Task 10. Multiuser support

Task description

In this task you will submit the form and create users on server side.

Time

2 hours

Detailed description

10.1 Show sections and notes for the current user

1) Execute mongo
2) Run

```
use tutor
db.notes.update({},{$set:{userName:"demo"}}, {multi:true})
db.users.insert({userName:"demo",password:"demo",
sections:[{title:"Work"},{title:"Vacations"},{title:"Children"}]})
```

3) Check updates:

```
show collections
db.users.find()
db.notes.find()
```

4) In **server.js**

1. Add function **setUser** to add **userName** to query objects:

```
function setUserQuery(req) {
req.query.userName = req.session.userName || "demo";
}
```

2. Update all queries to include user:

```
app.get("/notes", function(req,res) {
setUserQuery(req);
db.notes.find(req.query)
                .toArray(function(err, items) {
                res.send(items);
        });
});
```

Also update post notes by adding this:

```
req.body.userName =  req.session.userName || "demo";
```

3. Define sections as the array in users table.
   New structure of user in users collection will be like this:

```
{ userName: "John",
  password:"jpass",
  sections: [{title:"Work"}, {title:"Vacation"}, {title:"Hobby"}]
}
```

For this:
4.1 Update app.get("/sections"):

```
app.get("/sections", function(req,res) {
var userName = req.session.userName || "demo";
db.users.find({userName:userName})
```

```
        .toArray(function(err, items) {
        var user = items[0];
        res.send(user.sections||[]);
});
});
```

4.2 Rewrite app.post("/sections/replace"):

```
app.post("/sections/replace", function(req,res) {
var userName = req.session.userName || "demo";
db.users.update({userName:userName},
                {$set:{sections:req.body}},
function() {
        res.end();
});
});
```

10.2 Implement login form

Now lets implement the login functionality.

1) Create **LoginService** in **app/service**

```
@Injectable()
export class LoginService {
        private loginUrl = 'login';  // URL to web api
        private logoutUrl = 'logout';  // URL to web api
        loggedIn: boolean = false;
        constructor(private http: Http) { }
}
```

2) Define class **LoginUser** in **LoginService**:

```
export class LoginUser {
   username: string;
   password: string;
}
```

3) Add **login()** and **logout()** methods to **LoginService**:

```
login(user: LoginUser): Observable<boolean> {
   return this.http.post(this.loginUrl, user)
      .map(response => response.json() as boolean)
      .do(res => { if (res) this.userLogin(user) });
}

logout() {
   return this.http.get(this.logoutUrl)
      .do(res => this.userLogout());
}
```

Also define userLogin() and userLogout() methods:

```
userLogin(user: LoginUser) {
   this.loggedIn = true;
}

userLogout() {
   this.loggedIn = false;
}
```

4) Create login form component app/loginForm.component.ts

```
@Component({
   selector: 'login-form',
   templateUrl: 'app/loginForm.component.html'
})
export class LoginFormComponent {
   userForm: LoginUser = new LoginUser();
   failedLogin: boolean;

   constructor(private loginService: LoginService, private router: Router)
{}

   login() {
      this.loginService.login(this.userForm)
         .subscribe(res=>res?this.onSuccessLogin():this.onFailLogin());
   }

   logout() {
      this.loginService.logout().subscribe(res=>this.onLogout());
   }


   onSuccessLogin() {
      this.router.navigateByUrl("/");
   }

   onFailLogin() {
      this.failedLogin = true;
   }

   onLogout() {
      this.router.navigateByUrl("/");
   }
}
```

Note that we are redirecting to start page on login and logout.

5) Define template for login form component in app/loginForm.component.html:

```
<form *ngIf="!loggedIn" style="margin-bottom: auto">
   <div class="form-group">
      <input type="text" placeholder="Username" name="username"
          class="form-control" [(ngModel)]="userForm.username"
          (keyup.enter)="login()">
   </div>
   <div class="form-group">
      <input type="password" placeholder="Password"
name="password" class="form-control"
[(ngModel)]="userForm.password">
   </div>
   <button type="submit" class="btn btn-success" (click)="login()">
       Sign in
   </button>

   <ng-content></ng-content>
   <div *ngIf="failedLogin" style="color:white">
      Wrong username or password
   </div>
</form>
<div *ngIf="loggedIn" style="color:white">
   User: <b>{{userForm?.username}}</b>
   <button type="submit" class="btn btn-success" (click)="logout()">
       Logout
   </button>
</div>
```

Here you see the login form with Sign in button. If user is already logged in, he will see user name and Logout button.
We will use **loggedIn** flag from **LoginService** to see if user logged in.
To retrieve it from component, add this method to **LoginFormComponent**:

```
get loggedIn() {
   return this.loginService.loggedIn;
}
```

Also we will show message in case of wrong login/password. We'll show it during 1 second, then it disappears. To implement this, modify **onFailLogin()** method by adding this line:

```
setTimeout(() => this.failedLogin = false, 1000);
```

6) Now add login component to **AppComponent** template (in place of Register button – now Register button will be inside the form):

```
<div class="navbar-form navbar-right">
   <login-form>
      <a class="btn btn-primary" routerLink="/register">Register</a>
   </login-form>
</div>
```

Also modify "Notes tutorial app" div to make it a link:

```
<div class="navbar-brand" routerLink="/" style="cursor:hand">
        Notes tutorial app
</div>
```

7) Implement login and logout on server side:

```
app.post("/login", function(req,res) {
   db.users.find(
      {userName:req.body.username,
         password:req.body.password})
      .toArray(function(err, items) {
         if (items.length>0) {
            req.session.userName = req.body.username;
         }
         res.send(items.length>0);
      });
});

app.get("/logout", function(req, res) {
   req.session.userName = null;
   res.end();
});
```

Now you can reload page and login.

10.3 Reload sections and notes on login

However, you will see that on login sections and notes are not updated. To reload it, we have to notify **SectionsComponent** to reread list of sections and update notes. To do it, let's add possibility to subscribe to Login event in **LoginService**:

1) Add properties

```
private userLoginSource = new Subject<LoginUser>();
userLogin$ = this.userLoginSource.asObservable();
```

Notice that you have to import Subject this way:

```
import { Subject }    from 'rxjs/Subject';
```

2) Also modify userLogin() and userLogout() methods to emit the event:

```
userLogin(user: LoginUser) {
   this.loggedIn = true;
   this.userLoginSource.next(user);
}

userLogout() {
   this.loggedIn = false;
   this.userLoginSource.next(null)
}
```

Now we can subscribe to the login event in **SectionsComponent**:
3) Add **LoginService** to constructor injection in **SectionsComponent**

4) Add subscription to constructor:

```
this.loginService.userLogin$.subscribe(user => this.readSections())
```

This will fire readSections() on login of the new user – and we'll see the updated list of sections and notes. You can try.

10.4 Login after registration

When we register a new user, we would like to login into system right after registering. To do this, modify **UserFormComponent**: add **LoginService** to constructor arguments to be injected. Also modify **onSubmit()** to do login after submitting the form:

```
onSubmit() {
   this.http.post("users", this.user).subscribe(res=>{
      this.loginService.login({username:this.user.userName,
         password: this.user.password})
         .subscribe(res=>{if (res) this.router.navigateByUrl("/")});
   });
}
```

Now after registration of the new user you will see user logged in.

Additional tasks

1) After page reload user needs to login again. How to fix it?

2) Implement edit of the user data