We wil use api gaetway to authenticate any request which comes

We tell api gateway if any request come please authenticate it if it is authenticated then please refer it tome else reject it.

A diagram of a service

Description automatically generated

We will make one microservice and who will all the security related stuff.

**How it runs?**

1.firxt of all the user will reach out to the auth service reigsiter himself and get a token

2.next he will make a call to the api gateway for some endpoint using that token

3.now api gateway will send request to the auth service microservice to authenticate if the token is valid or not.

A diagram of service registration

Description automatically generated

**DEVELOPING THE AUTH-SERVICE**

We will make a new spring boot project and add the below dependencies to it.

A screenshot of a computer

Description automatically generated

Now we add jwt dependencies.

A screenshot of a computer program

Description automatically generated

Now make some packages .

Entity,repository,service,controller,config

A screenshot of a computer

Description automatically generated

Now make entity and repositoty

A screenshot of a computer program

Description automatically generated

A screenshot of a computer

Description automatically generated

Repo

A screenshot of a computer

Description automatically generated

Service

First we need to save a user who wants to register

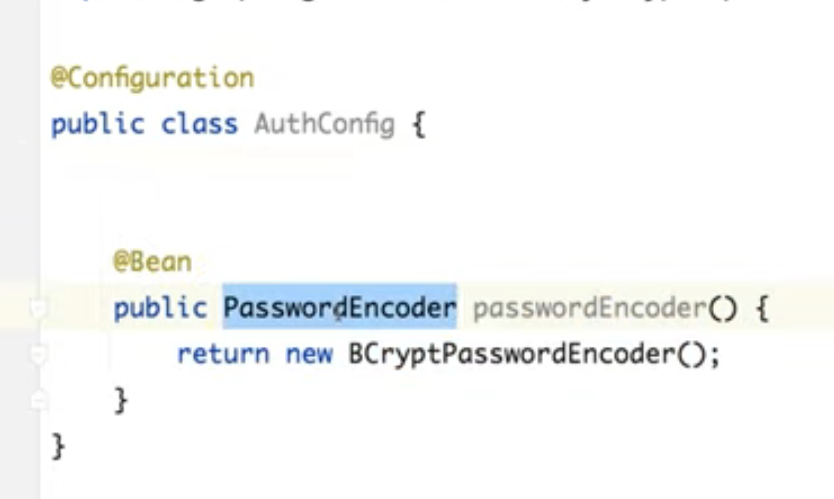
now while saving user we are using the usercredential entity and in this class the password is mentioned a s a string but we want to save as encrypted

A screen shot of a computer program

Description automatically generated



So we will make a config class



After declaring this bean e will inject it in our service.

A screenshot of a computer program

Description automatically generated

Now our password will be saved in db in encrypted form .

Now we want to make a util class in order to make a token and then validate the token

|  |
| --- |
| @Service  **public** **class** JwtService {  **public** **static** **final** String ***SECRET*** = "5367566B59703373367639792F423F4528482B4D6251655468576D5A71347437";  **public** **void** validateToken(**final** String token) {  Jwts.*parserBuilder*().setSigningKey(getSignKey()).build().parseClaimsJws(token);  }  **public** String generateToken(String userName) {  Map<String, Object> claims = **new** HashMap<>();  **return** createToken(claims, userName);  }  **private** String createToken(Map<String, Object> claims, String userName) {  **return** Jwts.*builder*()  .setClaims(claims)  .setSubject(userName)  .setIssuedAt(**new** Date(System.*currentTimeMillis*()))  .setExpiration(**new** Date(System.*currentTimeMillis*() + 1000 \* 60 \* 30))  .signWith(getSignKey(), SignatureAlgorithm.***HS256***).compact();  }  **private** Key getSignKey() {  **byte**[] keyBytes = Decoders.***BASE64***.decode(***SECRET***);  **return** Keys.*hmacShaKeyFor*(keyBytes);  }  } |

Now we come to auth service and write a method to generate token and validate a token

|  |
| --- |
| @Service  **public** **class** AuthService {  @Autowired  **private** UserCredentialRepository repository;  @Autowired  **private** PasswordEncoder passwordEncoder;  @Autowired  **private** JwtService jwtService;  **public** String saveUser(UserCredential credential) {  credential.setPassword(passwordEncoder.encode(credential.getPassword()));  repository.save(credential);  **return** "User added to the system";  }    **public** String generateToken(String username)  {  **return** jwtService.generateToken(username);    }    **public** **void** validateToken(String token) {  jwtService.validateToken(token);  }  } |

Now we need to call these methods from our controller

|  |
| --- |
| //Add or register a user  @RestController  @RequestMapping("/auth")  **public** **class** AuthController {  @Autowired  **private** AuthService service;  @Autowired  **private** AuthenticationManager authenticationManager;      @PostMapping("/register")  **public** String addNewUser(@RequestBody UserCredential user) {  **return** service.saveUser(user);  } |

|  |
| --- |
| //GENERATE TOEKN |
|  |
|  |

Now with the above method for each and evry user we will get a token but we not want so we want to first authenticate who is entering our system so we will make use of authentication manager fpr that

We will go to auth config class and specify what endpoint needs to be allowed to enter without authentication.

|  |
| --- |
| @Configuration  @EnableWebSecurity  **public** **class** AuthConfig {  @Bean  **public** UserDetailsService userDetailsService(){  **return** **new** CustomUserDetailsService();  }  @Bean  **public** SecurityFilterChain securityFilterChain(HttpSecurity http) **throws** Exception {  **return** http.~~csrf~~().disable()  .~~authorizeHttpRequests~~()  .requestMatchers("/auth/register", "/auth/token", "/auth/validate").permitAll()  .~~and~~()  .build();  } |
|  |

DEFINE A BEAN OF AUTHENTICATION MANAGER IN AUTHCONFIG



And now nject it in our controller to authenticate who is asking for a token



Now authentication manager wants to talk to the db in order to check for authenticated users in the db so it will need someones help.

So userdetails service will help in that case.

We will create a user details class who will connect to the db.

We will make our own user detailsservic custom class who will ftech data from the dba d give it back to authentication provider who in turn will give it back to the authentication manager.

A close-up of a computer screen

Description automatically generated

A screenshot of a computer

Description automatically generated

The new class.

Now go to db and check for the user.

A screenshot of a computer

Description automatically generated

Now we need to convert our usercredential to userdetaisl object and send it back so we will take help of custom userdetaisl class to do so.

|  |
| --- |
| package com.microservice.practise.SecurityService.config;  import java.util.Collection;  import org.springframework.security.core.GrantedAuthority;  import org.springframework.security.core.userdetails.UserDetails;  import com.microservice.practise.SecurityService.entity.UserCredential;  public class CustomUserDetails implements UserDetails{    private String username;  private String password;        public CustomUserDetails(UserCredential userCredential) {  super();  this.username = userCredential.getName();  this.password = userCredential.getPassword();  }  @Override  public Collection<? extends GrantedAuthority> getAuthorities() {  // TODO Auto-generated method stub  return null;  }  @Override  public String getPassword() {  // TODO Auto-generated method stub  return password;  }  @Override  public String getUsername() {  // TODO Auto-generated method stub  return username;  }  @Override  public boolean isAccountNonExpired() {  // TODO Auto-generated method stub  return true;  }  @Override  public boolean isAccountNonLocked() {  // TODO Auto-generated method stub  return true;  }  @Override  public boolean isCredentialsNonExpired() {  // TODO Auto-generated method stub  return true;  }  @Override  public boolean isEnabled() {  // TODO Auto-generated method stub  return true;  }  } |

Our custome user details service class will be like this

|  |
| --- |
| package com.microservice.practise.SecurityService.config;  import java.util.Optional;  import org.springframework.beans.factory.annotation.Autowired;  import org.springframework.security.core.userdetails.UserDetails;  import org.springframework.security.core.userdetails.UserDetailsService;  import org.springframework.security.core.userdetails.UsernameNotFoundException;  import com.microservice.practise.SecurityService.entity.UserCredential;  import com.microservice.practise.SecurityService.repository.UserCredentialRepository;  public class CustomUserDetailsService implements UserDetailsService {  @Autowired  private UserCredentialRepository repository;  @Override  public UserDetails loadUserByUsername(String username) throws UsernameNotFoundException {  Optional<UserCredential> credential =repository.findByName(username);    return credential.map(CustomUserDetails::new)  .orElseThrow(()->new UsernameNotFoundException("user not found"));    }  } |

Now we will define bean of authenticationprovidr

A screen shot of a computer

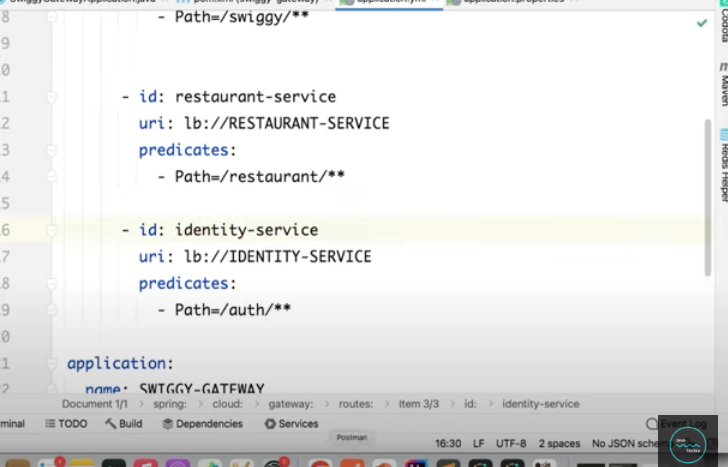
Description automatically generated

**INTEGRATION OF API GATEWAY**

**CONIFGURE SECUIRTY IN API GATEWAY**

**1.first define security service in the api gateway**

**In application.properties**

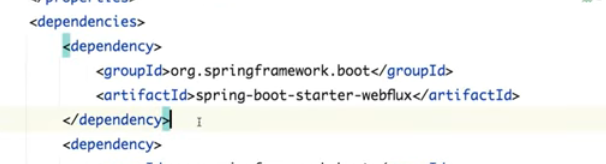
****

**Now get token from the api gateway.user cannot directly reach our identity service**

**Call security service form api gateway itself**

**NOW MAKE AN AUTHENTICATION FILTER IN THE API GATEWAY**

We need to have a webflux dependency in our project



We are adding this dependency because our api gateway will take input in the form of webflux.



We will write this method in the authentication filter clas

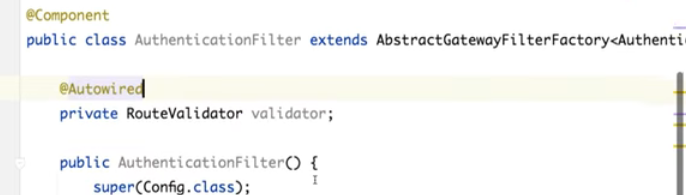
Inside hthis method we will write simple lgic to validate if it contains a header or not.then we will do the validate token call

But first we need to tell for what endpoints do we need to do the validation

We make a route validator class which will tell us which endpoints do not need to be validated.

|  |
| --- |
| @Component  **public** **class** RouteValidator {  **public** **static** **final** List<String> ***openApiEndpoints*** = List.*of*(  "/auth/register",  "/auth/token",  "/eureka"  );  **public** Predicate<ServerHttpRequest> isSecured =  request -> ***openApiEndpoints***  .stream()  .noneMatch(uri -> request.getURI().getPath().contains(uri));  } |
|  |
|  |

Now we autowire this route validator in the authentication filter class



No we will write the logic to check if the request header has the token then we will make a call to the security service to check if the token is valid or not



It Is not a good practice to make a call from api gateway to the security service it can be hacked so

We will implement something else

You can add the validate token method to api gateway only.

Add jwt dependencies to the application

We need same secret to encrypt and decrypt.

No we want that before sending the request to any other microservice we need to call this filter 

Now before making a call to some other microservice from our gateway we need to execute the filter so we will specify in the application.properties file for the same

