Task 8. Routes

Task description

In this exercise we will add the support of the routing in application.

Time

2 hours

Detailed description

8.1 Adding routes

1) Update app.module.ts by adding routes:

```
const appRoutes: Routes = [
  { path: '', component: NotesEditorComponent },
  { path: '**', component: PageNotFoundComponent }
];
```

2) Add RouterModule to imports with definition of appRoutes:

```
imports:     [ BrowserModule, RouterModule.forRoot(appRoutes), …],
```

3) Define NotesEditorComponent by moving bottom <div> container with notes and sections from app.component.html to NotesEditorComponent template and replacing it by

```
<router-outlet></router-outlet>
```

Also move logic from AppComponent to NotesEditorComponent (section field and setSection method).

4) Define component PageNotFoundComponent with template having text “page not found”.

5) Add base href to index.html body:

```
<base href="/">
```

Now you can check if the application works correctly.

8.2 View section

This feature will allow to preview only one section and send it as URL to someone, or add to bookmarks.

1) Create component ViewSectionComponent

2) Add viewSection.component.html should have this code:

```
<div class="col-md-8 col-md-offset-2">
```

```
      <div class="panel panel-primary ">
        <div class="panel-heading">
          <h3 class="panel-title">{{section}}</h3>
        </div>
        <ul class="list-group">
          <li *ngFor="let note of notes"
            class="list-group-item">{{note.text}}</li>
        </ul>
      </div>
    </div>
```

3) Create constructor of ViewSectionComponent this way:

```
constructor(private route: ActivatedRoute) {}
```

4) Define section property in ViewSectionComponent.

In ngOnInit() retrieve the section:

```
ngOnInit() {
  this.section = this.route.snapshot.params["name"];
}
```

Route snapshot returns the values of the parameters at the moment of component initialization. If parameter will change thereafter, it will not be updated.

http://localhost/viewSection/Vacation
http://localhost/viewSection/Work

5) Add viewSection route to appRoutes in app.module.ts:

```
{ path: 'viewSection/:name', component: ViewSectionComponent }
```

Note: add this route before '**' path in appRoutes.

6) Add link to view section:
Edit notes.component.html and change {{section}} to make it a link:

```
<a [routerLink]="['/viewSection',section]">{{section}}</a>
```

Now you can open this link and see the title of the section (but without the notes).

7) Now we need to read the list of notes for the selected section. We already have this logic in NotesComponent, so let we extract this logic to the service.

8) Create services folder and **NotesServerService** inside it:

```
@Injectable()
export class NotesServerService {
  private notesUrl = 'notes';  // URL to web api

  constructor(private http: Http) { }
}
```

Add NotesServerService to providers list is **AppModule**:

```
providers:   [ NotesServerService ]
```

Move method **getNotes(): Observable<Note[]>** from **NotesComponent** to **NotesServerService**, retrieve section as method parameter:

```
getNotes(section): Observable<Note[]>
```

Also mark Note interface in notes.component.ts with **export** to make it available from outside.

Inject **NotesServerService** into **NotesComponent** constructor:

```
constructor(private http:Http, private notesServer: NotesServerService)
{}
```

9) Also inject **NotesServerService** to **ViewSectionComponent**:

```
constructor(private route: ActivatedRoute,
   private noteServer: NotesServerService) {
}
```

10) Create method getNotes() in **ViewSectionComponent**:

```
getNotes() {
   return this.noteServer.getNotes(this.section);
}
```

11) Add field **notes: Note[]** and call to getNotes() in ngOnInit():

```
ngOnInit() {
   this.section = this.route.snapshot.params["name"];
   this.getNotes().subscribe(notes=>this.notes=notes);
}
```

Now you can open notes and use the link to open view section.

8.3 Page reload: process virtual URL on server side

If you will go to view section and reload page, you will get 404 error from the server. Problem is that server thinks that it should load this URL and it doesn't know that client side is responsible to process it.

To fix it, we should change server side. It should return index.html in case of URLs controlled from Angular application.

First, open server.js and define variable root in the beginning:

```
var root = __dirname + '/..'
```

Also update express.static call to

```
app.use(express.static(root));
```

Also add this route to the end of server js (it has to be in the bottom to not override all other routes):

```
app.get("*", function(req, res, next) {
   res.sendFile('index.html', { root : root });
});
```

Now you can reload page like http://localhost:8080/viewSection/Work
However, you will get the error. The reason is that all scripts like

```
<script src="node_modules/core-js/client/shim.min.js"></script>
```

return index.html instead of the correct code. The cause is that this path is relative, so if we go to http://localhost:8080/viewSection/Work
the real URL is

```
http://localhost:8080/viewSection/node_modules/core-js/client/shim.
min.js
```

You should change index.html to return all scripts from the root:

```
<script src="/node_modules/core-js/client/shim.min.js"></script>
```
Or
```
<script src="/systemjs.config.js"></script>
```
Also not forget about CSS link.

Now we have all URLs loaded from the root, however app itself is loaded relatively. To fix it, you should open **systemjs.config.js** and add this:
```
System.config({
  baseURL: "/", ...
```

It will load app from root folder.
Note that this approach will allow to deploy application only to the server root.

Alternate way to fix this is to implement custom route express handler like this:
```
app.get("/viewSection/*", function(req, res, next) {
   var url = req.originalUrl.replace("/viewSection/","");
   if
(url.match("app/*|node_modules/*|systemjs.config.js|css/*|fonts/*")
)
      res.sendFile(url, { root : root });
   else res.sendFile('index.html', { root : root });
});
```

8.4 Navigating sections

Now we want to provide section navigation using URL. It will allow using Back/Forward buttons in browser, as well as adding section to bookmarks.

So, if we go to Work section, browser should show this URL:
```
http://localhost:8080/Work
```

To implement this, we should update **NotesEditorComponent**.

1) In constructor inject router and route:
```
constructor(private route: ActivatedRoute, private router: Router) {}
```

2) In setSection() method, navigate to URL having name of the section:
```
setSection(section:string) {
  this.section = section;
  this.router.navigate([section]);
}
```

3) Also add this route to appRoutes in app.module.ts:
```
{ path: ':name', component: NotesEditorComponent },
```
You have to insert it after path '**viewSection/:name**', but before path '**', so that it wouldn't override viewSection path.

4) If you will run application, you will see that URL changes, but if you will try to go forward or backward, nothing will change. This is because we need to handle the change of the URL by subscribing to the changes. To do it, update constructor in NotesEditorComponent and subscribe to the changes in route params:

```
constructor(private route: ActivatedRoute, private router: Router) {
  this.route.params
     .map(params=>params["name"])
     .subscribe(section=>this.section=section);
}
```

It will change section on every change of URL. Now, notes are loaded for the section in URL because section if transmitted to <notes> component with this:

```
<notes [section]="section"></notes>
```

However, <sections> component do not get updated from URL active section. To change it, you should update sections component by injecting updated section inside component:

```
<sections [section]="section"
(sectionChanged)="setSection($event)"></sections>
```

And in **SectionsComponent** introduce setter for section:

```
@Input()
set section(section:string) {
   if (section && section.length>0) {
      this.activeSection = section;
   }
}
```

So when URL will be changed, this setter will be executed, and **activeSection** will be retrieved from URL and be set.

8.5 Using async pipe in ViewSection [optional]

We was retrieving list of notes in ViewSectionComponent by setting notes value. There's the alternate way to keep notes as the Observable and resolve it in the view. To do it, you should:

1) Introduce field

```
notes$: Observable<Note[]>;
```

in ViewSectionComponent
2) Update ngOnInit(): instead of subscribing to notes:

```
this.getNotes().subscribe(notes=>this.notes=notes);
```

we just keep the Observable link:

```
this.notes$ = this.getNotes();
```

3) Update the template in ViewSectionComponent:

```
<li *ngFor="let note of notes$ | async">
```

Now we resolve observable not in the component code, but in the view.

8.6 Prevent changing section if the note text was entered [optional]

We want to ask user if he really wants to go to another section in case if he has entered note text. To implement this, we will be using the mechanism CanDeactivate, which will check the possibility to change the route. To do this:

1) Create service CanDeactivateNote with this code:

```
@Injectable()
export class CanDeactivateNote implements
CanDeactivate<NotesEditorComponent> {

   canDeactivate(
      notesEditorComponent: NotesEditorComponent,
      route: ActivatedRouteSnapshot,
      state: RouterStateSnapshot
   ): Observable<boolean>|Promise<boolean>|boolean {
        return true;
   }
}
```

2) Now we should use CanDeactivate in our route change mechanism. To do that, change appRoutes configuration in app.module.ts:

```
{ path: '', component: NotesEditorComponent,
        canDeactivate: [CanDeactivateNote] }
```

And

```
{ path: ':name', component: NotesEditorComponent,
        canDeactivate: [CanDeactivateNote] },
```

Now you can try the mechanism by returning false in CanDeactivateNote.
It's still not working because we should rely solely on route change, otherwise we would not prevent to change the section.
3) We have to update setSection() in NotesEditorComponent:

```
setSection(section:string) {
  //  this.section = section;
  this.router.navigate([section]);
}
```

You see that now we do not change section, we only navigate to another route. Section will be changed after router mechanism will notice the URL change and retrieve section name from URL (see route.params subscribe in constructor).

4) Also we need to update showSection method in SectionsComponent: now it should not change activeSection, but only emit the event of changing section (which will fire setSection method in NotesEditorComponent):

```
showSection(section:Section) {
   //this.activeSection = section;
   this.sectionChanged.emit(section.title);
}
```

Now if you will return false from CanDeactivateNote and restart page, you will see that section change is not happening.

5) Now in CanDeactivateNote we want to check if note text is entered and prevent change of the section. In CanDeactivateNote we have access only to NotesEditorComponent, but note text is defined in NotesComponent. To access NotesComponent from NotesEditorComponent, add this field to NotesEditorComponent:

```
@ViewChild(NotesComponent) notesComponent:NotesComponent;
```

It will be give access from parent component (NotesEditorComponent) to child component (NotesComponent).

6) Now update canDeactivate method in CanDeactivateNote: we will get the note text and show confirmation if it was entered. If user will cancel the route change, section will not be changed.

```
const note = notesEditorComponent.notesComponent.text;
if (note && note.length>0) {
   return window.confirm(
     `You have entered the note.
      Do you really want to change section?`);
} else return true;
```

Now you can restart page and see it in action. If you will enter note text and try to change the section, you will see the confirmation. If you press cancel, change of the section will not occur.

Additional tasks

1) Move/copy note to another section.

2) Create web site on the base of notes. It should have the sections as menu items, and notes as the page contents. Show it on the separate route.

3) Implement subsections.