padding: 20px;
    }

    /* Add spacing between form and task list */
    .tasks {
      margin-top: 20px;
    }

    /* Style for empty state message */
    .no-tasks {
      text-align: center;
```

```
      color: #6c757d;  /* Grey color for secondary text */
      margin-top: 20px;
    }
  `]
})

// Component class implementation
export class TaskListComponent implements OnInit {
  // Observable that will hold the stream of tasks
  // '!' is the non-null assertion operator indicating it will be initialized
  tasks$!: Observable<Task[]>;

  // Inject TaskService through constructor
  constructor(private taskService: TaskService) {}

  // Lifecycle hook that runs when component initializes
  ngOnInit(): void {
    // Get tasks Observable from service
    this.tasks$ = this.taskService.getTasks();
  }
}

import { Component } from '@angular/core';
import { RouterOutlet } from '@angular/router';

@Component({
  selector: 'app-root',
  standalone: true,
  template: <h1>Hello World</h1>
})
export class AppComponent {
  title = 'angularfirst';
}

import { Component } from '@angular/core';
import { RouterOutlet } from '@angular/router';

@Component({
  selector: 'app-root',
  standalone: true,
  template: <div class=profile-card><h2>{{title}} </h2> <p>{{bio}}</p></div>,
  styles: [`
    .profile-card {
      background-color: #f0f0f0;
```

```
      padding: 20px;
      border-radius: 10px;
    }
  `]
})
export class AppComponent {
  title = 'angularfirst';
  bio = 'I am a software engineer';
}


import { Component } from '@angular/core';
import { CommonModule } from '@angular/common';
import { RouterOutlet } from '@angular/router';

@Component({
  selector: 'app-root',
  standalone: true,
  imports: [CommonModule],
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {
  title = 'angularfirst';
  bio = 'I am a software engineer';
  tasks = ['task1', 'task2', 'task3'];
}


<div class=""task-container>
  <h3>Tasks</h3>
  <ul>
    <li *ngFor="let task of tasks">{{tasks}}</li>
  </ul>
</div>


.task-container {
  background-color: #f0f0f0;
  padding: 20px;
  border-radius: 10px;
  margin: 20px;
}
```

```
.task-container ul {
  list-style-type: none;
  padding: 0;
}

.task-container li {
  margin-bottom: 10px;
  border-bottom: 1px solid #ccc;
}


import { Component } from '@angular/core';
import { CommonModule } from '@angular/common';
import { ChildComponent } from './child.component';
import { RouterOutlet } from '@angular/router';

@Component({
  selector: 'app-root',
  standalone: true,
  imports: [CommonModule,ChildComponent],
  templateUrl: './app.component.html'
})
export class AppComponent {
  parentMessage = 'Jinesh is trying to teach Angular';

  handleResponse(response:string){
    this.parentMessage = response;
  }
}


import { Component,Input,Output,EventEmitter } from '@angular/core';

@Component({
  selector: 'app-child',
  standalone: true,
  template: <div><h2>{{message}}</h2> <button (click)="sendResponse()">Send Response to
parent</button></div>
})
export class ChildComponent {
  @Input() message: string = '';
  @Output() response = new EventEmitter<string>();

  sendResponse(){
```

```
    this.response.emit('Response from child class');
  }
}

<div>
  <h2>Parent Component</h2>
  <app-child [message]="parentMessage" (response)="handleResponse($event)">
  </app-child>
</div>



PS C:\WINDOWS\system32> cd D:\CodePython\
PS D:\CodePython> cd angular
PS D:\CodePython\angular> cd basicangular
PS D:\CodePython\angular\basicangular> cd .\angularfirst\
PS D:\CodePython\angular\basicangular\angularfirst> ng serve -o
Initial chunk files | Names | Raw size
polyfills.js | polyfills | 90.20 kB |
 main.js | main | 3.75 kB |
 styles.css | styles | 95 bytes |
            | Initial total | 94.05 kB
Application bundle generation complete. [14.284 seconds]
Watch mode enabled. Watching for file changes...
NOTE: Raw file sizes do not reflect development server per-request transformations.

➜ Local: http://localhost:4200/
➜ press h + enter to show help
```

Component Structure: The @Component decorator defines the component's metadata standalone: true makes it a standalone component selector defines how to use the component in HTML () template contains the component's HTML structure styles contains component-specific CSS

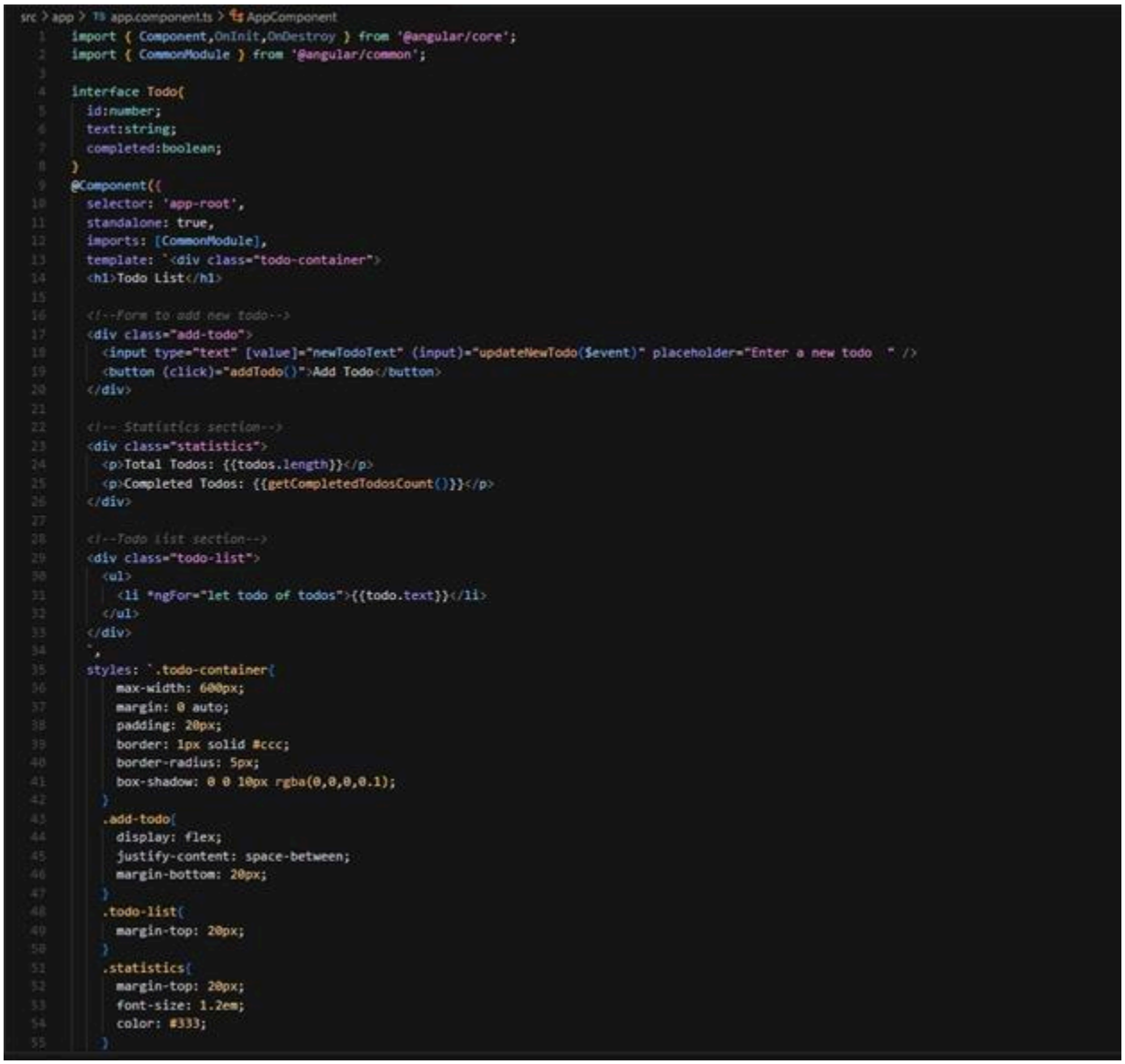
State Management: Component state is managed through class properties (todos, newTodoText, nextId) TypeScript interface (Todo) ensures type safety State is modified through class methods

Template Features:
Data binding:
{{ }} for displaying values
[] for property binding
() for event binding

@for directive for iterating over lists

Conditional styling with [class.completed]
User Interaction: Input handling with event binding
Click handlers for buttons
Checkbox state management

TO DO APPLICATION



```
src > app > TS app.component.ts > AppComponent
import { Component,OnInit,OnDestroy } from '@angular/core';
import { CommonModule } from '@angular/common';

interface Todo{
  id:number;
  text:string;
  completed:boolean;
}
@Component({
  selector: 'app-root',
  standalone: true,
  imports: [CommonModule],
  template: `<div class="todo-container">
  <h1>Todo List</h1>

  <!--Form to add new todo-->
  <div class="add-todo">
    <input type="text" [value]="newTodoText" (input)="updateNewTodo($event)" placeholder="Enter a new todo " />
    <button (click)="addTodo()">Add Todo</button>
  </div>

  <!-- Statistics section-->
  <div class="statistics">
    <p>Total Todos: {{todos.length}}</p>
    <p>Completed Todos: {{getCompletedTodosCount()}}</p>
  </div>

  <!--Todo list section-->
  <div class="todo-list">
    <ul>
      <li *ngFor="let todo of todos">{{todo.text}}</li>
    </ul>
  </div>
  `,
  styles: `.todo-container{
      max-width: 600px;
      margin: 0 auto;
      padding: 20px;
      border: 1px solid #ccc;
      border-radius: 5px;
      box-shadow: 0 0 10px rgba(0,0,0,0.1);
    }
    .add-todo{
      display: flex;
      justify-content: space-between;
      margin-bottom: 20px;
    }
    .todo-list{
      margin-top: 20px;
    }
    .statistics{
      margin-top: 20px;
      font-size: 1.2em;
      color: #333;
    }
```

```
    .todo-list li{
      margin-bottom: 10px;
      padding: 10px;
      border: 1px solid #ccc;
      border-radius: 5px;
    }
    .todo-list li.completed{
      background-color: #f0f0f0;
      color: #888;
    }`
})
export class AppComponent {


  todos:Todo[] = [
    {id:1,text:"Learn Angular",completed:false},
    {id:2,text:"Build a project",completed:false},
    {id:3,text:"Deploy the project",completed:false}
  ];

  newTodoText:string = '';
  nextId:number = 4;

  updateNewTodo(event:Event):void{
    const input =event.target as HTMLInputElement;
    this.newTodoText = input.value;
  }

  addTodo():void{
    if(this.newTodoText.trim() === ''){
      return;
    }
    this.todos.push({id:this.nextId++,text:this.newTodoText,completed:false});
    this.newTodoText = '';
  }


  getCompletedTodosCount():number{
    return this.todos.filter(todo => todo.completed).length;
  }
}
```

FULL STACK WEB APPLICATION WITH ANGULAR FRONTEND, SPRING BOOT BACK END AND MY SQL DATABASE FOR EMPLOYEE MANAGEMENT SYSTEM.

// springboot backend

//application.properties

```
spring.application.name=springboot-backend
spring.datasource.url=jdbc:mysql://localhost:3306/employee_management_syst
em?useSSL=false
spring.datasource.username=root
spring.datasource.password=root
spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.MySQLDialect
```

```
spring.jpa.hibernate.ddl-auto=update
```

// spring boot back end application.java

```
package net.javaguides.springboot;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class SpringbootBackendApplication {

    public static void main(String[] args) {
        SpringApplication.run(SpringbootBackendApplication.class, args);
    }

}
```

// Employee model.java

```
package net.javaguides.springboot.model;

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

// JPA (java persistence API) it a specification of java that is used to
persist data between java objects and
// relational database , it acts as a bridge between object oriented
domain model and relational database systems

@Entity  // to specify that this class is a JPA entity
@Table(name="employees") // to specify the name of the table that this
entity should be mapped to
public class Employee {
    @Id // used to specify a class memeber as an unique identifier for an
entity in database
```

```
    @GeneratedValue(strategy = GenerationType.IDENTITY) //automatically
generate id's as primary key
    private long id;

    @Column(name="first_name") //used to add columns in our database
    private String firstName;
    @Column(name="last_name")
    private String lastName;
    @Column(name="email_id")
    private String email;

    // default contructor because hibernate internally uses proxies to
create proxy objects
    public Employee() {
    }
    public Employee(String firstName,String lastName, String email){
        this.firstName=firstName;
        this.lastName=lastName;
        this.email=email;
    }
    // getters and setters methods
    public void setId(long id){
        this.id=id;
    }
    public long getId(){
        return id;
    }

    public void setFirstName(String firstName){
        this.firstName=firstName;
    }
    public String getFirstName(){
        return firstName;
    }

    public void setLastName(String lastName){
        this.lastName=lastName;
    }
    public String getLastName(){
        return lastName;
```

```
    }

    public void setEmail(String email){
        this.email=email;
    }
    public String getEmail(){
        return email;
    }
}
```

//Employee repository.java

```
package net.javaguides.springboot.repository;

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

import org.springframework.stereotype.Repository;

import net.javaguides.springboot.model.Employee;
@Repository

// jpa repository basically provides us with all the methods required to
perform crud operations on our data base
// like insert, delete, update and read, so we do not need to write the
methods and queries by ourself
public interface EmployeeRepository extends JpaRepository<Employee, Long>{
// here we are passing our jpa entity
    //that we created and the type of our primary key which is long


}
```

//ResourceNotFoundException.java

```
package net.javaguides.springboot.exception;

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

```
@ResponseStatus(value = HttpStatus.NOT_FOUND)
public class ResourceNotFoundException extends RuntimeException {
    private static final long serialVersionUID=1L;
    public ResourceNotFoundException(String message){
        super(message);
    }

}
```

//EmployeeController.java

```
package net.javaguides.springboot.controller;

import java.util.List;

import org.springframework.beans.factory.annotation.Autowired;
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.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

import net.javaguides.springboot.model.Employee;
import net.javaguides.springboot.repository.EmployeeRepository;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestBody;



@CrossOrigin(origins = "http://localhost:4200")
@RestController
@RequestMapping("/api/v1/")
public class EmployeeController {

    @Autowired
    private EmployeeRepository employeeRepository;
```

```
    //get all employees
    @GetMapping("/employees")
    public List<Employee> getAllEmployees() {
        return employeeRepository.findAll();
    }

    // create employee
    @PostMapping("/add")
    public void addEmployee(@RequestBody Employee employee) { // to map
request json into java object
        employeeRepository.save(employee);
    }

    //get employee by id
    @GetMapping("/update/{id}")
    public Employee getEmployeeById(@PathVariable Long id) { // to access
the variable present in path
        return employeeRepository.findById(id).orElse(null);
    }

    //update employee by id
    /*
     * understand the working on putMapping()
     * and also the ResponseEntity<Employee> return type of methods that
returns
     * ResponseEntity.ok(Employee)
     }

     */
    @PostMapping("/update/{id}")
    public void updateEmployee(@PathVariable Long id, @RequestBody
Employee employee) {
        Employee existingEmployee =
employeeRepository.findById(id).orElse(null);
        if (existingEmployee != null) {
            existingEmployee.setFirstName(employee.getFirstName());
            existingEmployee.setLastName(employee.getLastName());
            existingEmployee.setEmail(employee.getEmail());
            employeeRepository.save(existingEmployee);
        }else{
```

```
            employeeRepository.save(employee);
        }
    }

    // delete employee
    @DeleteMapping("/delete/{id}")
    public void deleteEmployee(@PathVariable long id){
        employeeRepository.deleteById(id);
    }
}
```

//Angular front end for employee management

//app.config.ts

```
import { ApplicationConfig, provideZoneChangeDetection } from
'@angular/core';
import { provideRouter } from '@angular/router';

import { routes } from './app.routes';
import { provideClientHydration } from '@angular/platform-browser';
import { provideHttpClient, withFetch } from '@angular/common/http';


export const appConfig: ApplicationConfig = {
  providers: [provideZoneChangeDetection({ eventCoalescing: true }),
provideRouter(routes),
provideClientHydration(),provideHttpClient(withFetch())]
};
```

//app.routes.ts

```
import { Routes } from '@angular/router';
import { EmployeeListComponent } from
'./myComponent/employee-list/employee-list.component';
import { AddEmployeeComponent } from
'./myComponent/add-employee/add-employee.component';
import { UpdateEmployeeComponent } from
'./myComponent/update-employee/update-employee.component';
```

```
import { DeleteEmployeeComponent } from
'./myComponent/delete-employee/delete-employee.component';

export const routes: Routes = [
    {path: 'employees', component: EmployeeListComponent},
    {path: '', redirectTo: '/employees', pathMatch: 'full'},
    {path: 'add', component: AddEmployeeComponent},
    {path: 'update/:id', component: UpdateEmployeeComponent},
    {path: 'delete/:id', component: DeleteEmployeeComponent}
];
```

//employee.ts

```
export class Employees {
    id!: number
    firstName!: string
    lastName!: string
    email!: string

    constructor(){
    }
}
```

//app.component.html

```
<nav class="navbar navbar-expand-lg navbar-light bg-light">
  <a class="navbar-brand" routerLink="/employees"
routerLinkActive="active">Employee List</a>
  <button class="navbar-toggler" type="button" data-toggle="collapse"
data-target="#navbarNav" aria-controls="navbarNav" aria-expanded="false"
aria-label="Toggle navigation">
    <span class="navbar-toggler-icon"></span>
  </button>
  <div class="collapse navbar-collapse" id="navbarNav">
    <ul class="navbar-nav">
      <li class="nav-item active">
        <a class="nav-link" routerLink="/add"
routerLinkActive="active">Add Employee</a>
```

```
      </li>
    </ul>
  </div>
</nav>
<router-outlet></router-outlet>
<footer class="bg-body-tertiary text-center text-lg-start">
  <!-- Copyright -->
  <div class="text-center p-3">
    © 2024 Copyright:
    <a class="text-body"
href="https://www.linkedin.com/in/ayush-arya-0500281b2/">Ayush Arya</a>
  </div>
  <!-- Copyright -->
</footer>
```

//app.component.ts

```
import { Component } from '@angular/core';
import { RouterOutlet } from '@angular/router';
import { EmployeeListComponent } from
"./myComponent/employee-list/employee-list.component";
import { CommonModule } from '@angular/common';

import { RouterLink } from '@angular/router';

@Component({
  selector: 'app-root',
  standalone: true,
  imports: [RouterOutlet, EmployeeListComponent,CommonModule,RouterLink],
  templateUrl: './app.component.html',
  styleUrl: './app.component.css'
})
export class AppComponent {
  title = 'angular-frontend';
}
```

//employee-list.component.html

```
<div class="container">
```

```
    <h1 class="center">Employee list</h1>
    <table class="table table-striped">
        <thead>
            <tr>
                <th>First Name</th>
                <th>Last Name</th>
                <th>Email</th>
                <th>Update</th>
                <th>Delete</th>
            </tr>
        </thead>
        <tbody>
            <tr *ngFor="let employee of employees">
                <td>{{employee.firstName}}</td>
                <td>{{employee.lastName}}</td>
                <td>{{employee.email}}</td>
                <td>
                    <button (click)="onUpdate(employee.id)" class="btn
btn-info">Update</button>
                </td>
                <td>
                    <button (click)="onDelete(employee.id)" class="btn
btn-danger">Delete</button>
                </td>
            </tr>
        </tbody>
    </table>
</div>
```

//employee-list.component.ts

```
import { CommonModule } from '@angular/common';
import { Component } from '@angular/core';
import { Employees } from '../../employee';
import { EmpService } from '../myService/emp.service';
import { NavigationEnd, Router } from '@angular/router';

@Component({
  selector: 'app-employee-list',
```

```
  standalone: true,
  imports: [CommonModule],
  templateUrl: './employee-list.component.html',
  styleUrl: './employee-list.component.css'
})
export class EmployeeListComponent {
  employees!:Employees[];
  constructor(private empservice: EmpService, private router: Router){
    this.empservice.getEmployeesList().subscribe(data => {
      this.employees = data;
    });
  }
  ngOnInit() {
    this.router.events.subscribe((event) => {
      if (event instanceof NavigationEnd && event.url === '/employees') {
        this.empservice.getEmployeesList().subscribe(data => {
          this.employees = data;
        });
      }
    });
  }

  onUpdate(id: number){
    this.router.navigate(['/update', id]);
  }

  onDelete(id:number){
    this.router.navigate(['/delete', id]);

  }
}
```

//add-employee.component.html

```
<div class="container">
    <h1 class="center">Add Employee</h1>
  <form (ngSubmit)="onSubmit()">
      <div class="form-group">
        <p>First Name</p>
```

```
        <input type="text" class="form-control"
[(ngModel)]="employee.firstName" name="firstName" id="firstName"
placeholder="Enter FirstName">
      </div>
      <div class="form-group">
        <p>Last Name</p>
        <input type="text" class="form-control"
[(ngModel)]="employee.lastName" name="lastName" id="lastName"
placeholder="Enter LastName">
      </div>
      <div class="form-group">
          <p>Email address</p>
          <input type="email" class="form-control"
[(ngModel)]="employee.email" name="email" id="email"placeholder="Enter
email">
        </div>
      <button type="submit" class="btn btn-primary">Submit</button>
  </form>
  </div>
```

//add-employee.component.ts

```
import { Component } from '@angular/core';
import { CommonModule } from '@angular/common';
import { FormsModule } from '@angular/forms'
import { Employees } from '../../employee';
import { EmpService } from '../myService/emp.service';
import { Router } from '@angular/router';

@Component({
  selector: 'app-add-employee',
  standalone: true,
  imports: [CommonModule,FormsModule],
  templateUrl: './add-employee.component.html',
  styleUrl: './add-employee.component.css'
})
export class AddEmployeeComponent {
  constructor(private empservice: EmpService, private router: Router){}
  employee: Employees= new Employees();
```

```
  onSubmit(){
    this.empservice.addEmployee(this.employee).subscribe(data=>
      console.log(data)
    ),
    (error: any) => console.error(error);
    this.goToEmployeeList();
  }

  goToEmployeeList(){
    this.router.navigate(['/employees']);
  }

}
```

//update-employee.component.html

```
<div class="container">
    <h1 class="center">Update Employee</h1>
  <form (ngSubmit)="onSubmit()">
      <div class="form-group">
        <p>First Name</p>
        <input type="text" class="form-control"
[(ngModel)]="employee.firstName" name="firstName" id="firstName"
placeholder="Enter FirstName">
      </div>
      <div class="form-group">
        <p>Last Name</p>
        <input type="text" class="form-control"
[(ngModel)]="employee.lastName" name="lastName" id="lastName"
placeholder="Enter LastName">
      </div>
      <div class="form-group">
          <p>Email address</p>
          <input type="email" class="form-control"
[(ngModel)]="employee.email" name="email" id="email"placeholder="Enter
email">
        </div>
      <button type="submit" class="btn btn-primary">Submit</button>
  </form>
  </div>
```

//update-employee.component.ts

```
import { Component } from '@angular/core';
import { Employees } from '../../employee';
import { FormsModule } from '@angular/forms';
import { EmpService } from '../myService/emp.service';
import { ActivatedRoute } from '@angular/router';
import { Router } from '@angular/router';

@Component({
  selector: 'app-update-employee',
  standalone: true,
  imports: [FormsModule],
  templateUrl: './update-employee.component.html',
  styleUrl: './update-employee.component.css'
})
export class UpdateEmployeeComponent {
  employee: Employees=new Employees();
  id!:number;
  constructor(private empService: EmpService, private activatedRoute:
ActivatedRoute, private router: Router){}
  ngOnInit(): void {
    this.id=this.activatedRoute.snapshot.params['id'];
    this.empService.getEmployeeById(this.id).subscribe(data=>
      this.employee=data
    );
  }
  onSubmit(){
    this.empService.updateEmployee(this.id, this.employee).subscribe(data
=>
      console.log(data)
    );
    this.router.navigate(['/employees']);
  }

}
```

//delete-employee.component.html

```
<div class="container">
```

```
    <h1>Delete Employee</h1>
    <table class="table table-striped">
        <thead>
            <th>FirstName</th>
            <th>LastName</th>
            <th>email</th>
        </thead>
        <tbody>
            <td>{{employee.firstName}}</td>
            <td>{{employee.lastName}}</td>
            <td>{{employee.email}}</td>
        </tbody>
    </table>
    <h3 class="mrg">The details of above employee will be permanently
deleted</h3>
    <button class="mrg btn btn-info" (click)="onCancel()">Cancel</button>
    <button class="mrg btn btn-danger"
(click)="onConfirm()">Confirm</button>
</div>
```

//delete-employee.component.ts

```
import { Component } from '@angular/core';
import { EmpService } from '../myService/emp.service';
import { ActivatedRoute } from '@angular/router';
import { Router } from '@angular/router';
import { Employees } from '../../employee';

@Component({
  selector: 'app-delete-employee',
  standalone: true,
  imports: [],
  templateUrl: './delete-employee.component.html',
  styleUrl: './delete-employee.component.css'
})
export class DeleteEmployeeComponent {
  id!:number;
  employee: Employees = new Employees();
  constructor(private router: Router, private empService:
EmpService,private activatedRoute: ActivatedRoute){
```

```
    this.id=this.activatedRoute.snapshot.params['id'];
    this.empService.getEmployeeById(this.id).subscribe(data =>{
      this.employee=data;
    })
  }

  onCancel(){
    this.router.navigate(['/employees']);
  }
  onConfirm(){
    this.empService.deleteEmployee(this.id).subscribe(data=>{
      console.log(data);
    });
    this.router.navigate(['/employees']);
  }
}
```

//employee.service.ts

```
import { Injectable } from '@angular/core';
import { HttpClient } from '@angular/common/http';
import { Observable, of, tap } from 'rxjs';
import { Employees } from '../../employee';

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

  employeeCache: Employees[]|null=null;

  getEmployeesList(): Observable<Employees[]>{
    return
this.httpClient.get<Employees[]>("http://localhost:8080/api/v1/employees")
  }

  addEmployee(employee: Employees): Observable<Object>{
    return
this.httpClient.post<Object>("http://localhost:8080/api/v1/add",
employee);
```

```
  }
  getEmployeeById(id:number): Observable<Employees>{
    return
this.httpClient.get<Employees>("http://localhost:8080/api/v1/update/"+
id);
  }
  updateEmployee(id:number, employee: Employees): Observable <object> {
    return
this.httpClient.post<Object>("http://localhost:8080/api/v1/update/"+id,emp
loyee);
  }
  deleteEmployee(id:number): Observable<object>{
    return
this.httpClient.delete<Object>("http://localhost:8080/api/v1/delete/"+id);
  }

  constructor(private httpClient: HttpClient) {}
}
```