REACT TRAINING

-----------------------------------------------------------

js is dynamically typed language

no data type need to put

same variable can m=be given different types of data

automatically inferred

restructured assignment--

let eid=101;

let ename="Vamsy";

let basicpay=45000;

let emp={eid, ename, basicpay}

console.log(emp);

setInterval(callback, duration);

|----duration---|callback()|----duration---|callback().....(until clearInterval)

setTimeout(callback, duration);

|----duration---|callback()

Promise p = new Promise((resolve, reject)=>{

//resolve is used to signal successful completion of the job

//reject is used to signal error some abruption of the job

});

p.then(okcallBack, errCallback)

p.then(okCallBack).catch(errCallBack)

async is used to convert a normal function into asynchronous function

async functions always return a Promise

const f1 = async () => {

//a very time consuming algorithm

return result;

}

await is used to wait for the completion of an async function and extract the value from the Promise

await can only be used inside a method that is marked as async

let x = f1(); //x refers to Promise

let y = await f1(); //y receives the result after completion of f1

Lab Setup

NodeJS Dev platform

VSCode IDE

Chrome Browser/Client

NodeJS supports execution of the development tools like building tool, testing tool, package/dependency management, etc

But the final app is executed in a browser

Working with NodeJS

npm as a build tool (node package manager) (package refers to one JS project)

To create a JS project on NodeJS--

md proj-name

cd proj-name

npm init -y

'package.json' is the build file for npm

npm i package-name --save

npm i package-name --dev-save

npm i -g package-name

npm uninstall package-name --save

'node\_modules' folder that houses all the downloaded dependencies

'npm i' will get back the 'node\_modules' folder along with all the required dependencies as per the list of dependencies in the package.json

Each JS file is a module, able to share resources with help of export and import

REACT JS--

It is a SPA (single page application) framework based on JavaScript

A spa is a web application that has only one html page (index.html)

The javascript attached to the application will generate html dynamically on the client itself and will update the content of index.html from time to time. Which means no html needs to be generated on the server or transfer from the server to the client or unload and reload of a page wont be needed.

ReactJS like many other SPA-frameworks also works based on html extendability

HTML extendability means that we can create our own html elements in RaectJS

HTML elements created in ReactJS are called ReactJS components

Creating ReactJS App--

Create a Node project

Install all the required react-scripts into the project

configure the react-scripts and build configs

(or)

npm i -g creat-react-app

creat-react-app app-name

(or)

npx create-react-app app-name

React Components--

JSX- JavaScript eXtended

JSX is a html embedded JavaScript

Function Components-

A function component is any function that returns an html-element or another react-component

example- const Header = () => (

<header>

<h2> HR Automation Portal </h2>

</header>

);

const Dashboard = () => (

<div>

<Header />

<h2> This is my dashboard </h2>

</div>

);

<Dashboard />

Class Components-

It is a JavaScript class that extends Component from 'react.js' module

React.Component provides-

(a)state

(b)setState()

(c)render

(d)and a few more lifecycle methods

example- import {Component} from 'react';

class EmployeesList extends Component {

constructor(){

super();

//state contains all the data needed to be managed by the component

this.state={

title:"My HR App";

};

}

render(){ //it is this method that gives the html dom of the component

return (

<div>

<h2>{this.state.title}</h2>

</div>

);

}

}

High Order Components-

It is a component that returns another component

JSX- JavaScript eXtended

JSX is a html embedded JavaScript

.js--

let myPara = document.createElement("p");

myPara.innerText = "Hello, this is a para";

let friends = ["Vamsy","Rahul","Sharma","Avinash"];

let myList = document.createElement("ol");

for(let i=0;i<friends.length;i++){

let myLi = document.createElement("li);

myLi.innerText = friends[i];

myList.append(myLi);

}

.jsx--

let myPara = <p>Hello, this is a para</p>

let friends = ["Vamsy","Rahul","Sharma","Avinash"];

let myList = (

<ol>

{friends.map(f=> <li>{f}</li> )}

</ol>

);

JSX Rules--

1. JSX is case sensitive

2. use all html in lower case only

3. any react components are expected to be initial capitals

4. attribute name should follow camel case

5. 'class' is not a valid attribute as it is a keyword in JavaScript so use 'className' instead

React Application Design--

index.html

|-body

|-div#root //nothing from this point to the top is altered

|-App //top level component, from this point to the bottom is what we control

|-Component1 |-InnerComponent1 |-MoreInnerComponent1

|-Component2

|-Component3

state, render(), setState() --

state is the field inherited from React.component and is used to hold all the data the component needs to work with

state whenever gets modified, will invoke the 'render()' and updates the html-content on the screen

state is immutable (means we cannot change any part of the state)

state is always replaced using 'setState()' method

'setState()' is going to accept full or partial state

Working with Forms--

Controlled Form Component-

Eact Input element of a form is attached to a 'ref' variable

a 'ref' variable is a variable created outside the component state

whenever the form is submitted, we have to collect data from all the 'ref' variables

'ref' variables are very costly on the memory and they actually have a diffferent purpose

Uncontrolled Form Component-

Each input element is attached to a field in the 'state'

as state is immutable, the input fields are not editable

we will use onChange event of the input field to receive the new value into the respective state field

when the form is submitted, the data is already inside the state

this ensures 'single source of truth' and less costly memory wise

ReactJS Props--

'props' stand for properties is used to ship the data from parent to child or vice-versa

a parent component can pass the data through attributes of a child component

and a child component can access that data through 'props'

Integrating Bootstrap with ReactJS--

npm i bootstrap

import bootstrap.bundle JS and bootstrap.min.css CSS file into index.js

Inventory App

Item

id

title

unit

costPrice

sellingPrice

packageDate

Shadow DOM / Virtual DOM--

It is a temporary DOM copy maintained by React

Any rendering will result in regenrating the Shadow DOM first and this is very less costly as it is not directly linked to the browser rendering

After the entire Shadow DOM is regenerated, it is compared with the actual DOM and the differences are identified and then the actual DOM is managed to suit the differences. This it is less costly than regenerating the entire actual DOM

"key" is an attribute that is expected to have unique value for each record as it is used to compare records from shadow dom with records on actual dom

Class Component Life Cycle Methods--

constructor() //purpose of initializing state -> render() //generate and return the html-dom -> componentDidMount() //execute any task immediately after the first render()

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

|--->any sort of event handling can happen

| as part of it, 'setState()' may get called

|/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

|setState() is if invoked, then render() will be called again

|setState() -> render() -> componentDidUpdate() --->| //executing any tasks after each render() from 2nd render onwards

|---------------------------------------------------|

React Hooks--

A react hook is a pure function that provides additional enhancement to a function component.

A hook must be invoked in the top-most lines of code of a function component

useState Hook-

let [reader,writer]=useState(fieldInitialValue);

let [fullName,setFullName]=useState("Vamsy");

let [skills,setSkills]=useState(["java","angular","reactjs"]);

let [address,setAddress]=useState({flatNo:9015,area:"MVV City",city:"Visakhapatnam"});

useEffect Hook-

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

the callBack is equivalent to componentDidMount(), the callBack is executed only once that too after the first render of the component

useEffect(() => {},[field1,field2,...]);

the callBack is equivalent to componentDidUpdate(), the callBack is updated after every render of the component but only when any of the fields in the dependency array had their values changed

useEffect(() => {});

the callBack executes unconditionally after every render

React Hook form--

npm i react-hook-form

useForm({default Values}); return an object that has

a register method

a handleSubmit method

a setValue method

a reset method

a formState: {errors} to pick errors

State Management Using Redux and React Redux--

npm i redux react-redux

Redux is a state management library that can handle state centrally and isolatedly for an app

store is a global object that has all the data/state of the application.

an app has one and onle one store.

let myStore=createStore(myReducer);

reducer is a user defined pure javascript function. it takes the currentState and an action as params and returns the modified state.

const myReducer=(state,action)=>{

//based on the action, state is modified

return modifiedState;

}

action is a object that has a payload and an action type.

'type' is the operation to be executed,

'payload' is the data needed for the operation to take place.

const action1={type:"ADD",emp};

const action2={type:"REMOVE",empID:101};

dispatch is a built-in method used by a component to send an action to the reducer.

dispatch(action1);

react-redux is the integration library for redux into react.

<Provider></Provider>

it is used to wrap the store on to the top-level-component of our Application

<Provider store={myStore}>

<App />

</Provider>

useSelector()

it is a hook that is used to select only the required part of the global state into our component.

let emps=useSelector(state=>state.emps);

let empCount=useSelector(state=>state.emps.length);

useDispatch()

it is going to return a 'dispatch' method related to the current store.

const dispatch=useDispatch();

Using these libraries, we can eliminate state management from components.

store(all state)->----------------------------------------------

^ | |

| | |

| | |

| useSelector() useSelector()

| | |

| Component1 Component2

| | |

| | dispatch=useDispatch() | dispatch=useDispatch()

| | dispatch(action) | dispatch(action)

| | |

| | |

| | |

|modifiedState | |

reducer<----------------------------------------------------

Fake Rest-Api using json-server--

md contacts-api

cd contacts-api

npm init -y

npm i json-server@0.17.4

create a contacts-api/data.json and populate it with hypothetical data

modify contacts-api/package.json file to put "start":"json-server --port 9999 --watch ./data.json"

npm start

Rest-Api calls using Axios--

npm i axios

axios.get(endPoint) returns Promise

axios.delete(endPoint) returns Promise

axios.post(endPoint,reqBody) returns Promise

axios.put(endPoint,reqBody) returns Promise

Thunk Middleware to handle asynchronous calls--

npm i redux-thunk

Generally action are objects

But a thunkAction is a function that sits between component and reducer and

keeps dispatching action objects to the reducer alongside an asynchronous call.

store (all state) →------------------------------------------

↑ | |

| ↓ ↓

| | |

| useSelector() useSelector()

| ↓ ↓

| Component1 Component2

| | |

| | dispatch = useDispatch() |

| | dispatch(action) | dispatch = useDispatch()

| | | dispatch(thunkAction)

| | |

↑ | ↓

| modifedState ↓ -------[Thunk Action Function]------

reducer | ←---------------------← | |

| ←-------------------------------| dispatch(waitActionObj) |

| | axiosCall |

| ←-------------------------------| ifOK, dispatch(dataActionObj) |

| ←-------------------------------| ifERR, dispatch(errActionObj) |

| |

------------------------------------

React Routing--

npm i react-router react-router-dom --save

<BrowserRouter>

<Header />

<Routes>

<Route path="/" element={<C1 />} />

<Route path="/a" element={<C2 />} />

<Route path="/b" element={<C3 />} />

<Route path="/c/:pathVariable" element={<C4 />} />

</Routes>

<Footer />

</BrowserRouter>

<Link to"" ></Link> instead of <a></a>

<Navigate to="" />

================================================================================================

ANGULAR TRAINING

---------------------------------------------------------

single page application spa framework--

server REQ <- client

spa-bundle index.html along with JS code is loaded in the browser

(index.html+JS code) any event/form submission/etc occurs is handled by the JS on the client itself and the relevant html is generated

--(spa-bundle)RESP-> the generated html can be tailored to the existing html page (no concept of unloading or reloading a page)

rest-api <-data operation request----spa

----->response(data.json/data.xml)-->spa

npm install -g @angular/cli

Angular Architecture

node js is an alternate runtime for JS

node js offers a developmet platform where tools for compiling, testing, bundling, etc can be executed upon and application is developed

Any angular application comprises of modules, components, directives, services and pipes

Each of these is a typescript class

Each of these is marked with a Decorator for role identification

Each of these decorators are passed with a json object called meta-data that contains the config

Module

@NgModule({

declarations:[],

imports:[],

exports:[],

providers:[],

bootstrap:[]

})

class SalesModule{

}

Component

@Component({

selector:"",

templateUrl:"",

styleUrls:[],

providers:[]

})

class DashboardComponent{

}

Directive

@Directive({

selector:"",

providers:[]

})

class FastMovingSirective{

}

Service

@Injectable({

providedIn:"root",

providers:[]

})

class SalesService{

}

Pipe

@Pipe({

name:'',

providers:[]

})

class numberToWordsPipe{

}

Angular CLI--

ng new app-name

ng new app-name --standalone=false

cd app-name

ng generate module (no shortcut) ModuleName

ng generate (g) component (c) ComponentName --skip-tests

ng generate directive (no shortcut) DirectiveName --skip-tests

ng generate service (no shortcut) ServiceName --skip-tests

ng generate pipe (no shortcut) PipeName --skip-tests

ng generate class (no shortcut) ClassName --skip-tests

ng generate interface (no shortcut) InterfaceName

ng build -- build the application and the bundle is placed inside a 'dist' folder

ng test -- build the application and execte the test cases

ng serve -- build the application and the bundle is hosted on a development server @4200

the port number can be customised using '--port' flag

Angular Modules--

what is a module in javascript, what is import, export, export default

Javascript Module--

Each file is a module in JS/TS

Resources contained in a file can be exported and imported using export and import keywords

Angular Modules--

A module in Angular is a class that encapsulates Components, Pipes, Directives, Services and other Angular Modules

@NgModule({

declarations:[/\*list of components,pipes and directives that belong to this module\*/],

imports:[/\*list of modules we want to import into the current module\*/],

exports:[/\*list of components,pipes and directives we want to export from the current module\*/],

providers:[/\*list of services that are injectable at the current module level\*/],

bootstrap:[/\*list of components to be loaded initially when the current module loads\*/]

})

class SalesModule{

}

Every Angular App is housed inside a top level module call ROOT MODULE generally named as app.module

Other sub-modules are called as FEATURE MODULES

the 'exports' section of the metadata is absent for Root Module

the 'bootstrap' section is absent for other Feature Modules

index.html calles main.ts

Angular Component--

Angular offer html extendability, means we can create our own html elements

Components are custom html elements

Component = Behaviour + DOM + Style

component.ts component.html component.css

(fields & methods) (display) (style)

Dashboard.component.ts

@Component({

selector:"app-dashboard",

templateUrl:"dashboard.component.html",

styleUrls:["dashboard.component.css"],

providers:[]

})

class DashboardComponent{

noOfMsgs:number;

noOfReadMsgs:number;

constructor(){

this.noOfMsgs=100;

this.noOfReadMsgs=90;

}

}

dashboard.component.html--

<div>Message: {{noOfMsgs-noOfReadMsgs}}/{{noOfMsgs}}</div>

dashboard.component.css--

div{border:1px solid black;}

<app-dashboard></app-dashboard> --> <div>Message: 10/100 </div>

Data Binding--

using the fields and methods of component.ts inside the component.html is called data binding

different types of data binding-

Interpolation-

we can render the output of any angular expression on to the html-dom

<tag>{{angular-expression}}</tag>

for any reason, if the fields are updated, the expression is reevaluated and the output is also automatically updated

Two-way Data Binding-

we use this to bind a field with an input element of an html form

we have to use an attribute called ngModel from 'FormsModule' from '@angular/forms' to perform two-way data binding

<input type="text" name="tb1" [(ngModel)]="field" />

One-way Data Binding-

Attribute Binding-

to bind the value of a field with an attribute of an element

<tagName [attrubuteName]="field"> </tagName> //property binding also called

<tagName attrubuteName="value"> </tagName>

<td colspan="5"></td>

<td [colspan]="x"></td>

Style Binding-

to bind the value of a field with a css-property of an element

<tagName [style.cssProperty]="field"> </tagName>

Css Class binding-

to switch a css class on or off using the booleanField

<tagName [class.className]="booleanField"> </tagName>

Event Binding-

to invoke a method when an event occurs

<tagName (event-directive)="method()"> </tagName>

html elements-- event-directives--

------------------------------------------------------

onload load

onblur blur

onfocus focus

onchange change

onclick click

ondblclick dblclick

onsubmit ngSubmit

Angular Directives--

Angular offers html extendability, means we can create our own html elements and attributes

Components are custom html elements

Directives are custom html attributes

in-built attribute directives--

ngModel

ngForm

ngClass

...etc

in-built structural directives--

\*ngIf

\*ngFor

ngSwitch \*ngSwitchCase \*ngSwitchDefault

custom directive--

@Directive({

selector:"[directiveName]",

providers:[]

})

class DirectiveName{

}

Integrating Bootstrap with Angular--

install bootstrap

npm i bootstrap

include bootstrap js and bootstrap css into scripts and styles section of angular.json

Angular Pipes--

a pipe is used to transform a piece of data into another just before rendering

in-built pipes-

lowercase

uppercase

titlecase

number

percent

currency

date

custom pipe-

@Pipe({

name:'',

providers:[]

})

class numberToWordsPipe implements PipeTransform{

transform(value:any):any {

return transformedValue;

}

}

EMI Assignment--

Principal

Time Period

Rate of Interest

EMI =

Pipe Assignment--

assuming num=1234567.89

{{num|formatNumber}}

12,34,567.89

Angular Routing--

Routing allows us to display only one component at a designated area of the index page depending on the URL

Router Module from @angular/router provides the below services, classes and methods to manage the same

Routes--array of routes

[{path:'abc', component:Component1},

{path:'xyz', component:Component2},

]

RouterModule.forRoot(routes);

router-outlet--

it is a built in component to preserve the designated area where we want the components to appear

routerLink--

it is a directive for anchor 'a' tag, this takes the path where to go when clicked on the link

routerLinkActive--

it is a directive that takes a css-class and applies that class to the link that is currently active

Router--

it is a service that offers two methods-

navigate(['/sales','/offers']);

navigateByUrl("/sales/offers");

Angular Services--

Service is a class that hold business or data logic

Angular supports Dependency Injection for Services

@Injectable

constructor(private myService:MyService){

}

Inter-Component Communication--

The parent component can pass data through attributes of the child component

We can mark fields of a child component as an attribute using '@Input()' decorator

Event Propagation-

The child component can create its own events using EventEmitter

And this objects must be marked with '@Output()' decorator

'emit' is a method of EventEmitter that raises an event and can pass some data to the parent

The event handler of the parent can receive this data using $event object

default type of a button is submit

Template Driven Forms vs Reactive Forms (Model Driven Forms)--

FormsModule ReactiveFormsModule

ngModel FormControl

ngForm FormGroup

RxJS - Observables--

Observable is an enhanced Promise

let ob = new Observable( observer => {

//the asynchronous operaion job

//observer.next(val) to emit intermediate results of the job

//observer.error(err) to emit any error while doing the job

//observer.complete() to indicate the successful completion of the job

});

ob.subscribe({

next: val => {/\* the values are received \*/}

err: err => {/\* the error is received \*/ }

complete: () => {/\* we will do something after completion of the job \*/ }

}); //job execution starts

Rest-Api Communication--

HttpClientModule from '@angular/commons/http'

|

|-HttpClient

get(endPoint): Observable<data>

post(endPoint,reqBody): Observable<data>

put(endpoint,reqBody): Observable<data>

delete(endPoint): Observable<void>

Fake Rest-Api using json-server

md contacts-api

cd contacts-api

npm init -y

npm i json-server@0.17.4

create a contacts-api/data.json and populate it with hypothetical data

modify contacts-api/package.json file to put "start":"json-server --port 9999 --watch ./data.json"

npm start

Angular LifeCycle Hooks for a Component--

constructor-

This is invoked when Angular creates a component or directive by calling new on the class.

ngOnChanges-

Invoked every time there is a change in one of the input properties of the component.

ngOnInit-

Invoked when given component has been initialized.

This hook is only called once after the first ngOnChanges

ngDoCheck-

Invoked when the change detector of the given component is invoked. It allows us to implement our own change detection algorithm for the given component.

ng-template is used for reserving space in the DOM

ng-container is used for holding ngIf or ngFor conditions when you do not want to apply them in an existing DOM

Custom Directives--

@Input() for receiving any values from the Host

@HostListener() is used to handle events that occur on the element

ElementRef represents the element on which the directive is applied

==============================================================================================

In a dynamic web application, we will have things that are capable of generating HTML dynamically

Angular Introduction--

Angular is a typescript based SPA framework

AngularJS

Angular 2

Angular 4

......

Angular 15

Typescript--

TypeScript=JavaScript+DataTypes

Typescript is transpiled into Javascript

increases the prodcutivity of our application

Lab Setup--

VSCode IDE

NodeJS Development Platform https://nodejs.org (16.x-18.x) node --version

Angular CLI PMT npm i -g @angular/cli@16.1.3 ng version

Why NodeJS--

NodeJS is a alternate runtime for javascript

NodeJS is used for javascript that doesnt need any UI

Browsers are used for javascript that needs UI

An application development involvs:

compose IDE VSCode

compile/transpeelling compiler tsc

manage dependencies build tool npm

build and pack the app-bundle build tool npm

test testing lib jasmine and karma

deploy on to a server build tool/PMT angular cli

execution run-time Browser

NodeJS is used to execute the tools needed for developing the application.

NPM-Node Package Manager

NPX-Node Package Executor

TypeScript--

Creating a project:

npm init -y

npm i typescript --save-dev

npx tsc --init

npx tsc filename.ts

Datatypes:

number

string

boolean

bigint

symbol

Special Datatypes:

any - can allow any type of value in the memory, will not force you to verify the datatype of the variable

unknown - can allow any type of value in the memory, will force you to verify the datatype of the variable

never - variable is not assigned any value but cannot use return statement, similar to void

void - variable is not assigned any value but void allows return statement

undefined - no variable in the memory

null - variable in the memory but no value

typeof x ==="number"

tscofig.json--

{

"compilerOptions": {

"module": "commonjs",

"esModuleInterop": true,

"target": "es6",

"moduleResolution": "node",

"sourceMap": false,

"outDir": "dist"

},

"lib": ["es2015"]

}

? operator means optional parameter meaning may pass or may not pass value to a parameter

?? coalesce operator is similar to isNull function in oracle, if variable is there it will be used otherwise default value will be used, it is a new syntax for ternary operator

Angular Architecture--

Any Angular application is made up of five main artifacts-Modules,Directives,Component,Pipes,Services

Each Angular artifact is a class

Each of these classes are marked with decorators to identify their roles

These artifacts have config and the config is passed as json object to the decorator and is called meta-data

Modules

@NgModule({

declarations:[],

imports:[],

exports:[],

providers:[],

bootstrap:[]

})

class SalesModule{

}

Directives

@Directive({

selector:'[fastMovingStock]'

})

class FastMovingStockDirective{

}

Components

@Component({

selector:'app-dashboard',

templateUrl:'',

styleUrls:[]

})

class DashboardComponent{

}

Pipes

@Pipe({

name:'intoWord'

})

class InToWordsPipe{

}

Services

@Injectable({

providedIn:'root'

})

class OrdersService{

}

Angular CLI--

Angular CLI is a frontier tool provided by angular team to manage the project structure and project development phases like creating, building, executing, testing, etc

CLI commands-

ng version

ng new app-name

cd app-name

ng g (generate) module ModuleName

ng g c ComponentName --skip-tests

ng g directive DirectiveName --skip-tests

ng g service ServiceName --skip-tests

ng g pipe PipeName --skip-tests

ng g interface InterfaceName

ng g class ClassName

ng build - will compile .ts into .js , package and put them in 'dist' folder

ng serve - will build and hosts the package on development server on port 4200

ng serve --port 8888 - will build and hosts the package on development server on port 8888

ng serve --port 8888 -o - will build and hosts the package on development server on port 8888, launches the app on a browser

ng test - will build and invoke all the test cases

Angular Modules--

In JS, each file is one module, resources defined in one file can be used in another file, it helps to manage/modularize the source code

In Angular, Angular Modules helps to define the scope of access

Each Angular Module can group components, pipes, directives, services and other modules also into them

Artifacts belonging to a specific module have access to one another but they wont have access to artifacts of other modules

A module (moduleA) can be imported into another module (moduleB) so that moduleB artifacts can access exported artifacts of moduleA

Each Angular app must be mandatorily housed in one top-level module and is referred to as ROOT MODULE

AppModule is the default name for ROOT MODULE

@NgModule({

declarations:[], --list of components,pipes and directives that are grouped under this module

imports:[], --list of modules to be imported into this

exports:[], --list of artifacts of this module that are allowed to be accessed out side

providers:[], --list of services that are managed by the injector of this module

bootstrap:[] --list of components to be instantiated immediately after loading this module

})

class SalesModule{

}

meta-data of ROOT MODULE will not have 'exports' section

meta-data of ROOT MODULE only has the 'bootstrap' section

Angular Directives and Components--

Angular offers html-extendability as a feature

It means that we can create our own html elements and attributes in angular

Attribute Directive / Directive - are custom made attributes in angular

@Directive({

selector:'[fastMovingStock]'

})

class FastMovingStockDirective{

}

<input type="text" ngModel="x" /> --ngModel is inbuilt directive

<div fastMovingStock="true"> --fastMovingStock is custom directive

<!--details of stock items-->

</div>

Structural Directives - are attribute directive that can control the rendering of an element

\*ngIf

\*ngFor

ngSwitch \*ngSwitchCase \*ngSwitchDefault

Component Directive / Components - are custom made elements/tags in angular

An Angular Component is made up of 3 things

Component = State & Behaviour (controlled by .component.ts file)

+

HTML DOM Content (controlled by .component.html file)

+

Style (controlled by .component.css file)

dashboard.component.ts--

@Component({

selector:'app-dashboard',

templateUrl:'dashboard.component.html',

styleUrls:['dashboard.component.css']

})

class DashboardComponent{

numberOfMessagesRead:number=12;

}

dashboard.component.html--

<div class="dashboard">

You have {{numberOfMessagesRead}} messages read already

</div>

dashboard.component.css--

.dashboard{

margin:10px;

border: 1px solid black;

}

<app-dashboard></app-dashboard>

Angular Application Flow--

index.html --> main.ts --> AppModule --> AppComponent

Data Binding--

It is about accessing the fields and the methods of component.ts file inside the component.html file

It can be done in the following ways-

Interpolation-

It is to bind an angular field or expression onto the content directly

Whenever the field bound here has its value changed, the rendered content also gets updated automatically

<tagName>{{fieldOrAngularExpression}}</tagName>

Two-Way Data Binding-

It is to bind the field with an input-element, so that the input-elemt will show the value of the field initially and whenever the input-element is edited, the value of the field also is updated

We have to use ngModel attribute-directive defined in 'FormsModule' from '@angular/forms' for two-way binding

<input type="text" [(ngModel)]="userName" />

One-Way Data Binding-

Attriubute Data Binding-

It is to bind a field with an attribute

<tagName attribute="value></tagName>

<tagName [attribute]="fieldOrAngularExpression></tagName>

<img src="abcd" /> <!-- abcd itself is the image url -->

<img [src]="abcd" /> <!-- abcd is the field that has the image url -->

Event Binding-

It is to bind a method with an event

<tagName (event-directive)="method()"></tagName>

html-event-attribute event-directive

-------------------------------------------------

onSubmit ngSubmit

onClick click

onblur blur

onFocus focus

onChange change

.....etc

Style Binding-

It is to bind a field with a css-property

<tagName [style.cssProperty]="field"></tagName>

field={"css-property1:angularExpression1","css-property2:angularExpression2"};

<tagName [ngStyle]="field"></tagName>

CSS Class Binding-

It is to control if a css class is applied on an element or not

<tagName [class.className]="booleanExpression"></tagName>

<tagName ngClass="cssClass1 cssClass2 cssClass3"></tagName>

myClasses:string="cssClass1 cssClass2 cssClass3";

<tagName [ngClass]="myClasses"></tagName>

myClasses:string[]=["cssClass1","cssClass2","cssClass3"];

<tagName [ngClass]="myClasses"></tagName>

myClasses={cssClass1:true,cssClass2:false};

<tagName [ngClass]="myClasses"></tagName>

ngClass is an attribute binding

check for ngStyle attribute binding as well

Custom Directive--

<div fastMovingStock="true">

<!-- details of stock items -->

</div>

<p fastMovingStock="true">

<!-- details of stock items -->

</div>

@Directive({

selector:'[fastMovingStock]'

})

class FastMovingStockDirective{

//access the hostElement via ElementRef class

//@HostListener decorator to respond to the event that happen on the host-element

//@Input decorator is used to receive the data from a host-element into the attribute directive

}

Structural Directives--

ng-template is an in-built component

\*ngIf-

<ng-template [ngIf]="booleanExpression">

<p>This is a test para</p>

</ng-template>

<p \*ngIf=booleanExpression">This is a test para</p>

\*ngFor-

<ng-template [ngFor]="let ele of anArray;index as i">

<p>{{ele}} is available at {{i}}</p>

</ng-template>

<p \*ngFor=let ele of anArray;index as i">{{ele}} is available at {{i}}</p>

ngSwitch \*ngSwitchCase \*ngSwitchDefault

<div [ngSwitch]="day">

<span \*ngSwitchCase="1">Monday</span>

<span \*ngSwitchCase="2">Tuesday</span>

<span \*ngSwitchCase="3">Wednesday</span>

<span \*ngSwitchCase="4">Thursday</span>

<span \*ngSwitchCase="5">Friday</span>

<span \*ngSwitchCase="6">Saturday</span>

<span \*ngSwitchCase="7">Sunday</span>

<span \*ngSwitchDefault>No Such Day</span>

</div>

default type of button element is type="submit"

ng-container is also an in-built component for holding ngIf and ngFor conditions and the ng-template can hold the view

Angular Pipes--

A pipe is used to transform a value into another just before rendering

{{valueOrFieldOrExpression | pipeName:'pipeInputs'}}

in-built pipes-

lowercase

uppercase

titlecase

number

currency

date

...etc

custom pipe-

@Pipe({

name:'intoWord'

})

class IntoWordPipe implements PipeTransform{

transform(value:any):any{

//here goes the transformation code..

return transformedValue;

}

}

Angular Services--

A service is a injectable object carrying business logic or api calls

Assignment-create a pipe called 'capitals'

let stateName:string="Andhra Pradesh";

{{stateName|capitals}} ---> Hyderabad

Integrating Angular with Bootstrap--

npm i bootstrap

include the below paths-

node\_modules/bootstrap/dist/css/bootstrap.min.css - in 'styles' section of angular.json

node\_modules/bootstrap/dist/js/bootstrap.min.js - in 'scripts' section of angular.json

Angular Routing--

Routing allows us to load one component at a time based on the url

We can map urls one for each component and to load one component at a time based on the url

RouterModule from '@angular/router'

Routes -- model Route[]

Route -- model {path:'',pathMatch:'full|startsWith',component:Component,redirect:''}

For Example: {path:'abc',component:DashboardComponent}

assuming port 8888

http://localhost:8888/abc

http://localhost:8888/abc/123

http://localhost:8888/abc/xyz

will result in loading Dashboard Component

For Example: {path:'abc',pathMatch:'full',component:DashboardComponent}

assuming port 8888

http://localhost:8888/abc

will result in loading DashboardComponent

http://localhost:8888/abc/123

http://localhost:8888/abc/xyz

will not work!

router-outlet -- component this is used in the top-level component to reserve space for the router output

routerLink -- attribute directive it is used on 'a' tag instead of its 'href' attribute

routerLinkActive -- attribute directive it accepts a css-class and that css-class is applied only when that link is active

Router -- service navigateByUrl("/sales/dailyReport")

navigate(['/sales','/dailyReport'])

ActivatedRoute -- service it is used to extract the data (like path parameters/query string,..etc) from the current URL

Angular Forms--

Template Driven Form Model Driven Form

FormsModule from '@angular/forms' is used ReactiveFormsModule from '@angular/forms' is used

ngForm FormGroup

ngModel FormControl

formControlName

Validations are limited Has an extensive Validation framework and custom validators are seamlessly supported

Used in simple context where not more Used in all complex and nested form and regular

than two input controls exist form scenarios

Testing is difficult as the form structure Testing is easy as the form is modelled in the TS class and is only structured

and behaviour are defined in the template in the template

Both ngForm and FormGroup support valid and invalid properties

Both ngModel and FormControl support touched and untoched, dirty and pristine, valid and invalid properties

InBuilt Validators:

Validators from '@angular/forms'

offers

requried

min

max

minLength

maxLength

email

pattern

Custom Validators:

is any function that accepts an object of 'AbstractControl' and returns an object of ValidationErrors or null.

customValidator(control:AbstractControl) : ValidationErrors | null {

let isValid:boolean = true;

//validation logic.

return isValid ? null : {"validatorName":true };

}

//Employee Management System--exercise

Assignment#1

Create an angular SPA that does CRUD operation on

Employee

id

fullName

basicPay

dateOfJoining

designation ASSOCIATE / Jr ASSOCIATE / Sr ASSOCIATE / MANAGER

Custome Validator to check that the dateOfJoining can not be a future date.

Observable from 'rxjs'--

Observable is an enhanced Promise

A Promise can emit only one value that too after the completion of the async job

An Observable can emit any number of values while the async job is in progress

A Promise has to be subscribed using 'then()' method for the async job to start and only one subscription can happen

An Observable has to be subscribed using 'subscribe()' method for the async job to start and any number of subscription can happen

A Promise once subscribed can not be unsubscribed

An Observable can be unsubscribed using 'unsubscribe()' method

let bgJob = (resolve,reject) => {

//resolve is used to signal completion and also to carry the end result

//reject signals error

};

let p = new Promise(bgJob);

p.then(

val => {/\*success callback\*/ },

err => {/\*error callback\*/ }

);

let bgJob2 = observer => {

//observer.next(val) used to emit intermediate results while the job is in progress

//observer.error(err) used to signal error

//observer.complete() used to signal completion

};

let ob = new Observable(bgJob2);

ob.subscribe({

next: val => {},

error: err => {},

complete: () => {}

});

Observable class has a method called 'pipe()' used to pipe operations on an existing observable to generate a new observable

let ob2 = ob1.pipe(operation1(), operation2());

Calling Rest-Api using HttpClient--

HttpClient is a service from HttpClientModule (import in AppModule) from '@angular/common/http'

get(endPoint): Observable

post(endPoint,reqBody): Observable

put(endPoint,reqBody): Observable

delete(endPoint): Observable

Generating fake rest-api using json-server--

md adb-api

cd adb-api

npm init -y

npm i json-server@0.17.4

create a json file containing hypothetical data in the adb-api folder

say adb-api/data.json

include the start command in package.json as below-

"start":"json-server --watch ./data.json --port 9999"

npm start

Content vs View--

All the DOM written inside the template of a component is called its view (ViewChildren or ViewChild)

Anything written inside the <tagName></tagName> of a component is called its content (ContentChildren or ContentChild)

<app-component-a>here goes the content</app-component-a>

Angular Component or Directive Life Cycle Hooks--

constructor() gets invoked after the allocation of the object

ngOnChanges(SimpleChanges) gets invoked every time when the @Input() fields change

ngOnInit() gets invoked after the first render (only once)

ngDoCheck() is used to detect and act upon changes that angular cannot detect on its own. invoked after ngOnInit() for the first

time and then after ngOnChanges()

ngAfterContentInit() gets invoked after the content is initialized

ngAfterContentChecked() gets invoked after the content is checked

ngAfterViewInit() gets invoked after the view is initialized

ngAfterViewChecked() gets invoked after the view is checked

ngOnDestroy() gets invoked just before the component in unloaded

Content Projection--

ng-content

ng-container

ng-template

We can't put form in a table

Command for creating environment files-

ng generate environments

ng build --production

ng build --test

ng build --dev

lazy module--

ng g module Dummy --route dummy --module app.module

normal module--

ng g module Dummy2 --module app.module

Learn for Advanced Concepts--TODO

ng update @angular/cli@15.1.0

================================================================================

HTML, CSS, JAVASCRIPT

---------------------------------------------------------

Lab Setup--

Chrome(or any other browser)

VSCode(or any other text editor)

NodeJS(as javascript runtime)

HTML was started to design/create documents

HTML- Hyper Text Mark Up Language--

It is used to write a document that can be shared across HTTP protocol

Hyper Text (Hyper Media) is a collection of plain text, formatted text, graphics and links

Mark up is a practice used in literature

WWW (World Wide Web) is quite ancient now.

"This is your first day. Have a blast" said my manager.

Vizag - the steel city - is my hometown

The symbols used to mark up are called HTML elements

Each html element is made up of a opening tag + content + closing tag

<p id="para1">This is a para element and para content goes here</p>

HTML is a scripting language because it is not compiled, it is only interpreted by other softwares like a browser

a sample html document

<!DOCTYPE html> --preprocessor statement

<html>

<head>

<!--meta data about about the current page, metadata is data about data-->

</head>

<body>

<!-- the page content -->

</body>

</html>

Typography--

h1 to h6 heading elements block element

p para block elelemt

b,u,i bold,underline,italicized inline element

strong,em sematic elements for bold and italics inline element

sup,sub superscript, subscript inline element

div - container block element

nav,header,section,footer,article,main,aside - sematic container block elements, alternates to div

span - inline group element

kbd,code,date,mail,address,phone - inline group element, alternates to span

Sematic elements are only introduced in html5, will alert the search engine for highlighting search mechanisms in the web

purpose of a sematic element is search engine optimization

div is called a container and span is an inline element

No 2 block elements will appear side by side, it will appear one below the other

Inline element means inline to text, it will appear side by side

Lists - both ul and ol are block level elements

ul - unordered lists

type circle/dot/square

ol - ordered lists

type 1/a/i/I/A

li - list item

HTML Tables--

table - to create a table

border - attribute takes 1 to 5

width - attribute takes a value in px/%/em/pt/in/cm

height - attribute takes a value in px/%/em/pt/in/cm

cellspacing - attribute takes a value in px/%/em/pt/in/cm and controls the spacing between two cells

cellpadding - attribute takes a value in px/%/em/pt/in/cm and controls the spacing between the cell border and its content

caption - sub-element to give a heading to the table

thead - sub-element of table element and holds the header rows

tfoot - sub-element of table element and holds the footer rows

tbody - sub-element of table element and holds the data rows

tr - table row is a sub element representing one row and can be used inside thead/tbody/tfoot/table

th - table header cell, a sub-element of tr

td - table data cell, a sub-element of tr

rowspan - attribute of th and td representing the number of rows to be occupied, default is 1

colspan - attribute of th and td representing the number of columns to be occupied, default is 1

cellspacing

^

------------------------------ | -------------------------

| | | | | |

cellpadding -------|--->| | | | |

| | |-----------| |

| Some content in the cell | | |

| | | |

------------------------------ -------------------------

HTML Media--

img - inline element that injects an image into the page and is a self-closing element

src - attribute of img tag to hold the relative or absolute or virtual path of the image

audio - inline element to inject an audio

video - inline element to inject a video

source - sub-element of audio and video to supply the audio file or video file, a single audio or video element can house any number of source tags each representing a different format of audio or video

src - attribute of source to hold the relative or absolute or virtual path of the audio or video file

iframe - used to embed an external web page into our own web page

src - attribute to hold the web address of the webpage you wish to embed

HTML Links--

a - anchor tag

href - attribute takes the destination path

target - attribute takes \_self,\_blank,iframeName

HTML Form Elements--

form - it is used to group a set of fields that have to be submitted to a server program

action - attribute takes the path of the server program to which the data has to be submitted

method - attribute that takes GET/POST

label - it is used to carry a text related to a field like field names

for - attribute takes the id of the input element to which the label is related to

input - it is used to create a form field control that enable inputting data

type - text

file

passowrd

checkbox

radio

number

range

decimal

email

date

datetime-locale

id - attribute is used to give an id to the input element

name - attribute is used to give a field name that is carried along with its value to the server

value - attribute takes the inital value of the field

required - attribute ensures that the field is given a value mandatorily

min - attribute that takes a minimum possible value for type="number/decimal/date"

max - attribute that takes a maximum possible value for type="number/decimal/date"

minlength - attribute that takes a minimum length of the value while type="text"

maxlength - attribute that takes a maximum length of the value while type="text"

pattern - attribute that takes a regular expression to validate the value while type="text"

textarea - used to accept multi line text

select - used to create a drop-down or a list box

multiple - attribute makes the drop-down into a list enabling multiple option selection

option - is a sub-element of select used to provide the options

value - attribute of option tag that takes the value to be submitted if this option is selected

button - used to create a push button

type - submit (default)

reset

button

HTML Layouts and sematic elements--

<header> - Defines a header for a document or a section

<nav> - Defines a set of navigation links

<section> - Defines a section in a document

<article> - Defines an independent, self-contained content

<aside> - Defines content aside from the content (like a sidebar)

<footer> - Defines a footer for a document or a section

<details> - Defines additional details that the user can open and close on demand

<summary> - Defines a heading for the <details> element

<figure> - Defines a self-contained content, like illustrations, diagrams, photos, code listings..etc

<figcaption>- Defines a caption for a <figure> element.

<time> - Defines a date/time

HTML entities--

Result Description Entity Name Entity Number

=============================================================================

non-breaking space &nbsp; &#160;

< less than &lt; &#60;

> greater than &gt; &#62;

& ampersand &amp; &#38;

" double quotation mark &quot; &#34;

' single quotation mark &apos; &#39;

¢ cent &cent; &#162;

£ pound &pound; &#163;

¥ yen &yen; &#165;

€ euro &euro; &#8364;

© copyright &copy; &#169;

® registered trademark &reg; &#174;

College Admission Form- exercise

id attribute is used in the javascript logic

name attribute is used to pass the field value name to the server

CSS3-

CSS 3 - Intro

-----------------------------------

Cascading Style Sheet

purpose is to provide styling to an html document.

+ CSS offers common uniform properties unlike html

+ CSS can provide styling across grouped elements.

+ CSS can isolate styling from content. It adds to maintainability.

Mode Of CSS

Inline Style Sheets

Embeded Style Sheets

External Style Sheets

Inline Style Sheet

it is applied using 'style' attribute.

Inline style sheet is the least used method. It is applied only to the element

on which it is written.

<tagName style="css-property:value;css-property:value;">

content

</tagName>

<p style="text-align:justified;font-size:11pts;">

this is a apra.

</p>

Embeded Style Sheet

it is applied using a tag called <style></style>, style elements

is a sub-element of <head></head>

<head>

<style>

selector {

css-property:value;

css-property:value;

}

</style>

</head>

External Style Sheet

style is defiend in a seperate file with extension .css

and that cna be linked to any number of html pages using

<link href="fileName.css" rel="stylesheet" />

Selector is a string that qualfies the elements to be applied with the style.

1. Tag Name

each tag name itself is a selector.

tagName{

css-property:value;

}

p{

text-align:center;

}

2. Attribute

attribute selector groups elements having the mentioned attribute.

[attributeName]{

css-property:value;

}

[attributeName=""]{

css-property:value;

}

[src] {

width:50%;

}

[type="number"] {

text-align:right;

}

3. Class

is any usr defiend string. that should start with dot(.)

to apply the class on a element the element class attribute

should be assigned with the class name.

.className{

css-property:value;

}

<tagName class="className1 className2">

</tagName>

.important {

background-color:#00000;

color:#ffffff;

}

.highlight{

border:1px solid white;

}

<p class="important"></p>

<h3 class="important highlight"></h3>

<tr class="important"></tr>

4. Id

if a style swhould be applied to an elemnt hving

a specific id, then id selector is used.

#id{

css-property:value;

}

5. Psuedo

CSS - unit of measure

------------------------------------

absolute (irrespective of screen size)

in

mm

cm

pt 1 in = 72 pt

pc 1 in = 6 pc 1 pc = 12 pt

relative (to screen size)

px

%

em relative the font-size

1 em = complete font-size

font-size: 12pt;

width: 8em; 8\*12pt = 96pt;

vh 1% of viewport height

vw 1% of viewport width

vmin min of vh or vw

CSS Box Model Properties--

margin - amount of space we have to maintain across the element

margin: top left bottom right;

margin-top

margin-bottom

margin-left

margin-right

border: border-size border-style border-color;

border-top

border-bottom

border-left

border-right

padding - space between the content and the element border

padding: top left bottom right;

padding-top

padding-bottom

padding-left

padding-right

border-radius - radius of the corners of the element

border-top-left-radius

border-top-right-radius

border-bottom-left-radius

border-bottom-right-radius

width

height

min-width

min-height

max-width

max-height

width and height will work for only block/inline-block elements

display - inline/block/inline-block

position - relative/absolute

top

left

bottom

right

z-index

CSS Background color and image related properties--

background-color

color

background-image: url("image/path")

background-repeat: no-repeat/repeat/repeat-x/repeat-y;

background-position

CSS Font related properties--

font-family

font-size

font-variant normal or small-caps

font-weight bold or bolder or boldest or any number like 300,400,500,..etc

font-style italic or oblique (default)

CSS Text related properties

color

direction

letter-spacing

word-spacing

text-align

text-indent

text-decoration none | underline| line-through | overline

Table related CSS properties--

border-collapse collapse or separate

border-spacing

caption-side top left right bottom

CSS List related properties--

list-style-type disc circle square none for ul

decimal decimal-leading-zero lower-alpha upper-alpha lower-roman upper-roman for ol

list-style-position outside or inside

list-style-image: url('imgs/note.png')

marker-offset

controlling scroll bars

------------------------------------

overflow

visible

scroll

auto

hidden

CSS Operators

-------------------------------------

s1 applies to elements that have s1 as tag name

.s1 applies to elements that have s1 as class

#s1 applies to elements that have s1 as id

s1,s2 applies to elements that match both selectors

s1 s2 applies to all elements that match s2 and are inside s1

s1>s2 applies to all elements that match s2 whose parent is s1

s1+s2 applies to all elements that match s2 and are immediately after s1

s1~s2 applies to every element that match s2 and is preceded by s1

<div>

<p></p> div p {} div>p {}

<section>

<p></p> div p {}

<p></p> div p {}

</section>

<p></p> div p {} div>p {}

</div>

<p></p> div~p {} div+p {}

<p></p> div~p {}

applies to all elements that have

------------------------------------

[s1] s1 as attribute

[s1='v1'] s1 attrib with v1 value

[s1~='v1'] s1 attrib value containes word v1

[s1\*='v1'] s1 attrib value containes word v1

[s1!='v1'] s1 attrib value not equal to v1

[s1^='v1'] s1 attrib value starts with the word v1

[s1$='v1'] s1 attrib value ends with the word v1

Psuedo selectors

----------------------------------------------------------------------------

input:enabled Selects every enabled <input> element

p:first-child Selects every <p> element that is the first child

of its parent

p::first-letter Selects the first letter of every <p> element

p::first-line Selects the first line of every <p> element

p:first-of-type Selects every <p> element that is the first <p>

element of its parent

input:focus Selects the input element which has focus

a:hover Selects links on mouse over

input:in-range Selects input elements with a value within a specified range

input:indeterminate

Selects input elements that are in an indeterminate state

input:invalid Selects all input elements with an invalid value

input:optional Selects input elements with no "required" attribute

p:last-child Selects every <p> element that is the last child of its parent

p:last-of-type Selects every <p> element that is the last <p> element of its parent

a:link Selects all unvisited links

:not(p) Selects every element that is not a <p> element

p:nth-child(2) Selects every <p> element that is the second child of its parent

p:nth-last-child(2)

Selects every <p> element that is the second child of its parent,

counting from the last child

p:nth-last-of-type(2)

Selects every <p> element that is the second <p> element of its parent,

counting from the last child

p:nth-of-type(2)

Selects every <p> element that is the second <p> element of its parent

HTML & CSS Assignment

---------------------------------------------------------------------------

Digital resume:

Banner Section

Photo

Name

Contact details

Skill Set

Technical Qualifications

Academic Qualification

Experience And Recent Projects

Achivments And Publications

Personal Details

JavaScript (ES6)--

Javascript Basics

is a scripting language that can execute its scripts

both on a browser and in a stand alone mode too.

we need a javascript runtime to execute the script

without a browser. And NodeJS is one such runtime.

1. javascript inherits it syntx from c and cpp,hence case sensitive

2. is a dynamically typed language

a. variables need not be declared

b. even if we declare using var/let/const keywords, we dont specfiy the datatype

c. the type of a variable is decided after it is assigned a value

d. a variable can hold data of different types.

number,strings,null,array,function,undefined

Datatypes

Variables and scopes (let,var), operators

Control Structures

Program Flows

String, Math, Date

String Methods

String length

String slice()

String substring()

String substr()

String replace()

String replaceAll()

String toUpperCase()

String toLowerCase()

String concat()

String trim()

String trimStart()

String trimEnd()

String padStart()

String padEnd()

String charAt()

String charCodeAt()

String split()

String indexOf()

String lastIndexOf()

String search()

String match()

String matchAll()

String includes()

String startsWith()

String endsWith()

Math Methods

Math.E // returns Euler's number

Math.PI // returns PI

Math.SQRT2 // returns the square root of 2

Math.SQRT1\_2 // returns the square root of 1/2

Math.LN2 // returns the natural logarithm of 2

Math.LN10 // returns the natural logarithm of 10

Math.LOG2E // returns base 2 logarithm of E

Math.LOG10E // returns base 10 logarithm of E

Math.round(x) Returns x rounded to its nearest integer

Math.ceil(x) Returns x rounded up to its nearest integer

Math.floor(x) Returns x rounded down to its nearest integer

Math.trunc(x) Returns the integer part of x (new in ES6)

abs(x) Returns the absolute value of x

acos(x) Returns the arccosine of x, in radians

acosh(x) Returns the hyperbolic arccosine of x

asin(x) Returns the arcsine of x, in radians

asinh(x) Returns the hyperbolic arcsine of x

atan(x) Returns the arctangent of x as a numeric value between -PI/2 and PI/2 radians

atan2(y, x) Returns the arctangent of the quotient of its arguments

atanh(x) Returns the hyperbolic arctangent of x

cbrt(x) Returns the cubic root of x

ceil(x) Returns x, rounded upwards to the nearest integer

cos(x) Returns the cosine of x (x is in radians)

cosh(x) Returns the hyperbolic cosine of x

exp(x) Returns the value of Ex

floor(x) Returns x, rounded downwards to the nearest integer

log(x) Returns the natural logarithm (base E) of x

max(x, y, z, ..., n) Returns the number with the highest value

min(x, y, z, ..., n) Returns the number with the lowest value

pow(x, y) Returns the value of x to the power of y

random() Returns a random number between 0 and 1

round(x) Rounds x to the nearest integer

sign(x) Returns if x is negative, null or positive (-1, 0, 1)

sin(x) Returns the sine of x (x is in radians)

sinh(x) Returns the hyperbolic sine of x

sqrt(x) Returns the square root of x

tan(x) Returns the tangent of an angle

tanh(x) Returns the hyperbolic tangent of a number

trunc(x) Returns the integer part of a number (x)

Javascript OOPs+

Javascript Functions

Arrow Functions

Memeber Functions and this keyword

call and apply

call backs and closures

Javascript Arrays

1. let arr = [];

2. let arr = new Array();

3. arrays are not homoginous

4. arrays are not fixed in size

Array properties

arr.length

Array functions

arr.push(ele);

arr.pop();

arr.concat(anotherArr);

arr.splice(startIndex,deleteCount);

Array Prototype Functions

arr.forEach(ele => {});

//will execute the given callback function on each and every ele of the array

arr.sort((ele1,ele2) => { return -1 if ele1<ele2 or return 1 if ele1>ele2 else return 0 })

//will sort the array and uses the given callback function to compare.

//and the sort function returns the sorted array.

arr.map(ele => { write code to transform ele to anuthing else and return it })

//the map function will execute the givne callback on each ele

//of the array and returns a new array containing the transformed values.

let arr = [1,2,3,4,5];

let arr2 = arr.map( ele => ele\*ele);

arr2 will be [1,4,9,16,25]

arr.reduce((ele1,ele2) => { return any binary output of ele1 and ele2 })

//the reduce function will execute the passed binary operation on

//first two eles of the array, and then that result paired with third ele

//of the array is again passed to the binary operation and son on...

//until no more eles left and final reduced value is returned.

let arr = [1,2,3,4,5];

let sum = (n1,n2) => n1+n2;

arr.reduce(sum);

//is as same as

// sum(sum(sum(sum(arr[0],arr[1]),arr[2]),arr[3]),arr[4])

// which is 15

ES6 Enhancements

Modules

Template Literals

Arrow Functions

Destructuring

Spread and Rest operators

Javascript Asynchronous Programming

setInterval

clearInterval

setTimeout

clearTimeout

async, await and Promise

a Promise is a class whose object accepts a job-function.

a job-function accepts two callbacks resolve and resposne and

excutes a task asynchronously. it invokes resilve incase of

successful completion of the job and invoke reject incase of error.

let p = new Promise(

(resolve,reject) => {

....async code will go

if done call resolve

if err call reject

}

);

p.then(

() => { to respond if job done},

() => { to respond if job got error}

);

PROMISE--

Promise is a javascript class

It encapsulates any async function and allows the function to communicate either an error or a completion of the job to the main executor (the place where the function is called)

const theJob = (resolve,reject) => {

//goes here, the asynchronous code...

//resolve is called here to signal completion, can also carry any end-result

//reject is called to signal error, can carry the error

};

let p = new Promise(theJob);

p.then(

endResult => { /\*gets invoked once the job is done \*/ },

error => { /\*gets invoked if the job encounters an error \*/ }

)

p.then(endResult => { /\*gets invoked once the job is done \*/ })

.catch(error => { /\*gets invoked if the job encounters an error \*/ });

Javascript BOM & DOM--

Browser Object Model-

window

navigator

appName --browser name

appCodeName --usually will be browser name

history

back()

forward()

location --related to the url

href

hostname

protocol

assign(url)

document --body of the window

alert()

prompt()

confirm()

open()

close()

Document Object Model-

document

elements

forms

createElement()

removeElement()

replace(oldEle,newEle)

append() --element adding after

prepand() --element adding before

getElementById(id)

getElementByName(name)

querySelector("css-selector") --can be id,class,tag,etc

Event Handling-

htmlEventAttribute="method()"

eleObject.addEventListener("eventName",methodRef)

eventName htmlEventAttribute

--------------------------------------

click onClick

dblClick onDblClick

mouseOver onMouseOver

mouseLeave onMouseLeave

keyUp onKeyUp

keyDown onKeyDown

keyPress onKeyPress

blur onBlur

change onChange

focus onFocus

submit onSubmit

blur, change, focus are only for Input Elements

submit is only for Forms

===============================================================

Spring Framework--

Objetives

1.Spring Core and Dependency Injection

2.Spring Data anf Spring Data JPA

3.Spring Boot

4.Spring Web MVC

Pre-Requisites

1.Maven

2.JDK 1.8/Java 8

3.JPA-Hibernate/any other ORM

Lab Setup

1.JDK 1.8

2.STS (latest) IDE

3.MySQL for RDBMS / H2DB

Introduction--

Spring Framework is a development platform and a JavaEE framework that offers dependency injection as a core-feature.

Dependency-

If an artifact (say A) of an application is functionally dependent on another artifact (say B), then A is said to depend on B or B is a dependency for A

interface EmpDao{

Employee add(Employee emp);

}

class EmpDaoImpl implements EmpDao{

//all code here...

}

interface EmpService{

Employee add(Employee emp);

}

class EmpServiceImpl implements EmpService{

private EmpDao dao;

public EmpServiceImpl(){

this.dao=new EmpDaoImpl();

}

public Employee add(Employee emp){

dao.add(emp);

}

}

//Service depends on DAO

//DAO is the dependency

//Service is the site

Dependency Injection-

Supplying dependencies into the site (place of injection) externally is called dependency injection

interface EmpDao{

Employee add(Employee emp);

}

class EmpDaoJdbcImpl implements EmpDao{

//all code here...

}

class EmpDaoJpaImpl implements EmpDao{

//all code here...

}

interface EmpService{

Employee add(Employee emp);

}

class EmpServiceImpl implements EmpService{

private EmpDao dao;

public EmpServiceImpl(EmpDao dao){

this.dao=dao;

}

public Employee add(Employee emp){

dao.add(emp);

}

}

//somewhere in main or so

EmpService service = new EmpServiceImpl(new EmpDaoJdbcImpl());

//or

EmpService service = new EmpServiceImpl(new EmpDaoJpaImpl());

Dependency Injection helps in optimizing development cost and faster development time

Spring Modules--

Spring is highly modularized

Spring

Core Base fundamental module needed for all other modules to work, this is where DI is offered

Context Offers ApplicationContext and auto-wiring on top of DI

SpEL Stands for Spring Expression Language - offers injecting expressive part of beans

AOP Stands for Spring Aspect Oriented Programming

Web Helps building MVC web apps and rest-apis

Data Offers dynamically implemented Repositories/DAOs

Boot Rapid Application Development through auto-configuration...

Batch Used to build batch processing applications

WebFlux Used to build asynchronous applications

Secure Offers Authorization and Authentication

Test It is a testing framework for all spring modules based applications

...etc.,

Spring Core, Spring Context, Spring EL--

These modules of Spring Framework offer Inversion Of Control (an implementation style of DI concept)

Inversion of Control (IoC) is a principle of DI that says "DO NOT CALL; WE WILL CALL"

The Life Cycle of the Artifact (Service/DAOs) Objects are managed automatically. Like object creation/injection or destruction is all automated

Terms To Remember-

Component is a class whose objects are to be managed (by Container)

Bean is an object of the Component OR is the object being managed by the Container

Container is the class offered in Spring Framework that manages the life cycle of the bean. Container creates, injects and destroys a Bean

where the class of a Bean is called a Component

Bean Configuration is to inform the container the list of Components or list of Beans (only when the class is 3rd party) and their dependencies

Container--

Spring Framework offers two interfaces as containers:

BeanFactory from Spring Core Module

ApplicationContext from Spring Context Module

BeanFactory is less capable than ApplicationContext

Bean Configuration--

It is to inform the Container - list of Components, list of Beans and dependency between them

Xml Based Configuration

Annotation Based Configuration

Java Based Configuration

Annotation Based Configuration

@Configuration

@ComponentScan(basePackage)

@Component("beanName")

|--@Service

|--@Controller

|--@Repository

|--@RestController

|--@Advise ......etc,

@Scope("singleton|prototype|request|session|global-session")

//lifecycle of an instance, directly used on the class itself

request|session|global-session are used for web apps

default scope value is singleton

whenever scope is singleton, single object is created and passed everytime

whenever scope is prototype, new object is created each time and passed everytime

@PropertySource("") //inform the name of the external properties source file

@Value("springExpression") //inject primitive or strings values from external source into our component

//Field Injection

@Value("${app.name:UnNamed Application}")

private String appName;

//Constructor Injection

@Value("${app.name:UnNamed Application}")

public className(String appName){

this.appName=appName;

}

//Setter Injection

@Value("${app.name:UnNamed Application}")

public void setAppName(String appName){

this.appName=appName;

}

@Autowired

It is used to inject a bean into another bean

default is byType

byType --a site is inject with a dependency as and when they are of the same type, @Primary annotation can be used to pick a default matching bean

in case of having multiple matchings

byName --injects a dependency whose name is mentioned using @Qualifier("beanName")

Java Based Configuration--

create a bean programmatically and register it with the Container

we can only do it in the @Configuration class

@Bean will register it with the container

@Configuration

class MyConfig(){

@Bean

public className beanName(){

return new className();

}

}

the objects managed are called as Bean

the classes managed are called as Component

Spring Boot--

It is a spring framework module that offers auto-configuration resulting in Rapid Application Development

Configuration as in-

Dependency Injection Spring Context Module, Configuration Class, Property Source, etc, has to be configured

WebApplication Spring Web Module, DispatcherServlet, View Resolvers, etc, has to be configured

Add Security Spring Security Module, UserDetailsService, AccessRules, AuthenticationType, etc, has to be configured

Add Data Impl Spring Data JPA, Connection Pools, Transaction Management , etc, has to be configured

and so on....

Spring Auto-Configuration is about providing default configs for any spring module, eliminating the necessity of the repetitive bolier-plate code and that makes the application ready-to-cook out of the box adding to the productivity

Spring Auto-Configuration is possible due to '-starter-' dependencies. Each spring module is defined with a '-starter-' module that encapsulates the original dependency along with the default configs needed

Spring Boot also offers embedded servers. Servers like Tomcat can be embedded into the application as a dependency making Spring Boot Applications stand-alone. This makes the spring boot application well suited for Containerization

@SpringBootApplication = @Configuration

@AutoConfiguration

@PropertySource //application.properties and/or application.yaml

@ComponentScan //consider the package of the current class as basePackage

SpringApplication.run(ConfigClass.class, commandLineArgs)

the run method is the entire application box-

1.it creates a container (ApplicationContext)

2.invoke all Spring Runner Classes (if any)

3.start the server (if any)

4.waits until the server shuts down (if any)

5.destroys the container

A Spring Runner is any class that implements the below interfaces

CommandLineRunner interface offers public void run(String[] args)

ApplicationRunner interface offers public void run(Argument[] args)

Assignment1--AddressBook

We need a console based menu driven application to ADD/LIST/DELETE contacts where each contact has an id, fullName, dateOfBirth, mobileNumber, mailId as properties

class com.cts.adb.models.Contact

interface com.cts.adb.service.ContactService

Contact add(Contact contact)

void deleteById(int contactId)

List<Contact> getAll()

class com.cts.adb.service.ContactServiceImpl implements ContactService

is expected to a hypothetical list of contacts

class com.cts.adb.view.ContactsView

Spring Data--

It is a spring framework module that offers dynamic implementation of repositories

Spring Data JPA--

It is a sub-module of spring data that deals with repositories based on RDBMS

CrudRepository

|

|-JpaRepository<E,pkType>

E save(E);

Optional<E> findById(pkType id);

List<E> findAll();

boolean existsById(pkType id);

void deleteById(pkTypeId);

@Entity

@Table(name="contacts")

public class Contact {

@Id

@GeneratedValue(strategy=GenerationType.IDENTITY)

private Integer contactId;

private String fullName;

private String mobileNumber;

private String mailId;

private LocalDate dateOfBirth;

//.....

}

public interface ContactRepo extends JpaRepository<Contact, Integer>{

boolean existsByMailId(String mailId);

boolean existsByMobileNumber(String mobileNumber);

Optional<Contact> findByMailId(String mailId);

Optional<Contact> findByMobileNumber(String mobileNumber);

List<Contact> findAllByFullName(String fullName);

@Query("SELECT c from Contact c WHERE c.dateOfBirth BETWEEN :start AND :end")

List<Contact> findAllBornBetween(LocalDate start, LocalDate end);

}

Assignment2--Employee HR

We need a console based menu driven application to ADD/LIST/DELETE employees where each employee has an id, fullName, dateOfJoining, mailId, salary as properties

Use Spring Data JPA for persistence

class com.cts.adb.models.Employee

interface com.cts.adb.service.EmployeeService

Employee add(Employee employee)

void deleteById(int employeeId)

List<Employee> getAll()

class com.cts.adb.service.EmployeeServiceImpl implements EmployeeService

is expected to a hypothetical list of employees

class com.cts.adb.view.EmployeesView

Use custom methods in the Employee App also

Include the clause by functions for data retrieval in the repos.

Spring Web--

Dynamic Web Applications are applications that are capable of generating html content dynamically on the server and pass that content as a response to the client

Dynamic Web Application construction in JEE-

First was using servlets and JSPs.

Servlets were used to receive the http-request, parse the request and extract the data from the request and execute relevant operations and the final result of the operations were shared with JSP-pages

JSP-pages used to receive the data(model) from Servlets and embed the data in html-content and the final html-content rendered/generated is sent as a response

The above thing is called as Model-View-Controller (MVC) Architecture

Repositories <--entities--> Services <--model--> Controllers <---Request-----Client

| |

|(model) |

| |

| |

Views --------Response------>|

Controllers were servlets whose job is to parse the request and extract data from request and execute the relevant services and pass the result as model to the View

Views were JSP-page / JFC / XHTML / ThymeLeaf ...etc

The problem with MVC is we have dont have standards set, how many servlets we can have, how to modularize controllers, etc

Model-View-Controller FrontController Architecture is followed in Spring Web--

Repositories <--entities--> Services <--model--> Controllers <--model--> FrontController <---Request------Client

| | | |

|---modelAndView--------| |(model) |

| |

| |

Views --------Response------>|

FrontController is provided by Spring Web. DispatcherServlet from Spring Web acts as FrontController.

This DispatcherServlet as a FrontController is responsible to

1. Receive all requests from Client

2. Extract any data or parameters from the request

3. Identify the respective action (methods) in a Controller for the incoming requests

4. FrontController Invokes the identified action from the Controller and pass the extracted data as model to the Controller

5. FrontController Receive the resultant model (if any) and ViewName from the action

6. Share the resultant model with the view (that is identified by the viewName)

Views were JSP-page / JFC / XHTML / ThymeLeaf ...etc

Controller is not a Servlet in this architecture

Controllers are any POJO class marked with @Controller annotation

These Controller Classes are expected to have methods called actions

These action methods must have relevant arguments to receive data (model) from FrontController

These action methods must return a viewName as a string or viewName and model as an object of ModelAndView class which is a predefined class

Advantages-

Controller class can be tested easily

the boiler plate code is taken care by the FrontController

How would a FrontController identify a relevant action from a Controller for an Incoming Request?

Answer-- UrlHandlerMapping is an Interface

|

|-BeanNameUrlHandlerMapping (obsolete)

|-SimpleUrlHandlerMapping

SimpleUrlHandlerMapping-

It expects each action to be marked with @RequestMapping(url,httpMethod)

Based on the url and method of the incoming request, the relevant action is identified

For Example, assuming the server is running on localhost:8080

@Controller

public class HomeController{

@RequestMapping("/") /\* http://localhost:8080 any method like GET or POST will be called \*/

public string indexAction(){

return "index";

}

@RequestMapping(value="/about",method=HttpMethod.GET) /\* http://localhost:8080/about only for GET \*/

public String aboutUsAction(){

return "aboutUs";

}

}

How would a FrontController identify a VIEW for a given viewName?

ViewResolver is an Interface

|

|-BeanResourceViewResolver (we have to give a .properties file mapping viewName=ViewPath)

|-XmlResourceViewResolver (we have to give a .xml file mapping viewName=ViewPath)

|-InternalResourceViewResolver (default for SpringBoot)

InternalResourceViewResolver-

ViewPath = prefix + viewName + suffix (where prefix and suffix are configurable properties of InternalResourceViewResolver)

assuming prefix is '/WEB-INF/pages' and suffix is '.jsp'

for a given viewName 'index',

View is resolved as '/WEB-INF/pages/index.jsp'

DispatcherServlet is automatically configured by SpringBoot

@RequestMapping

|

|-@GetMapping

|-@PostMapping

The method parameter can be avoided by using above short hand mapping

@RequestParam used to retrieve one single parameter from the request, only available when the form is submitted

@ModelAttribute used to retrieve all data from a form into a java object, also used to pass data from controller to view

prepare New Contact page and perform operation

perform the same operation on employee entity

Thymeleaf--

The Thymeleaf is an open-source Java library that is licensed under the Apache License 2.0. It is a HTML5/XHTML/XML template engine. It provides full integration with Spring Framework.

Thymeleaf supports-

variable expressions (${...}) like Spring EL and executes on model attributes

asterisk expressions (\*{...}) execute on the form backing bean (modelAttributes)

hash expressions (#{...}) are for internationalization

link expressions (@{...}) are for rewrite URLs

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-thymeleaf</artifactId>

</dependency>

To activate thymeleaf on .html (HTML 4)

<html lang="en" xmlns:th="http://www.thymeleaf.org">

application.properties

spring.thymeleaf.cache=false

spring.thymeleaf.suffix=.html

Thymeleaf HTML attributes-

th:text

th:value

th:field

th:object

th:href

th:if

th:class

th:each="loopingVariable : ${arrayOrListOrSet}"

th:insert

th:replace

data-th-text

data-th-field

data-th-value

...etc, FOR HTML 5

For a deep reading:

https://www.thymeleaf.org/doc/tutorials/2.1/usingthymeleaf.html

controller advice means it will work for the entire application, like a global data supplier

spring aop - addressing cross cutting concerns

define a funntion that can execute before a function/after a function or both before and after the functions

example like maintaining log files for success or exceptions

we will define aspects for success and exceptions

https://www.baeldung.com/spring-aop

https://www.baeldung.com/spring-profiles

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-core</artifactId>

<version>5.3.27</version>

</dependency>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-context</artifactId>

<version>5.3.27</version>

</dependency>

[12:41 PM] Vamsy Kiran

<!-- https://mvnrepository.com/artifact/com.h2database/h2 -->

<dependency>

<groupId>com.h2database</groupId>

<artifactId>h2</artifactId>

<version>2.2.220</version>

<scope>test</scope>

</dependency>

=======================================================================================================

Spring Boot--

Objective-

1.Spring Boot

2.Profiling

3.Testing

4.Rest Api

Pre-Requisites-

1.Spring IOC / DI

2.Spring Expression Language and Application Context

3.Autowiring and External Value Injection

4.Spring Data / Spring Data JPA

Lab Setup-

1.STS latest IDE

2.JDK 1.8

3.Maven

4.Spring Boot 2.x

5.MySQL (or other RDBMS) / H2DB

Introduction--

Spring Boot is one of the modules of Spring Framework that offers RAD as its primary feature

RAD stands for Rapid Application Development through auto-config

For each spring module, a new '-starter-' module is created

This '-starter-' is a conjunction of the module dependencies and default configs

Spring Boot supports embedded servers. This feature was proposed to support containerization.

Dependency Injection--

the word dependency means If an artifact (say A) of an application is functionally dependent on another artifact (say B), then A is said to depend on B or B is a dependency for A

Supplying dependencies into the site (place of injection) externally is called dependency injection

service needs dao so dao is dependency and service is the site

if the lifecycle of an object is managed then its a Bean otherwise its an object

the software that manages the lifecycle of a bean is a container

container is going to create, inject, destroy a bean

class in which a bean is created is called a component class

@SpringBootApplication = @Configuration

@ComponentScan

@PropertySource

@AutoConfiguration

SpringApplication.run(ConfigClass.class,commandLineArgs);

When run method is invoked-

1.ApplicationContext is created (Container is created)

2.CommandLineRunner and ApplicationRunner are invoked (if any)

3.Server gets started (if any)

4.The mainExecutor is held until the server shuts down (if any)

5.Container is closed and destroyed

we can have multiple runner classes and it will be executed alphabetically

Creating a Spring Boot Application--

Using 'Spring Starter Project' option from STS

Using SpringInitializr at 'http://start/spring.io'

Using Spring Boot CLI

Spring Data--

It is a spring framework module that offers dynamically implemented repositories

Spring Data JPA--

It is a sub-module of Spring Data that provides implementations based on JPA

JPA is an ORM mapping for talking to RDBMS databases

CrudRepository

|-JpaRepository<E,primaryKeyType> - an interface

|-List<E> findAll()

|-Optional<E> findById(primaryKey)

|-boolean existsById(primaryKey)

|-E save(entity)

|-void deleteById(primaryKey)

@Entity

@Table(name="contacts")

public class Contact {

@Id

@GeneratedValue(strategy=GenerationType.IDENTITY)

private Integer contactId;

private String fullName;

private String mobileNumber;

private String mailId;

private LocalDate dateOfBirth;

//.....

}

interface ContactRepository extends JpaRepository<Contact,Integer>{

boolean existsByMobileNumber(String mobileNumber);

Optional<Contact> findByMobileNumber(String mobileNumber);

List<Contact> findAllByFullName(String fullName);

void deleteByMobileNumber(String mobileNumber);

@Query("SELECT c FROM Contact c WHERE c.dateOfBirth BETWEEN :start AND :end")

List<Contact> getAllBornIn(LocalDate start, LocalDate end);

}

create database addb;

Assignment--

Develop a spring boot and spring data jpa powered

console application to ADD/List/DELETE an Employee where

Employee entity has empId, fullName, dateOfJoining, mailId and salary as fields.

Spring Web--

Spring Web is a Spring Framework module that helps in developing Dynamic Web applications based on MVC architecture and for developing Rest API as well

MVC stands for Model View Controller

handle the workflow of the application for a particular request

JSP is for views

Controller and Views are going to exchange data through Models

MVC Architecture--

Repos <--entities--> Services <--models--> Controllers <----------Request

|

| model

|

|

Views ---------->Responses

Servlets are used as Controllers

JSP/JFC/ThymeLeaf are used as Views

MVC FrontController Architecture

Repos <--entities--> Services <--models--> Controllers <--model--> FrontController <----------Request

|

| model

|

|

Views --------------------------------------->Responses

FrontController DispatcherServlet

Controller is any POJO that has actions mapped to specific urls

JSP/JFC/ThymeLeaf are used as Views

REST Api--

REpresentational State Transfer - REST

It is a modern model of Web Services where instead of SOAP protocol, http protocol itself is used to publish and consume web services

<------> UI For Desktop

DataBase <------> WebServices <------> UI For AndroidMobile

<------> UI For IOSMobile

Disadvantages of SOAP-

only xml data can be used

no standard exchange system

no support of media type of data

REST Api is Web Services hosted on http protocol deriving the advantage of Standrd Exchange System and have support for a large variety of media (including JSON and XML)

REST Api standards (taking the advantage of http protocol)--

In Rest Api, a single endPoint (URL) is enough to manage all CRUD operations, thanks to the HttpMethod system in http protocol

/emps is the endPoint if Employee is a resource assumed

Operations http Method endPoint reqBody respBody Success Failure Failure

Status Status @Client Status @Server

------------------------------------------------------------------------------------------------------------------------------------------

Create POST /emps emp(json/xml) emp(json/xml) CREATED-201 400-BAD REQ 500-Internal Server Error

Retrieve A Record GET /emps{empID} NONE emp(json/xml) OK-200 404-Not Found 500-Internal Server Error

Retrieve All Records GET /emps NONE emp[](json/xml) OK-200 404-Not Found 500-Internal Server Error

Update PUT /emps emp(json/xml) emp(json/xml) ACCEPTED-202 400-BAD REQ 500-Internal Server Error

PATCH /emps emp(json/xml) emp(json/xml) ACCEPTED-202 400-BAD REQ 500-Internal Server Error

Delete DELETE /emps{empID} NONE NONE NoCONTENT-204 404-Not Found 500-Internal Server Error

Patch is for updating specific property in the record

Put is for updaing entire record

Http Status Code System--

1xx --not used by developers, signal indicating that the request is received and is being processed, between browser and server

2xx --signals that a request is successfully processed and a response is sent

3xx --signal redirections

4xx --signal errors due to client side issues

5xx --signal errors due to server side issues

Spring Web Rest Api--

Controller -- a holder of actions

@RestController -- the views are directly models

@RestController = @Controller

@ResponseBody (return models)

@RequestMapping(value="endPoint",method=HttpMethod)

@RequestMapping()

|-@GetMapping("endPoint")

|-@PutMapping("endPoint")

|-@PostMapping("endPoint")

|-@DeleteMapping("endPoint")

@RequestBody -- will automatically read the request and put it in the model object

@Consumes -- consuming the media type (pre-configured for json and xml)

@Produces -- producing the media type (pre-configured for json and xml)

@Valid -- check internet

@RestControllerAdvice -- class which is intended to handle exceptions

@ExceptionHandler -- to be put in the methods of rest controller advice class

spring boot devtools -- monitor the entire source code and whenever the source code changes, it restarts our development server automatically

server.port=0 -- it will take random port number

response entity -- encapsulation of request body and http status

BuildFactory design pattern -- builder classes

Assignment--

Rest-api for an ecommerce application Vendor Module:

Using this rest-api, a UI application must be able to

1.Register a Vendor

2.Raise a Purchase Order for the Vendor

3.A vendor can have any number of Purchase Orders

4.Each Purchase Order has a OrderNumber, OrderDetails (string), OrderedDate, OrderValue and Status

5.The default status of any Purchase Order is 'BOOKED'

6.The status can change to 'SERVED' or 'CANCELLED' or 'DROPPED' (use PATCH request)

7.One must be able to retrieve a list of Booked Orders, Cancelled Orders, Dropped Orders and Served Orders Vendor wise

8.One must be also able to retrieve the list of Vendors

create own method in repo

custom model

path variable

link vendor and order

@Modifying("update query")

public Order updateOrderStatus(@Param)

[2:49 PM] Vamsy Kiran

@Modifying

@Query("update User u set u.active = false where u.lastLoginDate < :date")

void deactivateUsersNotLoggedInSince(@Param("date") LocalDate date);

[2:50 PM] Vamsy Kiran

https://www.baeldung.com/spring-data-jpa-modifying-annotation

Spring Data JPA @Modifying Annotation | Baeldung

Create DML and DDL queries in Spring Data JPA by combining the @Query and @Modifying annotations

Spring Profiles

---------------------------------------------------------------------------------------------------

Profile?

an isolated set of configuaratiosna and choice of components for executing our application in specific

stage of project life cycle and/or environment.

dev,testing,deployment/production ...etc.,

@Profile("")

is apllied on a component or a configuaration class.

@Component

publi class DummyComponent{

@Autowired

@Qualifier("projectOwner")

private String projOwner;

}

@Configuaration

@Profile("dev")

public class Config1 {

@Bean

public String projectOwner(){

return "Vamsy Kiran Aripaka";

}

}

@Configuaration

@Profile("test")

public class Config2 {

@Bean

public String projectOwner(){

return "Sagar";

}

}

@Configuaration

@Profile("prod")

public class Config3 {

@Bean

public String projectOwner(){

return "Nimal Kumar";

}

}

application.properties

spring.profiles.active=dev

application-dev.properties

application-test.properties

application-prod.properties

application.properties

spring.profiles.active=dev

spring.jpa.show-sql = false

spring.jpa.hibernate.ddl-auto = update

#---

spring.config.activate.on-profile=dev

server.port = 8888

spring.datasource.url = jdbc:h2:mem:db

spring.datasource.driver-class-name=org.h2.driver

spring.datasource.username = sa

spring.datasource.password = sa

#---

spring.config.activate.on-profile=prod

server.port = 9999

spring.datasource.url = jdbc:mysql://localhost:3306/hrappdb

spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver

spring.datasource.username = root

spring.datasource.password = root

spring.jpa.properties.hibernate.dialect = org.hibernate.dialect.MySQL8Dialect

=============================================================================================

Microservices

-----------------

Pre-Requisites-

Spring Framework

Spring Core, Spring Context

Spring EL

Spring Boot

Spring Web(Web MVC) & REST Api

Spring Data JPA

Lab SetUp-

JDK 1.8

STS 4

Maven

MySQL Community Server 8.0

Introduction--

Monolithical Architecture-

1.The entire application is in one deployment-unit

2.The entire application is modularized through layers

3.Layers are going to offer loosely coupled components

The user expectations have gone beyond the Monolithical Architecture

The following are the problems of Monolith-

1.Load Management via Scaling

2.Modules cant be separated and scaled separately

3.Interoperability-technical upgradation of modules is tough (if dao is jdbc, then like convert to orm/jpa)

4.Management-size of the monolith application makes running the test cases bulkier

Microservices--

1.A microservice is one isolated module of an application that is independently managed as a deployment unit

2.A group of inter-communicating microservices form an application eco-system

3.Load Management via Scaling

4.Interoperability - can be developed using multiple technologies but should be Rest Api based

5.Management-managing the application is easier

Challenges of Microservices-

1.Decomposiition-moving from monolithical to microservices is tough

2.Inter Communication-communication between different modules,due to addresses of the servers where each microservice is deployed

3.Distributed Tracing-tracing as part of debugging process, but there are different modules in microservies so tracing becomes challenging and that's why we have distributed tracing

4.Fault Tolerance (Fall Backs)

5.Monitoring and Configuration

Microservices Design Patterns--

Decomposiition Design Patterns

Decomposiition by Domain -- divide your each functionality module of monolithic to microservice module based on basic idea of module

Decomposiition by Sub-domain -- context bounded-distinct context where all terms and entities have a clear and unambiguos meaning,

god classes-A god class is a class which is present everywhere in the application, no need to repeat the properties

Integration Design Patterns -- one single unit of ecosystem

Api Gateway Pattern -- giving single toll number for all issues, it will be redirected to the concerned department upon dialing

Aggregator Pattern -- everything related will be given as a single pack

Client Side Component Pattern -- not in scope, whatever will be ready will be served first, related to angular/react

Database Design Patterns

Database Per Service -- each microservice shall have its own dedicated database, for greenfield projects

Shared Database -- only one single db for entire ecosystem, in case of db cannot be segregated

Saga Pattern -- later

CQRS Pattern -- later

Observability Design Patterns

Log Aggregator -- logging that is generated in diff microservices should be collectively maintained at a central location

Performance Metrics Aggregator -- measure the performance of the application, the metrics like response time,threshold,etc should be collected

from diff microservices and send to a central location

Distributed Tracing -- log aggregator + performance metrics aggregator, each and upcoming request is given a unique request identification

number, when the request is forwarded from one microservice to another microservice, using this id's we can monitor the request and send to a cental microservice and reported

Cross Cutting Design Patterns

Discovery Service -- is another microservice for helping in inter communication of microservices, it will maintain a register, each time a

microservice comes up, the server address and other details will be stored in the register and each time, the microservice will hit the discovery service and find the status of the microservice and then hit it

Circuit Breaker -- concept of fallback will come here, a visitor came from diff location to meet us, he is going to approach security desk and

confirm us about the visitor before allowing. when the visitor has come but employee is busy for 1 hour and so the visitor has to wait or go back, for same employee, another visitor has come in the same duration, so the security no need to check with the employee again till the time employee is busy, when the employee got free after 2 hours, another visitor comes so the security will ask to the employee again. Microservice A needs to check with Microservice B before it answers a request, maybe Microservice B not running at that time so Microservice A will go into a fallback mechanism with a threshold which can be number of request (10 request will have same fallback) or the time duration (10 mins it will give same fallback). this concept is called circuit breaker

External Configuration -- it is going to all managing microservices configuaration extternally

fresh project--greenfield projects

existing project-brownfield projects

A Case Study--

Budget Tracker

1.Each user is represented as an AccountHolder

2.Each AccountHolder can save a transaction into the system

3.A Transaction can be either CREDIT or DEBIT

4.The system must be able to generate Periodic statements

Microservice Design of Budget Tracker--

Decomposiition by Domain-

Profiles -- A microservice for User Management

Txns -- A microservice for Transaction Management

Statement -- A microservice for generating Statements

Decomposiition by Sub-Domain-

Identify the God Classes and Define Bounded Contexts for them

Profiles -- A microservice for User Management

AccountHolderEntity -- the god class here

int ahId --bounded context

String fullName --bounded context

String mobile --bounded context

String mailId --bounded context

Txns -- A microservice for Transaction Management

AccountHolderEntity -- the god class here

int ahId --bounded context

double currentBalance --bounded context

List<Txn> txns --bounded context

TxnEntity -- the god class here

int txnId --bounded context

double txnAmount --bounded context

double txnType --bounded context

LocalDate txnDate --bounded context

AccountHolderEntity holder --bounded context

Statement -- A microservice for generating Statements

StatementModel

AccountHolderModel details

List<TxnModel> txns

LocalDate startDate

LocalDate endDate

double totalCredit

double totalDebit

AccountHolderModel

ahId

fullName

mobile

mailId

currentBalance

TxnModel

int txnId

double txnAmount

double txnType

LocalDate txnDate

Database Per Service-

profiles-service txns-service stmt-service

| |

profilesDB txnsDB

Aggregator Pattern- Internal Communication

when a group of data is required, user should send request to one microservice

profiles-service

going to send subsequent requests to profile and txn microservices<--------stmt-service<-------REQ(for statement)----Client

txns-service

profiles-service <-------=-------- |

---profileData---> | | <-----REQ (for statement)<---------Client

| stmt object --> stmt-service|

txns-service <---------------- | generated | ----->RESP (aggregated statement)---->Client

---txnsList------> |

Api Gateway Pattern-it is also a microservice

responsible to receive any kind of request and response from the client, external communication till statement service and then internal communication from statement service to profile and txns microservices

client

up-arrow

| |

| |

| |

| |

down-arrow

api-gateway

||

-------------------------------------------------------------------------------------

|| || ||

|| || ||

profiles-service txns-service stmt-service

| |

profilesDB txnsDB

Distributed Tracing-

client

up-arrow

| |

| |

| |

| |

down-arrow

api-gateway

||

-------------------------------------------------------------------------------------

|| || ||

|| || ||

profiles-service txns-service stmt-service

| |

profilesDB txnsDB

tracingClient tracingClient tracingClient

|| || ||

|| || ||

-------------------------------------------------------------------------------------

||

tracing-service

||

tracingInfoDB

Every microservice will have a Component called as TracingClient, record the data of all the requests

tracing-service --another microservice,may have its own database

Discovery Service-

one microservice cannot talk to each other because they are dynamic in nature

It will maintain registry of service locations

api-gateway will also depend on discovery service to know the location of any microservice

client

up-arrow

| |

| |

| |

| |

down-arrow

api-gateway <--------------------------------------> discovery-service(registry of service locations)

|| ||

---------------------------------------------------------------------------------------------

|| || ||

|| || ||

profiles-service txns-service stmt-service

| |

profilesDB txnsDB

tracingClient tracingClient tracingClient

|| || ||

|| || ||

-------------------------------------------------------------------------------------

||

tracing-service

||

tracingInfoDB

Circuit Breaker-it doesnt create any microservice, its only implementation

External Configuaration-

It provides us with another microservice called ConfigService, its attached to a file repository

we will provide all our configuration properties files here

when application is starting, it will hit the config service which will have bootstrap properties in the properties files for the startup

client

up-arrow

| |

| |

| |

| |

down-arrow

api-gateway <--------------------------------------> discovery-service(registry of service locations)

|| ||

---------------------------------------------------------------------------------------------

|| || ||

|| || ||

profiles-service txns-service stmt-service

| |

profilesDB txnsDB

tracingClient tracingClient tracingClient

|| || ||

|| || ||

-------------------------------------------------------------------------------------

|| ||

tracing-service config-service

|| ||

tracingInfoDB fileRepo(git/...)

profiles-service.properties

txnx-service.properties

stmt-service.properties

api-gateway.properties

OAuth Authentication Module, JWT Token -- check internet, it will be handle by api-gateway

open feign - third party dependency for inter service commuication

rest template - has a lot of boiler plate code so open feign is taken to avoid boiler plate code, all the things will be automated

rest template for inter service as well as inter application communication

Step#1 implementing decomposed services and do inter-service communication and aggregator

in.bta:bta-profiles

dependencies

org.springframework.boot:spring-boot-starter-web

org.springframework.boot:spring-boot-devtools

org.springframework.cloud:spring-cloud-openfeign

mysq1:mysql-connector-java

org.springframework.boot:spring-boot-starter-data-jpa

configuaration

spring.application.name=profiles

server.port=9100

spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver

spring.datasource.username=root

spring.datasource.password=root

spring.datasource.url=jdbc:mysql://localhost:3306/bapsDB?createDatabaseIfNotExist=true

spring.jpa.hibernate.ddl-auto=update

in.bta:bta-txns

dependencies

org.springframework.boot:spring-boot-starter-web

org.springframework.boot:spring-boot-devtools

org.springframework.cloud:spring-cloud-openfeign

mysq1:mysql-connector-java

org.springframework.boot:spring-boot-starter-data-jpa

configuaration

spring.application.name=txns

server.port=9200

spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver

spring.datasource.username=root

spring.datasource.password=root

spring.datasource.url=jdbc:mysql://localhost:3306/batxnsDB?createDatabaseIfNotExist=true

spring.jpa.hibernate.ddl-auto=update

in.bta:bta-statement

dependencies

org.springframework.boot:spring-boot-starter-web

org.springframework.boot:spring-boot-devtools

org.springframework.cloud:spring-cloud-openfeign

configuaration

spring.application.name=statement

server.port=9300

bta-profiles--account holders record on the db

bta-txns--transactions record on the db

https://github.com/avamsykiran/SpringBootMicroservices\_20Mar24\_12Apr24\_2to4.git

ChildVaccinationTracker

-----------------------------------

1.Each Parent must register with fatherName,motherName,mobile,mailId

2.A Parent may have multiple children

3.Each child must be registered with childName,gender,dateOfBirth

4.Each time a child is given a vaccine, we need to record that info as vaccineName,dateOfAdministration,dosage

5.A child can be administered with multiple vaccines or multiple times

6.Generate a report of vaccines given to a child for a given childId

7.Generate a report listing all the children who are administered a specific vaccine

load balancer client -> discovery client -> eureka server

load balancer follows round robin algorithm

always start discovery server first

api gateway -> discovery service -> resp microservice

always start discovery followed by gateway

cohesion--single unit of code for similar dependecies and coupling

cohesion must be high and coupling must be low

Single Responsibility Principle--

Each class should be responsible for only task

Open Closed Principle--

units of code should be open for extension but closed for modification

Liskov Substitution Principle & Interface Segregation Principle--

interface Vehicle{

void accelerate();

}

interface Stoppable{

void applyBreaks();

}

interface Ignitable{

void turnOnEngine();

}

class BullockCart implements Vehicle{

public void accelerate(){

}

}

public MotorCar implements Vehicle,Ignitable,Stoppable{

public void turnOnEngine(){

}

public void accelerate(){

}

public void applyBreaks(){

}

}

class ElectricCar implements Vehicle,Stoppable{

public void accelerate(){

}

public void applyBreaks(){

}

}

Dependency Inversion Principle--

you dont call us, we will call you

solid principles design patterns--

structural and behavioral design patterns

vamsy.kiran@iiht.com

put all the application.properties of diff microservices in btaconfigrepo folder and rename them accordingly

then do git init in btaconfigrepo

then do git add and git commit

external config

circuit breaker

bear minimum are the 6 steps

aes algorithms

@json property write only

inner join using jpql query

=====================================================================================

Java SE 8--

Objectives-

1 Introduction to Java

2 Classes and Objects

3 Methods, Encapsulation, and Conditionals

4 Data Manipulation and Inheritance

5 Encapsulation, Polymorphism, and Abstraction

6 Interfaces, Lambda Expressions, Collections, and Generics

7 Inheritance, Interfaces, Exceptions, and Deployment

8 Date/Time API, I/O and File I/O (NIO.2), and Concurrency

9 Concurrency, Parallelism, The JDBC API, and Localization

10 JDBC

Introduction to Java--

History-

James Gosling, Sun Microsystems-taken over by Oracle

Jave is shipped in two main parts - JDK and JRE

JDK-Java Development Toolkit (all the java libraries, java source code, java tools, JRE)

JRE-Java Runtime Environment (class loaders, JIT, JVM, GC, etc)

Evolution-

Java SE- Java Standard Edition (Core Java) - Serve as the very base and support standalone application development

Java EE- Java Enterprise Edition - Supports developing Distributed Applications

Java ME - Java Mobile Edition - Supports developing Mobile based Java Application

Java 1.1, Java 1.2(Java 2), Java 1.4(Java 4), Java 1.5(Java 5), Java 1.8(Java 8), Java 1.11(Java 11), Java 1.17(Java 17), Java 1.21(Java 21)

Features-

Simple - minimum keywords

Robust - very good exception handling

Distributed - java ee edition

MultiThreaded - already available

Self Documented - good documentation technique

Platform Independent - check below

Architecture Neutral - can be developed in any system

JRE contains JVM which runs the class file which makes the Java platform independent

JDk, JRE, JVM are not platform independent

But the compiled codes are platform independent

Characteristics-

Case sensitive

Each statement terminates with a semi-colon (;)

Each block of statements are enclosed in a pair of curly-braces {}

Standards or Coding Conventions-

Tokens-

a smallest unit of a program that we are going to write

based on tokens we are going to define syntax

Tokens-

Identifiers-is a name of a variable / object/ user defined datatype/ method / etc

Variable Identifier or a Method Identifier or a Object Identifier

must start and continue in lowercase and if it comprises of more than one word,

each word from the second word onwards must be initial capitalized

Examples-

length

boxLength

equals()

toString()

empObj

contractEmpObj

UDT (classes, enums, interfaces, annotations) Identifier

must follow initial capitalization

Examples-

Exception

RuntimeException

ArrayIndexOutOfBoundsException

Constant Identifier

must be fully capitalized and must use '\_' underscore to separate words

PI

MAX\_LIMIT

MIN\_LIMIT

Keywords-

reserved words / vocabulary of the language

most of the keywords are lowercase

designed to identify control structure, data type of a variable, etc

Comments-

End-Of-Line Comment- used to describe a declaration

Examples-

int empNo; //represents employee number

Block Comment- used to describe a method or a class or an algorithm

/\*

\*The swap method will swap the values of the given params

\*params: obj1 and obj2

\*return nothing

\*/

public void swap(Object o1, Object o2){

Object t=o1;

o1=o2;

o2=t;

}

Operators-

Binary Operators-

Arithmetic + - \* / %

Assignment + += -= \*= /= %=

Relational == < > <= >= !=

Logical && || !

Unary Operators-

Unary Minus a = -b;

Increment ++

Decrement --

Conditional Operator- cond ? exp\_to\_evaluate\_if\_true : exp\_to\_evaluate\_if\_false;

a = x>y ? x : y;

Miscellaneous Operators- ; , () [] {} instanceof

Bitwise Operators- & | ^ >> <<

Literals-

===============================================================

REACT JS

-----------------------------------

Lab Setup--

Node JS

VS Code

Chrome

Pre-Requisites--

HTML 5

CSS 3

Bootstrap 5

JavaScript (ES6)

ReactJS Intro--

It is a javascript based Single-Page-Application Framework

Static Website

It is a folder of pre-scripted html documents hosted on a http-server.

Dynamic Web Application

HttpServer Browser

ServerSidePrograms

(Servlet & JSP/ASPX/PHP, etc)

<-----------------REQ1--------------------------------------

based on the data in the request, an html

content is generated on the go ("dynamically")

------------------(generated html content)RESP1------------>

<-----------------REQ2--------------------------------------

based on the data in the request, an html

content is generated on the go ("dynamically")

------------------(generated html content)RESP2------------>

Single Page Application

HttpServer Browser

spa-bundle

(index.html + \*.js)

<---------------REQ-----------------------------------

---------------(spa-bundle)RESP----------------------> index.html along with the JS is loaded

any event or form submissions or links, etc are all

handled by the JS on the client itself

JS on the client can generate html content dynamically

on the client side and can replace the index.html content

as and when needed

rest-api <-------json/.xml------------------------> spa-bundle

SPA-Frameworks

Angular

ReactJS

VueJS

...etc

ReactJS Features

1.Html Extendability (we can create our own html-elements and html-attributes using ReactJS)

2.State Binding (the changes to the data or state is monitored and the output on the screen is updated)

3.Shadow DOM/Virtual DOM

4.Server Side Rendering/Differential Loading

Creating a ReactJS application

npx create-react-app app-name

npm install -g create-react-app (if above command not working)

followed by create-react-app app-name

ReactJS Components

The html-elements created in ReactJS are called ReactJS Components.

A Single Page App is made up of a stack of these Components

index.html

<html>.....</html>

| - <head>...</head>

| - <body>...</body>

|- <div id="root">...</div>

|- <App>...</App> (Top Level Component)

|- <Component1 />

|- <Component2 />

|- <Component3 />

|- <Component4 />

|- <Component5 />

|- <Component6 />

We can create components in 2 different ways--

Function Components

It is any javascript function that accepts 'props' as argument and return one and only one Html-Element.

We use JSX for scripting. Function Components are also called as state-less components.

cont ComponentName = (props) => HtmlElement;

const TitleBar = () => <h1>My First ReactJS App</h1>;

<TitleBar />

const TitleBar2 = function(){

return <h1>My First ReactJS App</h1>;

}

<TitleBar2 />

Class Components

It is any sub-class of React.Component class.

From React.Component, a Class Component inherits

1.state is used to hold all the data related to the component

2.setState() is a method used to change state

3.render() must be overridden to return the html-element that has to be displayed for this component

4.componentDidMount()

5.componentDidUpdate()

6.componentWillUnMount()

7.and a few more life cycle methods

class EmployeeList extends React.Component{

constructor(props){

super(props);

this.state={}; //initial data into the state

}

render(){

return(

<table>

<thead>

</thead>

<tbody>

</tbody>

</table>

);

}

}

<EmployeeList />

JSX?

JavaScript Extended. It is an amalgamation script of html with javascript to facilitate easier html manipulation.

JS

let pObj=document.createElement("p");

pObj.innerText="This is a sample content of the para";

JSX

let pObj=<p>This is a sample content of the para</p>;

JS

let userName="Vamsy Kiran";

let pObj=document.createElement("p");

pObj.innerText="Hello! " + userName + "! Welcome Here...!";

JSX

let userName="Vamsy Kiran";

let pObj=<p>Hello! {userName} Welcome Here...!</p>;

JS

let friends=["Swathi","Suma","Sudha","Sita"];

let olObj=document.createElement("ol");

for(let i=0;i<friends.length;i++){

let liObj=document.createElement("li");

liObj.innerText=friends[i];

olObj.appendChild(liObj);

}

JSX

let friends=["Swathi","Suma","Sudha","Sita"];

let olObj=(

<ol>

{friends.map(f=><li>{f}</li>)}

</ol>

);

JSX Rules

1.'class' attribute is not to be used, 'className' attribute is used instead

2.All html-element tag-names must be lower in case

3.All attributes must follow camelCase

4.All ReactJS Component must be InitialCapitalized

ReactJS Component State

1.'state' is used to hold all the data related to a component

2.'state' is immutable, we cannot go and modify a state directly

3.'state' is only replaceable using 'setState()' method

4.Each time 'state' is replaced, the 'render()' method is invoked

npm start -- to start a react app

Shadow DOM

Each time when an iterative collection of data in state gets modified, regenerating the entire DOM is a costly process

Instead, ReactJS maintains a temporary copy of the DOM called Shadow DOM. Actual DOM is directly linked to the screeen whereas

the Shadow DOM has no link with the screen

This means modifying the shadow dom is less costlier than modifying the actual DOM

Now changes are first made on shadow DOM and after all the state is placed on shadow DOM, the shadow DOM is super imposed on

the actual DOM and the actual DOM will have only the final changes

To facilitate the comparison between the shadow DOM and actual DOM, each element iterated is expected to have a unique 'key'

Integrating ReactJS with Bootstrap

npm i bootstrap

bootstrap.min.css and bootstrap.bundle.js must be imported into index.js

props is used for receiving the data from parent to child

a parent can pass data through attributes

the child receives the attributes through props

state is immutable, we cannot modify state directly so we use setState

we have to put/maintain the data in a state so we use class component

we cannot store data using function component

Controlled Components are used in app03

Uncontrolled Components in ReactJS, In case of uncontrolled components, we use a reference

Working with Forms - what is the difference between controlled and uncontrolled components

Component Lifecycle Methods-

constructor() receive the 'props' and initialize the 'state'

|

|

render() returns the DOM to be displayed for the current component

|

|

componentDidMount is used to execute an operation after the first render (like rest-api calls)

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

whenever setState() is called...

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

|

|

render()

|

|

componentDidUpdate() is used to execute an operation after each render except the first time

componentWillUnMount() is used to execute an operation just before the component is unmounted

when one calculation is dependent on another calculation, it is called side effect

this is handled by componentDidUpdate

do not call setState() without a condition in componentDidUpdate

ReactJS Hooks-

A Hook is a special function used to add features to a function component

function components dont have state and lifecycle methods

Some of the Hooks are defined below-

useState is a hook that allows a function component to have a state

let [reader,writer] = useState(initialValue);

let [x,setX] = useState(0);

'x' is used to read the value and 'setX()' is like a method that can be used to to change the value

useEffect is a hook that provides lifecycle methods to a function component

useEffect(callBack, dependencyArray)

if the dependencyArray is empty, the callBack is executed only once, that too after the first render

useEffect( ()=> {

//this is equivalent to componentDidMount

}, []);

if the dependencyArray is not empty, the callBack is executed after every render each time the fields in the array

change their value

useEffect( ()=> {

//this is equivalent to componentDidUpdate

}, [x,y]);

function components are easy to test and less of code to write

98% people are using function components only

Employee- id, fullName, salary, mailId

Application Level State Management - Redux

Redux is a third party library , used to isolate state management from the UI-Components

npm i redux

Store is a object that holds all the data related to an application

one application has only one store

const store = createStore(reducer)

reducer is a function that accepts an action, and existing state, and returns the

modified state as per the given action

const reducer = (currentState,action) => {

//create a new modified state based on the action and the currentState

return modifiedNewState

}

action is an object that has two fields 'type' and 'payload'

const action1 = {type:"ADD\_EMP", emp:{id:1,fullName:"Vamsy",salary:45000}}

const action2 = {type:"DEL\_EMP", empId:101}

dispatch is a method provided by 'redux'. it is used to throw an action from a ui-component into the reducer

dispatch(action) ----------> reducer(store.state,action)

npm i react-redux

react-redux is an integration library between redux and reactjs

useSelector is a hook provided by 'react-redux'. it is used to select a part of the state

into a component. useSelector is bound to the state and hence everytime the state

is modified, useSelector will automatically update the component too

let empsList = useSelector( state => state.emps );

useDispatch is a hook that gives access to 'dispatch' method

const dispatch = useDispatch();

Provider is a component from 'react-redux' that is used to wrap the store on the 'App' component

<Provider store={store}>

<App />

</Provider>

reducer and action to be written by developer

store-------------------> ----(useSelector)----------

| | |

| | |

| Component1 |

| | Component2

| | |

| dispatch(action) |

| | dispatch(action)

| | |

|(modified state) | |

| | |

reducer <----------(action)------------------------

actions are the standards/guidelines for keeping the reducer and the component on the same page

Application Flow--action->reducer->store->component->action...

Create a fake rest-api using json-server

json-server is a javascript tool that generates fake rest-api from a .json

md adb-api

cd adb-api

npm init -y

npm i json-server

create adb-api/data.json

change the 'start' script in the package.json as 'json-server --port 9999 --watch ./data.json'

npm start

Calling rest-api using axios

npm i axios

axios.get(apiEndPoint) : Promise

axios.post(apiEndPoint,reqBody) : Promise

axios.put(apiEndPoint,reqBody) : Promise

axios.delete(apiEndPoint) : Promise

axios.get("http://localhost:9999/contacts")

.then( data => { } )

.catch( err => { } );

Integrating axios calls with redux reducer through redux-thunk (middleware)

'thunk' means a function that returns another function

while integrating rest-api calls with redux, the rest-api calls are managed by an actionFunction

Here, actions are not objects but they are functions

npm i redux-thunk

'thunk' is applied on the reducer using 'applyMiddleware' method on the store

const store = createStore(reducer, applyMiddleware(thunk));

We use action as object if the operation is local (like markEditable or unMarkEditable)

We use action as function (thunkAction) if the operation is a rest-api call (like add or update or delete)

store-------------------> ----(useSelector)----------

| | |

| | |

| Component1 |

| | Component2

| | |

| dispatch(thunkAction) |

| | dispatch(thunkAction)

| | |

|(modified state) | |

| | |

reducer <----------(action)---------------------|------------[thunkAction]---------------|

| | |

|<------(actionSignallingWait)----------------| dispatch(waitAction) |

| | call to rest-api |

|<------(actionCarryingData)------------------| (a)get data, dispatch(refreshAction) |

|<------(actionCarryingErr)-------------------| (b)get error, dispatch(errAction) |

| |----------------------------------------|

React Routing

npm i react-router react-router-dom

<BrowserRouter>

<Header />

<Routes>

<Route path="/" element={<C1 />} />

<Route path="/a" element={<C2 />} />

<Route path="/b" element={<C3 />} />

<Route path="/c/:pathVariable" element={<C4 />} />

</Routes>

<Footer />

</BrowserRouter>

<Link to=""></Link> instead of <a></a>

<Navigate to="" />

controlled forms are used nowadsys because uncontrolled use references which doesnt sit well with React's single instance of truth

state management, creating multiple reducers

combined reducers, ognl state management

Next.js go through now

create multiple entities react project

=====================================================================================================

Angular Intro

------------------------------------

It is a javascript based Single-Page-Application framework.

Static Website

It is a folder of pre-scripted .html and .css documents hosted on a http-server.

Dynamic Web Application

'dynamic' here refers to that the html content is not pre-scripted but rather is generated on the fly.

HttpServer Client/Browser

Server Side Programs

(servlets/jsp/asp/aspx/php,etc)

<-------------------REQ1----------------------------------------

is received by a program, and after processing the

data in the request, some dynamic html content is generated

-----------------------(html-content)RESP----------------------->

<-------------------REQ2-------------------

is received by a program, and after processing the

data in the request, some dynamic html content is generated

-----------------------(html-content)RESP----------------------->

Single Page Application

Http Server Client/Browser

spa-bundle

(index.html+\*.js)

<-------------------------REQ-----------------------

----------------(spa-bundle)RESP-------------------> index.html is loaded along with .js

the JS will execute on the client machine and any events

including links/form-submissions,etc are all handled by the

JS on the client

the JS can generate html content dynamically as needed and

can replace the content of the index.html as needed

rest-api <-----------------json/.xml------------> spa-bundle

SPA-Frameworks

AngularJS

Angular

ReactJS

VueJS

...etc

AngularJS vs Angular

AngularJS uses JavaScript for scripting

Angular supports Typescript (Angular 2 to Angular 18)

TypeScript

It is a superset of JavaScript. JavaScript plus type-safety is TypeScript. TypeScript is not understood by any browser but scripting using typescript will reduce the chance typing errors and eliminates a lot of runtime debugging.

.ts must be compiled in .js before it is executed on the browser. And this process is called transpiling(compiling).

Lab Setup

NodeJS above 18.x node --version https://nodejs.org

Angular CLI ng version npm i -g @angular/cli

VSCode as an IDE

NodeJS?

It is a javascript runtime. alternative to a browser to execute javascript. NodeJS does not support any UI like html.

Now, while developing an application we have to execute various tools like

angular-cli for project-artifact-management

tsc for compilation

jasmine for testing

grunt/gulp for packaging and so on

And these tools are executed on the developer machine, not on the client.

And for this purpose we need nodejs.

Angular CLI

It is a frontier of commands or macros that help in creating and managing the structure of an angular application. And also command for testing, packaging, and executing angular apps are found.

ng new app-name creates a new angular app

cd app-name

ng serve compile, build, and host the package into a development server running on 4200 port

ng serve --port 9899 -o compile, build, and host the package into a development server running on 9899 port,

and will launch the browser pointing to http://localhost:9899

ng build compile, build, and stores the package into 'dist' folder

ng test compile, build, and runs the test cases

ng g artifact-type artifact-name is used to generate angular artifacts

Angular Architecture

It talks about the list of artifacts an angular application is made up of

1.Each angular artifact is a typescript class

2.Each class is marked with a decorator to identify its role

3.Each of these decorators are passed a json-object called meta-data which has of the configuration information

Modules

@NgModule({

declarations:[],

imports:[],

exports:[],

providers:[],

bootstrap:[]

})

class SalesModule(){

}

Directives

@Directive({

selector:'[appFastMovingStock]',

providers:[]

})

class FastMovingStockDirective(){

}

Components

@Component({

selector:'app-dashboard',

templateUrl:'app-dashboard.component.html',

styleUrls:['app-dashboard.component.css'],

providers:[]

})

class SalesDashboardComponent(){

}

Pipes

@Pipe({

name:'inwords'

})

class IntoWordsPipe{

}

Services

@Injectable({

providedIn:'root'

})

class StockService(){

}

Angular Modules

Module is a isolated logical unit of an application

logical unit means one unseparable part which contains some functionality

isolation is always important for separating and maintaining your code

Angular Modules and JavaScript/TypeScript modules live alongside one another in an angular app

Angular Modules is a class by itself

@NgModule({

declarations:[ /\* list of all components, directives and pipes that belong to this module \*/ ],

imports:[ /\* list of other modules we want to import from \*/ ],

exports:[ /\* list of components or pipes or directives that belong here but are allowed to be used elsewhere \*/ ],

providers:[ /\* list of services that belong to this module \*/ ]

bootstrap:[ /\* list of Components that must be loaded along with the Module \*/ ]

})

class SalesModule(){

}

Each angular application is housed inside a top-level-module called 'ROOT MODULE'. The 'ROOT MODULE'is generally named as 'AppModule'.

'ROOT MODULE' will not have 'exports' section in its meta-data

Only 'ROOT MODULE' has 'bootstrap' section in its meta-data

ng g module ModuleName is the command used for creating a module

Angular Feature - Html Extendability

This allows us to create our own html elements and html attributes using angular framework

These html-attributes created using angular framework are called Directives

These html-elements created using angular framework are called Components

index.html

<html>.....</html>

|-<head>...</head>

|-<body>...</body>

|-<app-root>...</app-root>

|-<app-header>...</app-header>

|-<app-emps-list>...</app-emps-list>

|-<app-vendors-list>...</app-vendors-list>

|-<app-orders-list>...</app-orders-list>

npm is the build tool

package.json is the descriptor file like pom.xml in java

Angular Components

Component is an html-element created using Angular Framework

ng g c SalesDashboard --skip-tests

SalesDashboard.component.ts Component Class Fields and Methods

SalesDashboard.component.html Component Template HTML MarkUp

SalesDashboard.component.css Component Style Style

SalesDashboard.component.ts

@Component({

selector:'app-sales-dashboard',

templateUrl:'SalesDashboard.component.html',

styleUrls:['SalesDashboard.component.css'],

providers:[]

})

class SalesDashboardComponent(){

}

Data Binding

It is the way to consume the fields and methods of a component class in its template

We bind the fields with DOM in the html and the methods with the events in the html

Interpolation

To render the value of an expression in the content of an element

<element> {{expression}} </element>

<p> Sum of {{n1}} and {{n2}} is {{n1+n2}} </p>

Two-Way Data Binding

To bind a field with a form-element. The value of the field is loaded into the form-element for the first time.

Whenever the form-element is edited, the value of the field is as well updated

<input [(ngModel)]="fieldName" />

'ngModel' is a directive defined in 'FormsModule' from '@angular/forms'. we have to import it in AppModule class

One-Way Data Binding

Attribute Binding

It is about binding the value of a field to an attribute

<element [attribute]="field"></element>

<img [src]="myLogo" />

<img src="../imgs/logo.png" />

Event Binding

It is to bind a method with an event-directive

<element (eventDirective)="method()"></element>

html-event-attribute eventDirective

onClick click

onDblClick dblClick

onChange change

onFocus focus

onBlur blur

onSubmit ngSubmit

Style Binding

It is about binding an expression to a cssProperty or a cssClass

[style.cssProperty]="expression"

let abcd:string="CENTER";

<p [style.textAlign]="abcd">Some Text</p>

[ngStyle]="jsonObjectHavingCssPropertyNamesAsKeysAndValuesAsValues"

let obj1={width:"20px",height:"100px",margin:"10px"};

<p [ngStyle]="obj1">Some Text</p>

[class.className]="booleanExpression"

let flag:boolean=true;

<p [class.highlight]="flag">Some Text</p>

[ngClass]="arrayOfClassNames"

let myClasses:string[]=["highlight","important","bordered"];

<p [ngClass]="myClasses">Some Text</p>

[ngClass]="jsonObjectHavingClassNamesAsKeysAndBooleanAsValues"

let myClasses={"highlight":true,"important":false,"bordered":true};

<p [ngClass]="myClasses">Some Text</p>

Directives

Attribute Directives- ngModel, ngClass, ngStyle

Event Directives- mentioned above

Structural Directives- for manipulating the structure of the DOM

ngIf

<ng-template [ngIf]="booleanExpression">

<p>This para will appear only if the boolean expression evaluates to true.</p>

</ng-template>

<p \*ngIf="booleanExpression">This para will appear only if the boolean expression evaluates to true.</p>

<ng-container \*ngIf="booleanExpression then template1; else template2">

</ng-container>

<ng-template #template1>

<p>Template one content</p>

</ng-template>

<ng-template #template2>

<p>Template two content</p>

</ng-template>

ngFor

<ng-template [ngFor]="let loopingVar of array">

<p>{{loopingVar}}</p>

</ng-template>

<p \*ngFor="let loopingVar of array">

{{loopingVar}}

</p>

ngSwitch

<div [ngSwitch]="anExpression">

<p \*ngSwitchCase="value1">if expression evaluates to value1</p>

<p \*ngSwitchCase="value2">if expression evaluates to value2</p>

<p \*ngSwitchCase="value3">if expression evaluates to value3</p>

<p \*ngSwitchCase="value4">if expression evaluates to value4</p>

<p \*ngSwitchCase="value5">if expression evaluates to value5</p>

<p \*ngSwitchDefault>if expression evaluates to none of those values</p>

</div>

Pipes

It is a construct used to alter a value into another before rendering

built-in-pipes

lowercase

uppercase

titlecase

number

percent

date

currency

async (observables)

json

{{expression|pipeName:'pipe-inputs'}}

Integrating Bootstrap

bootstrap helps to design application in a responsive mode

npm i bootstrap

include

'node\_modules/bootstrap/dist/css/bootstrap.min.css' into 'styles' section &&

'node\_modules/bootstrap/dist/js/bootstrap.min.js' into 'scripts' section

of angular.json

Angular Routing

Routing feature allows navigating between components

import 'RouterModule' from '@angular/router'

RouterModule gives router outlet

router-outlet is a component used to reserve space for output of routing on the top level component

Routes is a data type of Route array

Route is an interface which contains path, redirect, pathMatch, component, children, etc

path: path of the component to be displayed

redirect: for redirecting to a certain component

target component will appear as and when url matches with the path

path match is 'startsWith|full'

we can also define children paths

routerLink is an Attribute directive

router-outlet component used to reserve space for output of routing on the top-level component

routes Route[]

Route interface path:''

redirect:''

pathMatch: 'startsWith|full'

component: TargetComponent

children: []

routerLink directive is used on 'a' tag instead of its 'href' attribute

routerLinkActive directive takes a css-class-name and applies that class on the 'a' tag only when it is active

Router service navigate(['sales','dashboard','settings'])

navigateByUrl('sales/dashboard/settings')

ActivatedRoute service used to extract info like pathParameters, queryString, hostName, etc from the current url

Angular Services

A service is an injectable artifact. services are used to hold business logic and supply it

whenever it is needed through dependency injection

Validation logic, calculations, rest-api calls, etc are all held in services

ng g service ServiceName --skip-tests the service gets created in the 'app' folder

ng g service folderName/ServiceName --skip-tests the service gets created in the specified folder

private will make the service a data member which will be accessible to use

services will be injectable through constructor without using the new keyword

@Input is used for passing data from parent to the child

that field becomes attribute of the component which is marked with @Input

! after a variable name means its optional, initialization not needed, only for undefined

? after a variable name means its optional, nulls and undefined

Interceptor have a look

Angular Forms

Template Driven Form 'FormsModule' from '@angular/forms'

1.the structure and behaviour of the form is defined using html

2.each input control is bound to a field of the component

3.these forms are difficult to test, as always testing JS is far more easier than testing html

4.these forms do not support nested or complex models

ngForm valid,invalid

ngModel valid,invalid,dirty,pristine,touched,untouched

Template Driven Forms are recommended only when the form has not more than two fields

Model Driven Form 'ReactiveFormsModule' from '@angular/forms'

Also called as Reactive Forms

1.the structure and behaviour of the form is defined using javascript

2.these forms are easy to test, as always testing JS is far more easier than testing html

3.these forms support nested or complex models

FormGroup valid,invalid

FormControl valid,invalid,dirty,pristine,touched,untouched

Bootstrap should design the webpage as its responsive

Angular Material should design the components

Angular Material is not responsive

snapshot is for accessing the app url

params is for accessing the path params

paramMap is for accessing the whole path params

custom pipes and directives

lifecycle methods, chain detection, view encapsulation

observables

rest api integration

route guards

see the mphasis pdf

Angular LifeCycle Hooks for a Component/Directive

constructor

This is invoked when Angular creates a component or directive by calling new on the class

ngOnChanges

Invoked every time there is a change in one of the input properties of the component

ngOnInit

Invoked when the given component has been initialized

This hook is only called once after the first ngOnChanges

ngDoCheck

Invoked when the change detector of the given component is invoked. It allows us to implement our own change detection algorithm for the

given component

Custom Directives

ng g directive DirectiveName --skip-tests

@Directive({

selector:'[appFastMovingStock]'

providers:[]

})

class FastMovingStockDirective{

}

@Input() for receiving any values

@HostListener() is used to handle event that occur on the element

ElementRef represents the element on which the directive is applied

Custom Pipes

ng g pipe PipeName --skip-tests

@Pipe({

name:'inwords'

})

class IntoWordsPipe implements PipeTransform{

transform(value:any,...args:unknown):any{

}

}

Observable from RxJS

An Observable is an enhanced Promise

const bgJob = (resolve,reject) => {

//we use resolve(val) to indicate successful completion of the job

//we use reject() to indicate abortion due to an error

};

let p = new Promise(bgJob);

p.then(

(val) => { /\* is the implementation of resolve \*/ },

() => { /\* is the implementation of reject \*/ }

);

- only one value can be returned from the bgJob after its completion through resolve

- once the bgJob is triggered we can not cancel it or unsubscribe from it

- only one subscription per promise is allowed

const bgJob2 = (observer) => {

//observer.next(val) is called every time the bgJob2 wants to emit a value

//observer.error(err) is called to indicate abortion of the job due to an error

//observer.complete() is called to indicate the successful completion

};

let ob = new Observable(bgJob2);

ob.subscribe({

next: val => { /\* consume the value \*/ }

error: err => { /\* handle the error \*/ }

complete: () => { /\* do something once the job is done \*/ }

})

+ any number of values can be returned from the bgJob2 while it is in progress

+ once the bgJob2 is triggered we can cancel by calling 'unsubscribe'

+ any number of subscriptions per observable are allowed

HttpClient from 'HttpClientModule' from '@angular/common/http'

HttpClient provides

get(apiEndPoint) : Observable

post(apiEndPoint, reqBody) : Observable

put(apiEndPoint, reqBody) : Observable

delete(apiEndPoint) : Observable

one thousand twenty four

Create a fake rest-api using json-server

json-server is a javascript tool that generates fake rest-api from a .json

md hr-api

cd hr-api

npm init -y

npm i json-server@0.17.4

create hr-api/data.json

change the 'start' script in the package.json as 'json-server --port 9999 --watch ./data.json'

npm start

'async' pipe helps to extract data from Observable, read more on this

View Encapsulation

allows us to isolate the css classes to the specified component level

encapsulation: ViewEncapsulation.NONE

encapsulation: ViewEncapsulation.Emulated (default)

encapsulation: ViewEncapsulation.ShadowDOM

Router Guards

canActivate

canActivateChild

canDeActivate

canMatch

pure and impure pipes

observables (rxjs) learn more, dependency injection learn more, @Output-old school thought, use input only, ViewChild, ContentChild,

debugging-only developer tools, subjects, form builder, form array, go with form group and form control

angular material, angular library, lazy loading

https://medium.com/@mattias.trnqvist/10-best-practices-for-building-scalable-angular-applications-with-examples-efdfe2a5b4e8

https://www.freecodecamp.org/news/best-practices-for-a-clean-and-performant-angular-application-288e7b39eb6f/

====================================================================================

Java 8

--------------------------------------

Topics to be Covered:

1 Introduction to Java

2 Classes and Objects

3 Methods, Encapsulation, and Conditionals

4 Data Manipulation and Inheritance

5 Encapsulation, Polymorphism, and Abstraction

6 Interfaces, Lambda Expressions, Collections, and Generics

7 Inheritance, Interfaces, Exceptions, and Deployment

8 Date/Time API, I/O and File I/O (NIO.2), and Concurrency

9 Concurrency, Parallelism, The JDBC API, and Localization

Lab Setup

JDK 8 or above.

EClipse / IntelJ/ STS

Maven

MySQL 8 community Server

Intro

History

1.1,

1.2,1.3,1.4

1.5

1.8

1.17

1.21

Features

Simple

Robust

Multi-Threaded

Distributed

Self-Documented

Platform Independent

Architecture Neutral

Characteristics

Partially but Strictly Object Oriented

Type Safe

Case Sensitive

Has c-family syntax

Tokens And Standards

Identifiers

variables names and method names

camelCase

type names (classes, interfaces, etc)

initialCapitals

constants Names

ALL\_CAPITALS

Keywords

Operators

Comments

Literals

"hello" string literal

'$' char literal

12 integer literal (decimals)

034 integer literal (octal)

0xA1 integer literal (hexa-decimal)

12.56 floating point literal

true, false boolean literals

null null literal

z1 holds the reference of the object but not the object directly

copy constructor is accpeting an object of the same class

'this' means reference to the current object

Object Oriented Programming Concepts

Class and Object

class is a user defined datatype. It is used to define a model with properties and behaviours. Properties are represented through variables and behaviours through Methods

Object is a variable of class type

class ComplexNumber{

int real;

int imaginary;

ComplexNumber(){

/\* is a default constructor - no arg constructor

is a special method that gets invoked as and when an object is allocated

is used to initialize the fields of the object

\*/

this(8,17);

//here 'this' keyword is used to call paramaterized-constructor from default constructor

}

ComplexNumber(int real, int imaginary){

//a paramaterized constructor

this.real=real;

this.imaginary=imaginary;

//'this' keyword represents the current object

}

ComplexNumber(ComplexNumber z){

//a copy constructor

this(z.real, z.imaginary);

//here 'this' keyword is used to call paramaterized-constructor from copy constructor

}

}

ComplexNumber z1=new ComplexNumber();

ComplexNumber z2=new ComplexNumber(10,11);

ComplexNumber z3=new ComplexNumber(z2);

ComplexNumber z4=z2;

new keyowrd creates the object. memory location

z2,z4 two difeerenet names of the same object

Encapsulation

It is a means of applying restriction of access on fields or methods

This is done through access specifiers and setters and getters

Access Specifiers

<default>

private

protected

public

setter is a method that allows a value into a field (writing into a field)

getter is a method that allows the value of the field to be returned (reading a field)

If a field is marked private,

we can make it both readable and writable by defining both setter and getter upon it

we can make it read only by defining only getter

we can make it write only by defining only setter

class ComplexNumber{

private int real;

private int imaginary;

public ComplexNumber(){

this(8,17);

}

public ComplexNumber(int real, int imaginary){

this.real=real;

this.imaginary=imaginary;

}

public ComplexNumber(ComplexNumber z){

this(z.real, z.imaginary);

}

public void setReal(int real){

this.real=real;

}

public int getReal(){

return this.real;

}

//we are making the field 'imaginary' as read-only (as a setter is not defined)

public int getImaginary(){

return this.imaginary;

}

}

'static' keyword is used in various context

static fields a field is called a instance variable as each instance (object) of the class has its own copy of the field but if

the field is marked static, it is called class variable, as it is allocated as a single copy for all instances (objects)

to share

static methods methods that are marked static can access only static fields and method of that class directly

public static fields or methods can be accessed with the class name directly, memory efficiency, global access, object independence

static classes Inner Classes vs Nested Classes

A class inside a class is called Inner class. Inner class has access to all the other memebr of the

enclosing class.

A static class inside a class is called Nested class. Nested class is a treated as memeber of the enclosing

class.

static blocks static { /\* some code goes here \*/ }, what it is, what is the purpose, where to use it

A class can have any number of static blocks but are all clubbed into one.

The code enclsoed in these blocks gets executed just before the class being accessed

for the first time.

Polymorphism

It is a technique to define multiple methods with the same name

It has two ways of achieving it-

Overloading

two methods of the same class or two methods from a super and a sub class

1.having same name

2.but different no. of args or different types of arguments

are said to be overloaded

class Human{

public void eat(Apple apple){

wash(apple);

ApplePieces[] pieces = cut(apple);

for(ApplePieces piece : pieces){

chewAndSwallow(piece);

}

}

public void eat(IceCream iceCream){

while(iceCream!=null){

lickAndSwallow(iceCream);

}

}

}

Overriding

two methods from a super and a sub class, having the same signature (same name, same args and same return type)

are said to be overridden

class Monkey{

public void eat(Apple apple){

while(apple!=null){

biteAndChewAndSwallow(piece);

}

}

}

class Human extends Monkey{

public void eat(Apple apple){

wash(apple);

ApplePieces[] pieces = cut(apple);

for(ApplePieces piece : pieces){

chewAndSwallow(piece);

}

}

}

Inheritance

It is about defining a new class from an existing class

class Pen{

Nib nib;

Barrel barrel;

Refill refill;

public void write(Paper paper){

}

}

class Marker extends Pen{

public void write(WhiteBoard board){

}

}

Simple/Single

super<---------------sub

MultiLevel

super<---------------sub1<--------------sub2<---------------sub3

Hierarchical

|<------------sub1

super<----------|

|<------------sub2

Multiple

super1<---------|

|<------------sub

super2<---------|

Hybrid

|<-----------sub1

super<----------|

|<-----------sub2<------------sub3<----------sub4

Classes in Java do not support 'multiple' inheritance

Constructors are chained from super to sub class, when an object of a sub-class is allocated

A reference of a super class can refer to an object of its sub class

SuperClass s = new SubClass();

A Super class reference can be typecasted to a sub class reference

SubClass x = (SubClass) s;

'super' keyword can be used to invoke any super class constructor from a sub class constructor

'super' keyword can be used to invoke a super class method from its overridden copy in the sub-class

class SuperClass{

public void method(){

}

}

class SubClass extends SuperClass{

public void method(){

super.method();

}

}

Abstraction

It is the process of declaring a method and implementing it in later stages

In main words, it is the process of declaring and the implementation follows later

It has three ways of achieving it-

abstract class

a class that can not have an object allocated

you can declare an object but cannot allocate an object

abstract class Person id,fullName,gender,bloodGroup

|

|-class Student class,section,fee

|-class Teacher subject,salary

|-class NonTeachingStaff designation,salary

|-class Parent mobileNumber,mailId

abstract method

a method that is declared now but implemented later

an abstract method can be declared only inside an abstract class or an interface

abstract class Shape{

private int[] sides;

public int perimeter(){

int sum=0;

for(int s : sides){

sum += s;

}

return sum;

}

public abstract int area();

}

abstract classes when inherited, the inheriting sub class must implement all the abstract methods

of the abstract super class, failing which - that sub class also must be marked as abstract

interface

It is a user defined abstract data type that can not have instance variables (non-static fields)

a class can implement multiple interfaces

while defining an abstract data type.....

if we have to declare fields -> no choice -> we write an abstract class

if we have no fields -> we choose interface over abstract class, as interface support multiple inheritance

Classes represent Entities and Interfaces represent Roles

interface Mother giveBirth()

interface Father protect(),feed()

interface Sibling showAffectionAndLove()

abstract class Animal

|-class FemaleAnimal implements Mother,Sibling

|-class MaleAnimal implements Father,Sibling

abstract class Bird

|-class FemaleBird implements Mother,Sibling

|-class MaleBird implements Father,Sibling

'final' keyword

final variables are constants

final methods can not be overridden

final classes can not be inherited further

'final' and 'abstract' keywords contradict each other and will never appear together

MarkerInterface an interface having no methods at all is called a MarkerInterface

example- java.io.Serializable

Modernly MarkerInterfaces are replaced with Annotations

FunctionalInterface an interface having only one abstract method is called a functional interface

introduced in 1.8

interface BinaryOperator{

int operate(int n1, int n2);

}

interface Greet{

String doGreet(String unm);

}

interface GetText{

String get();

}

Functional interfaces can be implemented and instantiated without a class,

using a special functional syntax called Lambda Expression

A Lambda Expression 'maps' a list of parameters with its return value

(paramList) -> returnValue

BinaryOperator sum = (a,b) -> a+b;

System.out.println(sum.operate(10,20));

BinaryOperator maxOf = (x,y) -> x>y?x:y;

System.out.println(maxOf.operate(10,20));

Greet greet1 = userName -> "Hello " + userName;

System.out.println(greet1.doGreet("Vamsy"));

GetText companyName = () -> "Cognizant";

System.out.println(companyName.get());

(paramList) -> {

//complex function body

return someResult;

};

pass a function as a parameter to another function is known as functional programming

@Override is a compiler check annotation, it ensures that we overriden the right method name

@FunctionalInterface is a compiler-check annotation that ensures that the interface has one and only abstract method

if we have two abstract methods in functional interface, then lambda function wont know to where to go because we didnt mention the function name in implementation so to avoid that, the lambdas are used with functional interface since they only have one abstract method

Types of Functional Interfaces

Suppliers returns a value but has no-args

Consumers doesn't return any value but has args

Predicates returns boolean irrespective of having args

Functionals any other than the above

java.lang

Object the default super class for all classes in java

offers some important methods like-

public boolean equals(Object) //obj1.equals(obj2)

public int hashcode()

public String toString()

if two objects have same hashcode, it is not necessary that they have the same magnitude

overriding the hashcode gives the same hashcode based on the same magnitude

if not overridden, then it gives hashcode based on the new object creation

== compares pointing to the same object location or not

.equals compares the value

when object is created, java maintains a register

hashcode is an easier method to compare objects for good performance

when hashcode is not equal, no comparison happens

when hashcode is equal, then further comparison happens

System

Math

Wrapper Classes-

Byte

Short

Long

Integer

Float

Double

Boolean

Void

Character

String

Strings are immutable, we cannot modify the existing string, if modified a new string object is created and assigned to the variable

StringBuffer

it is mutable, String buffer is thread safe and used in multi threaded environments

StringBuilder

it is mutable, String builder is not thread safe and used in uni threaded environments

Exception Handling

Throwable interface

|-Exception

|-<sub classes of Exception class> are called CHECKED Exceptions

|-RuntimeException

|-<sub classes of RuntimeException class including RuntimeException> are called UN\_CHECKED Exceptions

CHECKED Exceptions are checked by the compiler, if they are handled or not. And if not handled

the compilation will not continue. It is compulsory to handle CHECKED Exceptions

try{

Connection con = DriverManager.getConnection(dbUrl,dbUserName,dbPassword);

//in this case we cannot avoid if the username or password are changed by the time the above line executes

//we cannot control the power-off of the database server by the time the above line executes

}catch(SQLException exp){

}

UN\_CHECKED Exceptions are NOT checked by the compiler, if they are handled or not. It is NOT compulsory to handle UN\_CHECKED Exceptions. Because the UN\_CHECKED exceptions MUST NOT BE HANDLED USING TRY...CATCH, BUT THEY MUST BE AVOIDED.

public void increaseSal(Employee emp,double percent){

if(emp!=null){ //avoiding NullPointerException, we are not handling it

emp.setSalary(emp.getSalary() + (emp.getSalary()\*percent));

}

}

//try-with-multiple-catch

try{

//statements that may raise an exception

}catch(ExceptionType1 exp){

//alternate code or exception handling happens

}catch(ExceptionType2 exp){

//alternate code or exception handling happens

}catch(ExceptionType3 exp){

//alternate code or exception handling happens

}finally{

//any code that has to be executed irrespective an exception occured or not

//is used to close resources like streams, connections, etc

}

//try-with-multi-catch

try{

//statements that may raise an exception

}catch(ExceptionType1 | ExceptionType2 | ExceptionType3 exp){

//alternate code or exception handling happens

}finally{

//any code that has to be executed irrespective an exception occured or not

//is used to close resources like streams, connections, etc

}

//try-with-resources

try(/\*declare all closeable resources like files, streams, connections, etc, here\*/){

//statements that may raise an exception

}catch(ExceptionType1 | ExceptionType2 | ExceptionType3 exp){

//alternate code or exception handling happens

}

'throw' is a keyword used to raise an exception

'throws' is a keyword used to transfer the exception handling from a method to its caller-method

CHECKED Exceptions cannot be avoided

UN\_CHECKED Exceptions can be avoided

'throws' keyword transfers the exception to the UI layer, meaning it transfers first to the calling method and then handled there

followed by transfer to the UI

Runnable interface

|-Thread

java.time

LocalDate .now(),.of(year,month,day)

LocalTime .now(),.of(hour,min,sec)

LocalDateTime .now(),.of(year,month,day,hour,min,sec)

ZonedDateTime .now(zoneId),.of(year,month,day,hour,min,sec,zoneId)

getDay(),getMonth(),getYear(),getHour() .....etc.,

plusDays(int),plusMonths(int),plusYears(int),...etc.,

minusDays(int),minusMonths(int),minusYears(int),...etc.,

isAfter,isBefore,isEquals

parse(String),parse(String,dateTiemFormatter),format(dateTiemFormatter)

TemporalAdjusters

Period .between(startDate,startDate);

Duration .between(startTime,startTime);

DateTimeFormatter.ofPattern("");

java.util

Scanner (class)

Comparator

is an interface used to provide comparison strategy of any model

int compare(T obj1, T obj2); //comparatorObject.compare(obj1,obj2);

is expected to return a

positive number if obj1 is greater than obj2 OR

negative number if obj1 is lesser than obj2 OR

zero meaning both objects are equal

java.lang.Comparable is also an interface expected to be implemented by a model class to facilitate the

comparison strategy

int compareTo(Object); //obj1.compareTo(obj2)

Comparable interface is used for default comparison strategy, whereas

Comparator interface facilitates customized comparison strategy

Collection<T> (interface) add(ele),remove(ele),contains(ele),size(),clear(),stream()

|

|-Set (i)

| represents a non-linear data-structure

| elements in set will not have index. set doesn't allow duplicates

|

| |-HashSet (c) doesn't have any order in retrieving elements (internally implements HashTable)

| |-LinkedHashSet (c) it follows entry order in retrieving elements (internally implements LinkedList and HashTable)

| |-TreeSet (c) it follows sorted order in retrieving elements (internally implements Binary Search Tree)

|

|-List (i)

| represents a linear data structure

| elements have index(0 based). duplicate elements are allowed

|

| add(int index,element),removeAt(int index),first(),last(),etc

|

| |-Vector (c) a synchronized growable array, thread-safe (internally implements Array)

| |-ArrayList (c) a growable array (not synchronized), not thread-safe(internally implements Array)

| |-LinkedList (c) a doubly linked list implementation (in which there is a node which contains 3 parts-previous element index,

current node value, next element index)

Linked List is used for extensive modification like insertion and deletion, heavy traversing is there in linked list

Array List or Vector is used for retrival purposes and when insertion and deletion are lesser

Map (i) represents a group of key value pairs (internally implements Dictionary)

| put(key,value),set(key,value),containsKey(key),keySet(),size(),clear()

| key cannot be duplicate

|

| - HashMap (c) doesn't have any order in retrieving elements (internally implements HashTable)

| - LinkedHashMap (c) it follows entry order in retrieving elements (internally implements LinkedList and HashTable)

| - Tree Map (c) it follows sorted order on KEY in retrieving elements (internally implements Binary Search Tree)

Collections (c) is a class that provides a variety of utility functions

javap java.util.Comparator --> command to see the structure

java.util.regex

Pattern pattern = Pattern.compile(regexp);

java.lang.string matches(regexp)

java.util.function

this package provides a list of pre-declared functional interfaces

:: -> scope resolution operator

System.out::println; //Method Referencing

Suppliers returns a value but has no-args

Consumers doesn't return any value but has args

Predicates returns boolean irrespective of having args

Functionals any other than the above

java.util.stream

it gives a class called Stream

Stream is a class that represents 'a flow of data'

DataSource - flow of data starts

DataSink - flow of data ends

DataSource<-------------stream---------------->DataSink

DataSource can be an array or a list or a set or any other collection

DataSink can be an array or a list or a set or any other collection or a single object or nothing

Stream s1 = Stream.of(ele1,ele2,ele3,...);

Stream s2 = Arrays.stream(array);

Stream s3 = ListorSetorAnyOtherCollectionObj.stream();

Streams support functional programming

DataSource

|----------Stream----------|

OPERATION

|--------------Stream----------|

OPERATION

|----------Stream----------|

DataSink

As functional programming is a new paradigm that provides higher maintainability of data

manipulation when compared to their alternate looping-based-solutions

Intermediate and Terminal Stream

Instance Methods of Stream Class

void forEach(Consumer)

execute the consumer on each and every element of the stream

this is a terminal operation as the stream terminates into nothing

Collection collect(Collector)

is used to collect a stream into a collection

this is a terminal operation as the stream terminates into a collection

Collectors.toList()

Collectors.toSet()

Collectors.toMap()

...etc, are Collectors

Set set = streamObj.collect(Collectors.toSet());

T reduce(identity, binaryOperator) - used for any logical operation like sum, multiply, etc

Optional<T> reduce(binaryOperator) - used for operation on objects like Employee, Item, etc

it is going to reduce a stream into a single element

this is a terminal operation as the stream terminates into a single element

int[] nums = new int[] {1,2,3,4,5}

BinaryOperator prd = (x,y) -> x\*y;

int result = Arrays.stream(nums).reduce(1,prd);

// prd(prd(prd(prd(prd(1,1),2),3),4),5) -> 120

Optional<Integer> result2 = Arrays.stream(nums).reduce(prd);

// prd(prd(prd(prd(1,2),3),4),5) -> 120

Stream filter(predicate)

it executes the predicate on each element of a stream and returns a new stream containing

elements that satisfy the predicate

this is a intermediate operation as it returns a new stream

Stream map(mapper)

a mapper is a functional interface that has a function which accepts one arg and returns one result

x -> x\*x

x -> Math.sqrt(x)

bankAccount -> bankAccount.getBalance()

'map' will execute the mapper on each element of the stream and returns a new stream containing

all the results

this is a intermediate operation as it returns a new stream

Multi-Threads

Thread is a isloated process

if multiple processes share the same heap, they are said to be a group of threads or multi-threading

Each java application is a thread by itself

java.lang.Runnable void run()

|

java.lang.Thread

Thread()

Thread(Runnable)

Thread(String name)

Thread(Runnable, String)

String getName()

void setName(String name)

int getPriority()

void setPriority(int p)

void start()

void join()

static void sleep(long durationInMilli)

static Thread currentThread();

Lifecycle-

constructor()

|

| start()

↓

[...queue READY] ------ the cpu and other resources are available --|

↑ |

| ↓

[PAUSED...]-------------sleep()----------------------- [Running (run()) ]

|

↓

[Terminates]

Multi-Layer Archetecture

DAO <-------models------> SERVICE <-------models------> UI

UI - User Interface - is responsible to

(a) accept data or instruction from the user.

(b) to display information or resposne from the application

SERVICE is responsible to

(a) provide bussiness logic like caliculations, computations, validations ..etc.,

(b) it generally accepts a model from UI layer, validates or computes on the model

and passes model to the DAO layer for persistence.

DAO - DataAccessObject - is responsible to

(a) interact with a data store like a file/ or a databases etc.,

and persist the data into the data store.

(b) it receives the model from service layer and after persisting

will inform the service layer with the outcome of the

data operation and the service layer will

pass on the outcoem to the UI layer

and the UI layer will according inform the user.

java.io, java.nio

java.io provides a classical InputOutput API

java.nio provides a novel / new InputOutput API, and also a few non-blocking InputOutput API

Files

Paths

Path

Directories

Non-blocking api replaced IO-Streams with Channels, Buffers and Selectors

Channel is a flow of data

Buffer is a temporary memory location to store data from a channel or into a channel

Selector is a monitor on a buffer that responds when ever a reading or writing happens on the buffer

JDBC

Oracle <-> Thin Driver <->

MySQL <->ConnectorJ Driver <-> JDBC-API <--> [DAO <-model-> SERVICE <-model-> UI ] <--> End User

Ms SQL Server <-> Jet Driver <->

A Jdbc is a database connectivity specification (only declarations [interfaces and abstract class])

Each Database driver is an implementation of this specification

Oracle jdbc:oracle:thin:@serverName:5314/OracleServiceName

MySQL jdbc:mysql://serNameOrIP:3306/databaseName

java.sql

Connection con=DriverManager..getConnection(dbConnectionString,userId,password)

|

--------------------------------------------------------------------------

| | |

Statement st=con.createStatement(); | CallableStatement cst=con.preppareCall(call);

PreparedStatement pst=con.prepareStatement(qry)

Statement and PreparedStatement are used to execute DDL/DML/DRL queries

CallableStatement is used to execute a database procedure or function

Statement used to handle dynamic queries

int executeUpdate(qry); used to execute insert/update/delete queries and return affectedRowCount

boolean execute(qry); used to execute create/alter/drop queries and return isDoneOrNot

ResultSet executeQuery(qry); used to execute Select query and the data is returned as a ResultSet Obj

PreparedStatement used to handle static queries and supports parameters

int executeUpdate(); used to execute insert/update/delete queries and return affectedRowCount

boolean execute(); used to execute create/alter/drop queries and return isDoneOrNot

ResultSet executeQuery(); used to execute Select query and the data is returned as a ResultSet Obj

void set<Type>(paramIndex,paramValue)

ResultSet

is an interface that hold the data returned by the Select query

boolean next()

<Type> get<Type>(colIndex)

<Type> get<Type>(colLabel)

create database adbdb;

use adbdb;

create table contacts(

cid int primary key auto\_increment,

fnm varchar(50) not null,

mob char(10) not null,

mid varchar(100) not null,

dob date not null,

age double

);

wrapper classes are required to provide object status

wrapper classes can hold null but primitive variables cant

wrapper classes have inbuilt methods to help us perform certain operations

1. Accept the date of birth of a user, assuming 70 years as the retriment age, compute and print his retirement date.

2. Accept the principal, rate, time period, and disbursal date of a loan, compute and print the EMI and loan closure date.

Assignment:

(a) Create a Model called "Item". Item will have itemCode:int,title:string,price:double,packageDate:LocalDate as fields

(b) Make the Item implement Comparable interface and override compareTo method to compare two items based on their itemCode

(c) Create a List<Item> and display them using an ApplicationClass

(d) Pick up the names of all those items that has price between 100 and 300 and print them

(e) Add two more fields to the Item model, expiryDate:LocalDate and category:String ["BEVERAGES","CEREALS","PULSES","ACCESSORIES"]

(f) Assuming the shelfLife of each time as 40 days for CEREALS and PULSES, 80 days for BEVERAGES and 5 years for ACCESSORIES,

compute the expiryDate of each item

(g) Pick up the costliest beverage and print it

LocalDate cerealsAndPulseExpiryDate = LocalDate.of(2024,6,13).plusDays(cerealsAndPulseShelfLife);

System.out.println("Cereals And Pulse will be expired on: "+cerealsAndPulseExpiryDate);

LocalDate beveragesExpiryDate = LocalDate.of(2024,6,13).plusDays(beveragesShelfLife);

System.out.println("Beverages will be expired on: "+beveragesExpiryDate);

LocalDate accessoriesExpiryDate = LocalDate.of(2024,6,13).plusDays(accessoriesShelfLife);

System.out.println("Accessories will be expired on: "+accessoriesExpiryDate);

=====================================================================

Spring Boot and Microservices

-------------------------------------

Spring Boot is a spring framework module that offers auto-configuration and thereby provides RAD - Rapid Application Development

Spring Boot also has embedded servers

auto-configuration->what, how and why

auto-config-------->Convention over coding...!

@SpringBootApplication = @Configuration

@ComponentScan

@EnableAutoConfig

@PropertySource

Spring Starter Project can be created via

1.STS Starter Project Wizard

2.SpringInitializr -> http://start.spring.io

3.Spring Boot CLI

@SpringBootApplication

public class MyApplication{

public static void main(String args[]){

SpringApplication.run(MyApplication.class);

/\*

\*ApplicationContext is instantiated

\*Context initialization like component scan, auto-config, etc

\*Invoke all Spring Runners (if any)

\*Start the embedded server (if any)

\*Once Server is shut down, the context is destroyed and the app terminates

\*/

}

}

SpringRunner

is any class that implements CommandLineRunner or ApplicationRunner

Assignment 1-

Develop a rest-api on Spring Boot 3.0

To facilitate the CRUD operation on Contact Entity

Contact Entity has contactId, fullName, mailId, dateOfBirth as fields

Spring Data JPA for ORM

Validation framework for Server Side Validations

Use Lombok if available

@Query("select c from Contact c where c.dob between :startingDate and :endingDate")

List<Contact> findByRange(String startingDate, String endingDate)

no need to put annotation in repository interface

employeeDAO class -> employeeService

interface is very important as it maintains a contract between the controller and service

@responsebody tells the controller that the action methods inside the controller wont provide views, it will return the data as the response body

put and patch both mean update

patch is a small update, put is update at the object level

a patch request will depend on path variable for modifying one or two variable, like updating status of a service ticket

spring profiles, spring security, spring testing

Micro-services

-------------------------------------

Database <------------> Application[logic + UI] <-------------> EndUser

A Webservice is any method that can be invoked through a request, the parameter if any will be passed as request parameters or request body and the result is received through a response

web services were introduced because it encapsulates logic for the different types of user interfaces

Persistence Layer Middleware UI/UX

Java-Swing-Standalone-App <---> EndUser

Java-SpringWebMVC-App <---> EndUser

Database <------------> Webservice <------------>

Java-Android-Mobile-App <---> EndUser

JavaScript-Angular-SPA <---> EndUser

Web-Services are of two ways to build:

SOAP Web Services

REST-ful Web Services / REST api

The approach of building these middleware as a single unit of deployment is called Monolithical approach. All the modules of an application including all the entities, services, controllers are housed together in a single deployment unit.

This approach has three main limitations-

(a) lack of Scalability

(b) lack of Interoperability

(c) lack of Adaptability

Scalability is to make application available to maximum nunmber of users in a given point of time

Interoperability is developing application in different languages

Adaptability is to adapt to certain changes or additions

Micro-services is an ecosystem of isolated independent web-services aimed at a single application that can correlate data and operations

(+) As the services are isolated into individual deployments, independent scaling is possible

(+) As the services are independent, each of them can be of any technology

(+) As and a new services has to be added to the ecosystem, the new service need not comply with the technologies of the existing services and can independently adapt to new standards and technologies

Challenges to Build a Microservices Ecosystem

(-) Decomposition - how many microservices are need for the application

(-) Inter-Service-Communication - how to identify the service for exchange of information

(-) Monitoring and Tuning - for monitoring the services

(-) Distributed Transactions - if one operation fails, all the other operations have to be rolled back residing in diff microservices

(-) Distributed Tracing - if something went wrong, then debugging is to start at which microservices

Micro-Services Design Patterns

-------------------------------------------

Decomposition Desing Patterns-

Decomposition by Domain

Decomposing a monolithic application into a microservice. We can identify the modules of a monoliothic application and consider each module as a microservice.

Based on the anticipated workload, further microservices can be merged or split.

We will decompose the budgetTackingApp into 3 microservices

(a)Profiles-Service

(b)Transactions-Service

(c)Statement-Service

Decomposition by Sub-Domain

Identify the god-classes (classes that appear in most of the microservices)

Define their bounded-context and avoid redundancy, consider an employee having personal details needed by the HR department and the finance details needed by the finance department, so each of these is a bounded context

Database Design Patterns-

Shared Database Pattern

Having a single DB for all microservices in brown field apps (an application already existing and enhancements happening)

Database Per Service Pattern

Each microservice has its own database in all green field apps (an application constructed from scratch)

CQRS Pattern

DML and DDL operations are handled separately

SAGA Pattern

To handle distributed transactions across microservices

Integration Design Patterns-

API Gateway Pattern

Aggregation Pattern

Client Side Component Pattern

Observation Design Patterns-

Log Aggregation Design Pattern

Distributed Tracing Design Pattern

Cross-Cutting Design Patterns-

Discovery Service Pattern

Load Balancer Design Pattern

External Configuration Design Pattern

Circuit Breaker Design Pattern

Discovery Service Pattern

It solves the problem of inter service communication

microservices urls are dynamic

it is a microservice running independently

each microservice will register its address with discovery service

when one microservice wants to communicate with another microservice, it will ask to discovery service for the address and then connect

discovery-service

|

|-all microservices will register their address with discovery-service

|-the address are retireved from here by the needy microservices

Data Aggregation Pattern

Aggregation is about designing a microservice that can collect info from other microservices, analyze and aggregate the data and pass the aggregagted data to the client, saving the client from making multiple requests for different parts of the data

going to a restaurant and ordering biryani, we get all the other things like salad, raita all together with biryani

similarly, a user logged in bank website, he is asking for statement, then all the details of the transactions should come together

the 'statement-microservice' is an example for this pattern

Client Side Component Pattern

Each component of the UI/UX application can place their individual requests to different microservices parallely and should be receiving the responses as well parallely

example-one client will have 3 different components, it will have 3 diff requests and the responses will come parallely with their own time

Distributed Tracing Design Pattern

Tracing - Service, this is a separate microservice maintained centrally

Whenever a request comes to any of the microservices in our app-ecosystem, that request is given a unique ID and is reported to the Tracing-Service every time, the request goes from one service to another service until the final response is sent to the client. And the tracing-service will record all the track of this request along with any performance metrics and log info attached with the request

Load Balancing Design Pattern

Load balancing means mapping the incoming requests to multiple instances of the same microservice based on some (round-robin) algorithm

tools like Ribbon/Spring Cloud Load Balancer, etc are used to perform load balancing

API Gateway Design Pattern

it is like a receptionist in a office

gateway-service <------------(all reqs)----------------any-client

|

|->forward that request to the respective microservice

|<-receives the response from that microservice

|

|--------------------------------(response)-------->client

Circuit-Breaker Design Pattern

circuit-breaker-threshold, time limit or subsequent number of requests

when the first request could not reach a specific microservice (due to its downtime), a fallback mechanism is triggered

After that, the circuit is made open (broken), means that the fallback mechanism will address all the other consecutive request targeting that microservice

when a request to that same microservice is inbound after the threshold, then the circuit is half-closed, means that a new attempt to reach the microservice is made,

|-on successful contact, the circuit is closed

|-or if that microservice is still unavailable, the circuit continues to be open

tools like Resilience4j, etc, are used for the purpose

open- circuit not working

closed- circuit working

half-open-neither open nor closed, it is going to test whether the microservice will be able to reach or not for communication

External-Configuration Design Pattern

repository(github) [contains a list of all config of all microservices]

|

|

config-service

|<-when ever a microservice has to start, it will first send a fetch req to the config-service

|

|the config-service will check for the config file in the repo

|

|<-the config file is passed to the microservice by the config-service

|

|<-when ever the config files are modified and pushed into the repo, the config service will automatically notify all the respective microservices and the microservices will receive the updated config-file and restart all by themselves

Case Study Budget Tracking App

1.We need to have different consumer or account holders to register

2.Each accountHolder must be able to record his spending or earning transactions

3.Generate a statement periodically displaying the total spending, the total earning and the balance

Decomposing by domain

budgetTracking

profiles service

txns service

statement service

Decomposing by sub-domain

budgetTracking

profiles service

AccountHolder Entity

Long ahId

String fullName

String mobile

String mailId

txns service

AccountHolder Entity

Long ahId

Double currentBalance

Set<Txn> txns

Txn Entity

Long txnId

String header

Double amount

TxnType type

LocalDate txnDate

AccountHolder holder

statement service

AccountHolder Model

Long ahId

String fullName

String mobile

String mailId

Double currentBalance

Txn Model

Long txnId

String header

Double amount

TxnType type

LocalDate txnDate

Statement Model

LocalDate start

LocalDate end

AccountHolder profile

Set<txn> txns

totalCredit

TotalDebit

statementBalance

A class that represents a domain object, anything that is related to project domain but not a component. domain is the area for which the project is being created like healthcare, banking, etc

Connections, threads, etc are all components

person, etc are all domain objects

if the domain object is mapped to a table, then it is an entity

if the domain object is not mapped to a table, then it is a model

statement service acts an an Aggregator

data will be stored in profiles and txns service

Aggregator Pattern-

req for statement--------------->statement-service-------------->profile service

<------account holder data------

-------------------------->txns service

<--------list of txns-----------

does the composition and computation

<--------statement obj---------into statement obj

Zuul-used for API Gateway

Discovery Service Design Pattern-

discovery-service

(spring cloud netflix eureka discovery service)

||

registration of urls

and retrieval of urls

||

-----------------------------------------

| | |

profile-service txns-service statement-service

Api Gateway Pattern Design Pattern-

Android App/Angular App/ReactJS App

||

api-gateway

(spring cloud api gateway)

|

|

|<----> discovery-service

| (netflix eureka discovery service)

| ||

| registration of urls

| and retrival of urls

| ||

------------------------------------

| | |

profile-service txns-service statement-service

sleuth-tracking the request and giving request id, request and response data

prometheus-capture the performance metrics

zipkin job is to collect info and store it, info related to all api's

Distributed Tracing-

Android App/Angular App/ReactJS App

||

api-gateway

(spring cloud api gateway)

|

|

|<----> discovery-service

| (netflix eureka discovery service)

| ||

| registration of urls

| and retrival of urls

| ||

------------------------------------

| | |

profile-service txns-service statement-service

(sleuth) (sleuth) (sleuth)

| | |

--------------------------------------

||

distributed tracing service

(zipkin-server)

External Configuration-

Android App/Angular App/ReactJS App

||

api-gateway

(spring cloud api gateway)

|

|

|<----> discovery-service

| (netflix eureka discovery service)

| ||

| registration of urls

| and retrival of urls

| ||

------------------------------------

| | |

profile-service txns-service statement-service

(sleuth) (sleuth) (sleuth)

| | |

---------------------------------------

|| ||

distributed tracing service configuration-service

(zipkin-server) (spring cloud config service)

|

|

git-repo

profile.properties

txns.properties

statement.properties

gateway.properties

open feign is for communicating to another service

all account holder to register, retrieve accountholders, for transaction to be recorded, an account holder exists or not

open feign is a framework that generates inter service communication automatically

open feign-

communication between 2 webservices, the methods are very lot

to eliminate the boiler plate code, we have open feign

we will have to enable feign clients on the main Spring Boot application class

for transaction management, we use transactional annotation for persisting to db or rolling back from db

Design Algorithm-saved in snipping tool snapshot

POST call for adding a transaction->incoming transaction contains accountHolder id or not->if accountHolder id is not there then throw an exception->now accountHolder is present in 2 db's which are profiles db and transaction db->first check will be if the account with the given accountHolder id exists in the transaction db or not->if it exists then record the transaction->if not exists then the transaction db doesnt have that data but it may be present in the profiles db->so we do an inter service communication call to the profiles microservice from transaction microservice->if the accountHolder id is present in the profiles db then record the accountHolder in transaction db and record the transaction object->if not present in the profiles db then throw an exception

fork and join

discovery client-register to eureka server and fetch list of available microservices

address of one microservice to be provided automatically without hardcoding urls

client will send the request to gateway, which will call to microservices

zuul is not getting used nowadays for api gateway

gateway will help in only single url to communicate with all the other microservices

zipkin for distributed tracing, alternate is telemetry

for tracing the requests, zipkin server is used

External Configuration Steps--

git init

git add .

git commit -m "all configs"

from resources, delete application properties file

and put bootstrap properties file

================================================================