Angular Routing

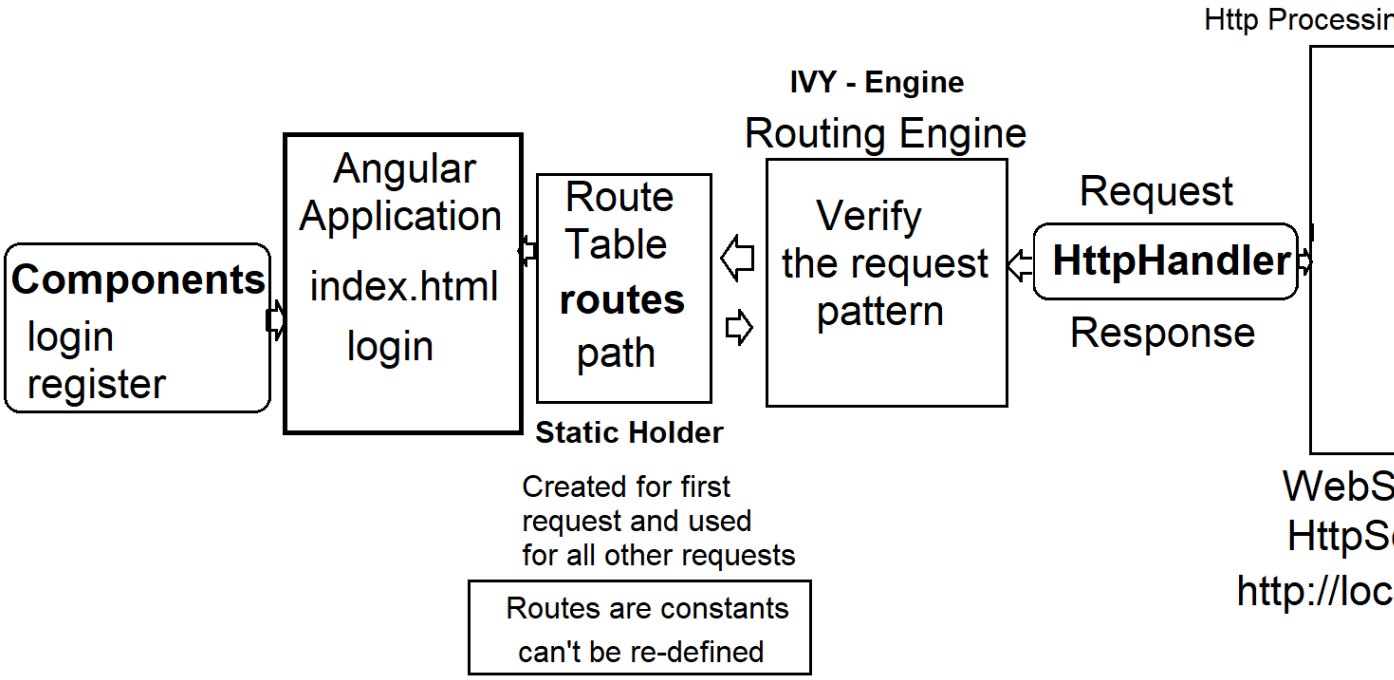
- Web development requires several built-in objects provided by technologies.
    - Session Object
    - Application Object
    - Cookie Object
    - Response Object
    - Request Object
- Web development uses several techniques
    - Caching
    - Hosting
    - Deploying
    - Testing
    - Routing
- Routing is a technique used for web application.
- Routing technique was introduced into web application to create and configure User and SEO friendly URL’s.

- User friendly URL allows the user to access any content directly by querying from URL.
- It also makes the content accessible without much querying.

  Previous:
  http://www.amazon.in/electronics.html?category=electronics&subcat=mobiles&model=samsung

  Routing:
  http://www.amazon.in/electronics/mobiles/samsung
- SEO friendly URL can identify the exact location on your page. And provide suggestions for next time with relative content.
- Routing in SPA application allows the user to stay on one page and get access the everything from the page.
- In SPA new details are added into the page without reloading the complete page.

- Routing uses built-in AJAX for accessing the components and rendering into page.
- Routing can be defined
  - Server-Side
  - Client-Side
- Angular implements routing client side.



Angular Routing Library

- **"@angular/router"** is a package that provides set of modules used to configure

the routes and export routes for application.

- The basic modules required to configure routes for angular application.
  - RouterModule
  - Routes

Routes	- It configures a route object that comprises of collection of routes. - The routes collection is a constants collection with various details like path, component, outlet etc. Syntax: const routes: Routes = [{path:'' }, {path:''}];
RouterModule	It imports the routes

	and exports into route table. Whenever client request comes, it verifies from route table and renders to client.

- Routing is configure in a separate routing module

“app-routing.module.ts”

```
import { NgModule } from ‘@angular/core’;
import { Routes, RouterModule } from ‘@angular/router’;
const routes: Routes = [ { }, { } ];
@NgModule({
    imports: [RouterModule.forRoot(routes)]
    exports: [RouterModule]
```

})

- In Routing the resulting markup will be rendered at specified location by using a “<router-outlet>”, which is a member of RouterModule.
- Every page can be defined with only one <router-outlet>.
- Router outlet can be nested in hierarchy not multiple.
- You have to configure routes for your application using a “Routes” collection.
- Routes is a collection of route objects.
- Every route object comprises of following properties

Property	Description
path	It is the request path used to access any resource in application. It specifies the request name.

	Syntax: { path: 'home' } Request: http://localhost:4200/home Path can be defined with wild card routes: Syntax: { path: ' ' } – If no component is requested then what to do? { path: '**'} – If the component requested is not available in application then what to do?
component	It refers the component to be rendered when any specific path is requested by client. **Syntax:**

	{path: ‘home’, component: HomeComponent}
redirectTo	It specifies the path for redirection. It uses an existing path instead of loading a new path when ever the condition is matching.
pathMatch	It specifies the URL to access when requested path is similar to existing path. No need to re-define the path. - Full: to match all details defined. - Prefix: to match only the component.

- Hyperlinks are designed to navigate to any specific route path by using “routerLink” attribute, which is used as “href”
  Syntax:
  **<a routerLink=”pathname”> Link Text / Image **

Route Parameters

- Web application use “Query String” technique to transport data across requests.
- Query String appends the data into URL, so that any page can access data from URL.
  Syntax:
  www.amazon.in/electronics.php?category=mobiles&model=samsung
- Applications that using routing will use route parameters instead of query string.
- Route parameters provides an easy technique to query any content directly from URL.
  Syntax:
  www.amazon.in/electronics/mobiles/samsung

FAQ: Can we have query string along with route parameter?

A. Yes.

- **Route parameters are configured in route path.**

  {path: ‘electronics/:parameterName1/:parameterName2’}

- **Actual values are passed into Route parameters from URL or any Link reference.**

  URL: http://localhost:4200/electronics/value1/value2

  Anchor:

<a routerLink=”electronics/value1/value2”>

ParameterName1 = value1

ParameterName2 = value2

- You can access and use the route parameters in component by using “**ActivatedRoute”**object.

  Syntax:
  private route: ActivatedRoute

  this.route.snapshot.paramMap.get(“RouterParameterName”);

Summary

- Configure Route Parameters
  Syntax:

{path: ‘search/:id/:name/:price’,
component: SearchComponent}

- Pass values into parameters
  http://localhost:4200/search/1/tv/34000
  <a routerLink=”search/1/tv/34000>
  Search

- Access and use parameters

  public id =
  this.route.snapshot.paramMap.get(‘id’)

Configuring Child Routes

- Child routes are used to define navigation within the component.
- A route collection can be maintained within the context of existing route.
- You can query and access any content with in the current component.

- The child routes are defined by using “children[]” collection

  Syntax:

```
const routes: Routes = [
  {
    path: ‘parent’, component: ‘ParentComponent’, children: [
            { path: ‘childPath’, component: ‘ChildComponent’ }
        ]
  }
]
```

- Redirection to child routes the defined by using “navigate()” of “RouterModule”.

  Syntax:

```
private router: Router;
this.router.navigate([‘path’, parameters], { relativeTo: this.route })
```

Lazy Loading of Routes

- Eager Loading
- Lazy Loading
- All Angular modules “NgModules” of application are eagerly loaded.
- Modules are loaded along with application.
- It makes application heavy on browser.
- It makes page rendering slow.
- Lazy loading allows to load “NgModules” only when they are required.
- It keeps initial bundle size smaller.
- It improves the render time.
- It decreases the page load time.
- It is more light weight on browser.
- It can reach broad range of devices like mobile to browser.
- Modules can be loaded with lazy loading pattern

  Syntax:

  { path: ‘electronics’, component: ElectronicsComponent } // not lazy

{ path: ‘electronics’, loadChildern:() => import(‘./electronics.component’).then(c => c.ElectronicsComponent) } // lazy

Ex:

- Create a new Application
  > ng g application shopping --routing
- Open “app” folder of shopping project in terminal
- Generate the following modules
  > ng g module customers --route customers --module app.module
  > ng g module vendors --route vendors --module app.module

- Every module act as an application within the root application.

- Every module comprises of its own routing and module registrations.

- Every module is a collection of components, services, pipes etc.

- Every module comprises of routes loaded “forChild()”.

Syntax:

Customers-routing.module.ts

```
import { NgModule } from '@angular/core';
import { Routes, RouterModule } from '@angular/router';


import { CustomersComponent } from './customers.component';


const routes: Routes = [{ path: '', component: CustomersComponent }];


@NgModule({
  imports: [RouterModule.forChild(routes)],
  exports: [RouterModule]
})
```

export class CustomersRoutingModule { }

- The lazy routes are configured at root level in the route configuration

Syntax:

App-routing.module.ts

{ path: ‘moduleName’, loadChildren: (import the module).then(load the module) }

Ex:

```
const routes: Routes = [
{ path: 'vendors', loadChildren: () => import('./vendors/vendors.module').then(m => m.VendorsModule) },
{ path: 'customers', loadChildren: () => import('./customers/customers.module').then(m => m.CustomersModule) }
];
```

Authorization

- Authorization is the process of restricting access to the resources in application.
- You can configure components so that they are accessible only to the user when authentication is successfully.
- Authentication is the process of verifying user id, password, security token etc.
- You can restrict access to any component by using “Route Guards”.
- It prevents users from navigating to any specific location without proper authentication.
- You can secure the route path.
- A route guard allows to configure a custom logic and functionality, where we can verify the user credentials and allow access or restrict access.
- Angular provides various route guards:

CanActivate	It restricts access to a specific route.

CanActivateChild	It restricts access to child route.
CanDeactivate	It is used to restrict the user to exit the route. [can he come out of the route?]
Resolve	It is used to access data from any API before route activation.
CanLoad	It is authorizing the lazy routes.

- All route guards can return different types of values. Usually a boolean value is used to confirm the user authentication.
- You have to generate route guard for the route you want to restrict.
  Syntax:
  > ng generate guard <guard-name>

- The route guard contains logic that verifies user credentials and gives access to components only when authentication is successful.

Ex:

- Add a new components
  > **ng g c manager-home**
  > ng g c login
- Generate a route guard for Manager component
  > **ng g guard manager-guard**
- Go to “data.service.ts” and a new service method

```
GetUsers(){
    return [
     {Name: 'John', Role: 'Admin', Pwd: 'admin12'},
      {Name: 'David', Role: 'Manager', Pwd: 'mng12'}
   ];
 }
```

- Go to “manager-guard.guard.ts”

```
import { Injectable } from
'@angular/core';
import { CanActivate,
ActivatedRouteSnapshot,
RouterStateSnapshot, UrlTree, Router }
from '@angular/router';
import { Observable } from 'rxjs';
import { DataService } from
'./data.service';

@Injectable({
  providedIn: 'root'
})
export class ManagerGuardGuard
implements CanActivate {
  constructor(private data: DataService,
private router: Router){

  }
  public users = [];
```

```
  public username = 'David';
  public password = 'mng12';

  canActivate(
    route: ActivatedRouteSnapshot,
    state: RouterStateSnapshot): boolean {
     this.users = this.data.GetUsers();
     for(var user of this.users){
       if(user.Name==this.username &&
user.Pwd==this.password) {
         return true;
       }
     }
     this.router.navigate(['login']);
  }
}
```

- **Go to “app-routing.module.ts”**

```
  {path: 'login', component:
LoginComponent},
  {path: 'manager', component:
ManagerHomeComponent,
canActivate:[ManagerGuardGuard]},
```

- **Go to “app.component.html”**

```
<li>
        <a
routerLink="manager">Manager</a>
</li>
```

- **Go to login.component.html**

```
<h2>Manager Login</h2>
<div>
  <dl>
    <dt>Name</dt>
    <dd><input type="text"></dd>
    <dt>Password</dt>
    <dd><input type="password"></dd>
  </dl>
  <button>Login</button>
</div>
```