# Routes with PrimeNg

## Simple Routes with PrimeNg with navigation programmatically

Create an angular app with the PrimeNg UI components. Start up project can be cloned from <https://github.com/primefaces/primeng>

Include the panel from <https://www.primefaces.org/primeng/#/panel>

Changes to be followed from a simple Anuglar Cli base project, follow the steps

1. Install font-awesome and primeng ["font-awesome": "^4.7.0","primeng": "^4.2.2",]
2. Copy the priming/resources and font-awesome css under assets folder.
3. Update the index.html to include

<link rel="stylesheet" type="text/css" href="/assets/resources/themes/omega/theme.css" />

<link rel="stylesheet" type="text/css" href="/assets/resources/primeng.min.css" />

<link rel="stylesheet" type="text/css" href="/assets/css/font-awesome.min.css" />

1. Include the modules in the app.module.ts

import { BrowserAnimationsModule } from '@angular/platform-browser/animations';

import { PanelModule } from 'primeng/primeng';

import { TabViewModule} from 'primeng/primeng';

Add it in the imports section

1. Review the following links

<https://www.primefaces.org/primeng/#/panel>

<https://www.primefaces.org/primeng/#/tabview>

1. Update the app.component.html

<p-panel>

<p-tabView (onChange)="onTabChange($event)">

<p-tabPanel header="Home">

</p-tabPanel>

<p-tabPanel header="About">

</p-tabPanel>

<p-tabPanel header="Contact Us">

</p-tabPanel>

<p-tabPanel header="Disabled" [disabled]="true">

</p-tabPanel>

</p-tabView>

<router-outlet></router-outlet>

<p-footer>

Footer Contents

</p-footer>

</p-panel>

1. Update app.component.ts

constructor(private router: Router){

}

onTabChange(event) {

console.log(event.index);

switch(event.index){

case 0:{

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

break;}

case 1:{

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

break;}

case 2:{

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

break;}

}

}

1. Run and see the routes.

## Simple Child routes with params

1. Update the constants to include the child. (change the ‘about’ in app.component.ts to ‘product’)

export const routes: Routes = [

{ path: '', redirectTo: 'home', pathMatch: 'full' },

{ path: 'home', component: HomeComponent },

{ path: 'product', component: ProductDetailComponent, children: [

{path: '', component: ProductDescriptionComponent},

{path: 'seller/:id', component: SellerInfoComponent}

]},

{ path: 'contact', component: ContactComponent },

{ path: 'contactus', redirectTo: 'contact' },

];

1. Create ProductDetailComponent (the component with children should have links and router-outlet)

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

@Component({

selector: 'product',

styles: ['.product {background: cyan}'],

template: `

<div class="product">

<h1>Product Detail for Product</h1>

<router-outlet></router-outlet>

<p><a [routerLink]="['./seller', sellerId]">Seller Info</a></p>

</div>

`

})

export class ProductDetailComponent {

sellerId = 5678;

}

1. Create ProductDescriptionComponent (simple component) and SellerInfoComponent.

Update SellerInfoComponent

@Component({

selector: 'seller',

template: 'The seller of this product is Mary Lou {{sellerID}}',

styles: [':host {background: yellow}']

})

export class SellerInfoComponent {

sellerID: string;

constructor(route: ActivatedRoute){

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

console.log(`The SellerInfoComponent got the seller id ${this.sellerID}`);

}

}

## Simple QueryParams

Two advantages over params

* Parameters can be optional
* Preserve the params across route navigation

1. Update the productdetail.component.ts in the previous example

<p><a [routerLink]="['./seller']" [queryParams]="{sellerId: sellerId}">Seller Info</a></p>

1. Remove the :id in the route configuration
2. Update the seller info component

this.sellerID = route.snapshot.queryParamMap.get('sellerId');

1. Run and see the result.
2. Go to contact component, the query string is not preserved. To preserve update the app.component.ts

this.router.navigate(['contact'],{ queryParamsHandling: 'preserve'} );

## Name the outlets (Secondary Routes)

1. Update the configurations in the previous example

{ path: 'contact', component: ContactComponent, outlet: "aux" },

1. Update the app.component.ts

onTabChange(event) {

console.log(event.index);

switch(event.index){

case 0:{

this.router.navigate([{ outlets: { primary: ['home'],

aux: null }}]);

break;}

case 1:{

this.router.navigate([{ outlets: { primary: ['product'],

aux: null }}]);

break;}

case 2:{

this.router.navigate([{ outlets: { aux: 'contact' }}],{ queryParamsHandling: 'preserve'} );

break;}

}

# TypeScript

Open the TypeScript playground and http://www.typescriptlang.org/Playground and try transpiling the following code samples one by one. If a fun

1. let and class

let foo: string;

class Bar {

}

console.log('Hello TypeScript');

2. Fat arrow function

let getName = () => 'John Smith';

console.log(`The name is ` + getName());

3. The this problem

function StockQuoteGeneratorArrow(symbol: string){

this.symbol = symbol;

setInterval(() => {

console.log("StockQuoteGeneratorArrow. The price quote for "

+ this.symbol + " is " + Math.random());

}, 1000);

}

let generator = new StockQuoteGeneratorArrow("IBM");

4. A class with constructor and a private variable

class Person {

public firstName: string;

public lastName: string;

public age: number;

private \_ssn: string;

constructor(firstName:string, lastName: string, age: number, ssn: string) {

this.firstName = firstName;

this.lastName = lastName;

this.age = age;

this.\_ssn = ssn;

}

}

var p = new Person("John", "Smith", 29, "123-90-4567");

console.log("Last name: " + p.lastName + " SSN: " + p.\_ssn);

5. A class with auto-generate member variables

class Person {

constructor(public firstName: string,

public lastName: string, public age: number, private \_ssn: string) {

}

}

var p = new Person("John", "Smith", 29, "123-90-4567");

console.log("Last name: " + p.lastName + " SSN: " + p.\_ssn);

6. Destructuring:

function getStock(){

return {

symbol: "IBM",

price: 100.00,

open: 99.5,

volume:100000

};

}

let {symbol, price} = getStock();

console.log(`The price of ${symbol} is ${price}`);

// outputs: The price of IBM is 100

# Template Driven – Forms

## Template Driven

<form #f="ngForm" (ngSubmit)="onSubmit(f.value)">

<p-panel header="Simple Form">

<div class="ui-grid ui-grid-responsive ui-grid-pad ui-fluid">

<div class="ui-grid-row">

<div class="ui-grid-col-2">

<label for="skuInput">SKU</label></div>

<div class="ui-grid-col-6">

<input type="text" id="skuInput" placeholder="SKU" name="sku" ngModel />

</div>

</div>

<div class="ui-grid-row">

<div class="ui-grid-col-2"></div>

<div class="ui-grid-col-2">

<button pButton type="submit" label="Submit"></button>

</div>

</div>

</div>

</p-panel>

</form>

Update app.module.ts

import {ButtonModule} from 'primeng/primeng';

import {GrowlModule} from 'primeng/primeng';

Update app.component.ts to include a simple submit handler. Include growl component. In the html

<p-growl [value]="msgs" sticky="sticky"></p-growl>

msgs:Array<Object>;

constructor(){

this.msgs = new Array<string>();

}

onSubmit(form:any){

this.msgs.push({severity:'info', summary:'Success', detail:'Form Submitted'});

console.log(form);

}