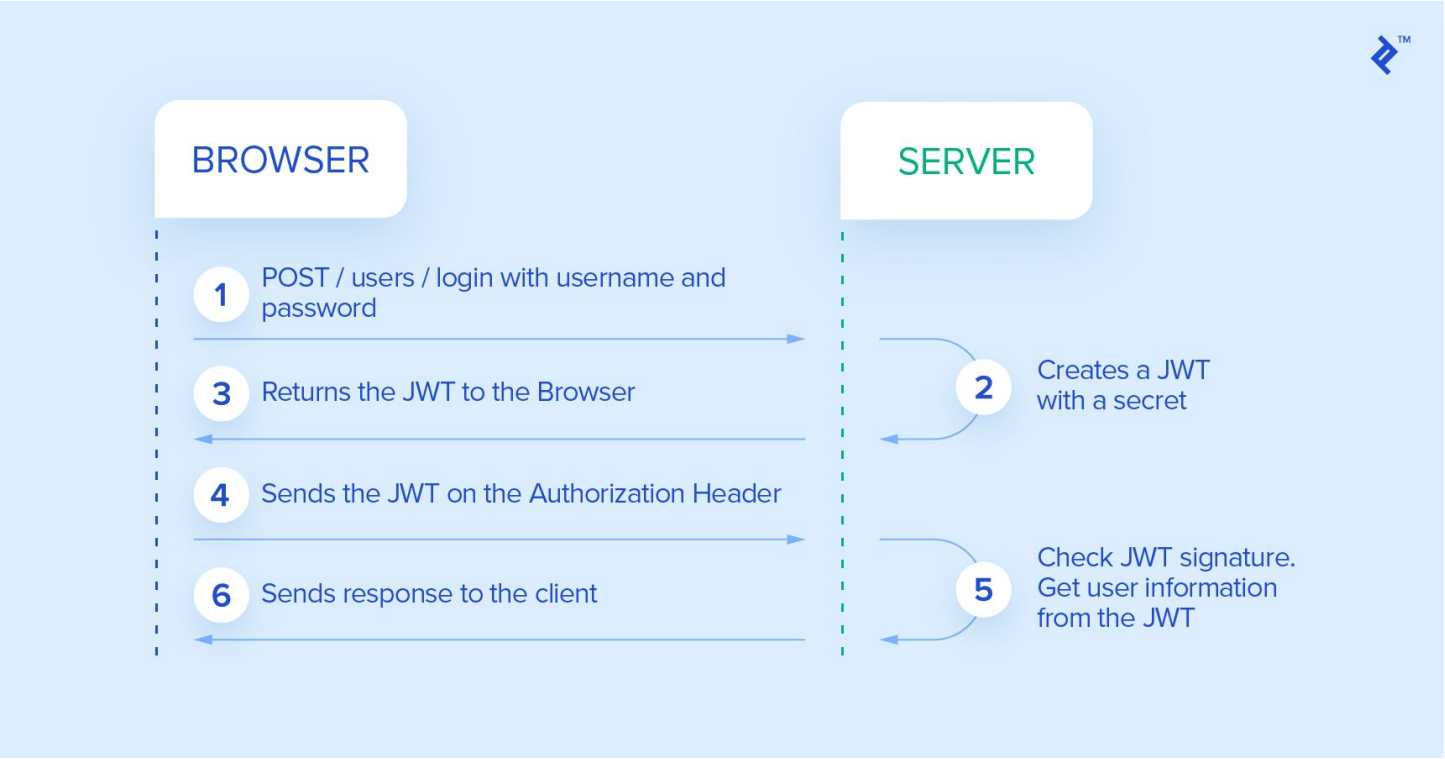
JWT, Angular integration

What JWT is and how it work?

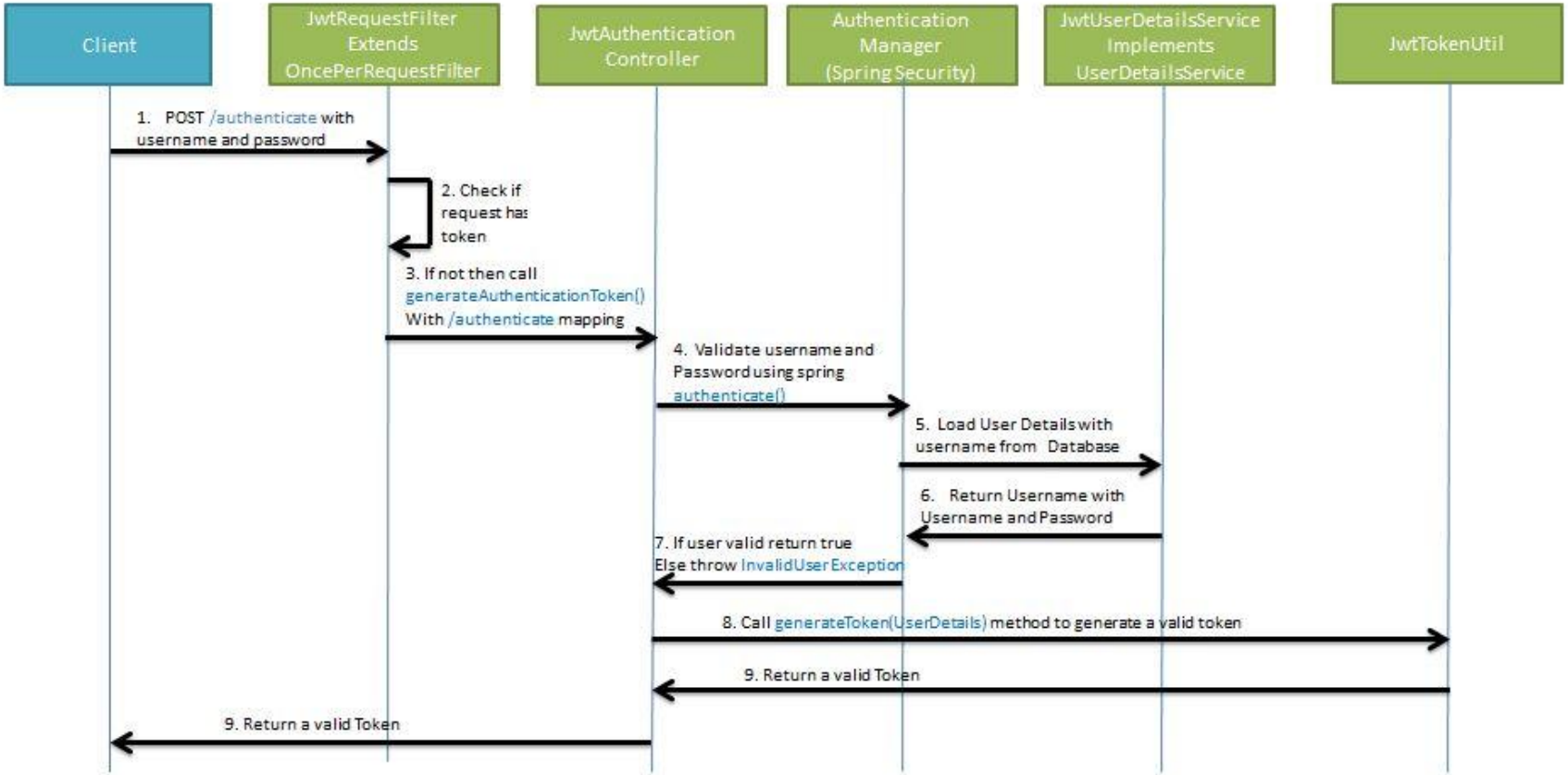
JWT (shortened from JSON Web Token) is the missing standardization for using tokens to authenticate on the web in general, not only for REST services. Currently, it is in draft status as [RFC 7519](#).



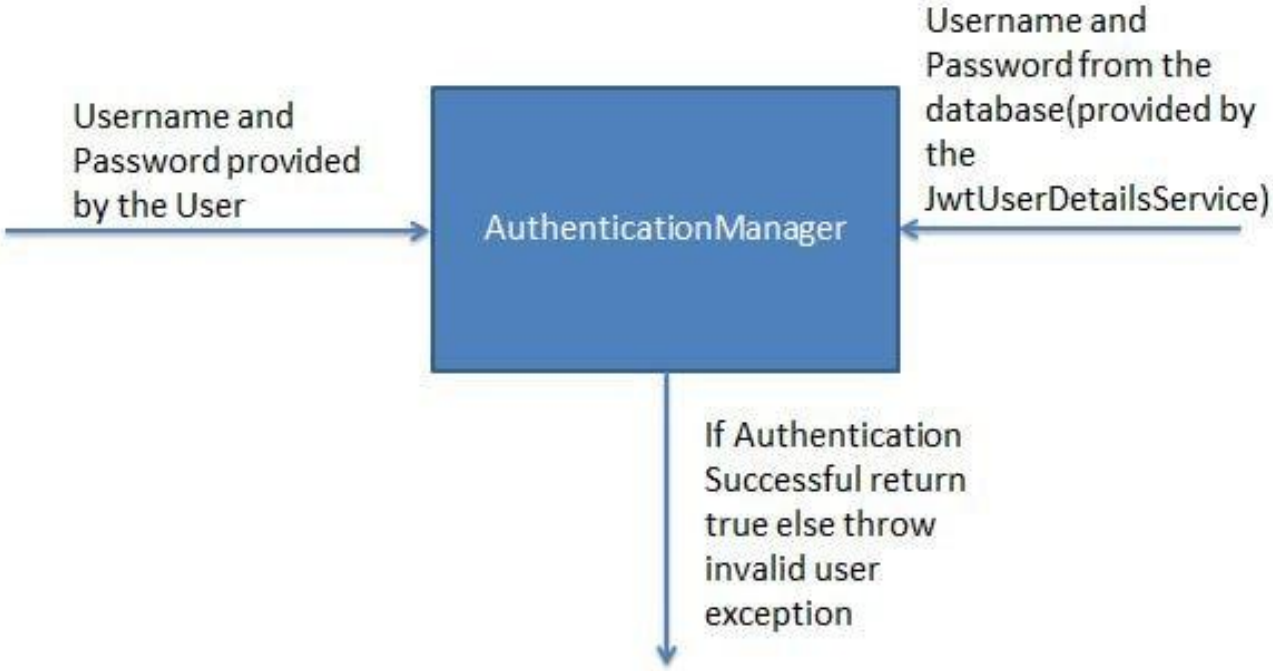
What JWT is and how it work?

- Clients logs in by sending their credentials to the identity provider.
- The identity provider verifies the credentials; if all is OK, it retrieves the user data, generates a JWT containing user details and permissions that will be used to access the services, and it also sets the expiration on the JWT (which might be unlimited).
- Identity provider signs, and if needed, encrypts the JWT and sends it to the client as a response to the initial request with credentials.
- Client stores the JWT for a limited or unlimited amount of time, depending on the expiration set by the identity provider.
- Client sends the stored JWT in an Authorization header for every request to the service provider.
- For each request, the service provider takes the JWT from the `Authorization` header and decrypts it, if needed, validates the signature, and if everything is OK, extracts the user data and permissions. Based on this data solely, and again without looking up further details in the database or contacting the identity provider, it can accept or deny the client request. The only requirement is that the identity and service providers have an agreement on encryption so that service can verify the signature or even decrypt which identity was encrypted.

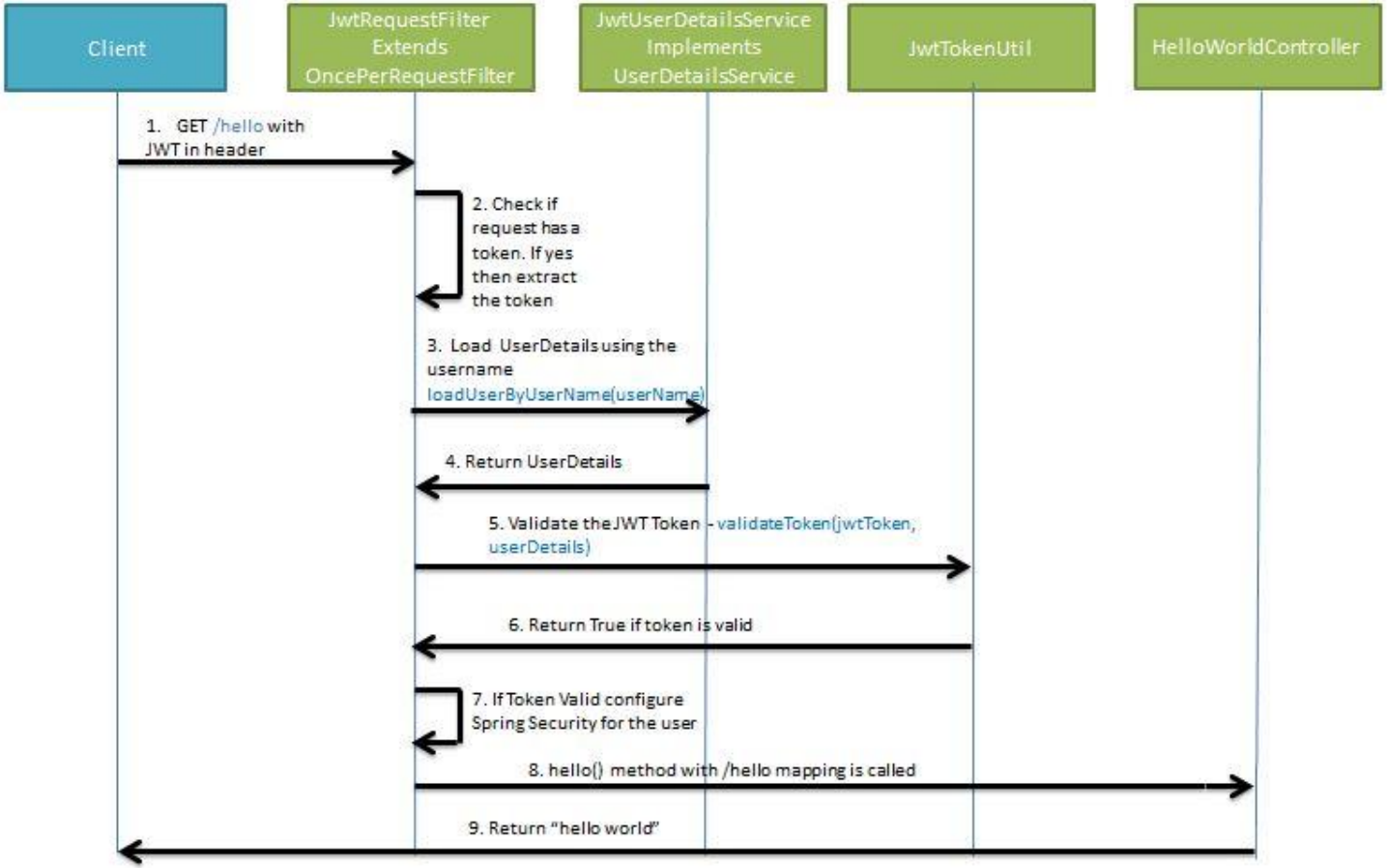
Implementation steps (JWT creation)



Implementation steps (Password check with AuthenticationManager)



Implementation steps (JWT validation)



Simple JWT implementation workshop

Practical part

Simple JWT implementation workshop

- Add missing dependencies in pom.xml

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-security</artifactId>
</dependency>

<dependency>
    <groupId>io.jsonwebtoken</groupId>
    <artifactId>jjwt</artifactId>
    <version>0.9.1</version>
</dependency>
```

Simple JWT implementation workshop

- Add JwtTokenUtil.java to auth package

1

```java
package com.sd.petclinic.auth;

import java.io.Serializable;
import java.util.Date;
import java.util.HashMap;
import java.util.Map;
import java.util.function.Function;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.security.core.userdetails.UserDetails;
import org.springframework.stereotype.Component;
import io.jsonwebtoken.Claims;
import io.jsonwebtoken.Jwts;
import io.jsonwebtoken.SignatureAlgorithm;

@Component
public class JwtTokenUtil implements Serializable {
  private static final long serialVersionUID = 1L;

  public static final long JWT_TOKEN_VALIDITY = 5 * 60 * 60;
  @Value("${jwt.secret}")
  private String secret;

  // Retrieve username from jwt token
  public String getUsernameFromToken(String token) {
    return getClaimFromToken(token, Claims::getSubject);
  }
...
```

2

```java
.../
/ Retrieve expiration date from jwt token
  public Date getExpirationDateFromToken(String token) {
    return getClaimFromToken(token, Claims::getExpiration);
  }

  public <T> T getClaimFromToken(String token, Function<Claims, T> claimsResolver) {
    final Claims claims = getAllClaimsFromToken(token);
    return claimsResolver.apply(claims);
  }

  // For retrieveing any information from token we will need the secret key
  private Claims getAllClaimsFromToken(String token) {
    return Jwts.parser().setSigningKey(secret).parseClaimsJws(token).getBody();
  }

  // Check if the token has expired
  private Boolean isTokenExpired(String token) {
    final Date expiration = getExpirationDateFromToken(token);
    return expiration.before(new Date());
  }

  // Generate token for user
  public String generateToken(UserDetails userDetails) {
    Map<String, Object> claims = new HashMap<>();
    return doGenerateToken(claims, userDetails.getUsername());
  }

  // While creating the token -
  // 1. Define claims of the token, like Issuer, Expiration, Subject, and the ID
  // 2. Sign the JWT using the HS512 algorithm and secret key.
  // 3. According to JWS Compact
  // Serialization(https://tools.ietf.org/html/draft-ietf-jose-json-web-signature-41#section-3.1)
  // compaction of the JWT to a URL-safe string
  private String doGenerateToken(Map<String, Object> claims, String subject) {
    return Jwts.builder().setClaims(claims).setSubject(subject).setIssuedAt(new Date(System.currentTimeMillis()))
        .setExpiration(new Date(System.currentTimeMillis() + JWT_TOKEN_VALIDITY * 1000))
        .signWith(SignatureAlgorithm.HS512, secret).compact();
  }

  // Validate token
  public Boolean validateToken(String token, UserDetails userDetails) {
    final String username = getUsernameFromToken(token);
    return (username.equals(userDetails.getUsername()) && !isTokenExpired(token));
  }
}
```

Simple JWT implementation workshop

- Create JwtRequest.java and JwtResponse.java models to hold user credentials and token

1

```java
package com.sd.petclinic.auth;

import java.io.Serializable;

public class JwtRequest implements Serializable {
  private static final long serialVersionUID = 1L;
  private String username;
  private String password;

  public JwtRequest() {
  }

  public JwtRequest(String username, String password) {
    this.setUsername(username);
    this.setPassword(password);
  }

  public String getUsername() {
    return this.username;
  }

  public void setUsername(String username) {
    this.username = username;
  }

  public String getPassword() {
    return this.password;
  }

  public void setPassword(String password) {
    this.password = password;
  }
}
```

2

```java
package com.sd.petclinic.auth;

import java.io.Serializable;

public class JwtResponse implements
Serializable {
  private static final long serialVersionUID = 1L;
  private final String jwttoken;

  public JwtResponse(String jwttoken) {
    this.jwttoken = jwttoken;
  }

  public String getToken() {
    return this.jwttoken;
  }
}
```

Simple JWT implementation workshop

- Create JwtUserDetailsService in auth package with hardcoded user

```java
package com.sd.petclinic.auth;

import java.util.ArrayList;
import org.springframework.security.core.userdetails.User;
import org.springframework.security.core.userdetails.UserDetails;
import org.springframework.security.core.userdetails.UserDetailsService;
import org.springframework.security.core.userdetails.UsernameNotFoundException;
import org.springframework.stereotype.Service;

@Service
public class JwtUserDetailsService implements UserDetailsService {

  @Override
  public UserDetails loadUserByUsername(String username) throws UsernameNotFoundException {
    if ("petclinic".equals(username)) {
      return new User("petclinic", "$2a$10$UQcyonuRbAVtqaa5HAJbj.5egqExgXWec22ZqkXwPVpRIXo.JU98y",
          new ArrayList<>());
    } else {
      throw new UsernameNotFoundException("User not found with username: " + username);
    }
  }
}
```

Simple JWT implementation workshop

- Create JwtAuthenticationController.java in auth package

```
...
@RestController
@CrossOrigin
public class JwtAuthenticationController {
  @Autowired
  private AuthenticationManager authenticationManager;
  @Autowired
  private JwtTokenUtil jwtTokenUtil;
  @Autowired
  private JwtUserDetailsService userDetailsService;

  @RequestMapping(value = "/authenticate", method = RequestMethod.POST)
  public JwtResponse createAuthenticationToken(@RequestBody JwtRequest authenticationRequest) throws Exception {
    authenticate(authenticationRequest.getUsername(), authenticationRequest.getPassword());
    final UserDetails userDetails = userDetailsService.loadUserByUsername(authenticationRequest.getUsername());
    final String token = jwtTokenUtil.generateToken(userDetails);
    return new JwtResponse(token);
  }

  private void authenticate(String username, String password) throws Exception {
    try {
      authenticationManager.authenticate(new UsernamePasswordAuthenticationToken(username, password));
    } catch (DisabledException e) {
      throw new Exception("USER_DISABLED", e);
    } catch (BadCredentialsException e) {
      throw new Exception("INVALID_CREDENTIALS", e);
    }
  }
}
```

Simple JWT implementation workshop

- Create JwtRequestFilter

1

```
@Component
public class JwtRequestFilter extends OncePerRequestFilter {
  @Autowired
  private JwtUserDetailsService jwtUserDetailsService;
  @Autowired
  private JwtTokenUtil jwtTokenUtil;

  @Override
  protected void doFilterInternal(
   HttpServletRequest request, HttpServletResponse response, FilterChain chain
  )
        throws ServletException, IOException {

    final String requestTokenHeader = request.getHeader("Authorization");
    String username = null;
    String jwtToken = null;
    // JWT Token is in the form "Bearer token". Remove Bearer word and get
    // only the token
    if (requestTokenHeader != null && requestTokenHeader.startsWith("Bearer ")) {
      jwtToken = requestTokenHeader.substring(7);
      try {
        username = jwtTokenUtil.getUsernameFromToken(jwtToken);
      } catch (IllegalArgumentException e) {
        System.out.println("Unable to get JWT Token");
      } catch (ExpiredJwtException e) {
        System.out.println("JWT Token has expired");
      }
    } else {
      logger.warn("JWT Token does not begin with Bearer String");
    }
```

2

```
// Once we get the token validate it.
      if (username != null &&
SecurityContextHolder.getContext().getAuthentication() == null) {
          UserDetails userDetails =
this.jwtUserDetailsService.loadUserByUsername(username);
          // If token is valid configure Spring Security to manually set
          // authentication
          if (jwtTokenUtil.validateToken(jwtToken, userDetails)) {
            UsernamePasswordAuthenticationToken
usernamePasswordAuthenticationToken = new
UsernamePasswordAuthenticationToken(
                    userDetails, null, userDetails.getAuthorities());
            usernamePasswordAuthenticationToken
                    .setDetails(new
WebAuthenticationDetailsSource().buildDetails(request));

            // After setting the Authentication in the context, we specify
            // that the current user is authenticated. So it passes the
            // Spring Security Configurations successfully.

SecurityContextHolder.getContext().setAuthentication(usernamePasswordAu
thenticationToken);
          }
        }
        chain.doFilter(request, response);
    }
}
```

Simple JWT implementation workshop

- Create JwtAuthenticationEntryPoint.java

```java
package com.sd.petclinic.auth;

import java.io.IOException;
import java.io.Serializable;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import org.springframework.security.core.AuthenticationException;
import org.springframework.security.web.AuthenticationEntryPoint;
import org.springframework.stereotype.Component;

@Component
public class JwtAuthenticationEntryPoint implements AuthenticationEntryPoint, Serializable {
  private static final long serialVersionUID = 1L;

  @Override
  public void commence(HttpServletRequest request, HttpServletResponse response,
        AuthenticationException authException) throws IOException {
    response.sendError(HttpServletResponse.SC_UNAUTHORIZED, "Unauthorized");
  }
}
```

Simple JWT implementation workshop

- Configure security in Java config (SecurityConfig.java)

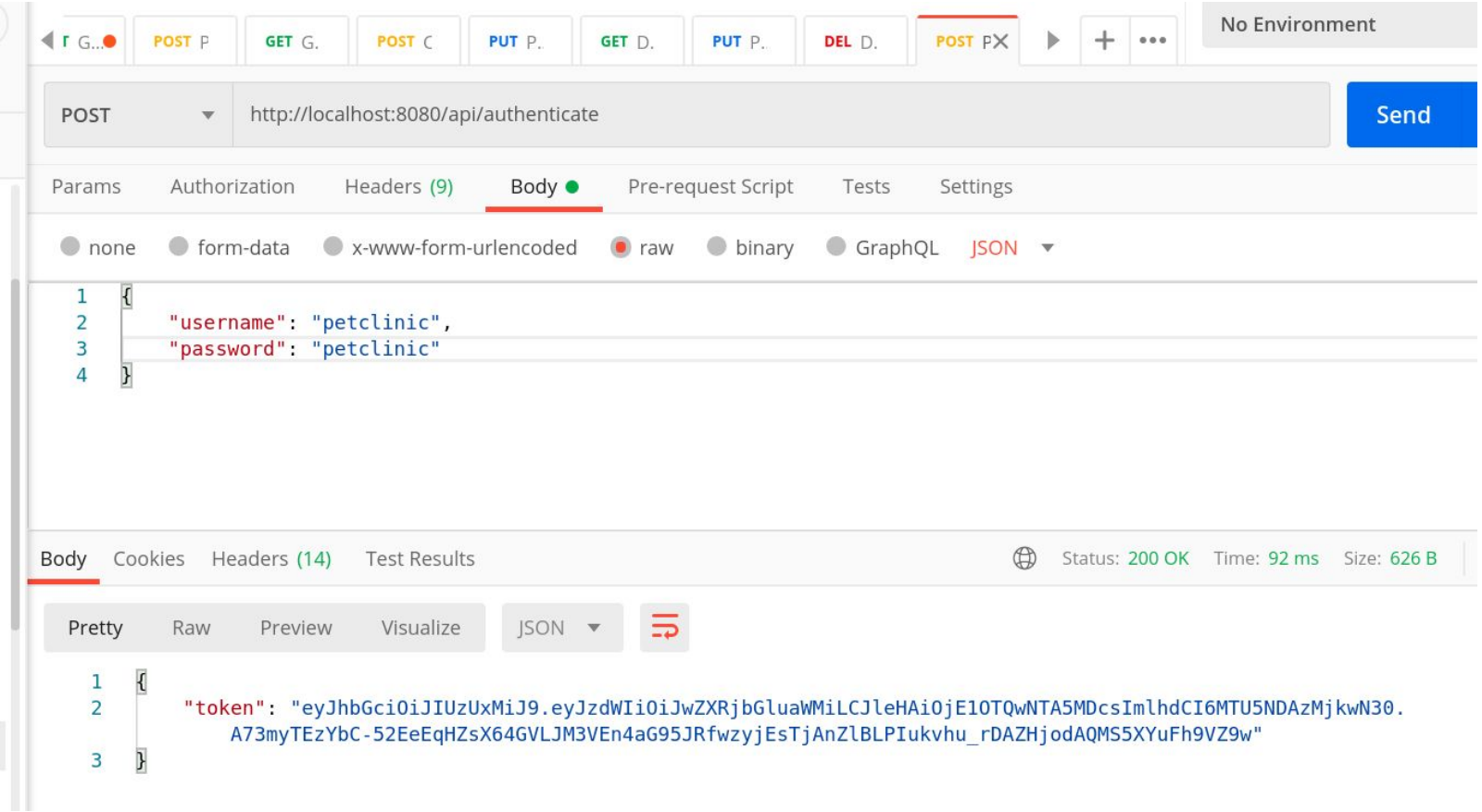
1

```
@Configuration
@EnableWebSecurity
@EnableGlobalMethodSecurity(prePostEnabled = true)
public class SecurityConfig extends WebSecurityConfigurerAdapter {
  @Autowired
  private JwtAuthenticationEntryPoint jwtAuthenticationEntryPoint;
  @Autowired
  private UserDetailsService jwtUserDetailsService;
  @Autowired
  private JwtRequestFilter jwtRequestFilter;

  @Bean
  @Override
  public AuthenticationManager authenticationManagerBean() throws
Exception {
    return super.authenticationManagerBean();
  }

  ...
```

2

```
...
@Autowired
  public void configureGlobal(AuthenticationManagerBuilder auth) throws
Exception {

auth.userDetailsService(jwtUserDetailsService).passwordEncoder(password
Encoder());
  }

  @Bean
  public PasswordEncoder passwordEncoder() {
    return new BCryptPasswordEncoder();
  }

  @Override
  protected void configure(HttpSecurity httpSecurity) throws Exception {
    // We don't need CSRF for REST based API
    httpSecurity.csrf().disable()
        // Don't authenticate this particular request
        .authorizeRequests().antMatchers("/api/authenticate").permitAll()
        // All other requests need to be authenticated
        .anyRequest().authenticated().and()
        // Make sure we use stateless session; session won't be used to
        // store user's state

.exceptionHandling().authenticationEntryPoint(jwtAuthenticationEntryPoint).a
nd().sessionManagement()
        .sessionCreationPolicy(SessionCreationPolicy.STATELESS);

    // Add a filter to validate the tokens with every request
    httpSecurity.addFilterBefore(jwtRequestFilter,
UsernamePasswordAuthenticationFilter.class);
  }
}
```

Simple JWT implementation workshop



Simple JWT implementation workshop

Let's integrate user model from lessons678 project so we could store users in database

a) Move user and role models

b) Add database creation SQL statements for user and roles

c) Create SQL seed statements for users

d) Create a UserRepository

e) Update JwtUserDetailsService for loading data from database

Simple JWT implementation workshop

Move user and role models

```
@Entity
@Table(name = "users")
@ScriptAssert(lang = "javascript", script = "_this.decryptedPassword ===
_this.passwordConfirm", message = "must match", reportOn = "decryptedPassword")
public class User extends BaseEntity {
  private static final long serialVersionUID = 1L;

  @NotBlank
  @Column(name = "username")
  private String username;

  @JsonIgnore
  @Column(name = "password")
  private String password;

  ....
```

```
@Entity
@Table(name = "roles")
public class Role extends BaseEntity {
 private static final long serialVersionUID = 1L;

 @Enumerated(EnumType.STRING)
  @Column(
    name = "name",
    columnDefinition = "ENUM('ADMIN','EDITOR', 'USER', 'ANONYMOUS')",
    nullable = false
  )
  private RoleName name;

  @ManyToMany(mappedBy = "roles")
  private Set<User> users;

  public void setName(RoleName name) {
    this.name = name;
  }

  public RoleName getName() {
    return name;
  }

  public Set<User> getUsers() {
    return users;
  }

  public void setUsers(Set<User> users) {
    this.users = users;
  }

  public enum RoleName {
      ADMIN,
      EDITOR,
      USER,
      ANONYMOUS
  }
}
```

Simple JWT implementation workshop

Add database creation SQL statements for user and roles and create SQL seed statements for users

```sql
create table if not exists users
(
 id bigint not null auto_increment primary key,
 username varchar(255) null,
 password varchar(255) null
);

create table if not exists roles
(
 id bigint not null auto_increment primary key,
 name enum ('ADMIN', 'EDITOR', 'USER', 'ANONYMOUS') not null
);

create table if not exists user_role
(
 user_id bigint not null,
 role_id bigint not null,
 primary key (user_id, role_id),
 constraint FK_user_role_users
   foreign key (user_id) references users (id),
 constraint FK_user_role_roles
   foreign key (role_id) references roles (id)
);
```

```sql
-- password - petclinic
INSERT IGNORE INTO users VALUES (1, 'petclinic',
'$2a$10$mWmWbwt6mDRikhJbPKmQDewJ8Of2BtdBd6xzvdGN8yKo06gFGb/p2');
-- password - user
INSERT IGNORE INTO users VALUES (2, 'user',
'$2a$10$UDW6q2eTpOdLG24crN/FgOdzxB03CM/fT0T84IU6f8elbZqxSXc2i');

INSERT IGNORE INTO roles VALUES (1, 'ADMIN');
INSERT IGNORE INTO roles VALUES (2, 'USER');

INSERT IGNORE INTO user_role VALUES(1, 1);
INSERT IGNORE INTO user_role VALUES(2, 2);
```

Simple JWT implementation workshop

Create a UsersRepository

```
package com.sd.petclinic.auth;

import java.util.Optional;

import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.stereotype.Repository;

@Repository
public interface UserRepository extends JpaRepository<User, Long> {
  Optional<User> findByUsername(String username);
}
```

Simple JWT implementation workshop

Update JwtUserDetailsService for loading data from database

```
@Override
@Transactional(readOnly = true)
public UserDetails loadUserByUsername(String username) throws UsernameNotFoundException {
    User user = userRepository.findByUsername(username).orElseThrow(
            (() -> new UsernameNotFoundException(String.format("User not found with username: %s", username))));

    Set<GrantedAuthority> grantedAuthorities = new HashSet<>();

    user.getRoles().stream().forEach(r -> {
        grantedAuthorities.add(new SimpleGrantedAuthority(r.getName().name()));
    });

    return new org.springframework.security.core.userdetails.User(user.getUsername(), user.getPassword(),
            grantedAuthorities);
}
```

Using JWT token in Angular for handling authentication

Let’s setup angular project petclinic-front with tailwind library for simpler styling

npm install -g @angular/cli@9

ng new petclinic-front --style=scss

npm i tailwindcss postcss-import postcss-loader postcss-scss @angular-builders/custom-webpack -D

npx tailwind init

Add apiUrl property to environment: apiUrl: 'http://localhost:4200/api'

Using JWT token in Angular for handling authentication

```
"build": {
      "builder": "@angular-builders/custom-webpack:browser",
      "options": {
        "customWebpackConfig": {
          "path": "./webpack.config.js"
        },
...

"serve": {
      "builder": "@angular-builders/custom-webpack:dev-server",
      "options": {
        "browserTarget": "petclinic-front:build",
        "customWebpackConfig": {
          "path": "./webpack.config.js"
        },
        "proxyConfig": "src/proxy.conf.json"
      },
...
```

```
{
 "/api/*": {
   "target": "http://localhost:8080",
   "secure": false,
   "logLevel": "debug"
 }
}
```

```
@tailwind base;
@tailwind components;
@tailwind utilities
```

```
module.exports = {
 module: {
  rules: [
   {
    test: /\.scss$/,
    loader: "postcss-loader",
    options: {
     ident: "postcss",
     syntax: "postcss-scss",
     plugins: () => [
      require("postcss-import"),
      require("tailwindcss"),
      require("autoprefixer"),
     ],
    },
   },
  ],
 },
};
```

Using JWT token in Angular for handling authentication

- Generate login component

  ng g login/login

- Add some nicely looking style for it
- Add login page to app-routing module

```
...
const routes: Routes = [
 { path: 'login', component: LoginComponent },
...
];

@NgModule({
 imports: [RouterModule.forRoot(routes)],
 exports: [RouterModule]
})
export class AppRoutingModule { }
```

```
<div id="app-login" class="flex container mx-auto items-center justify-center">
  <div class="w-full max-w-xs">
    <form class="bg-white shadow-md rounded px-8 pt-6 pb-8 mb-4">
      <div class="mb-4">
        <label class="block text-gray-700 text-sm font-bold mb-2" for="username">
          Username:
        </label>
        <input
          class="shadow appearance-none border rounded w-full py-2 px-3 text-gray-700 leading-tight focus:outline-none
focus:shadow-outline"
          id="username" type="text" placeholder="Username">

      </div>
      <div class="mb-6">
        <label class="block text-gray-700 text-sm font-bold mb-2" for="password">
          Password:
        </label>
        <input
          class="shadow appearance-none border rounded w-full py-2 px-3 text-gray-700 mb-3 leading-tight focus:outline-none
focus:shadow-outline"
          id="password" type="password" placeholder="******************">
      </div>
      <div class="flex items-center justify-between">
        <button
          class="bg-blue-500 hover:bg-blue-700 text-white font-bold py-2 px-4 rounded focus:outline-none focus:shadow-outline"
          type="submit">
          Sign In
        </button>
        <a class="inline-block align-baseline font-bold text-sm text-blue-500 hover:text-blue-800" href="#">
          Forgot Password?
        </a>
      </div>
    </form>
  </div>
</div>
```

Using JWT token in Angular for handling authentication

- Create user model
- Create role model

```
export enum Role {
    Editor = 'Editor',
    Admin = 'Admin'
}

import { Role } from './role';

export class User {
    id: number;
    username: string;
    password: string;
    roles?: Role[];
    token?: string;
}
```

Using JWT token in Angular for handling authentication

- Create authentication service

  ng g sevice auth/authentication

- Add logic to newly created service
- Add HttpModule to imports for DI

```
...
imports: [
  ...
  HttpClientModule,
  ...
 ],
...
```

```
import { Injectable } from '@angular/core';
import { HttpClient } from '@angular/common/http';
import { BehaviorSubject, Observable } from 'rxjs';
import { map } from 'rxjs/operators';
import { User } from './user';
import { environment } from 'src/environments/environment';

@Injectable({ providedIn: 'root' })
export class AuthenticationService {
  private currentUserSubject: BehaviorSubject<User>;
  public currentUser: Observable<User>;

  constructor(private http: HttpClient) {
    this.currentUserSubject = new BehaviorSubject<User>(JSON.parse(localStorage.getItem('currentUser')));
    this.currentUser = this.currentUserSubject.asObservable();
  }

  public get currentUserValue(): User {
    return this.currentUserSubject.value;
  }

  login(username: string, password: string) {
    return this.http.post<any>(`${environment.apiUrl}/authenticate`, { username, password })
      .pipe(map(user => {
        if (user && user.token) {
          localStorage.setItem('currentUser', JSON.stringify(user));
          this.currentUserSubject.next(user);
        }

        return user;
      }));
  }

  logout() {
    localStorage.removeItem('currentUser');
    this.currentUserSubject.next(null);
  }
}
```

Using JWT token in Angular for handling authentication

- Create authentication guard

  ng g guard auth/authentication

- Add logic to newly created service

```
import { Injectable } from '@angular/core';
import { CanActivate, ActivatedRouteSnapshot, RouterStateSnapshot, UrlTree, Router } from '@angular/router';
import { Observable } from 'rxjs';
import { AuthenticationService } from './authentication.service';

@Injectable({
 providedIn: 'root'
})
export class AuthenticationGuard implements CanActivate {
 constructor(
   private router: Router,
   private authenticationService: AuthenticationService
 ) { }

 canActivate(
   route: ActivatedRouteSnapshot,
   state: RouterStateSnapshot): Observable<boolean | UrlTree> | Promise<boolean | UrlTree> | boolean | UrlTree {

   const currentUser = this.authenticationService.currentUserValue;
   if (currentUser) {
     return true;
   }

   this.router.navigate(['/login'], { queryParams: { returnUrl: state.url } });
   return false;
 }
}
```

Using JWT token in Angular for handling authentication

- Generate jwt interceptor

  ng g interceptor auth/jwt

- Add logic to newly created intercepter

```
import { Injectable } from '@angular/core';
import {
 HttpRequest,
 HttpHandler,
 HttpEvent,
 HttpInterceptor
} from '@angular/common/http';
import { Observable } from 'rxjs';
import { AuthenticationService } from './authentication.service';
import { environment } from 'src/environments/environment';

@Injectable()
export class JwtInterceptor implements HttpInterceptor {

 constructor(private authenticationService: AuthenticationService) { }

 intercept(request: HttpRequest<any>, next: HttpHandler): Observable<HttpEvent<any>> {
  // Add auth header with jwt if user is logged in and request is to api url.
  const currentUser = this.authenticationService.currentUserValue;
  const isLoggedIn = currentUser && currentUser.token;
  const isApiUrl = request.url.startsWith(environment.apiUrl);

  if (isLoggedIn && isApiUrl) {
   request = request.clone({
    setHeaders: {
     Authorization: `Bearer ${currentUser.token}`
    }
   });
  }

  return next.handle(request);
 }
}
```

Using JWT token in Angular for handling authentication

- Generate error interceptor

  ng g interceptor auth/error

- Add logic to newly created intercepter

```
import { Injectable } from '@angular/core';
import { HttpRequest, HttpHandler, HttpEvent, HttpInterceptor } from
'@angular/common/http';
import { Observable, throwError } from 'rxjs';
import { catchError } from 'rxjs/operators';
import { AuthenticationService } from './authentication.service';

@Injectable()
export class ErrorInterceptor implements HttpInterceptor {
 constructor(private authenticationService: AuthenticationService) { }

 intercept(request: HttpRequest<any>, next: HttpHandler):
Observable<HttpEvent<any>> {
  return next.handle(request).pipe(catchError(err => {
   if ([401, 403].indexOf(err.status) !== -1) {
    // Auto logout if 401 Unauthorized or 403 Forbidden response returned from
api.
     this.authenticationService.logout();
     location.reload(true);
   }

   const error = err.error.message || err.statusText;
   return throwError(error);
  }));
 }
}
```

Using JWT token in Angular for handling authentication

- Add interceptors to app.module.ts

```
...
providers: [
  { provide: HTTP_INTERCEPTORS, useClass: JwtInterceptor,
multi: true },
  { provide: HTTP_INTERCEPTORS, useClass: ErrorInterceptor,
multi: true },
 ],
...
```

Using JWT token in Angular for handling authentication

- Generate component for dashboard

  ng g component dashboard/dashboard

- Add component to app-routing.module.ts

```
const routes: Routes = [
 { path: 'login', component: LoginComponent },
 {
   path: '',
   component: DashboardComponent,
   canActivate: [AuthenticationGuard],
 },
...
```

Using JWT token in Angular for handling authentication

- Add logic to login component so it would be possible to get token from api

1

```
import { Component, OnInit } from '@angular/core';
import { Router, ActivatedRoute } from '@angular/router';
import { FormBuilder, FormGroup, Validators } from '@angular/forms';
import { AuthenticationService } from '../auth/authentication.service';
import { first } from 'rxjs/operators';

@Component({
 selector: 'app-login',
 templateUrl: './login.component.html',
 styleUrls: ['./login.component.scss']
})
export class LoginComponent implements OnInit {
 loginForm: FormGroup;
 loading = false;
 submitted = false;
 returnUrl: string;
 error = '';


 constructor(private formBuilder: FormBuilder,
          private route: ActivatedRoute,
          private router: Router,
          private authenticationService: AuthenticationService) {
  // Redirect to home if already logged in
  if (this.authenticationService.currentUserValue) {
   this.router.navigate(['/']);
  }
 }

 ngOnInit(): void {
  this.loginForm = this.formBuilder.group({
   username: ['', Validators.required],
   password: ['', Validators.required]
  });

...
```

2

```
 ...
 // Get return url from route parameters or default to '/'
  this.returnUrl = this.route.snapshot.queryParams.returnUrl || '/';
 }

 get f() { return this.loginForm.controls; }

 onSubmit() {
  this.submitted = true;

  if (this.loginForm.invalid) {
   return;
  }

  this.loading = true;
  this.authenticationService.login(this.f.username.value, this.f.password.value)
   .pipe(first())
   .subscribe(
    data => {
     this.router.navigate([this.returnUrl]);
    },
    error => {
     this.error = error;
     this.loading = false;
    });
 }
}
```

Using JWT token in Angular for handling authentication

- Add reactive form in app.module.ts
- Add form controls in login template

```
imports: [
  BrowserModule,
  AppRoutingModule,
  ReactiveFormsModule,
  HttpClientModule,
],
```

```
...
<form class="bg-white shadow-md rounded px-8 pt-6 pb-8 mb-4" [formGroup]="loginForm"
(ngSubmit)="onSubmit()">
…
<input
        class="shadow appearance-none border rounded w-full py-2 px-3 text-gray-700
leading-tight focus:outline-none focus:shadow-outline"
        id="username" type="text" placeholder="Username"
formControlName="username">
...
```

Integrate data fetch from api

- Generate owners service

  ng g service dashboard/owners

- Add logic to service

```
export class Owner {
  id: number;
  firstName: string;
  lastName: string;
  address: string;
  city: string;
  telephone: string;
}
```

```
import { Injectable } from '@angular/core';
import { HttpClient } from '@angular/common/http';
import { BehaviorSubject, Observable } from 'rxjs';
import { Owner } from './owner';
import { environment } from 'src/environments/environment';;

@Injectable({
 providedIn: 'root'
})
export class OwnersService {
 constructor(private http: HttpClient) { }

 public getOwners(): Observable<Owner[]> {
  return this.http.get<Owner[]>(`${environment.apiUrl}/owners`);
 }
}
```

Integrate data fetch from api

- Add logic to dashboard component

1

```
import { Component, OnInit } from '@angular/core';
import { OwnersService } from './owners.service';
import { Owner } from './owner';

@Component({
 selector: 'app-dashboard',
 templateUrl: './dashboard.component.html',
 styleUrls: ['./dashboard.component.scss']
})
export class DashboardComponent implements OnInit {
 owners: Owner[];
 isLoading: boolean;

 constructor(private ownersService: OwnersService) { }

 ngOnInit(): void {
  this.getOwners();
 }

 ...
```

2

```
getOwners() {
  this.isLoading = true;
  this.ownersService.getOwners()
   .subscribe(
    response => this.handleResponse(response),
    error => this.handleError(error));
 }

 handleResponse(response: Owner[]): void {
  this.isLoading = false;
  this.owners = response;
 }

 handleError(error: any): void {
  this.isLoading = false;
  console.error(error);
 }
}
```

Integrate data fetch from api

- Add data binding in dashboard template

```
<div id="app-dashboard" class="flex container mx-auto items-center justify-center bg
shadow-md rounded">
  <table class="table-auto m-8">
    <thead>
      <tr>
        <th class="px-4 py-2">First name</th>
        <th class="px-4 py-2">Last name</th>
      </tr>
    </thead>
    <tbody>
      <tr *ngFor="let owner of owners">
        <td class="border px-4 py-2">{{owner.firstName}}</td>
        <td class="border px-4 py-2">{{owner.lastName}}</td>
      </tr>
    </tbody>
  </table>
</div>
```

Exercise

1. Extend owners service so it would be possible:
   - get owner by id
   - create owner by id
   - update owner by id
   - delete owner by id
2. Create owner form component for updating or creating owner and integrate it with dashboard