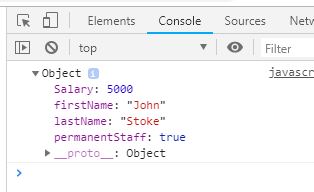
SPA Characteristics - Difference between a Single Page Application and Multi Page Application Settings  
  
Observe the difference between a Single Page Application and Multi Page Application  
  
Multi Page Application

1. Most of the new websites will be a multi-page application
2. In browser go to <https://www.thehindu.com/>
3. When a new article is clicked the page goes blank and the entire page is loaded

Single Page Application

1. In browser go to <https://translate.google.com/>
2. Provide a word in left hands side window
3. The result of the translation is displayed in the right hand side without page reload.
4. **JavaScript Object - Define JavaScript object for employee**  
     
   Create a JavaScript object variable and display the content in console using console.log() method.  
     
   Include the JavaScript in a HTML file and execute it in browser.

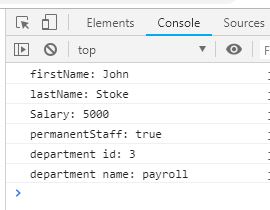
|  |  |  |
| --- | --- | --- |
| **Property** | **Value** | **Type** |
| First Name | John | String |
| Last Name | Stoke | String |
| Salary | 5000 | Number |
| Permanent Staff | true | Boolean |

1. **Sample Output**  
   
2. **JSON - Define JSON for employee details and parse**  
     
   Create JSON text based on the below given employee data and assign to a JavaScript string variable. Then convert it into a JavaScript object using parse() method and display the data by directly reading individual property values and printing using console.log() method. Write an html web page with JavaScript that will implement the above-mentioned features.

|  |  |  |
| --- | --- | --- |
| **Property** | **Value** | **Type** |
| First Name | John | String |
| Last Name | Stoke | String |
| Salary | 5000 | Number |
| Permanent Staff | true | Boolean |

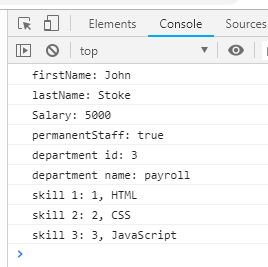
1. **Sample Output**  
   
2. **JSON - Add department details to employee**  
     
   Modify the JSON in the previous program to include department details as per the requirement below. Read the specific property value from the JavaScript object and display the department details.

|  |  |  |
| --- | --- | --- |
| **Property** | **Value** | **Type** |
| Department | Object with id and department name ID = 3 Name = “Payroll” | Object |

1. **Sample Output**

**SON - Add skill details to employee**  
  
Modify the JSON in the previous program to include skill details as per the requirement below. Get the skills property and iterate through the array to display the skill id and name.

|  |  |  |
| --- | --- | --- |
| **Property** | **Value** | **Type** |
| Skills | Array of skill with each skill having id and skill name. Skill 1: id = 1; value = HTML Skill 2: id = 2; value = CSS Skill 3: id = 3; value = JavaScript | Object Array |

**Sample Output**  


**Angular Benefits - Implement SPA using JavaScript**  
  
Get partially implemented code from [here](https://code.cognizant.com/genc-java/examples/blob/master/spa-ajax/ajax.html) and open it in browser.  
  
Clicking on “Get Names” button actually retrieves user details as JSON from internet (https://reqres.in/api/users) and displays them. Refer screen shot below.

|  |
| --- |
| https://cognizant.e-box.co.in/uploads/Image/01fseangular/users1.jpg |

Try accessing <https://reqres.in/api/users> in a browser to get a feel about the JSON data returned.  
  
SME to walk through the JavaScript code.  
  
Implement changes in this code so that it displays more details about each user in the following layout:

|  |
| --- |
| https://cognizant.e-box.co.in/uploads/Image/01fseangular/user2.png |

**Angular Benefits - Implement SPA using jQuery**  
  
The jQuery implementation of the previous problem is available [here](https://code.cognizant.com/genc-java/examples/tree/master/spa-jquery).  
  
SME to walkthrough and explain about the code in jquery.html file.  
  
Modify the code to bring the same layout as done in the previous problem.

**Angular Intro – Hello World in angular**  
  
SME to provide explanation about the following files in the generated code:

1. app.component.ts
2. app.component.html
3. app.module.ts

Modify app.component.html to display the message “Hello World” alone in the view.

**Angular Benefits - Implement SPA using Angular**  
  
The solution for the previous problem is implemented using angular and is available [here](https://stackblitz.com/edit/angular-fchphp?file=src%2Fapp%2Fapp.component.html).  
  
Please go through the code in the following files:

1. app.component.html
2. app.component.ts
3. user.service.ts
4. user.ts

**ES6 - Understanding ES6 features**  
  
The below link contains details and examples about the features of ES6.  
  
SME to walkthrough the following topics in this link:

* let
* const
* Arrow Functions
* Classes

Use the “Try it Yourself” option to demonstrate how each feature works.  
  
<https://www.w3schools.com/js/js_es6.asp>

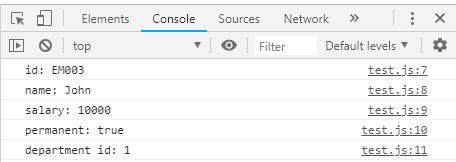
**TypeScript - Write, Compile and Execute a TypeScript program**  
  
Follow the steps below to understand how TypeScript works:

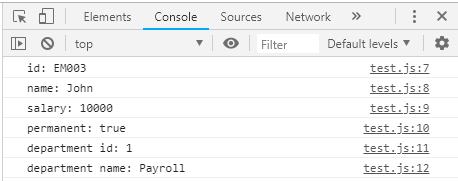
1. Ensure Node JS is already installed in the PC
2. Install TypeScript by executing the command ‘npm install -g typescript’ in command line
3. Go to the link specified below. Copy the TypeScript code under 'Classes' heading.
4. Paste the copied code into Visual Studio Code and save the file as hello.ts.
5. Compile hello.ts using command ‘tsc hello.ts’ in command line which will generate hello.js.
6. Open hello.js in Visual Studio Code and observe the difference in code between ts and js files.
7. Execute hello.js by creating hello.html (refer greeter.html code from the link below)

Reference: <https://www.typescriptlang.org/docs/handbook/typescript-in-5-minutes.html>

**TypeScript data types and interface – Define Employee interface and display**  
  
Define an interface for employee and display the details. Refer file wise specification below.  
  
**employee.ts**  
An interface with name Employee.  
Properties - id, name, salary, permanent with data types number, string, number and boolean types respectively.  
  
**employee-test.ts**  
TypeScript file to test the usage of Employee interface. Implement steps specified below:

* Import Employee interface
* Defines values based on Employee interface and assign to a variable
* Display the employee details using console.log() method.

**employee-test.js**  
Compile employee-test.ts to generate this file.  
  
**employee-test.html**  
Includes employee-test.js that displays the console output in browser.  
  
**Troubleshooting Tips**  
To avoid runtime error include the following line above the script definition tags in employee-test.html.  
    <script> var exports = {}; </script>  
    <script src="employee-test.js"></script>  
  
**Sample Output in Chrome Browser Console**  


**TypeScript data types and interface – Include Department details in Employee and display**  
  
Upgrade the previous problem to include department details in employee.  
  
**department.ts**  
An interface to represent Department.  
Properties – id and name with data types number and string respectively  
  
**employee.ts**  
Import department and include it as attribute.  
  
**employee-test.ts**  
Include code to display department id and name using console.log.  
  
**Expected output in the chrome browser console**  
**s**

**TypeScript data types and interface – Include Skill details in Employee and display**  
  
Make changes in the previous program to include skill details in employee.  
  
**skill.ts**  
An interface to represent Skill.  
Properties – id and name with data types number and string respectively.  
  
**employee.ts**  
Import skill and include an array of skills as attribute.  
  
**employee-test.ts**  
Include code to display all skill id and skill name by iterating through the skills.  
  
**Expected output in the chrome browser console**

**TypeScript data types and interface – Use class to display employee details**  
  
Display Employee details using a class following the steps below:

1. Convert code in employee-test.ts to have a class with class name EmployeeTest.
2. Include employee as property of EmployeeTest
3. Initialize employee property in constructor and include the code that sets the values for properties of employee.
4. Include display() method in EmployeeTest that will display employee details.
5. Instantiate EmployeeTest class, set the employee and invoke the display() method.

**Test Cases**

**TypeScript functions and for .. of**  
  
SME to provide a walkthrough of code as specified below.  
  
**Functions**  
<https://www.typescriptlang.org/docs/handbook/functions.html>  
Explain the code under ‘Type the function’ section.  
  
**for .. of**  
<https://www.typescriptlang.org/docs/handbook/iterators-and-generators.html>  
Explain the code under ‘for .. of statements’ section.

**Angular Intro - Angular App Creation and Testing**  
  
**Steps to create a new angular application and test it:**

1. Create a new application named “angular-learning” using ng new. [NOTE: Select routing support option during application creation]
2. Run the application in browser using ng serve command
3. Stop the service using Ctrl+C keyboard shortcut in command line
4. Test the application using ng test command in the command line
5. This will launch the test results in a browser window.

**Steps to fail a test case:**

1. Open app.component.spec.ts
2. Modify the title value in line 22 to fail the test case. Change angular-learning to angular-learn and save.
3. The test results in browser will display failed error test case details.
4. Revert the code back, so that all test cases succeed.
5. SME to walkthrough the code in app.component.spec.ts.

**Interpolation – Display favorite movie**  
  
Modify angular-learning application to display your favorite movie name.  
  
Create a property named favoriteMovie in app.component.ts  
  
Display the favorite movie in the below format:  
  
**Favorite Movie:** Lord of the Rings  
  
  
**Karma Testing**

* Include test case to validate if favorite movie property and HTML element value matches.
* Use 'expect' function of Karma to perform the validation. 'toEqual' of expect function can be used to do the data comparison.
* The sample snippet as shown below can be used to find the HTML element, h1.
  + const compiled = fixture.debugElement.nativeElement;
  + compiled.querySelector('h1').textContent
* **Interpolation – Display Employee details**  
    
  Modify angular-learning application to display employee’s id, name and salary.  
    
  Use Employee, Department and Skill interfaces created earlier.  
    
  Sample screen layout

|  |
| --- |
| https://cognizant.e-box.co.in/uploads/Image/01fseangular/dd1.JPG |

**Interpolation – Display employee permanent details using ngIf**  
  
Include employee permanent details using ngIf directive.  
  
Sample Layout

|  |
| --- |
| https://cognizant.e-box.co.in/uploads/Image/01fseangular/dd2.JPG |

**Interpolation – Display employee department details**  
  
Interpolation – Display employee department details  
  
Sample Layout

|  |
| --- |
| https://cognizant.e-box.co.in/uploads/Image/01fseangular/dd3.JPG |

**Interpolation – Display employee skill details using ngFor**  
  
Include employee skill details using ngFor directive.  
  
Sample Layout

|  |
| --- |
| https://cognizant.e-box.co.in/uploads/Image/01fseangular/dd4.JPG |

**Interpolation - Display employee details with date of birth**  
  
Inlude dateOfBirth property of type Date in employee.ts.  
  
Set dateOfBirth value in component.ts using new Date('12/31/2000') option.  
  
Include Date of Birth details in html.  
  
Sample Layout

|  |
| --- |
| https://cognizant.e-box.co.in/uploads/Image/01fseangular/dd5.JPG |

**Karma Testing**

* Include test case to check if the department name property and the value in HTML matches.
* Use the sample code snippet as shown below to find the HTML element & perform the comparison with model property
  + const compiled = fixture.debugElement.nativeElement;
  + compiled.querySelector('#departmentName').textContent
  + component.employee.department.name
* **Pipes – Display employee date of birth and salary in custom format**  
    
  Display Salary in currency format with any currency symbol of your choice.  
    
  Display date of birth as per the layout defined below.

|  |
| --- |
| https://cognizant.e-box.co.in/uploads/Image/01fseangular/dd6.JPG |

**Component and Routing – Create View Employee Component**  
  
Earlier employee details were created in the default app component itself.

* This now has to be moved to a new component named "view-emp". Follow the steps below:
* Create a new component in angular-learning application created earlier with component name "view-emp".
* Copy and paste relevant code from html and component source from app component to this new component.
* Create another component "edit-emp" to have a placeholder for another component that will be developed later. This is primarily to check how router navigation works.
* Include routing with router link “View Employee” and "Edit Employee" pointing to this new component. Clicking either one of the link should display the respective component. Refer sample layout below.

**View Employee Component Layout**

|  |
| --- |
| https://cognizant.e-box.co.in/uploads/Image/01fseangular/comp1.JPG |

**Edit Employee Component Layout**

|  |
| --- |
| https://cognizant.e-box.co.in/uploads/Image/01fseangular/comp2.JPG |

If required, use the CSS below to underline the component title that is currently active. The class active-link is the one that needs to be applied in routerLinkActive.

|  |
| --- |
| body {  font-family: Segoe UI, Arial  }  a:visited {  color: blue  }  a {  text-decoration: none  }  .active-link {  border-bottom: Solid 1px blue  } |

**Event Handling - Display value based on button click**  
  
In online food ordering apps, to add a specific food item with multiple quantity, there will be a plus and minus button that helps in increasing or decreasing the number of items. We are gradually going to build a component that will achieve this functionality.  
  
As part of this particular exercise we will create a button named "Click me". Clicking on this button should display the message "Click me button clicked!"  
  
Create a new component named "quantity-increment".  
  
In this component include a separate router link as per the layout below.  
  
Layout before button click

|  |
| --- |
| https://cognizant.e-box.co.in/uploads/Image/01fseangular/event1.JPG |

Layout after button click

|  |
| --- |
| https://cognizant.e-box.co.in/uploads/Image/01fseangular/event2.JPG |

Karma Testing

* Test if button click displays the message "Click me button clicked!"
* Include below imports:

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

import { DebugElement } from '@angular/core';

* Define debugElement after fixture:

let debugElement: DebugElement;

* Include form module import in beforeEach(async) after declaration:

imports:[FormsModule]

* Include debug element in beforeEach() after component:

debugElement = fixture.debugElement;

* Get value from DOM using debugElement. Below code gets the button element from DOM:

let clickMeButton = debugElement.nativeElement.querySelector('#clickMe');

* Use click() method on clickMeButton to trigger the click event:
* Use fixture.detectChanges() to get the changes made because of the click event.
* Use textContent property of DOM element to get the text content within it:

debugElement.nativeElement.querySelector('#simpleClick').textContent

**Event Handling and Two Way Binding - increment value in textbox based on button click**  
  
Component: quantity-selector  
  
Modify the “Click me” button as “Add”. Include a new textbox with default value zero. Clicking on the “Add” button should increment value in the textbox by 1.   
  
Sample Layout

|  |
| --- |
| https://cognizant.e-box.co.in/uploads/Image/01fseangular/event3.JPG |

**Event Handling and Two Way Binding – Implement quantity selector for Food App**  
  
Find below a sample layout for selecting quantity in a\ food app.  
  
This provides option to choose the quantity using a textbox having a plus icon in the right hand side and minus icon in the left hand side.  
  
The default value in the textbox should be zero.  
  
Clicking plus should increment the value. Clicking minus should decrement the value.  
  
If the value is zero, then minus icon should be disabled.  
  
Sample Layout

|  |
| --- |
| https://cognizant.e-box.co.in/uploads/Image/01fseangular/event4.JPG |

Note: Minus button is disabled

|  |
| --- |
| https://cognizant.e-box.co.in/uploads/Image/01fseangular/event5.JPG |

Note: Minus button is enabled  
  
**Karma Testing**

* Include test to check if increment works correctly.
* Include value attribute with interpolation to quantity textbox. This is required for Karma testing to pick the data.
* Include fixture.detectChanges() call in the start of the test method to ensure the test runs after initial value 0 is set in the DOM element.

[1 Template Driven Form](javascript:void(0);)[Scores](javascript:void(0))

**Objectives**

* Demonstrate implementation of forms using Template Driven Forms
  + template reference variable, [(ngModel)], FormModule, Validation Properties (invalid, touched, dirty, errors.required, errors.minlength, json pipe, binding ngModel with radio buttons using [value], binding ngModel with select drop dwon using [value], [selected], (change) and $event.target.value, karma testing with createNewEvent() and dispatchNewEvent()
    - Forms Overview - https://angular.io/guide/forms-overview
    - Template Driven Form - https://angular.io/guide/forms
    - Form Validation - https://angular.io/guide/form-validation

[](https://cognizant.e-box.co.in/projectComponent/show/15679) [](https://cognizant.e-box.co.in/quiz/show/15680)

[Reactive Form](javascript:void(0);)[Scores](javascript:void(0))

**Objectives**

* Demonstrate implementation of forms using Reactive Forms with validations
  + ReactiveFormsModule, FormControl, [formControl], formControlName, set form control values, FormGroup, Validators (required, minlength, maxlength), read value from FormGroup, definition of FormGroup in <form>, nested form groups, FormBuilder, Dynamic Forms, FormArray, formArrayName, FormBuilder.array(), FormBuilder.control(), push new form control
    - Reactive Form - https://angular.io/guide/reactive-forms
    - Dynamic Form - https://angular.io/guide/dynamic-form
    - Form Validation - https://angular.io/guide/form-validation

**Reactive Form - New Component and Name field inclusion**  
  
Similar to the hands on done for Template Driven Form, we will develop the same edit employee form using Reactive Forms. The form will be gradually built with increasing complexity.  
  
In this hands, we will create a new component and add the name field for employee.  
  
Follow instructions below:

* Create a new component named 'edit-emp-reactive'
* Include ReactiveFormsModule in app.module.ts
* Import FormControl in edit-emp-reactive component.ts.
* Include a new instance variable in component.ts. Refer code below

name = new FormControl('');

* Refer below code to include the form element and map it to the form control

Name: <input type="text" [formControl]="name">

* Set a default value for the field by changing the name property as specified below.

name = new FormControl('John');

**Sample Layout**

|  |
| --- |
| https://cognizant.e-box.co.in/uploads/Image/01fseangular/tdf2.png |

**Reactive Form - Include form group and validation for name field**  
  
Instructions for getting employee data from instance variable and include validations:

* Import Employee, Skill and Department into component.ts
* Copy and paste the Employee test data from template driven form instance variable
* Set the data for form control from employee property. Refer code below.

name = new FormControl(this.employee.name);

* Include FormGroup and Validators in import

import { FormGroup, FormControl, Validators } from '@angular/forms';

* Define form group property in component.ts as specified below:

this.empForm = new FormGroup({

'name': new FormControl(this.employee.name, [

Validators.required,

Validators.minLength(4),

Validators.maxLength(20)

])

});

* Include getter method for name. This is for the validator to retrieve the value in the form field.

get name() { return this.empForm.get('name'); }

* Modify template with inclusion of FormGroup and use the template driven form validation code.

**Reactive Form - Include salary, permanent and department fields**  
  
Implement rest of the form fields for edit employee form. Refer information below

* Utilize the HTML template code implemented in template driven form including display of employee JSON and form JSON
* Define department as FormControl
* Define the department drop down in HTML template as specified below:

    <div>

        <h3>Department</h3>

        <select formControlName="department">

            <option \*ngFor="let department of departments"

                    [ngValue]="department"

                    [selected]="department.id === employee.department.id">

                {{department.name}}

            </option>

        </select>

    </div>

* Include additional getter methods for each field
* Implement onSubmit() method to print the form details in console by using the value property of form group

**Reactive Form - Using FormBuilder**  
  
Implement employee edit using FormBuilder. Refer instructions below:

* Create a new component named 'edit-emp-form-builder
* Copy existing html and ts code from reactive form
* Modify FormControl defintion to use FormBuilder
* Inject form builder in constructor

**Reactive Form - Dynamic Forms**  
  
Implement adding multiple skills in edit employee form. This should dynamically add new textboxes for each skill.  
  
Refer live example [here](https://stackblitz.com/angular/olrgkrandqm) and implement this functionality.

**Component Interaction - Problem Definition**  
  
Following are the new features that needs to be incorporated in the angular-learning app:

* Include a new screen to display the list of employees
* This screen should also include option to search employees
* For each employee displayed there should be an option to edit
* Clicking on the edit link should lead to the reactive form edit screen
* The edit employee screen should be prepopulated with the respective employee details

Go to subsequent hands on problems to gradually build the above requirements.  
  
**NOTE:** No output is expected as part of this hands on and this is only for problem definition.  
  
**Sample Screen Layout for Employees List**

|  |
| --- |
| https://cognizant.e-box.co.in/uploads/Image/01fseangular/ci2.png |

**Component Interaction - Implement Search**  
  
Keying in a name in the search text box should modify the list of cards immediately.  
  
Implement this search feature based on the steps below:

* Create a property named searchKey and incorporate two way binding with search text box
* Incorporate keyup event in the search text box point to a function named search()
* Implement search() method that will filter using the filter() method of employeeList array with case insensitive comparision of the searchKey and name in the array.
* The result of the filter() method should be assigned to a component property named 'filteredEmployees'
* The \*ngFor in HTML template should be using filteredEmployees instead of employeeList

**Sample Layout**

|  |
| --- |
| https://cognizant.e-box.co.in/uploads/Image/01fseangular/ci5.PNG |

**Component Interaction - Create Component and Display Employee List**  
  
Implement the below steps for creating a component and display employee list:

* Create a new component with name 'employee-list'
* Import Employee, Department and Skill
* Create property named 'employeeList' in this new component of type array of Employee.
* Set data for array of Employee with five employee details with all necessary details filled.
* In the HTML Template incorporate the following:
  + Page title as "Employee List"
  + Include search text box
  + Using \*ngFor iterate through employeeList and display the employee cards as per the layout specified below
  + The 'Edit' link shall currently point to a dummy link '#'
* Include a new link in header as 'Employees' routing to this new component.

**Sample Layout**

|  |
| --- |
| https://cognizant.e-box.co.in/uploads/Image/01fseangular/ci3.png |

**Component Interaction - Master / Detail Component**  
  
The employee-list component needs to be split into two components:

* Parent Component will be employee-list - Contains the page heading and search text box.
* Child Component will be employee-info - Contains the view for displaying a single employee

Refer screen layout below that contains details about the component details included in red.

|  |
| --- |
| https://cognizant.e-box.co.in/uploads/Image/01fseangular/ci4.png |

Follow steps below to incorporate above mentioned changes:

* Create a new component employee-info
* Include property 'employee' of type Employee
* Assign static data to employee property
* Define the Edit link in employee-info HTML as specified below

<a [routerLink]="['/edit-emp',employee.id]">Edit</a>

* Include this component app selector in employee-list component HTML. This will display a single employee in employee-list component.
* Modify the app selector in previous step to be included into \*ngFor and pass the employee reference as input to employee-info component
* Include @Input decorator in employee-info component for the employee property

​​​​​​​**Component Interaction - Link Employee List with Edit Employee**  
  
The edit link in Employee List should open Edit Employee component with respective employee details.  
  
Follow steps below to implement this:

* In app-routing.module.ts modify edit-emp-reactive path to have id parameter for representing employee id. Refer code below.

{ path: 'edit-emp-reactive/:id', component: EditEmpReactiveComponent }

* In employee-info.component.html modify edit link with router link. Refer code below.

<a [routerLink]="['/edit-emp-reactive',employee.id]">Edit</a>

* In edit-emp-reactive.component.ts incorporate the following changes
  + Import ActivatedRoute

import { ActivatedRoute } from "@angular/router";

* Initiliaze the route using constructor. Refer code below.

constructor(private route : ActivatedRoute) { }

* Get the employee id parameter from the router link. Include the below code in ngOnInit() method.

const employeeId = this.route.snapshot.paramMap.get('id');

* Log the employeeId using console.log() to verify if we have got the employeeId appropriately.

**IMPORTANT NOTE**  
Now we need to retrieve the specific employee from the employee list based on employeeId, so that we can populate edit employee form. Again we are supposed to copy the hardcoded employee array from employee-list component. Instead we will use Service feature available in angular to achieve this. Using services we can use this data cross components. The following practice problems will help us implement Services after which we will complete the implementation of Edit Employee component.

**Service - Create service for handling employee data**  
  
Follow steps below the create the service and utilize necessary methods edit-employee-reactive component:

* Create a new service using following command. Execute this command in the root folder of the application.

ng generate service employee

* Create a property named employeeList as array of Employee type.
* Copy and paste the hardcoded employeeList from employee-list component.
* Implement the below methods in the service class.
  + getAllEmployees() : Employee[]
    - This method will return the employeeList property
  + getEmployee(employeeId: number): Employee
    - This method will iterate through employeeList and return based on the matching employeeId
* To use this service edit-employee-reactive component by changing the constructor as following, which will inject the service into this component. Include necessary import for EmployeeService.

constructor(private route : ActivatedRoute, private employeeService : EmployeeService) { }

* Assign the employee property of edit-employee-reactive component using employeeService.getEmployee() method passing the employee id obtained from the router link.

**Guard - Login Component**  
  
Restrict access to edit-employee-reactive component with a login page.  
  
The credentials can be hardcode to 'admin'/'password' for the sake of simplicity.  
  
We will implement guard feature gradually.  
  
In this hands on we will create a login component. Follow instructions below:

* Create new component with name 'login'
* Include properties username and password
* Include form in HTML template with following fields:
  + Username - textbox - include required validation
  + Password - password - include required validation
  + Login - Submit Button pointing to login() method in login component.
* Include code in login() to verify if username is 'admin' and password is 'password', if so navigate to edit-emp-reactive route. The status of successful login needs to be stored in a service. This will be done in next hands on.
* If the credentials does not match display message 'Invalid Username/Password' within the login component.
* Include a new header navigation link named Login and bind a path in router module

**Guard - AuthService**  
  
Implement AuthService that will be used to store and retrieve the authentication status. Follow steps below to implement and use it:

* Create new service named 'auth'
* Include property loggedIn of type boolean.
* Implement method login() that sets the loggedIn property to true.
* Implement method logout() that sets the loggedIn property to false.
* Implement method isLoggedIn() that returns the value of loggedIn property
* Inject this service in login component constructor
* Invoke authService.login() when Username and Password matches to 'admin'/'password'

**Guard - Protect Component using Guard**  
  
Only admins will be able to edit an employee detail. The edit-employee-reactive form needs to be available only after login. If this form is accessed through URL or router navigation link without login, then it should be redirected to login component.  
  
Steps to implement:

* Create a new guard with name auth

ng generate guard auth/auth

* Inject Router and AuthService in the constructor
* Implement canActivate() method that returns boolean
* If login status is true from authService return true else navigate to login component using router.navigate() method.
* In app-routing.module.ts include canActivate for edit-employee-reactive path with reference to AuthGuardService as guard.

**HttpClient - Get User List**  
  
Get list of users from the RESTful Web Service available in URL (https://reqres.in/api/users?page=2) using HttpClient and display the data.  
  
Following implementations are required:

* Create new service named 'user'
* Create new interface 'user' to handle user data
* Implement method getAllUsers() in the service that will retrieve data from the above mentioned URL and return array of User
* Create new component named user that injects user service and get the user array
* User component displays the data in HTML template using user array

**HttpClient - POST, PUT and DELETE calls**  
  
Implement following methods in user service that will make respective RESTful Web Service call

* createUser() - Create relevant JSON data required and invoke the POST CREATE method in reqres.in
* upateUser() - Create relevant JSON data required and invoke the PUT UPDATE method in reqres.in
* deleteUser() - Create relevant JSON data required and invoke the DELETE method in reqres.in

In the user component created in the previous step create buttons to invoke the above service methods and display the reponse in the component's HTML template.

**Test Cases**

**HttpClient - Error Handling**  
  
Demonstrate HTTP error handling by invoking the REGISTER UNSUCCESSFUL call available in https://reqres.in.  
  
Include a method in service named register() which will make the above call and fail. Write necessary error handler to get the error message.  
  
Include a button named "Register" of user component to invoke the above service method and display the error message returned in the HTML template.

**Test Cases**

**Protractor - Setup**  
  
Follow instructions below to setup protractor:

* Open google chrome and find the browser version using Menu > Help > About Google Chrome. Copy the version number and have it ready during subsequent steps.
* Execute the following commands in command line. This will install protractor and starts the webdriver-manager. Don’t forget to change the browser version number.

npm install -g protractor

webdriver-manager update --versions.chrome=[browser-version-number] --ignore-ssl --proxy http://proxy.cognizant.com:6050

webdriver-manager start --versions.chrome 72.0.3626.121

* Create two JavaScript files in a folder:
  + todo-spec.js

describe('angularjs homepage todo list', function() {

  it('should add a todo', function() {

    browser.get('https://angularjs.org');

    element(by.model('todoList.todoText')).sendKeys('write first protractor test');

    element(by.css('[value="add"]')).click();

    var todoList = element.all(by.repeater('todo in todoList.todos'));

    expect(todoList.count()).toEqual(3);

    expect(todoList.get(2).getText()).toEqual('write first protractor test');

    // You wrote your first test, cross it off the list

    todoList.get(2).element(by.css('input')).click();

    var completedAmount = element.all(by.css('.done-true'));

    expect(completedAmount.count()).toEqual(2);

  });

});

* conf.js

exports.config = {

  seleniumAddress: 'http://localhost:4444/wd/hub',

  specs: ['todo-spec.js']

};

* Execute the below command in command line in the folder where .js files were placed:

protractor conf.js

**Proctrator - Test Google**  
  
Test if google search website works correctly using Protractor.  
  
Access the URL https://www.google.co.in/ and validate if the title is 'Google'.  
  
Protractor Reference: <https://www.protractortest.org/#/tutorial>

**Protractor - Test angular-learning app**  
  
Using Protractor verify if login works correctly. Test both success scenario and failure scenario.  
  
After successful login let protract test the following flow:

* Click 'Employees' link and check if search field is available.
* Click 'Edit Employee Reactive' link and check if salary field is available
* Click 'Quantity Increment', click 'Add' button once and see if the value in the textbox is '1'

**Build and Deploy angular app**  
  
Build the angular-learning project using **ng build** and deploy the content of the dist folder into a web server.  
  
Validate if the angular app functions as expected.  
  
Instructions for installing in Tomcat Server

* Create new folder angular-learning in webapps folder of tomcat
* Copy the all files and folders available in 'dist' to the above created angular-learning folder
* Start tomcat server by running the command 'catalina run' in the command prompt in tomcat/bin folder
* Test the application

Instructions for installing in IIS Server

* Open IIS
* Create a Website pointing the physical directory to the application folder inside the 'dist' folder of the application that is being worked on
* Provide a new port address other than the default 80 in the Website bindings
* Browse the site to view and test