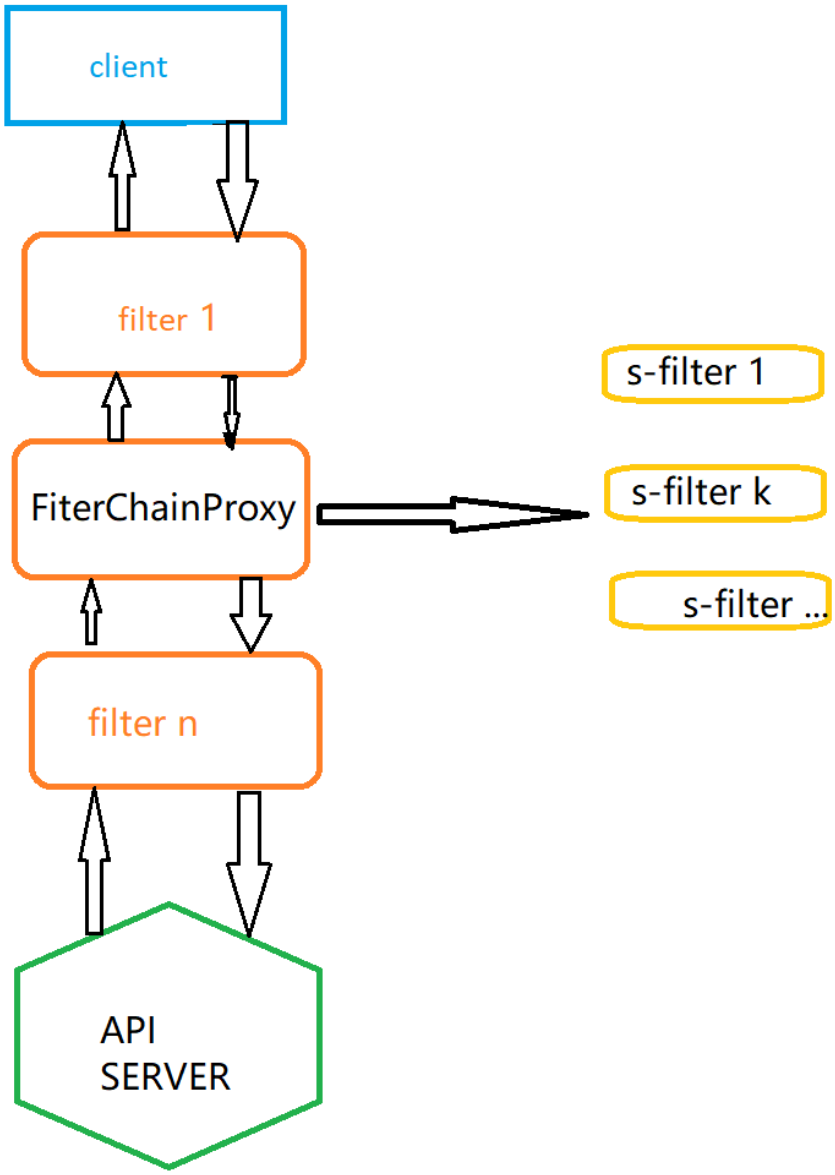
Spring Security + JWT

Authentication work flow



in `Servlet` filters chanis，Spring Security add `FilterChainProxy`. `FilterChainProxy`：

```
@Override
public void doFilter(ServletRequest request, ServletResponse response,
                     FilterChain chain) throws IOException, ServletException
{
    ...

    // get Spring Security filters
    List<Filter> filters = getFilters(request);
    // Spring Security
    VirtualFilterChain vfc = new VirtualFilterChain(fwRequest, chain,
filters);
```

```
    vfc.doFilter(request, response);

    ...
}

```

the filters：

`UsernamePasswordAuthenticationFilter` for Authentication

`FilterSecurityInterceptor` for Authorization.

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

```

```
@EnableWebSecurity
public class SecurityConfig extends WebSecurityConfigurerAdapter {
}

```

> `SecurityContextHolder` manager it and you can use it everywhere：

```
Authentication authentication =
SecurityContextHolder.getContext().getAuthentication();

```

the path：`SecurityContextHolder` 👉 `SecurityContext` 👉 `Authentication`。

📝 `Authentication`：save the user info

📝 `SeucirtyContext`：for get `Authentication`

📝 `SecurityContextHolder`：for get `SecurityContext`

the relationship：

`Authentication`：

📝 `Principal`：befor authentication is user name, after is user object.

📝 `Credentials`：usually is password

📝 `Authorities`：permissions

sql the user info from database👉check password👉true save it to SecurityContext👉SecurityContext contains authentication means user login sucess

Spring Security：

```
Authentication authentication = new
UsernamePasswordAuthenticationToken(userName, password, permissions);
SecurityContextHolder.getContext().setAuthentication(authentication);

```

the path：`Authentication` 👉 `SecurityContext` 👉 `SecurityContextHolder`。

```
// check user
if (!userService.login(userName, password)) {
    return "User or Password not match";
}
//
Authentication authentication = new
UsernamePasswordAuthenticationToken(userName, password, permissions);
SecurityContextHolder.getContext().setAuthentication(authentication);

```

Spring security using the `AuthenticationManager` for Authentication

```
@EnableWebSecurity
public class SecurityConfig extends WebSecurityConfigurerAdapter {
    @Bean
    @Override
    protected AuthenticationManager authenticationManager() throws Exception
{
        return super.authenticationManager();
    }
}


```

```
@RestController
@RequestMapping("/API")
public class LoginController {
    @Autowired
    private AuthenticationManager authenticationManager;

    @PostMapping("/login")
    public String login(@RequestBody LoginParam param) {
        //create token
        Authentication token = new
UsernamePasswordAuthenticationToken(param.getUsername(),
param.getPassword());
        // AuthenticationManagerverify the token
        Authentication authentication =
authenticationManager.authenticate(token);
        // save it to context

SecurityContextHolder.getContext().setAuthentication(authentication);
        return "success";
    }
}

```

Session `HttpSessionSecurityContextRepository.SPRING_SECURITY_CONTEXT_KEY` you can get `Authentication`:

```
Authentication = (Authentication)session.getAttribute(HttpSessionSecurityContextRepository.SPRING_SECURITY_CONTEXT_KEY)

```

:

```
@GetMapping("/logout")
public String logout() {
    SecurityContextHolder.clearContext();
    return "logout";
}

```

JWT

disable the `session`:

```
//
http.sessionManagement().sessionCreationPolicy(SessionCreationPolicy.STATELESS);

```

create `token`:

```
@Autowired
private UserService userService;

@PostMapping("/login")
public UserVO login(@RequestBody @Validated LoginParam user) {
    // call userService
    return userService.login(user);
}

```

```
public UserVO login(LoginParam param) {
    //
    UserEntity user = userMapper.selectByUsername(param.getUsername());
    //
    if (user == null || !passwordEncoder.matches(param.getPassword(), user.getPassword())) {
        throw new ApiException("Error");
    }

    //
```

```
    UserVO userVO = new UserVO();
    userVO.setId(user.getId())
        .setUsername(user.getUsername())
        //
        .setToken(jwtManager.generate(user.getUsername()));
    return userVO;
}

```

add filter：

```
@Component
public class LoginFilter extends OncePerRequestFilter {
    @Autowired
    private JwtManager jwtManager;
    @Autowired
    private UserServiceImpl userService;

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

        Claims claims =
jwtManager.parse(request.getHeader("Authorization"));
        if (claims != null) {

            String username = claims.getSubject();

            UserDetails user = userService.loadUserByUsername(username);

            Authentication authentication = new
UsernamePasswordAuthenticationToken(user, user.getPassword(),
user.getAuthorities());


 SecurityContextHolder.getContext().setAuthentication(authentication);
        }
        chain.doFilter(request, response);
    }
}

```

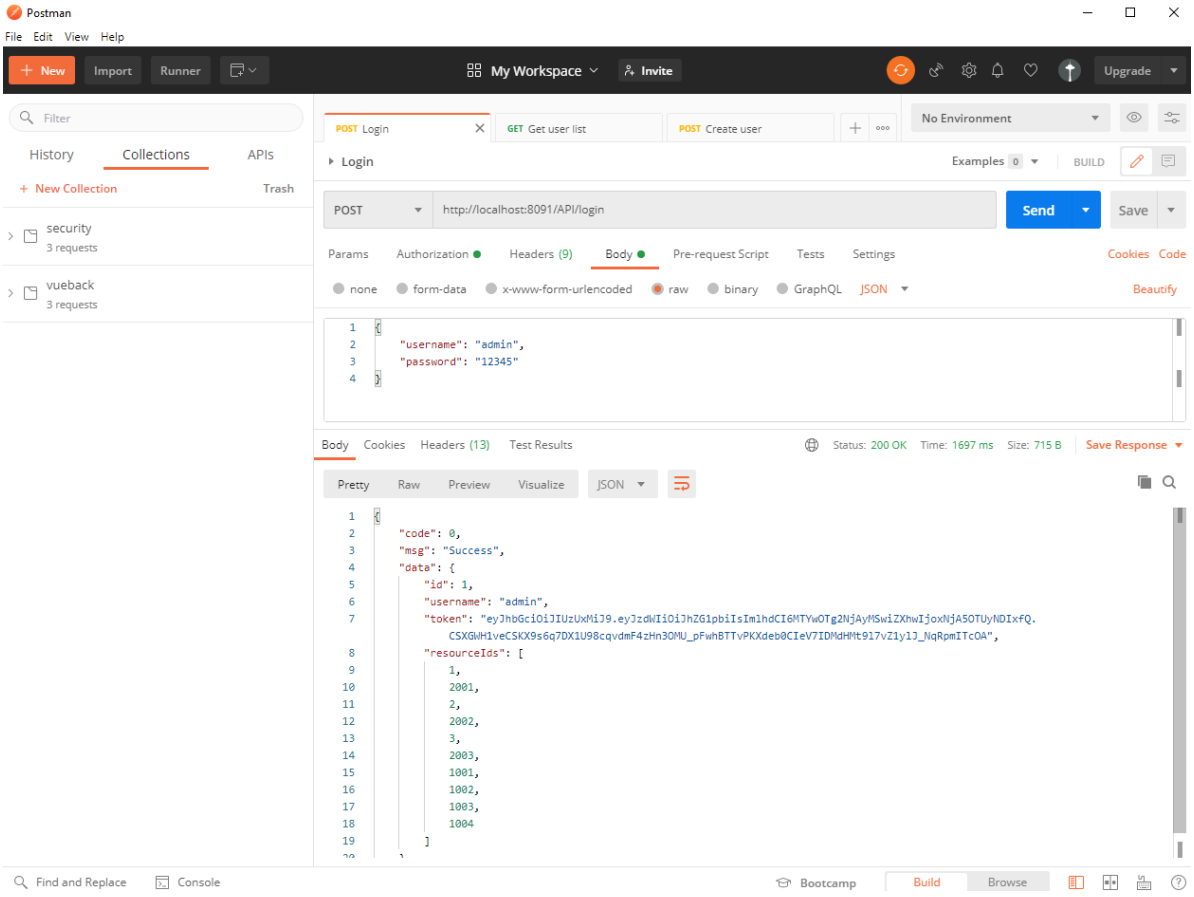
replace the filter in securityConfig：

```

http.addFilterBefore(loginFilter,
UsernamePasswordAuthenticationFilter.class);

```

login：



get user list