Angular Identity Management

Ajout et édition d'un élément LdapUser

L'ajout et l'édition partagent le même html et la même implémentation à quelques détails près. Nous allons donc créer 2 composants : un pour l'édition et un pour l'ajout. Ces composant hériteront du composant LdapDetail que nous avons déjà fait.

Création des composants

Placez-vous dans le répertoire src/app et exécutez les commandes suivantes :

```
ng generate component ldap-edit --module=app
ng generate component ldap-add --module=app
```

Les modifications à apporter :

- Les composants LdapEdit et LdapAdd vont partager le même html donc **les 2 composants ont le même template et même CSS**
- Le constructeur de chaque composant doit reprendre le constructeur du composant LdapDetail
- LdapDetail va avoir une méthode abstraite "validateForm" qui sera appelée par LdapDetail et implémentée par les classes filles

Le code sera alors le suivant (LdapEditComponent):

```
@Component({
  selector: 'app-ldap-edit',
  templateUrl: '../ldap-detail/ldap-detail.component.html',
  styleUrls: ['../ldap-detail/ldap-detail.component.scss']
})
export class LdapEditComponent extends LdapDetailComponent implements OnInit {

  constructor(private usersService: UsersService,
              private route: ActivatedRoute,
              fb: FormBuilder,
              router: Router) {
    super( addForm: false, fb, router);
  }

  ngOnInit(): void {
    super.onInit();
  }

  validateForm(): void {
    console.log('LdapEditComponent - validateForm');
  }
}
```

Le template html et le css pour le composant LdapEditComponent ne sont plus nécessaire, il faut les supprimer.

Le code pour LdapAddComponent:

```
@Component({
  selector: 'app-ldap-edit',
  templateUrl: '../ldap-detail/ldap-detail.component.html',
  styleUrls: ['../ldap-detail/ldap-detail.component.scss']
})
export class LdapAddComponent extends LdapDetailComponent implements OnInit {
  constructor(private usersService: UsersService,
              fb: FormBuilder,
              router: Router,) {
    super( addForm: true, fb, router);
  }

  ngOnInit(): void {
    super.onInit();
  }

  validateForm(): void {
    console.log('LdapAddComponent - validateForm');
  }
}
```

Le template html et le css pour le composant LdapAddComponent ne sont plus nécessaire, il faut les supprimer.

Modification du module

Il faut modifier app-module.ts pour **supprimer le composant ldap-details**.

Modification du composant LdapDetails

Puis nous allons modifier la classe LdapDetails de la manière suivante :

```
/*
Il faut supprimer @Component
*/
export abstract class LdapDetailComponent {
  ...
  // Le PlaceHolder pour les mots de passes en fonction de l'édition ou non
  passwordPlaceHolder: string;
  ...

  protected constructor(
    public addForm: boolean,
    /* A VOIR protected route: ActivatedRoute,*/
    private fb: FormBuilder,
    private router: Router,
  ) {
    this.passwordPlaceHolder = 'Mot de passe' + (this.addForm ? '' : ' (vide si inchangé)');
  }

  protected onInit(): void {
    // Permet d'initialiser le formulaire au cas où
    // Nous n'en avons pas besoin ici
  }
  ...
  isFormValid(): boolean {
    return this.userForm.valid
        // Exemple de validation d'un champ :
        &&  (!this.addForm || this.formGetValue( name: 'passwordGroup.password') !== '');
  }
  ...

  /*
  Cette méthode est déplacée dans LdapEditComponent

  private getUser(): void {
   const login = this.route.snapshot.paramMap.get('id');

   this.usersService.getUser(login).subscribe(
     user => { this.user = user; console.log('LdapDetail getUser ='); console.log(user); }
   );
  }*/
```

```
onSubmitForm() : void  {
  this.validateForm();
}
1 usage
isFormValid(): boolean {
  return this.userForm.valid
    // Exemple de validation d'un champ :
    &&  (!this.addForm || this.formGetValue( name: 'passwordGroup.password') !== '');
}

abstract validateForm(): void

5+ usages
private formGetValue(name: string): string {
  const control : AbstractControl<any, any> | nu...  = this.userForm.get(name);
  if (control === null) {
    console.error("L'objet '" + name + "' du formlaire n'existe pas");
    return "";
  }
  return control.value;
}

no usages
private formSetValue(name: string, value: string | number): void {
  const control : AbstractControl<any, any> | nu...  = this.userForm.get(name);
  if (control === null) {
    console.error("L'objet '" + name + "' du formlaire n'existe pas");
    return;
  }
  control.setValue(value);
}
```

```
// Permet d'afficher les propriétés de UserLdap dans le formulaire
no usages
protected copyUserToFormControl(): void {
  if (this.user === undefined) {
    return;
  }

  this.formSetValue( name: 'login', this.user.login);
  this.formSetValue( name: 'nom', this.user.nom);
  this.formSetValue( name: 'prenom', this.user.prenom);
  this.formSetValue( name: 'mail', this.user.mail);
  /* Il faudra ajouter les champs suivant au formulaire
  this.formSetValue('employeNumero', this.user.employeNumero);
  this.formSetValue('employeNiveau', this.user.employeNiveau);
  this.formSetValue('dateEmbauche', this.user.dateEmbauche);
  this.formSetValue('publisherId', this.user.publisherId);
  this.formSetValue('active', this.user.active);
   */
}
// Permet de récupérer les valeurs du formulaire et
// de retourner un objet UserLdap avec ces valeurs
no usages
protected getUserFromFormControl(): UserLdap {
  return {
    login: this.formGetValue( name: 'login'),
    nom: this.formGetValue( name: 'nom'),
    prenom: this.formGetValue( name: 'prenom'),
    nomComplet: this.formGetValue( name: 'nom') + ' ' + this.formGetValue( name: 'prenom'),
    mail: this.formGetValue( name: 'mail'),
    // Les valeurs suivantes devraient être reprise du formulaire
    employeNumero: 1, // this.formGetValue('employeNumero'),
    employeNiveau: 1, // this.formGetValue('employeNiveau'),
    dateEmbauche: '2020-04-24', // this.formGetValue('dateEmbauche'),
    publisherId: 1, // this.formGetValue('publisherId'),
    active: true,
    motDePasse: '',
    role: 'ROLE_USER'
  };
}
```

La classe LdapDetails est maintenant abstraite car elle a la méthode validateForm abstraite afin que les classes parents la redéfinissent et puisse gérer la validation des données. De plus l'attribut addForm a été ajouté afin de savoir si le formulaire est pour l'ajout ou pour l'édition.
Il faut aussi modifier le début du HTML pour afficher le titre correct et le PlaceHolder du mot de passe :

```
<h3 class="mat-h3">
  <button mat-icon-button (click)="goToLdap()">
    <mat-icon aria-label="Accueil">arrow_back</mat-icon>
  </button>
  {{ addForm ? "Ajout d'un utilisateur" : "Edition d'un utilisateur" }}
</h3>

<table class="user-full-width" formGroupName="passwordGroup">
  <tr>
    <td>
      <mat-form-field class="user-full-width">
        <input type="password" matInput
               [placeholder]="passwordPlaceHolder"
               id='password' formControlName='password'
        >
```

Modification du composant LdapListComponent

Nous allons ajouter le bouton pour aller vers la page d'ajout d'un utilisateur (ldap-list.component.html):

```
<div class="mat-elevation-z8">
  <button mat-raised-button color="primary"
          (click)="addUser()">
    Ajouter un utilsateur
  </button> <br /><br />
```

Et implémenter la méthode "addUser" pour aller vers la page d'ajout d'un utilisateur (ldap-list.component.ts):

```
export class LdapListComponent implements OnInit {
  ...
  addUser() {
    this.router.navigate(['/user/add']).then( (e) => {
      if (! e) {
        console.log('Navigation has failed!');
      }
    });
  }
```

Modification des routes

Il faut modifier la route pour aller au composant LdapEditComponent et non LdapDetailComponent (app-routing.module.ts) et ajouter la route pour l'ajout d'un utilisateur:

```
const routes: Routes = [
  { path: 'users/list', component: LdapListComponent },
  { path: 'user/add', component: LdapAddComponent},
  { path: 'user/:id', component: LdapEditComponent},
  { path: '**', component: PageNotFoundComponent }
];
```

Modification du service et des composants

Le service UsersService

Nous allons ajouter les méthodes d'ajout de de modification d'un utilisateur mais elles sont toujours factices :

```
@Injectable({
  providedIn: 'root'
})
export class UsersService {
  • • •
  1 usage
  addUser(user: UserLdap): Observable<UserLdap> {
    // Ajout dans la liste
    this.users.push(user);
    return of(user);
  }

  1 usage
  updateUser(userToUpdate: UserLdap): Observable<UserLdap> {
    // Modification de l'utilisateur
    const user : UserLdap | undefined  = this.users.find( u : UserLdap  => u.login === userToUpdate.login);
    if (user) {
      // Modif
      user.nom = userToUpdate.nom;
      user.prenom = userToUpdate.prenom;
      user.nomComplet = user.nom + ' ' + user.prenom;
      user.motDePasse = userToUpdate.motDePasse;

      return of(userToUpdate);
    }
    return throwError( errorFactory: () => new Error( message: 'Utilisateur non trouvé'));
  }
}
```

Les composants LdapDetailComponent, LdapAddComponent et LdapEditComponent

Pour LdapEditComponent : nous allons récupérer les infos puis valider les infos.
Pour LdapAddComponent: nous allons valider les infos.
Dans les 2 cas nous allons ajouter un message d'erreur si il y a un problème et afficher une notification (snackbar)

Modification des composants (suite)

LdapDetailComponent

Nous allons ajouter une balise pour afficher un message d'erreur dans le HTML en utilisant un nouveau composant "AlertComponent" que nous allons créer après :

```
<h3 class="mat-h3">
  <button mat-icon-button (click)="goToLdap()">
    <mat-icon aria-label="Accueil">arrow_back</mat-icon>
  </button>
  {{ addForm ? "Ajout d'un utilisateur" : "Edition d'un utilisateur" }}
</h3>

<section class="loading" *ngIf="processLoadRunning==true">
  <mat-spinner diameter="25" ></mat-spinner>
  <span>  En cours de chargement ...</span>
</section>

<app-alert *ngIf="errorMessage.length>0" [type]="'danger'">{{ errorMessage }}</app-alert>
```

Ajout de l'attribut "errorMessage":

```
export abstract class LdapDetailComponent {
  user: UserLdap;
  processLoadRunning = false;
  processValidateRunning = false;
  // Le PlaceHolder pour les mots de passes en fonction de l'édition ou non
  passwordPlaceHolder: string;
  // Message d'erreur
  errorMessage = '';
  ...
```

Le composant Alert

Ce composant se contente d'afficher un message dans le style de Bootstrap - Alert.

Création du composant :
Créez et placez vous dans le répertoire src/app/share puis créez le composant "alert" (vous devez maintenant connaître toutes les commandes nécessaires :)).

Le HTML :

```
<div class="{{cssClass}}">
    <ng-content></ng-content>
    <a class="close" (click)="removeAlert()">X</a>
</div>
```

La directive ng-content permet d'insérer le HTML contenu dans la balise alert.

Le CSS:

```
.alert {
 position: relative;
 padding: .75rem 1.25rem;
 margin-bottom: 1rem;
 border: 1px solid transparent;
 border-radius: .25rem;
}

.alert-hide {
 display: none;
}

.alert-dismissible .close {
 position: absolute;
 top: 0;
 right: 0;
 /* padding-top: .75rem;*/
 padding: .75rem 1.25rem;
 color: inherit;
}

.alert-warning {
 color: #856404;
 background-color: #fff3cd;
 border-color: #ffeeba;
}

.alert-info {
 color: #0c5460;
 background-color: #d1ecf1;
 border-color: #bee5eb;
}

.alert-danger {
 color: #721c24;
 background-color: #f8d7da;
 border-color: #f5c6cb;
}

.alert-success {
 color: #155724;
 background-color: #d4edda;
 border-color: #c3e6cb;
}
```

Le code :

```
@Component({
  selector: 'app-alert',
  templateUrl: './alert.component.html',
  styleUrls: ['./alert.component.css']
})
export class AlertComponent implements OnInit {
  @Input() type: string;
  cssClass: string[] = ['alert', 'alert-dismissible', 'fade'];

  no usages
  constructor() {
    this.type = 'info';
  }

  no usages
  ngOnInit(): void {
    let alert :string  = 'alert-info';
    switch (this.type) {
      case 'succes': alert = 'alert-success'; break;
      case 'danger': alert = 'alert-danger'; break;
      case 'warning': alert = 'alert-warning'; break;
    }
    this.cssClass.push(alert);
  }

  1 usage
  removeAlert() :void  {
    this.cssClass = ['alert-hide'];
  }
}
```

Le composant LdapEditComponent

Le composant va obtenir les données avec la méthode getUser et valider les données avec la méthode validateForm. Ces 2 méthodes utilise le service UsersService et le composant Material SnackBar pour afficher un message.

Il faut au préalable ajouter le module MatSnackBarModule au module Material et définir les valeurs par défaut (pour toute l'application) du composant SnackBar :

```
@NgModule({
 exports: [
     ...
 ],
 providers: [
   {provide: MAT_SNACK_BAR_DEFAULT_OPTIONS, useValue: {duration: 2500,
verticalPosition: 'top'}}
```

```
  ]
})
export class AppMaterialModule { }
```

Par défaut, la SnackBar s'affichera pendant 2,5 secondes pour toute l'application.

Le code :

```
export class LdapEditComponent extends LdapDetailsComponent implements OnInit {

  no usages
  constructor(
    private usersService: UsersService,
    private route: ActivatedRoute,
    fb: FormBuilder,
    router: Router,
    private snackBar: MatSnackBar
  ) {
    super( addForm: false, fb, router);
  }

  no usages
  ngOnInit(): void {
    super.onInit();
    this.getUser();
  }

  no usages
  validateForm() : void {
    console.log('LdapEditComponent - validateForm');
    this.processValidateRunning = true;
    this.usersService.updateUser(this.getUserFromFormControl()).subscribe( observerOrNext: {
      next: (value : UserLdap ) : void  => {
        this.processValidateRunning = false;
        this.errorMessage = '';
        this.snackBar.open( message: 'Utilisateur modifié !', action: 'X');
      },
      error: (err) : void  => {
        this.processValidateRunning = false;
        this.errorMessage = 'Une erreur est survenue dans la modification !';
        console.error('Modification utilisateur', err);
        this.snackBar.open( message: 'Utilisateur non modifié !', action: 'X');
      }
    });
  }
```

```
1 usage
private getUser() : void {
  const login : string | null  = this.route.snapshot.paramMap.get('id');

  if (login === null) {
    console.error("Can't retreive user id from URL");
    return;
  }

  this.usersService.getUser(login).subscribe( observerOrNext: {
    next: (user : UserLdap | undefined ) : void  => {
      this.user = user;
      this.copyUserToFormControl();
      console.log('LdapDetails getUser =', user);
    },
    error: (err) : void  => {
      this.processValidateRunning = false;
      this.errorMessage = "L'utilsateur n'existe pas !";
      console.error('Obtention utilisateur', err);
      this.snackBar.open( message: 'Utilisateur non trouvé !',  action: 'X');
    }
  });
 }
}
```

Le composant LdapAddComponent

Le code :

```
export class LdapAddComponent extends LdapDetailsComponent implements OnInit {

  no usages
  constructor(
    private usersService: UsersService,
    fb: FormBuilder,
    router: Router,
    private snackBar: MatSnackBar
  ) {
    super( addForm: true, fb, router);
  }

  no usages
  ngOnInit(): void {
    super.onInit();
  }
```

```
  validateForm() : void {
    console.log('LdapAddComponent - validateForm');

    this.processValidateRunning = true;
    this.usersService.addUser(this.getUserFromFormControl()).subscribe( observerOrNext: {
      next: (value : UserLdap ) : void  => {
        this.processValidateRunning = false;
        this.errorMessage = '';
        this.snackBar.open( message: 'Utilisateur ajouté !',  action: 'X');
      },
      error: (err) : void  => {
        this.processValidateRunning = false;
        console.error('Ajout utilisateur', err);
        this.errorMessage = 'L\'utilisateur n\'a pas pu être ajouté !';
        this.snackBar.open( message: 'Erreur dans l\'ajout de l\'utilisateur!',  action: 'X');
      }
    });
  }
}
```

Malheureusement, la modification et l'ajout ne sont pas reflétés dans la liste des utilisateurs. Nous allons changer cela dans le chapitre "Le service Users".

Validation du formulaire

Nous avons écrit du code pour valider les mots de passe mais la seule validation effectuée est que le mot de passe n'est pas vide. Or il faut vérifier que les 2 mots de passe concordent. Nous allons donc écrire une validation personnalisée, la validation intégré d'Angular ne suffit pas.

Validation du formulaire entier (rappel)

```
isFormValid(): boolean {
  return this.userForm.valid
      // Exemple de validation d'un champ :
      && (!this.addForm || this.formGetValue( name: 'passwordGroup.password') !== '');
}
```

Créer son validateur personnalisé

Le formulaire contient 2 champs pour le mot de passe et la confirmation du mot de passe. Nous voulons que les 2 mots de passe soient identiques.
Nous pouvons associer une validation à un groupe de données. C'est pour cette raison que les 2 mots de passe sont dans un groupe de données.

Création de la validation personnalisée :

- Dans le répertoire ldap-detail, ajoutez un fichier TypeScript nommé passwords-validator.directive.ts (c'est une directive)
- Ajoutez le code suivant :

```
2 usages
export const passwordMatchingValidator: ValidatorFn = (control: AbstractControl): ValidationErrors | null => {
  const password : AbstractControl<any, any> | nu...  = control.get('password');
  const confirmPassword : AbstractControl<any, any> | nu...  = control.get('confirmPassword');

  return password && confirmPassword && password.value === confirmPassword.value ?
    null :  { passwordMatching: true };
};
```

- Ajoutez la validation au groupe de données (ldap-detail.component.ts):

```
  // Groupe de données imbriqué
  passwordGroup: this.fb.group( controls: {
    password: [''],
    confirmPassword: ['']
  }, options: { validators: passwordMatchingValidator }),
  mail: {value: '', disabled: true},
});
```

- Modifiez le constructeur pour ajouter la validation des mots de passe si ajout d'un utilisateur

```
3 usages
get passwordForm() { return this.userForm.get('passwordGroup'); }

2 usages
protected constructor(
  public addForm: boolean,
  private fb: FormBuilder,
  private router: Router,
) {
  this.passwordPlaceHolder = 'Mot de passe' + (this.addForm ? '' : ' (vide si inchangé)');
  if (this.addForm) {
    this.passwordForm?.get('password')?.addValidators(Validators.required);
    this.passwordForm?.get('confirmPassword')?.addValidators(Validators.required);
  }
}
```

De plus nous voulons afficher l'erreur pour les 2 champs, nous allons devoir utiliser un ErrorStateMatcher personnalisé. Dans le fichier "passwords-validator.directive.ts", ajoutez le code suivant (à la fin) :

```
/**
 * Custom ErrorStateMatcher which returns true (error exists) when the parent form group
 * is invalid and the control has been touched
 */

2 usages
export class ConfirmValidParentMatcher implements ErrorStateMatcher {
  no usages
  isErrorState(control: FormControl | null, form: FormGroupDirective | NgForm | null): boolean {
    const isSubmitted : boolean | null  = form && form.submitted;
    return !!(control && control.parent
        && control.parent.invalid && (control.touched || isSubmitted));
  }
}
```

Ajout des messages d'erreur dans le HTML :
Pour les champs Nom, Prénom et Login, il faut ajouter les balises d'erreur dans les champs, exemple :

```
<mat-form-field class="user-full-width">
  <input matInput class="form-control" placeholder="Nom de l'utilisateur"
         id="nom" formControlName="nom" required
         (input)="updateLogin()"
  />
  <mat-error>Le nom est requis</mat-error>
</mat-form-field>
```

Pour les mots de passe, il faut ajouter, en plus de la balise d'erreur, la gestion de l'objet "errorStateMatcher" :

```
<table class="user-full-width" formGroupName="passwordGroup">
  <tr>
    <td>
      <mat-form-field class="user-full-width">
        <input type="password" matInput
               [placeholder]="passwordPlaceHolder"
               id='password' formControlName='password'
               [errorStateMatcher]="confirmValidParentMatcher"
        >
        <mat-error >{{getErrorMessage()}}</mat-error>
      </mat-form-field>
    </td>
    <td>
      <mat-form-field class="user-full-width">
        <input type="password" matInput
               placeholder="Vérification du mot de passe"
               id="confirmPassword" formControlName="confirmPassword"
               [errorStateMatcher]="confirmValidParentMatcher"
        />
        <mat-error >{{getErrorMessage()}}</mat-error>
      </mat-form-field>
    </td>
  </tr>
</table>
```

Et aussi ajouter et instancier l'attribut "confirmValidParentMatcher" dans le composant :

```
export abstract class LdapDetailsComponent {

  ....
  confirmValidParentMatcher : ConfirmValidParentMatcher  = new ConfirmValidParentMatcher();


  ....
  2 usages
  getErrorMessage() : string {
    if (this.passwordForm?.errors) {
      return 'Les mots de passe ne correspondent pas';
    }
    return 'Entrez un mot de passe';
  }

  ....
```

Liens

https://angular.io/start

https://blog.angular.io/

https://blog.angular-university.io/

https://guide-angular.wishtack.io/

https://openclassrooms.com/fr/courses/4668271-developpez-des-applications-web-avec-angular

https://www.typescriptlang.org/docs/home.html

https://blog.soat.fr/

Parent Child Two way binding
https://medium.com/@preethi.s/angular-custom-two-way-data-binding-3e618309d6c7

Organisation par module
Mise en place de la sécurité
https://angular.io/guide/route

Login:
https://loiane.com/2017/08/angular-hide-navbar-login-page/

Dialog:
https://blog.angular-university.io/angular-material-dialog/

Angular Material
https://material.angular.io/

https://medium.com/@ismapro/first-steps-with-angular-7-with-angular-cli-and-angular-material-d69f55d8ac51

https://www.positronx.io/create-angular-material-8-custom-theme/

https://akveo.github.io/ngx-admin/

https://auth0.com/blog/creating-beautiful-apps-with-angular-material/