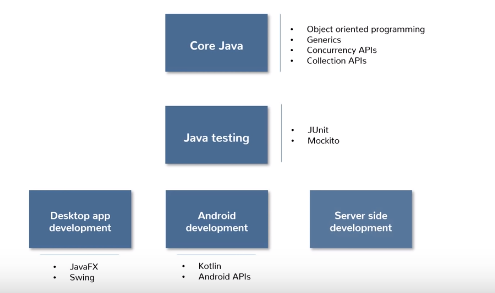
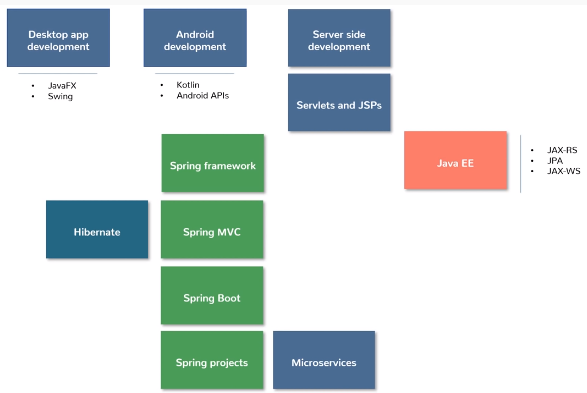
<https://www.udemy.com/>

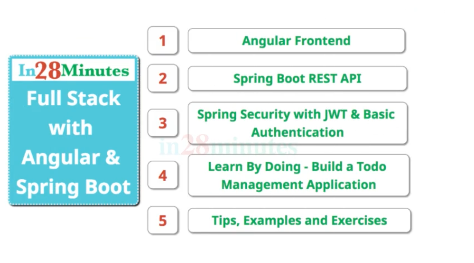
[tlin0328@gmail.com](mailto:tlin0328@gmail.com) Childtime.2000

Interview question <https://www.fullstack.cafe/blog/8-ultimate-full-stack-interview-questions-and-answers>









<https://github.com/in28minutes/full-stack-with-angular-and-spring-boot>

1. Install Node.js

Node –v 12.16.1

Npm –v //6.13.4

Npm init //as package name ….. create a package.json

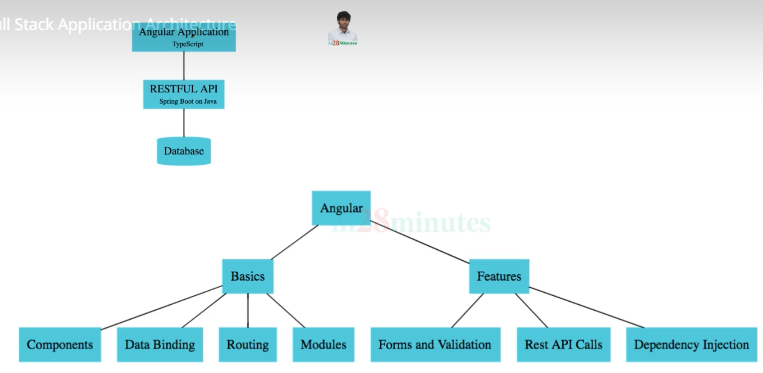
Npm install //….. create a package.json

Npm jquery // create a jquery as dependence in json file.

Review Visual studio code IDE <https://code.visualstudio.com/docs/getstarted/introvideos>

1. Not related to this training, I found another Angular 6 tutorial. <https://www.tutorialspoint.com/angular6/angular6_event_binding.htm>

Graphic draw tool for svg <https://inkscape.org/>



1. JWT – Json Web Token
2. Value : number declare value as number in type script

Value =1

Value = “Five” //compile error

Function add (p1 : number, p2: number) : number {

Return p1 +p2

} // return number

1. Install angular cli

npm install -g @angular/cli use npm to install cli

ng version

ng help // show all functions

ng new –help // show all options for new

ng build //create a dist folder with all js files

ng lint

ng test

ng e2e

ng serve

1. Create a workspace and such as angular/frontend and run cd to frontend dir and create a project named to do using ng command ng new todo //package successful created

Cd todo

Ng serve //compiler successful

open your browser on http://localhost:4200/

1. Go to visual studio code CTRL b to open navigator window

Update scr/index.html

1. Component used @Component as declare and contain three attributes ({selector, templateURL, styleURLs})
2. @Component({
3. selector: 'app-root',
4. templateUrl: './app.component.html',
5. styleUrls: ['./app.component.css']
6. })

Variable set up in component.ts and output to component.html, when output the variable call interpolation {{data}}

1. Create a component by using command line ng generate component login

Use login.component.html to build a field

Use [(ngModel)] to assign value, the type script will biding data, and used name and parameter don’t need interpolation [(ngModel)] = “username”, it will send username value from ts to html template

When you want to use ngModel, we need to import FormsModule

import { FormsModule } from '@angular/forms';

Use ts file to set variable and even handler

1. \*ngIf
2. <small \*ngIf = "showError">{{errorMessage}}</small>

showError is condition declare in ts event hander function

1. Use router to control the route (app-routing.module.ts) use error component to display the error. (\*\* mean the rest of router display error component )

In the login ts file constructor pass the parameter of router: Router and define the rout when it is successful.

constructor(private router: Router) { }

this.router.navigate(['welcome']);

1. How to pass parameter to route in routing-module

login

this.router.navigate(['welcome',this.username]);

routing-module

{ path: 'welcome/:name', component: WelcomeComponent },

And go to rout (welcome) ts file to accept the parameter by using ActivatedRoute

constructor(private route:ActivatedRoute) { }

assign new variable and assign the name value by using snapshot.param[]

ngOnInit(): void {

    this.name = this.route.snapshot.params['name']

  }

1. Create an Array and use \*ngFor

todos = [{

    id : 1,

    description: 'Learn how to dance'

  },

  {

    id : 2,

    description: 'Learn how to study'

  }

]

<tr \*ngFor="let todo of todos">

            <td>{{todo.id}}</td>

            <td>{{todo.description}}</td>

        </tr>

1. Use routerLink to redirect, the different between routerLink and href is the routerLink is not refresh page and used inside the router component that used in routing file and href can go outside of component

<div>Please go to To Do List, please click <a routerLink="/todo">here</a></div>

1. Use | to convert data such as date, uppercase, lowercase

<td>{{todo.targetDate | date}}</td>

1. Index.html =><app-root>(app.component.ts)=>app.module.ts=>bootstrap: [AppComponent] =>app.component.html call <app-router> , use html to control layout
2. Download bootstrap (unpkg bootstrap 4), use 4.10 css main.css

<https://unpkg.com/browse/bootstrap@4.1.0/dist/css/bootstrap.min.css>

@import in style.css

@import url(https://unpkg.com/bootstrap@4.1.0/dist/css/bootstrap.min.css)

1. CSS stylesheet

Menu:

<header>

    <nav class="navbar navbar-expand-md navbar-dark bg-dark">

        <div><a href="http://google.com" class="navbar-brand">Hello 2</a></div>

        <ul class="navbar-nav">

            <li><a href="/welcome/Eric in" class="nav-link">Home</a></li>

            <li><a href="/todo" class="nav-link">To Do List</a></li>

        </ul>

        <ul class="navbar-nav navbar-collapse justify-content-end">

            <li><a href="/login" class="nav-link">Login</a></li>

            <li><a href="/logout" class="nav-link">Logout</a></li>

        </ul>

    </nav>

</header>

Footer

<div class="container">

        <span class="text-muted">All Right Reserved @Eric Lin</span>

    </div>

To Do

<h1 class="text-center">To Do List</h1>

<div class="container">

<table class="table">

Login

<div class="alert alert-warning"

<button (click)=handleLogin() class="btn btn-success">Login</button>

1. Create a service ng generate service service/hardcodeAuthentication, create a service is because of same business logic, it shall create in service folder.

Create this service to store as session sessionStorage.setItem

authenticate (username, password){

    console.log('before login ' + this.isLoggedUser());

    if (username === 'My user name' && password === 'dummy'){

      sessionStorage.setItem('authenticatedUser',username);

      console.log('After login ' + this.isLoggedUser());

      return true;

    }

    return false;

  }

  isLoggedUser(){

    let user= sessionStorage.getItem('authenticatedUser');

    return !(user===null);

  }

Then in the login.component.ts can call this service, before you can use this, the service shall load into constructor

constructor(private router: Router, private hardcodeAuthenticate:HardcodeAuthenticationService)

then access the method and pass the parameter

if (this.hardcodeAuthenticate.authenticate(this.username,this.password)

then we can use isLoggedUser to show the menu

\*ngIf ="hardcodeAuthenticationService.isLoggedUser()

1. Use sessionStorage.removeItem to remove the session

sessionStorage.removeItem('authenticatedUser')

1. Create a service to secure the component

Create a service named routGuardService that implement CanActive class

export class RouteGuardService implements CanActivate {

  constructor(private hardcodeAuthenticationService:HardcodeAuthenticationService) { }

  canActivate(route: ActivatedRouteSnapshot, state: RouterStateSnapshot){

    if (this.hardcodeAuthenticationService.isLoggedUser())

    return true;

    else

    return false;

  }

Add redirect to login page if there is hacker attack,

constructor(private hardcodeAuthenticationService:HardcodeAuthenticationService,

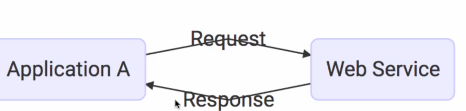
              private router:Router) { }

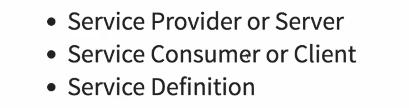
else

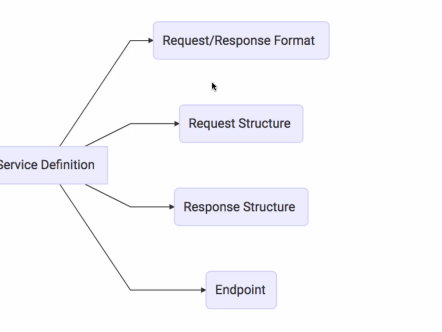
    this.router.navigate(['login']);

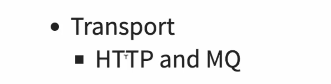
1. Web service terminology

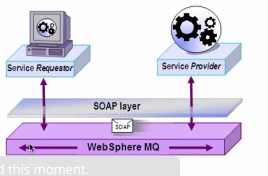


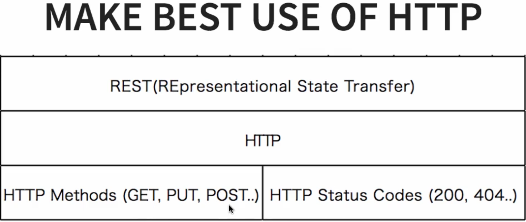


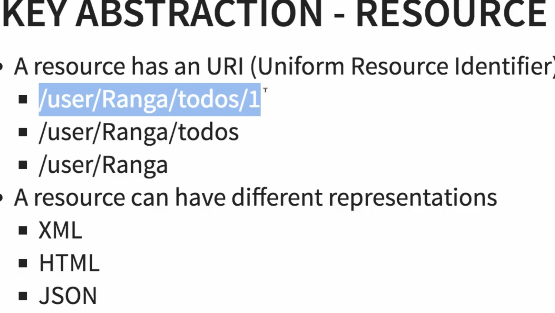


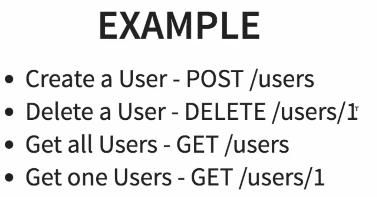












1. Start spring initializr https://start.spring.io/

Group: com.javademo.rest

Artifact: restful-web-services

Dependency

Web

Devtool

JAP

H2

Use import to and point to the workspace of unzip file, the spring maven application will be created

Open a Application.java and run

1. Spring boot makes it easy to create stand-alone, production-grade spring application.

Spring:

Application Framework

Programming and configuration model

Infrastructure support

1. Create a new Controller class under the same package

Use annotation to declare the Controller and path

@RestController

@RequestMapping (method=RequestMethod.***GET***,path="/hello-world") or @GetMapping (path = "/hello-world")

1. Java Bean: is a java class it should have a no-arg constructor, it should be Serializable, it should provide methods to set and get the values of the properties, used to encapsulate many objects into a single object (the bean)

Create helloWorldBean method in controller

@GetMapping (path = "/hello-world-bean")

**public** HelloWorldBean helloWorldBean() {

**return** **new** HelloWorldBean("Hello World Bean");

}

Create a HelloWorldBean class and generate setter and getter with property

For debug purpose set up in application.property

logging.level.org.springframework = debug;

1. Create a Path variable

//create a path variable

@GetMapping (path = "/hello-world/{name}")

**public** HelloWorldBean helloWorldPathVar(@PathVariable name) {

**return** **new** HelloWorldBean(String.*format*("Hello World %s", name));

}

1. How to connect JAVA API to Angular?

* (JAVA) Create a controller to identify the path (path variable) and return bean class, also identify the request URL which can access the class
* (JAVE) Create a bean class with getter and setter to return json property value.
* (Angular) create a component (html and ts) and inject a service.
* (Angular) Create a service (import HttpsClientModule in app.module first), the use the HttpClient method in constructor (service file)
* import { HttpClientModule } from '@angular/common/http';
* (Angular)Create a method to invoke the HttpClient and use get(URL) and return to component.
* When click the request and debug in console it shows observable but there is network traffic, which means the request is observable from request but the request has no response from end server. So use subscribe() to debug the issue from browser.
* 
* (JAVA) add @CrossOrigin annotation in control class

@CrossOrigin(origins= "http://localhost:4200")

1. How to display the console data into Angular?

* In inside the subscribe, pass the response as key to value array and pass to another method, this technic will avoid the time to wait within the subscribe.
* In the new method, set up a new property that is going to use in html for displaying purpose, the method can only see pass response “message”, and the “message” need to be uploaded before loading into the service, therefore, create another class with a constructor to load this “message” finally use the variable to display in html file

getresponse(response){

      //console.log(response);

      //console.log(response.message);

      this.weclomeMessageFromBean=response.message;

  }

export class WelcomeBean{

  constructor (private message:string){}

}

exectuteWelcomeData(){

    return this.http.get<WelcomeBean>("http://localhost:8080/hello-world-bean");

    //console.log('Welcome to this page');

  }

1. Display the error message.

Use RuntimeException to show the error in Java for testing purpose

@GetMapping (path = "/hello-world-bean")

**public** HelloWorldBean helloWorldBean() {

**throw** **new** RuntimeException ("Runtime error");

//return new HelloWorldBean("Hello World Bean - change");

}

Then in Angular to get error response

this.myservice.exectuteWelcomeData().subscribe(

      //pass response to method

      responseResulst=>this.getResponse(responseResulst),

      //pass error to method

      error=>this.getErrorResponse(error)

    );

getErrorResponse(response){

    //console.log("response below");

    //console.log(response);

    //console.log("response error below");

    //console.log(response.error.message);

    //console.log("response End");

    this.weclomeMessageFromBean=response.error.message;

1. Display in html:

<div class="container" \*ngIf="weclomeMessageFromBean">The message from Bean is {{weclomeMessageFromBean}}</div>

1. Accept parameter from API, use tick `

return this.http.get<WelcomeBean>(`http://localhost:8080/hello-world/${name}`);

1. CRUD and API

Create, Update and Delete methods: all used ResponseEntity.

Methods in service.

findAll()

deleteByID(**long** id)

findByID(**long** id)

save(TodoBean todo)

Read All Records in JAVE for example To Do List:

Create a TodoController: that class includes all API methods and calls service class to execute the actions.

Use @RestController

Use @CrossOrigin(origins= "http://localhost:4200") to indicate source API

Use @GetMapping("/user/{username}/todos") on the top of getAllList function

Use @PathVariable to pass Path variable

The following method is used for displaying all to do list.

@Autowired

**private** TodoService todoService = **new** TodoService();

@GetMapping("/user/{username}/todos")

**public** List<TodoBean> getAllTodo(@PathVariable String username) {

**return** todoService.findAll();

}

Create a TodoBean: that includes the to do property and getter and setter.

Create a TodoService and temporary hardcode the to do list

**private** **static** List<TodoBean> *todos* = **new** ArrayList<TodoBean>();

**private** **static** **int** *idCounter* = 0;

**private** **static** Date *date* = **new** Date();

**static** {

*todos*.add(**new** TodoBean(++*idCounter*, "tlin", "Learn Java", *date*, **false**));

*todos*.add(**new** TodoBean(++*idCounter*, "tlin", "Learn JavaScript", *date*, **false**));

*todos*.add(**new** TodoBean(++*idCounter*, "tlin0328", "Learn Drupal", *date*, **false**));

*todos*.add(**new** TodoBean(++*idCounter*, "tlin", "Learn Spring", *date*, **false**));

}

**public** List<TodoBean> findAll(){

**return** *todos*;

}

1. Create a Delete API first, because the method will be used for updated API.

Controller: pass username and ID. Call deleteByID method in service and then noContent().build for debug purpose.

//in Controller:

@DeleteMapping("user/{username}/todos/{id}")

**public** ResponseEntity<Void> deleteTodo(@PathVariable String username, @PathVariable **long** id) {

TodoBean todo = todoService.deleteByID(id);

**if** (todo != **null**) {

**return** ResponseEntity.*noContent*().build();

}

**return** ResponseEntity.*notFound*().build();

}

//In Service, the deleteByID call findByID (return the fining todo) then use todos.remove to remove the todo.

**public** TodoBean deleteByID(**long** id){

TodoBean todo = findByID(id);

**if** (todo ==**null**) **return** **null**;

**if**(*todos*.remove(todo)) {

**return** todo;

}

**return** **null**;

}

**public** TodoBean findByID(**long** id) {

// **TODO** Auto-generated method stub

**for** (TodoBean todo: *todos*) {

**if** (todo.getId() == id) {

**return** todo;

}

}

**return** **null**;

}

1. Create update (PUT) API

Controller: pass username and ID

//The getTodo method is used for displaying record before user update.

@GetMapping("/user/{username}/todos/{id}")

**public** TodoBean getTodo(@PathVariable String username,@PathVariable **long** id) {

**return** todoService.findByID(id);

}

Use spring ResponseEntity to pass three parameters (username, id, and RequestBody) to service. Then return the new responseEntity

@PutMapping("user/{username}/todos/{id}")

**public** ResponseEntity<TodoBean> updateTodo(@PathVariable String username, @PathVariable **long** id, @RequestBody TodoBean todo ) {

TodoBean updateTodo = todoService.save(todo);

**return** **new** ResponseEntity<TodoBean>(updateTodo, HttpStatus.***OK***);

}

In service class

**public** TodoBean save(TodoBean todo){

**if** (todo.getId()<=0) {

todo.setId(++*idCounter*);

*todos*.add(todo);

}**else** {

deleteByID(todo.getId());

*todos*.add(todo);

}

**return** todo;

}

1. Create add

//return a new URL with new ID

@PostMapping("user/{username}/todos/")

**public** ResponseEntity<Void> createTodo(@PathVariable String username, @RequestBody TodoBean todo ) {

TodoBean createTodo = todoService.save(todo);

URI uri = ServletUriComponentsBuilder.*fromCurrentRequest*().path("/{id}").buildAndExpand(createTodo.getId()).toUri();

**return** ResponseEntity.*created*(uri).build();

}

1. Set up JPA H2 database in application

spring.jpa.show-sql=true

spring.h2.console.enabled=true

the H2 console is http://localhost:8080/h2-console