Atelier Spring Boot/Angular :

Inscription des Utilisateurs

L'objectif général de cet atelier est de développer un module pour l'inscription des nouveaux utilisateurs. L'email de l'utilisateur doit être validé, en envoyant un code secret par email. L'utilisateur doit taper ce code pour valider son email et pouvoir se connecter. Le rôle par défaut attribué au nouvel utilisateur créé est USER.

Objectifs :

1. Créer un formulaire Angular pour l'inscription,
2. Développer l'api /register coté Spring Boot et l'appeler
3. Générer un code secret pour valider l'email
4. Envoyer par email le code généré
5. Créer l'api verifEmail pour valider l'email de l'utilisateur
6. Consommer l'api verifEmail à partir de Angular
7. Modifier le login pour vérifier si l'utilisateur est activé
8. Quelques améliorations

Créer un formulaire Angular pour l'inscription

Créer un component register

ng g c register

Ajouter le path au fichier app.routes.ts

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

Ajouter ce code à la page de login

```
<p>Vous n'avez pas un compte? <a routerLink="/register" mb-4 >S'inscrire </a></p>
```

Ajouter ces attributs au model User

```
email!: string ;
enabled!: boolean
```

register.component.ts

```
  imports: [FormsModule, ReactiveFormsModule, RouterLink],
```

Modifier le fichier **register.component.ts** comme suit :

```
import { AbstractControl, FormBuilder, FormGroup, Validators } from '@angular/forms';
```

```
export class RegisterComponent implements OnInit {

 public user = new User();
 confirmPassword?:string;
 myForm!: FormGroup;

 constructor(private formBuilder: FormBuilder) { }

 ngOnInit(): void {
  this.myForm = this.formBuilder.group({

    username : ['', [Validators.required]],
    email : ['', [Validators.required, Validators.email]],
    password : ['', [Validators.required, Validators.minLength(6)]],
    confirmPassword : ['', [Validators.required]]
 } );

 }

onRegister()
{
  console.log(this.user);
}
```

Modifier le fichier **register.component.html** comme suit :

```
<div class="container mt-5">
  <div class="row justify-content-md-center">
    <div class="col-md-4">

      <form [formGroup]="myForm">
        <div class="form-group">
          <label>Nom Utilisateur :</label>
       <input type="text" name="username" class="form-control"  formControlName="username"
[(ngModel)]="user.username">
        </div>
        @if (myForm.controls['username'].hasError('required')&&
myForm.controls['username'].touched) {
          <span class="text-danger">
            Nom Utilisateur est obligatoire
          </span>
        }

        <div class="form-group">
          <label>email :</label>
          <input type="text" name="email" class="form-control" formControlName="email"
[(ngModel)]="user.email">
        </div>
        @if (myForm.controls['email'].hasError('required')&&
myForm.controls['email'].touched) {
          <span class="text-danger">
            email est obligatoire
          </span>
        }
        @if (myForm.controls['email'].hasError('email')&& myForm.controls['email'].touched)
{
          <span class="text-danger">
            email non valide
          </span>
        }

        <div class="form-group">
          <label>Mot de passe :</label>
          <input type="password" name="password" formControlName="password"
[(ngModel)]="user.password" class="form-control">
        </div>
        @if (myForm.controls['password'].hasError('required')&&
myForm.controls['password'].touched) {
          <span class="text-danger">
            mot de passe est obligatoire
          </span>
```

```
        }
        @if (myForm.controls['password'].hasError('minlength')&&
myForm.controls['password'].touched) {
          <span class="text-danger">
            mot de passe doit contenir au moins 6 caractères
          </span>
        }

        <div class="form-group">
          <label>Confirmer Mot de passe :</label>
          <input type="password" name="confirmPassword"
            formControlName="confirmPassword" class="form-control">
          </div>
          @if (myForm.controls['confirmPassword'].hasError('required')&&
myForm.controls['confirmPassword'].touched) {
            <span class="text-danger">
              confirmer mot de passe est obligatoire
            </span>
          }

          <div>
            @if (myForm.get('confirmPassword')?.touched &&
myForm.get('confirmPassword')?.value != myForm.get('password')?.value) {
              <span  class="text-danger">
                Le mot de passe confrmé ne correspond pas au mot de passe tapé
              </span>
            }
          </div>

          <button    type="button" [disabled]="myForm.invalid"  class="btn btn-success"
          (click)="onRegister()" >S'insrire</button>

          <p>vous avez un compte? <a routerLink="/login">se connecter </a></p>
        </form>
      </div>
    </div>
  </div>
```

Développer l'api /register coté Spring Boot et l'appeler

1. Ajouter l'attribut email à l'entité User

```
private String email;
```

2. Ajouter au service *UserService* la méthode *registerUser* :

UserRepository :

```
Optional<User> findByEmail(String email);
```

```
        @Override
        public User registerUser(RegistationRequest request) {
        Optional<User> optionaluser = userRep.findByEmail(request.getEmail());
              if(optionaluser.isPresent())
                    throw new EmailAlreadyExistsException("email déjà existant!");

               User newUser = new User();
               newUser.setUsername(request.getUsername());
               newUser.setEmail(request.getEmail());

newUser.setPassword(bCryptPasswordEncoder.encode(request.getPassword()));
               newUser.setEnabled(false);
               userRep.save(newUser);

               //ajouter à newUser le role par défaut USER
               Role r = roleRep.findByRole("USER");
               List<Role> roles = new ArrayList<>();
               roles.add(r);
               newUser.setRoles(roles);

               return userRep.save(newUser);
        }
```

3. Créer, dans le package exception, la classe *ErrorDetails*

```
@Data
@AllArgsConstructor
@NoArgsConstructor
public class ErrorDetails {
    private LocalDateTime timestamp;
    private String message;
    private String path;
    private String errorCode;
}
```

4. Créer, dans le package exception, la classe *EmailAlreadyExistsException*

```
@ResponseStatus(value = HttpStatus.BAD_REQUEST)
public class EmailAlreadyExistsException extends RuntimeException{
    private String message;

    public EmailAlreadyExistsException(String message){
        super(message);
    }
}
```

5. Créer, dans le package exception, la classe GlobalExceptionHandler

```
@ControllerAdvice
public class GlobalExceptionHandler {


    @ExceptionHandler(EmailAlreadyExistsException.class)
    public ResponseEntity<ErrorDetails>
handleEmailAlreadyExistsException(EmailAlreadyExistsException exception,
                                                    WebRequest webRequest){

        ErrorDetails errorDetails = new ErrorDetails(
                LocalDateTime.now(),
                exception.getMessage(),
```

```
                webRequest.getDescription(false),
                "USER_EMAIL_ALREADY_EXISTS"
        );

        return new ResponseEntity<>(errorDetails, HttpStatus.BAD_REQUEST);
    }

    @ExceptionHandler(Exception.class)
    public ResponseEntity<ErrorDetails> handleGlobalException(Exception exception,

WebRequest webRequest){

        ErrorDetails errorDetails = new ErrorDetails(
                LocalDateTime.now(),
                exception.getMessage(),
                webRequest.getDescription(false),
                "INTERNAL SERVER ERROR"
        );

      return new ResponseEntity<>(errorDetails, HttpStatus.INTERNAL_SERVER_ERROR);
    }
}
```

6. Créer la classe *RegistationRequest* dans le package register :

```
package com.nadhem.users.register;

import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;

@Data @NoArgsConstructor @AllArgsConstructor
public class RegistationRequest {
      private String username;
      private String password;
      private String email;
}
```

7. Ajouter à la classe *UserRestController* la méthode :

```
    @PostMapping("/register")
      public User register(@RequestBody  RegistationRequest request)
      {
            return userService.registerUser(request);
      }
```

8. Modifier le classe *SecurityConfig*

```
.requestMatchers("/login","/register/**").permitAll()
```

9. Tester l'api register avec POSTMAN
10. Appeler l'api /register à partir de Angular

Auth.service.ts

```
 registerUser(user :User){
    return this.http.post<User>(this.apiURL+'/register', user,
{observe:'response'});
  }
```

register.component.ts

```
onRegister()
{

 this.authSerice.registerUser(this.user).subscribe({
   next:(res)=>{
     alert("veillez confirmer votre email");
    // this.router.navigate(["/verifEmail",this.user.email]);

   },
   error:(err:any)=>{
     if(err.status=400){
       this.err= err.error.message;

     }
   }
 }
 )
}
```

Register.component.html

```
 <form [formGroup]="myForm">
              @if (err) {
                <div class="alert alert-danger">
                    <strong>{{err}}</strong>
                </div>
               }
```

11. Modifier l'interceptor comme suit, pour exclure /register

```
const exclude_array : string[] = ['/login','/register','/verifyEmail'];

function toExclude(url :string)
 {
  var length = exclude_array.length;
  for(var i = 0; i < length; i++) {
      if( url.search(exclude_array[i]) != -1  )
          return true;
  }
  return false;
```

```
 }

export const tokenInterceptor : HttpInterceptorFn = (req, next) => {

  const authService = inject(AuthService);

   if (!toExclude(req.url))
      {
      let jwt = authService.getToken();
      let reqWithToken = req.clone( {
   setHeaders: { Authorization : "Bearer "+jwt}
   })
   return next(reqWithToken);
   }
   return next(req);
}
```

Générer un code secret

12. Créer l'entité *VerificationToken*

```
@Data
@Entity
@NoArgsConstructor
public class VerificationToken {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    private String token;
    private Date expirationTime;
    private static final int EXPIRATION_TIME = 15;

    @OneToOne
    @JoinColumn(name = "user_id")
    private User user;

    public VerificationToken(String token, User user) {
        super();
        this.token = token;
        this.user = user;
        this.expirationTime = this.getTokenExpirationTime();
    }

    public VerificationToken(String token) {
        super();
        this.token = token;
        this.expirationTime = this.getTokenExpirationTime();
    }

    public Date getTokenExpirationTime() {
        Calendar calendar = Calendar.getInstance();
        calendar.setTimeInMillis(new Date().getTime());
        calendar.add(Calendar.MINUTE, EXPIRATION_TIME);
        return new Date(calendar.getTime().getTime());
    }
```

```
}

public interface VerificationTokenRepository extends
JpaRepository<VerificationToken, Long> {
    VerificationToken findByToken(String token);
}
```

13. Ajouter à *UserServiceImpl* les méthodes qui enregistre l'utilisateur dans la BD et génère un token :

```
@Override
  public User registerUser(RegistationRequest request) {
         Optional<User> optionaluser = userRep.findByEmail(request.getEmail());
         if(optionaluser.isPresent())
                throw new EmailAlreadyExistsException("email déjà existant!");

            User newUser = new User();
         newUser.setUsername(request.getUsername());
         newUser.setEmail(request.getEmail());
         newUser.setPassword(bCryptPasswordEncoder.encode(request.getPassword()));
         newUser.setEnabled(false);
         userRep.save(newUser);

         //ajouter à newUser le role par défaut USER
         Role r = roleRep.findByRole("USER");
         List<Role> roles = new ArrayList<>();
         roles.add(r);
         newUser.setRoles(roles);

         userRep.save(newUser);

         //génére le code secret
         String code = this.generateCode();

         VerificationToken token = new VerificationToken(code, newUser);
         verificationTokenRepo.save(token);


         return newUser;
  }



  public String  generateCode() {
      Random random = new Random();
      Integer code = 100000 + random.nextInt(900000);

      return code.toString();
  }
```

Envoyer par email le code généré

Ajouter la dépendance suivante :

```
<dependency>
      <groupId>org.springframework.boot</groupId>
```

```
        <artifactId>spring-boot-starter-mail</artifactId>
</dependency>
```

Créer l'interface EmailSender dans le package util

```
public interface EmailSender {
    void sendEmail(String toEmail, String body);
}
```

Créer son implémentation EmailService dans le package util

```
import org.springframework.mail.javamail.JavaMailSender;
import org.springframework.mail.javamail.MimeMessageHelper;
import org.springframework.stereotype.Service;
import jakarta.mail.MessagingException;
import jakarta.mail.internet.MimeMessage;
import lombok.AllArgsConstructor;


@AllArgsConstructor
@Service
public class EmailService implements EmailSender{

           private final JavaMailSender mailSender;

           public void sendEmail(String to, String email) {
               try {
                   MimeMessage mimeMessage = mailSender.createMimeMessage();
                   MimeMessageHelper helper =
                           new MimeMessageHelper(mimeMessage, "utf-8");
                   helper.setText(email, true);
                   helper.setTo(to);
                   helper.setSubject("Confirm your email");
                   helper.setFrom("nadhem.belhadj.abdallah@gmail.com");
                   mailSender.send(mimeMessage);
               } catch (MessagingException e) {
                           throw new IllegalStateException("failed to send email");
               }
                 }
}
```

14. Ajouter à *UserServiceImpl* la méthode qui envoie un email

```
public void sendEmailUser(User u, String code) {
        String emailBody ="Bonjour "+ "<h1>"+u.getUsername() +"</h1>" +
      " Votre code de validation est "+"<h1>"+code+"</h1>";
       emailSender.sendEmail(u.getEmail(), emailBody);
}
```

Appeler la méthode *sendEmailUser* :

```
//envoyer par email pour valider l'email de l'utilisateur
sendEmailUser(newUser,token.getToken());
```

Pour configurer l'envoie des emails, Ajouter ces lignes au fichier application

```
spring.mail.default-encoding=UTF-8
spring.mail.host=smtp.gmail.com
spring.mail.port=587
spring.mail.username=nadhem.belhadj.abdallah@gmail.com
spring.mail.password=************
spring.mail.properties.mail.smtp.auth=true
spring.mail.properties.mail.smtp.starttls.enable=true
```

Créer l'api verifEmail pour valider l'email de l'utilisateur

UserService

```
public User validateToken(String code);
```

UserServiceImpl

```
	@Override
	public User validateToken(String code) {
		VerificationToken token = verificationTokenRepo.findByToken(code);
		if(token == null){
			throw new InvalidTokenException("Invalid Token");
		}

		User user = token.getUser();
		Calendar calendar = Calendar.getInstance();
if ((token.getExpirationTime().getTime() - calendar.getTime().getTime()) <= 0){
			verificationTokenRepo.delete(token);
			throw new ExpiredTokenException("expired Token");
		}
		user.setEnabled(true);
		userRep.save(user);
		return user;
	}
```

Créer, dans le package exception, les classes *InvalidTokenException* et *ExpiredTokenException*

```
@ResponseStatus(value = HttpStatus.NOT_FOUND)
public class InvalidTokenException  extends RuntimeException{
	private String message;

	public InvalidTokenException(String message){
		super(message);
	   }
}
@ResponseStatus(value = HttpStatus.BAD_REQUEST)
public class ExpiredTokenException extends RuntimeException{
	private String message;

	public ExpiredTokenException (String message){
		super(message);
	   }
```

```
}
```

UserController

```
   @GetMapping("/verifyEmail/{token}")
    public User verifyEmail(@PathVariable("token") String token){
            return userService.validateToken(token);
    }



  @ExceptionHandler(InvalidTokenException.class)
    public ResponseEntity<ErrorDetails>
handleInvalidTokenException(InvalidTokenException exception,
WebRequest webRequest){

        ErrorDetails errorDetails = new ErrorDetails(
                LocalDateTime.now(),
                exception.getMessage(),
                webRequest.getDescription(false),
                "INVALID_TOKEN"
        );

        return new ResponseEntity<>(errorDetails, HttpStatus.NOT_FOUND);
    }


    @ExceptionHandler(ExpiredTokenException.class)
    public ResponseEntity<ErrorDetails>
handleExpiredTokenException(ExpiredTokenException exception,
                                                    WebRequest webRequest){

        ErrorDetails errorDetails = new ErrorDetails(
                LocalDateTime.now(),
                exception.getMessage(),
                webRequest.getDescription(false),
                "EXPIRED_TOKEN"
        );
        return new ResponseEntity<>(errorDetails, HttpStatus.BAD_REQUEST);
    }
```

Tester l'api avec POSTMAN

Consommer l'api verifEmail à partir de Angular

Créer le composant verifEmail

ng g c verifEmail

```
  { path: 'verifEmail', component: VerifEmailComponent },
```

Modifier la classe AuthService

```
public regitredUser : User = new User();

setRegistredUser(user : User){
  this.regitredUser=user;
}

getRegistredUser(){
  return this.regitredUser;
}
```

Modifier la classe RegisterComponent

```
 this. authService.setRegistredUser(this.user);
 alert("veillez confirmer votre email");
 this.router.navigate(["/verifEmail"]);
```

verif-email.component.html

```
<div class="container mt-5">
    <div class="row justify-content-md-center">
        <div class="col-md-4">
          @if(err) {
            <div class="alert alert-danger" >
                        <strong>{{err}}</strong>
            </div>
           }
         <form >
              <div class="txt_field">
                  <label>Veuillez entrer le code envoyé par mail</label>
                  <input type="text" name="code" [(ngModel)]="code"   class="form-
control"   placeholder=" code de confirmation" >
              </div>
              <button  type="button"  class="btn btn-success mt-3"
(click)="onValidateEmail()">S'insrire</button>
         </form>
        </div>
    </div>
</div>
```

verif-email.component.ts

```
import { Component, OnInit } from '@angular/core';
import { User } from '../model/user.model';
import { ActivatedRoute, Router } from '@angular/router';
import { AuthService } from '../services/auth.service';

@Component({
  selector: 'app-verif-email',
  standalone: true,
  imports: [],
  templateUrl: './verif-email.component.html',
  styleUrl: './verif-email.component.css',
})
export class VerifEmailComponent implements OnInit {
  code: string = '';
  user: User = new User();
  err = '';

  constructor(
    private route: ActivatedRoute,
    private authService: AuthService,
    private router: Router
  ) {}

  ngOnInit(): void {
    this.user = this.authService.regitredUser;
  }

  onValidateEmail() {
    this.authService.validateEmail(this.code).subscribe({
      next: (res) => {
        alert('Login successful');
        this.authService.login(this.user).subscribe({
          next: (data) => {
            let jwToken = data.headers.get('Authorization')!;
            this.authService.saveToken(jwToken);
            this.router.navigate(['/']);
          },
          error: (err: any) => {
            console.log(err);
          },
        });
      },
      error: (err: any) => {
        if ((err.status = 400)) {
          this.err = err.error.message;
        }
        console.log(err.errorCode);
```

```
      },
    });
  }
}
```

auth.service.ts

```
validateEmail(code : string){
   return this.http.get<User>(this.apiURL+'/verifyEmail/'+code);
  }
```

Résoudre le problème de CORS, en ajoutant ces lignes à la méthode *filterChain* de la classe *SecurityConfig*

```
@Bean
     public SecurityFilterChain filterChain(HttpSecurity http) throws Exception {
                    http.csrf().disable()
                    .sessionManagement().sessionCreationPolicy(SessionCreationPolicy.STATELESS).and()

.cors(cors -> cors.configurationSource(new CorsConfigurationSource() {
                    @Override
                    public CorsConfiguration getCorsConfiguration(HttpServletRequest
request) {
                        CorsConfiguration cors = new CorsConfiguration();

cors.setAllowedOrigins(Collections.singletonList("http://localhost:4200"));
                        cors.setAllowedMethods(Collections.singletonList("*"));
                        cors.setAllowCredentials(true);
                        cors.setAllowedHeaders(Collections.singletonList("*"));

cors.setExposedHeaders(Collections.singletonList("Authorization"));
                        cors.setMaxAge(3600L);
                        return cors;
                    }
                }))

            .authorizeHttpRequests()

.requestMatchers("/login","/register/**","/verifyEmail/**").permitAll()

…
```

Améliorer la gestion des erreurs

```
if  (err.error.errorCode=="INVALID_TOKEN")
          this.err="Code invalide!"

 if  (err.error.errorCode=="EXPIRED_TOKEN")
          this.err="Code a expiré!"
```

Modifier le login pour vérifier si l'utilisateur est activé

Coté Backend

Modifier la méthode `loadUserByUsername` de la classe `MyUserDetailsService`

```
return new org.springframework.security.core.
userdetails.User(user.getUsername(),user.getPassword(),user.getEnabled(),true,true
,true,auths);
```

La classe *JWTAuthenticationFilter*

Ajouter la méthode suivante :

```
@Override
protected void unsuccessfulAuthentication(HttpServletRequest request,
HttpServletResponse response,   hentication Exception failed)
throws IOException, ServletException {

         if (failed instanceof DisabledException ) {
            response.setStatus(HttpServletResponse.SC_FORBIDDEN);
            response.setContentType("application/json");
            Map<String, Object> data = new HashMap<>();

            data.put("errorCause", "disabled");
            data.put("message", "L'utilisateur est désactivé !");

            ObjectMapper objectMapper = new ObjectMapper();
            String json = objectMapper.writeValueAsString(data);

            PrintWriter writer = response.getWriter();
            writer.println(json);
            writer.flush();

        } else {
            super.unsuccessfulAuthentication(request, response, failed);
        }
    }
```

Coté Front end

login.component.ts

```
  message : string = "login ou mot de passe erronés..";
```

```
onLoggedin()
    {
       this.authService.login(this.user).subscribe({
        next: (data) => {

          let jwToken = data.headers.get('Authorization')!;
          this.authService.saveToken(jwToken);
          this.router.navigate(['/']);
        },
        error: (err) => {
```

```
            this.err = 1;
            if (err.error.errorCause=='disabled')
             this.message="Utilisateur désactivé, Veuillez contacter votre Administrateur";

        }
        });
    }
```

login.component.html

```
<div class="col-md-4">
      @if(erreur==1)   {
            <div  class="alert alert-danger" >
                <strong>{{message}}</strong>
            </div>
          }
</div>
```

Quelques améliorations

Installer le module toastr

npm install ngx-toastr --save

app.config.ts

```
 providers: […
              provideAnimations(), // required animations providers
              provideToastr(), // Toastr providers
            ]
```

register.component.ts

```
Constructor(…
                private toastr: ToastrService) { }
```

```
   this.toastr.success('veillez confirmer votre email', 'Confirmation');
```

Ajouter toastr.css au fichier angular.json :

```
"styles": [
            …
              "node_modules/ngx-toastr/toastr.css"
            ],
```

Redémarrer le serveur Angular avant de tester

Ajouter un spinner en attenant l'envoie de l'email

```
@if(loading){
                <div class="d-flex justify-content-center">
                       <div class="spinner-border" role="status">
                       <span class="sr-only">Loading...</span>
                       </div>
                </div>
                }
```