export class RegisterComponent implements OnInit, OnDestroy {

titles$: Observable<Title[]>;

loading$: Observable<boolean>;

private subscription = new Subscription();

anonymousConsent$: Observable<{

consent: AnonymousConsent;

template: string;

}>;

userRegistrationForm: FormGroup = this.fb.group(

{

titleCode: [''],

name: ['', Validators.required],

lastName: ['', Validators.required],

email: ['', [Validators.required, CustomFormValidators.emailValidator]],

password: [

'',

[Validators.required, CustomFormValidators.passwordValidator],

],

passwordconf: ['', Validators.required],

newsletter: new FormControl({

value: false,

disabled: this.isConsentRequired(),

}),

termsandconditions: [false, Validators.requiredTrue],

},

{ validator: CustomFormValidators.matchPassword }

);

constructor(

protected userService: UserService,

protected globalMessageService: GlobalMessageService,

protected fb: FormBuilder,

protected router: RoutingService,

protected anonymousConsentsService: AnonymousConsentsService,

protected anonymousConsentsConfig: AnonymousConsentsConfig

) {}

ngOnInit() {

this.titles$ = this.userService.getTitles().pipe(

tap(titles => {

if (Object.keys(titles).length === 0) {

this.userService.loadTitles();

}

}),

map(titles => {

return titles.sort(sortTitles);

})

);

this.loading$ = this.userService.getRegisterUserResultLoading();

this.registerUserProcessInit();

// TODO: Workaround: allow server for decide is titleCode mandatory (if yes, provide personalized message)

this.subscription.add(

this.globalMessageService

.get()

.pipe(filter(messages => !!Object.keys(messages).length))

.subscribe((globalMessageEntities: GlobalMessageEntities) => {

const messages =

globalMessageEntities &&

globalMessageEntities[GlobalMessageType.MSG\_TYPE\_ERROR];

if (

messages &&

messages.some(message => message === 'This field is required.')

) {

this.globalMessageService.remove(GlobalMessageType.MSG\_TYPE\_ERROR);

this.globalMessageService.add(

{ key: 'register.titleRequired' },

GlobalMessageType.MSG\_TYPE\_ERROR

);

}

})

);

const { registerConsent } = this.anonymousConsentsConfig.anonymousConsents;

this.anonymousConsent$ = combineLatest([

this.anonymousConsentsService.getConsent(registerConsent),

this.anonymousConsentsService.getTemplate(registerConsent),

]).pipe(

map(([consent, template]: [AnonymousConsent, ConsentTemplate]) => {

return {

consent,

template: template ? template.description : '',

};

})

);

this.subscription.add(

this.userRegistrationForm.get('newsletter').valueChanges.subscribe(() => {

this.toggleAnonymousConsent();

})

);

}

submit(): void {

this.userService.register(

this.collectDataFromRegisterForm(this.userRegistrationForm.value)

);

}

titleSelected(title: Title): void {

this.userRegistrationForm['controls'].titleCode.setValue(title.code);

}

collectDataFromRegisterForm(formData: any): UserSignUp {

const { firstName, lastName, email, password, titleCode } = formData;

return {

firstName,

lastName,

uid: email.toLowerCase(),

password,

titleCode,

};

}

isConsentGiven(consent: AnonymousConsent): boolean {

return this.anonymousConsentsService.isConsentGiven(consent);

}

private isConsentRequired(): boolean {

const {

requiredConsents,

registerConsent,

} = this.anonymousConsentsConfig.anonymousConsents;

if (requiredConsents && registerConsent) {

return requiredConsents.includes(registerConsent);

}

return false;

}

private onRegisterUserSuccess(success: boolean): void {

if (success) {

this.router.go('login');

this.globalMessageService.add(

{ key: 'register.postRegisterMessage' },

GlobalMessageType.MSG\_TYPE\_CONFIRMATION

);

}

}

toggleAnonymousConsent(): void {

const { registerConsent } = this.anonymousConsentsConfig.anonymousConsents;

if (Boolean(this.userRegistrationForm.get('newsletter').value)) {

this.anonymousConsentsService.giveConsent(registerConsent);

} else {

this.anonymousConsentsService.withdrawConsent(registerConsent);

}

}

private registerUserProcessInit(): void {

this.userService.resetRegisterUserProcessState(); // 重置注册用户进程标志

this.subscription.add(

this.userService.getRegisterUserResultSuccess().subscribe(success => {

this.onRegisterUserSuccess(success);

})

);

}

ngOnDestroy() {

this.subscription.unsubscribe();

this.userService.resetRegisterUserProcessState();

}

}