Task 9. Form with validation

Task description

In this task you will create a registration form and validation for the form.

Time

3 hours

Detailed description

9.1 Create the form

1) Add the button Register to top right of the page:
open app.component.html, and after application title

```
<div class="navbar-header">
   <div class="navbar-brand">Notes tutorial app</div>
</div>
```

add Register div and button:

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

2) Define new route in app.module.ts:

```
{ path: 'register', component: UserFormComponent },
```

It should be before ':name' path!

3) Create userForm.component.ts having UserFormComponent. Define templateUrl as

```
templateUrl: 'app/userForm.component.html'
```

Don’t forget to register component in declarations of AppModule.

4) Create model/User.ts having User domail model object:

```
export class User {
   name:string;
   password: string;
   password2: string;
   subscribe: boolean;
   email: string;
   dateOfBirth: Date;
}
```

5) Add field **user:User** to **UserFormComponent**, initialize it with
new User()

6) Create file userForm.component.html having this code:

```
<div class="col-md-8 col-md-offset-2">
   <div class="panel panel-primary">
   <div class="panel-heading">User registration form</div>
   <div class="panel-body">
```

```
        <form class="css-form" name="userForm"
            #userForm="ngForm" (ngSubmit)="onSubmit()" >

        </form>

    </div>
  </div>
</div>
```
Here we define the form, use **#userForm="ngForm"** to define template variable having access to form metadata. Also we define callback on form submission.

7) Add username field to the form:
```
<div class="form-group">
   <label for="userName">User name</label>
   <input type="text" class="form-control" id="userName"
        name="userName" placeholder="Username"
        #name="ngModel" [(ngModel)]="user.name">
</div>
```

Here you can see #name which defines template variable referencing ngModel object with metadata (we can access validation results, etc.). Also we define two-way data binding with [(ngModel)]="user.name".
Required is the attribute to validate if this field value is not empty.

8) The same way add fields for **password**, **password2**, **dateOfBirth**, **email**.

9) We will add checkbox "subscribe for newsletters". If this checkbox will be checked, we will show e-mail input box. To implement this, add checkbox this way:
```
<div class="form-group">
   <div class="checkbox">
      <label><input type="checkbox" name="subscribe"
                [(ngModel)]="user.subscribe">
         Subscribe for newsletters
      </label>
   </div>
</div>
```

10) In div containing e-mail we will check if subscribe checkbox is checked:
```
*ngIf="user.subscribe"
```

E-mail will be shown only in this case.

11) In the end of the form define Submit button this way:
```
<button type="submit" class="btn btn-primary"> Submit</button>
```

9.2 Add validation to the form

1) We want to allow to press Submit only if the whole form is valid. To check this, add this code to Submit button:
```
[disabled]="!userForm.valid"
```

It will check userForm validity and disable button if any field in the form is validated to invalid value.

2) We will show the results of validation using CSS. To do that, add this code to UserFormComponent @Component decorator:

```
styles: [`
   input.ng-touched.ng-invalid {
      background-color: #ffe8f1;
   }
`]
```

It will show input fields which are touched but invalid with reddish background.

3) Add **required** attribute to <input> of name, password, password2 and dateOfBirth. Now you can reload page and see the form validation by visiting required fields but not entering any data. It will get reddish.

4) We will also apply regular expression validation. For **dateOfBirth** it will be

```
pattern="[0-9][0-9]\.[0-1][0-9]\.[1-2][0-9][0-9][0-9]"
```

5) To show the hint if user is using the wrong date format, add this code after dateOfBirth <input> tag:

```
<span *ngIf="dateOfBirth.errors && dateOfBirth.errors.pattern &&
dateOfBirth.touched">
   Date of birth should be in format dd.mm.yyyy
</span>
```

It will check if there're some errors in date format, and show hint.

6) Do the same for the e-mail field.

9.3 Add check that password and repeat password are the same

We want to check if password and password2 match. To do this, we will need to create custom validator validateEqual. We will be using it this way:

```
<input type="password" name="password2" placeholder="Retype
password" [(ngModel)]="user.password2" validateEqual="password">
```

You see validateEqual attribute – that's our custom directive. It will take the name of second field as the parameter and will check if these fields are equal. Lets create this directive.

1) Create file directives/EqualToValidator.ts.

2) Define class EqualToValidator implements Validator with this decorator:

```
@Directive({
   selector: '[validateEqual][ngModel]',
   providers: [{provide:NG_VALIDATORS,
               useExisting: EqualToValidator, multi: true}]
        })
```

Also add EqualToValidator to declarations in AppModule.

3) Define constructor where we will take attribute value of "validateEqual" field:

```
constructor( @Attribute("validateEqual") public validateEqual: string) {}
```
This will be used to pass the name of the field which we will be comparing to.

4) Define validate method:
```
validate(c: AbstractControl): {[key: string]: any} {
   let v = c.value;
   let e = c.root.get(this.validateEqual);
   if (e && v !== e.value) return { validateEqual: false };
   return null;
}
```

Here we take the value of the field and find another field with which we will compare to.

5) Add the results of validation to the form: after password2 <input> add this:
```
<span *ngIf="password2.errors && !password2.errors.validateEqual &&
password2.touched">
   Passwords should match
</span>
```

It will show error if password2 was touched and not equal to password.

6) However, there's still one problem: in case if we change password after we have set password2, validation is not re-executed. This happens because validation is working only after changing of the validated field. To fix this, do the following:

In EqualToValidator we will subscribe to all changes in password field:

```
// subscribe to future changes in password
e.valueChanges.subscribe((val:string)=> {
     if (val != v) c.setErrors({validateEqual: false});
     else c.setErrors(null);
   }
);
```

Now if you will change password, validation will be working correctly.

9.4 Add check that userName is unique

That's all good, but some validations can't be done on the client. For example, we cannot check the uniqueness of user name on client side, because we need to do database query. For such cases we can use asynchronous validation. Lets define UserUniqueValidator which will check user.

1) Create file directives/UserUniqueValidator.ts

2) Define class
```
UserUniqueValidator implements Validator
```
With constructor
```
constructor( private http: Http) {}
```
We need http service to access server.

3) Define this @Directive decorator:

```
@Directive({
   selector:
'[userUniqueValid][formControlName],[userUniqueValid][ngModel]',
   providers: [{provide:NG_ASYNC_VALIDATORS,
               useExisting: UserUniqueValidator, multi: true}]
       })
```

4) Define validate method:

```
validate(c: AbstractControl): Promise<{[key: string]: any}> {
   const user = c.value;
   const params: URLSearchParams = new URLSearchParams();
   params.set('user', user);

   return new Promise(resolve =>
      this.http.get("checkUserUnique", {search:params})
         .map(response => response.json())
         .subscribe(res =>
            res?resolve(null):resolve({userUniqueValid:false})));
}
```

Here validator returns not the value, but the promise.

5) Add "/checkUserUnique" route to server.js:
Since we don't have users table and can't check userName for uniqness, let create some mock implementation for a while (we'll implement it later):

```
app.get("/checkUserUnique", function(req,res) {
res.send(req.query.user.length>2);
});
```

6) Add **userUniqueValid** attribute to username <input> in the form

7) Add this error message after username <input>:

```
<span *ngIf="name.errors && name.value?.length>0
   && !name.errors.userUniqueValid && name.dirty">
   User name is not unique. Please select another one.
</span>
```

9.5 Add user to mongodb

1) In server.js add new collection users:

```
        db.collection('users', function(error, users) {
                db.users = users;
        });
```

2) Add possibility to post user to users collection:

```
app.post("/users", function(req,res) {
db.users.insert(req.body, function(resp) {
        req.session.userName = req.body.userName;
        res.end();
});
});
```

3) In UserFormComponent add **ngSubmit()** method:

```
        onSubmit() {
           this.http.post("users", this.user).subscribe(res=>{
              this.router.navigateByUrl("");
           });
        }
```

4) Also modify constructor of UserFormComponent:

```
        constructor(private http:Http, private router: Router) {}
```

Now you can register the user. To check that user was added to database, execute mongo and look at list of the users:

```
                        db.users.find();
```

9.6 Implement check userName for uniqueness

Implement it yourself.

Additional tasks

1) Check that user's age is >12 (create special directive)

2) Implement drop-down to select country and city (list of countries and cities should be loaded from database), so that if user selects country, the list of cities would be loaded