Міністерство освіти і науки України
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»
Факультет інформатики
Кафедра інформатики

# Аналіз методів авторизації та аутентифікації в бізнес застосування на базі платформи Spring

**Текстова частина до курсової роботи**

**за спеціальністю «Інженерія Програмного Забезпечення»- 121**

Керівник курсової роботи
к.т.н., ст. в. ___Шабінська М. О.
*(прізвище та ініціали)*
_____
*(підпис)*
“____” _____ 2020 р.
Виконав студент Ляш Д. В.
*(прізвище та ініціали)*
_____
*(підпис)*
“____” _____ 2020 р.

Київ 2020

Міністерство освіти і науки України
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»

Кафедра інформатики факультету інформатики

ЗАТВЕРДЖУЮ

Зав. кафедри інформатики,
к.ф.-м. н. С. С. Гороховський

_____

*(підпис)*

“____” _____ 2020 р.

ІНДИВІДУАЛЬНЕ ЗАВДАННЯ
на курсову роботу

Студента Ляша Данила Вячеславовича факультету інформатики 3 курсу
ТЕМА Аналіз методів аутентифікації та авторизації у бізнес застосування на базі платформи Spring

Вихідні дані:

Зміст ТЧ до курсової роботи:

Календарний план

Вступ

Розділ 1. Spring та Spring Security

Розділ 2. Авторизація та аутентифікація

Розділ 3. Реалізації та аналіз методів авторизації та аутентифікації платформи Spring

Висновки

Список використаної літератури

Дата видачі “____” _____ 2020 р. Керівник _____

*(підпис)*

Завдання отримав _____

*(підпис)*

**Тема:** Аналіз методів аутентифікації та авторизації у бізнес застосування на базі платформи Spring

**Календарний план виконання роботи:**

| № | Назва етапу | Термін виконання | Примітка |
|---|---|---|---|
| 1. | Отримання теми курсової роботи | 27.10.2019 | |
| 2. | Пошук тематичної літератури | 15.11. 2019 | |
| 3. | Ознайомлення з літературою | 1.12. 2019 | |
| 4. | Вивчення аналогів | 20.12. 2019 | |
| 5. | Планування веб-сайту | 05.01. 2020 | |
| 6. | Ознайомлення з платформою Spring | 12.01. 2020 | |
| 7. | Ознайомлення з модулем Spring Security | 23.01. 2020 | |
| 8. | Реалізація проекту Basic authentication | 15.02. 2020 | |
| 9. | Реалізація проекту Token authentication | 01.03. 2020 | |
| 10. | Реалізація проекту OAuth2 authentication | 27.03. 2020 | |
| 11. | Реалізація проекту X.509 authentication | 08.04. 2020 | |
| 12. | Оформлення сторінок | 12.04. 2020 | |
| 13. | Написання текстової частини | 21.04. 2020 | |
| 14. | Перегляд змісту роботи керівником | 07.05. 2020 | |
| 15. | Внесення змін до курсової роботи відповідно до зауважень наукового керівника | 09.05. 2020 | |
| 16 | Створення презентації | 10.05. 2020 | |

Студент Ляш Д. В.

Керівник Шабінська М. О.

“____” _____

# Зміст

# Анотація

У роботі розглянута задача аналізу методів авторизації та аутентифікації у бізнес застосування на базі платформи Spring.

В першому розділі розглянуті теоретичні відомості про платформу Spring та її модуль який відповідає саме за авторизацію та аутентифікацію Spring Security.

В другому розділі розглянуті та роз’яснені теоретичні відомості про те що таке авторизація та аутентифікація. Дається їх визначення послідовність, також пояснюється як вони відбуваються у Spring Security.

В третьому розділі розглядаються різні методи аутентифікації: Basic, Digest, Token, Certificate, OAuth2. Визначаються їх можливості, переваги та недоліки. Також розглянуті способи реалізації авторизації у Spring Security. У кінці цього розділу визначаються результати аналізу цих методів авторизації та аутентифікації.

# Вступ

**Актуальність:**

В сучасному світі веб-технологій проблема аутентифікації та авторизації, з вірною перевіркою користувача за наданими ідентифікаторами та вірним виділенням відповідних ролей з можливостями йому дуже актуальна [1].

Зараз ми майже кожен день можемо побачити новини про викрадення та використанням особистостей у інтернеті зловмисниками . Величезна кількість паролів викрадається із року в рік [1].

Також у користувачів можуть виникати труднощі або просто може бути не зручно проходити аутентифікацію у більшості випадків. Це відбувається через те, що користувачам потрібно запам’ятовувати досить велику кількість паролів щоб їх дані залишалися у безпеці. [1]

Тому аналіз методів авторизації та аутентифікації це дуже актуально для застосувань, користувачів яких хочуть захисти і допомогти зробити це легше та зручніше.

**Мета та завдання курсової роботи:**

**Мета**: Показати можливості та реалізації авторизації та аутентифікації фреймворку Spring.

**Завдання:** Пояснити та дати визначення термінам авторизації та аутентифікації, описати процес цих методів у Spring Security, зробити приклади реалізації різних методів та проаналізувати їх, проаналізувати отримані результати.

# Розділ 1 Spring та Spring Security

## 1.1 Spring

Що ж взагалі таке платформа Spring? Spring – це фреймворк або програмний каркас для платформи Java з підтримкою інверсії керування. [3]

Що тут мається на увазі під інверсією керування? Стандартна модель керування полягає в тому що застосування викликає методи бібліотек, у випадку IoC (Inversion of Control або інверсія управління) фреймворк сам надає точки розширень, через які він викликає методи користувацького коду [6].

Платформа Spring може бути використана будь-яким додатком Java, платформою Java EE і окремий випадок .NET, форк якого називається Spring.NET. [3]

Spring як фреймворк складається з менших фреймворків або модулів, що надають якісь послуги [3], ось деякі з них

Контейнер Інверсії управління: Конфігурація компонентів додатків і керування життєвим циклом об'єктів. [3]

Доступ до даних: робота з СКБД з використанням JDBC та ORM засобів.[3]

MVC: каркас на основі HTTP сервлетів для розробки веб-застосувань і підтримкою архітектури model-view-controller. [3]

Керування транзакціями: об’єднує кілька різноманітних API керування транзакціями і координує операції об’єктів Java. [3]

**Spring security**: інструментарій процесів аутентифікації і авторизації, що підтримує ряд стандартів протоколів, інструментів, практик через підпроект проект **Spring Security** [3], про який буде йти мова у моїй курсовій роботі.

## 1.2 Spring Security

Взагалі, Spring Security це минулий Acegi Security, започаткований Беном Алексом (Ben Alex) у 2003 році. Spring Security це Java/JavaEE framework, який надає можливість розробнику накладати обмеження на додатки, основані на базі Spring, він працює з авторизацією та аутентифікацією. [2]

Велика перевага цього фреймворку у тому, що він надає великі можливості у конфігуруванні та імплементації для розробника, хоча й дозволяється використовувати опції за замовчуванням. [7]

# Розділ 2 Авторизація та Аутентифікація

## 2.1 Аутентифікація

**Аутентифікація** – це підтвердження належності суб'єкта до якоїсь сутності у відповідність до наданого їм ідентифікатора. [8]

Для того, щоб підтвердити аутентифікацію, тобто так би мовити дати визначити себе, користувач надає ідентифікатор, який у більшості випадків собою являє username або e-mail, також можливі інші варіанти ідентифікатора (номер телефону, ІПН і так далі…) . Після ідентифікатора, користувач має надати різні данні, наприклад:

1. Пароль [8] (PIN-код, кодоване слово, графічний ключ тощо);
2. Біометричні дані [8] (відбиток пальця, «скан» обличчя або сітківки очей тощо);
3. Інший пристрій [8] (USB, картка доступу тощо);

У випадку Spring це можуть бути дуже різноманітні варіанти.

На наступному рисунку (рисунок 2.1) показано вікно для аутентифікації з мого проекту Basic authentication example.

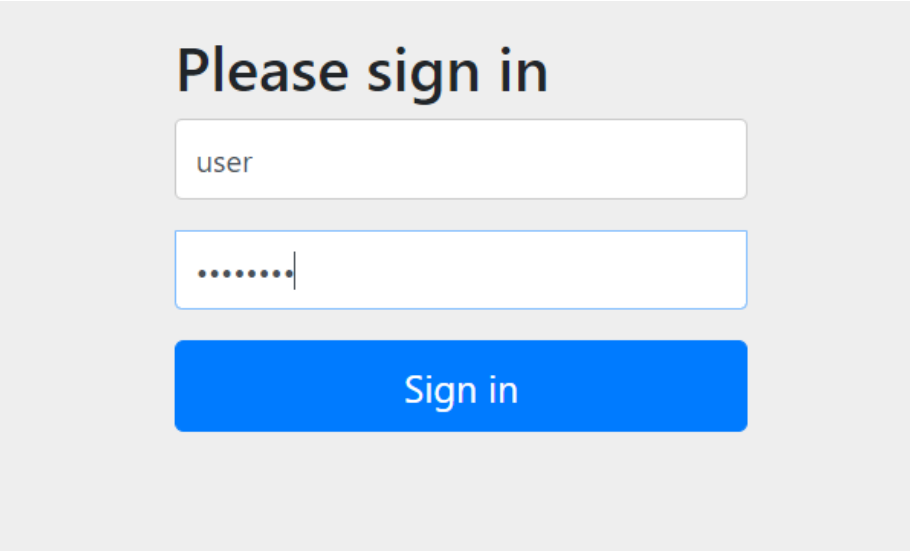


*Рисунок 2.1 - вікно аутентифікації, автоматично запропоноване*

## 2.2 Авторизація

**Авторизація** – це надання доступу до ресурсів, за визначеною аутентифікацією, тобто це останній етап перед користуванням ресурсами. [8]

Якщо казати простими словами, то через авторизацію ми можемо зрозуміти ким є користувач, чи то адміністратором, чи то звичайним користувачем.

Відповідно ми обмежуємо нашого суб’єкта в інформації, обмеження якої якимось чином визначено. Тобто адміністратор має права видаляти інших користувачів, але зазвичай користувачі не видаляють інших користувачів, окрім себе, і ми розуміємо, що звичайному користувачеві не треба давати доступ до прав адміністратора, бо він випадково або навмисно може нашкодити іншим користувачам або бізнесу.

## 2.3 Загальне

Отже, після визначення авторизації та аутентифікації можна зрозуміти послідовність цих дій [8]

Як показано на рисунку (рисунок 2.2), користувач спочатку ідентифікує себе, після цього дає інформацію для аутентифікації.. Після ідентифікації та аутентифікації система повинна авторизувати користувача та надати якісь права або можливості. У кінці можна зрозуміти, що терміни **аутентифікація** та **авторизація** мають послідовний характер.

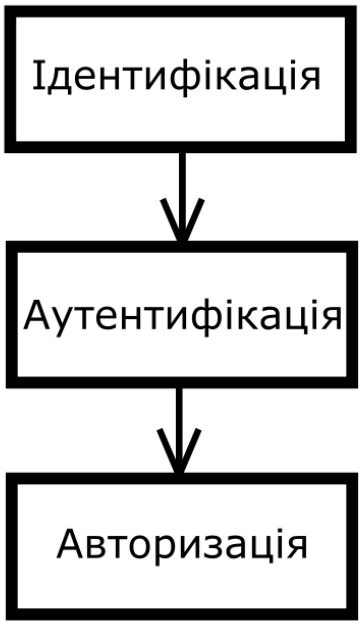


*Рисунок 2.2 послідовність входу у систему*

## 2.4 Загальні відомості Spring Security authorization and authentication

У Spring Security свій механізм для авторизації та аутентифікації користувачів. Для розуміння цього механізму треба надати зрозуміти деякі речі:

- Ключовим об’єктом вважається **SecurityContextHolder**, у якому зберігається інформація про контекст безпеки застосунку із інформацією про **Principal**, тобто користувача, що працює з додатком. [6]
- Система Spring Security використовує об’єкт **Authentication** щоб показати інформацію про нашого користувача. Якщо немає потреби, то створювати об’єкт **Authentication** не потрібно, але бувають випадки потреби звернутися до цього об’єкта [6], ось приклад метода для отримання **Principal** з контексту:

```
public Object getPrincipal(){
   return SecurityContextHolder.
                  getContext().getAuthentication().getPrincipal();
}
```

У більшості випадків нам буде виданий звичайний **UserDetails** з якого можна отримати потрібну інформацію. [6]

- **UserDetails** це так би мовити головний або центровий об’єкт Spring Security, який представляє нашого авторизованого суб’єкта з відповідними рисами відповідно до конфігурації застосунків. Для цього об’єкта існує інтерфейс **UserDetailsService** для єдиного метода loadUserByUsername(String username) який поверне **UserDetails** для вашої реалізації (у більшості випадків витяг з DAO) або запропонованої. У загальному можна сказати, що UserDetails відображає аутентифікацію якогось користувача. [6]

- Що ж до авторизації, то у випадку Spring Security краще усього використовувати ролі користувача, які допомагають нам визначити, до якої інформації його можна допускати [9]. Наприклад в нас є новий зареєстрований звичайний користувач, у більшості випадків йому видадуть роль ROLE_USER, через яку можна зрозуміти що суб'єкт – звичайний користувач і до якого ресурсу йому можна дати доступ до ресурсів [6]. У випадку Spring Security важливою частиною є **GrantedAuthorities**, що відображає роль, отримати їх можна з нашого **UserDetails** через метод getAuthorities(). [6]

Отже, після розуміння ключових сутностей авторизації та аутентифікації У Spring Security треба дати коротку відомість цього процесу:

1. Заданим чином отримується ім’я користувача та пароль та об’єднуються у екземпляр **UsernamePasswordAuthenticationToken** [6]
2. Токен передається екземпляру **AuthenticatioManager** для перевірки, і якщо аутентифікація успішна, то повертає заповний **Authentication**. [6]
3. Контекст безпеки визначається через цей повернутий **Authentication** через метод setAuthentication() через ключовий об’єкт Security **SecurityContexHolder.** [6]
4. За авторизацію відповідає інтерфейс **AccessDecisionManager**, в якому є метод, що вирішить для об’єкта **Authentication** який дає запит на доступ за допомогою атрибутів метаданих безпеки, які можна застосувати до нього. [6]

# Розділ 3 Реалізації та аналіз методів авторизації та аутентифікації платформи Spring

## 3.1 Загальне

У цьому розділі я аналізую, описую реалізації методів, більшість видів яких можна побачити у проектній частині курсової робити.

Для проектів я використовую REST контролери для відображення результатів авторизації та аутентифікації.

Що ж таке REST? REST - Representational State Transfer, що перекладається як передача репрезентативного стану.[11]

З цією архітектурою сервер не повинен зберігати дані про стан операцій [11]. Тоді треба сказати, що у більшості випадків, при успішній авторизації клієнт має зберігати інформацію про себе (у сессії або токенах) і відправляти її на сервер у Header для можливості надання доступу до якогось ресурсу.

Також треба сказати, що більшість конфігурацій я виконував за допомогою коду Java, хоча також підтримується конфігурація через XML.[2]

## 3.2 Basic authentication

Цей спосіб є найбанальнішим та найпростішим із усіх реалізацій, його ім’я каже це за себе basic – базовий, елементарний, найпростіший у перекладі.

Basic authentication являє собою надання користувачем логіна та паролю до REST сервіса [11]. У такого методу є один великій мінус, який робить його неможливим до застосування у бізнес застосунках через велику можливість його розшифрувати. Дані аутентифікації передаються по мережі як незашифрований текст закодований простішим Base64 (група схожіх binary-to-text encoding схем, які представляють двійкові дані в ASCII-форматі методом перекладу в radix-64 [12]). Якщо все ж таки використовувати такий підхід, то дані треба передавати тільки через протокол https [11]

Таким методом ми конфігуруємо WebSecurity, розширюючи WebSecurityConfigurerAdapter:

```
@Override
protected void configure(HttpSecurity http) throws Exception {

   http
        .httpBasic().authenticationEntryPoint(basicAuthenticationEntryPoint)
        .and()
        .authorizeRequests()
        .antMatchers(HttpMethod.GET, "/api/userinfo").hasRole("USER")
        .antMatchers(HttpMethod.GET, "/api/admininfo").hasRole("ADMIN")
        .and()
        .formLogin();
}
```

У цьому методі я конфігурую відображення інформації авторизованих користувачів для GET методів по URL /api/userinfo та /api/admininfo за допомогою hasRole, який допомагає системі зрозуміти хто є користувач за ролю: чи то ADMIN чи то USER [2].

authenticationEntryPoint(basicAuthenticationEntryPoint) відповідає за надання відповіді, якщо аутентифікація не вдалась [2]. formLogin() дозволяє використовувати для аутентифікації форму [2] (рис. 1).

Наступна конфігурація показує звідки система буде виокремлювати зареєстрованих користувачів та виділяти їх можливості

```
@Override
protected void configure(AuthenticationManagerBuilder
authenticationManagerBuilder) throws Exception {
   authenticationManagerBuilder.userDetailsService(userDetailsService)
                                .passwordEncoder(passwordEncoder());
}
```

Тут за прописаним userDetailService система розуміє де співставляти користувача, який намагається увійти. Також, я використовую passwordEncoder, для того щоб зберігати паролі хешованими.[2]

Для кодування використовую @Bean

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

BCryptPasswordEncoder - Реалізація PasswordEncoder, який використовує функцію сильного хешування BCrypt [2].

Далі у коді показані методи описані у додатку, для яких вище сконфігуровано доступ

```
@RestController
@CrossOrigin
@RequestMapping("/api")
public class InfoController {

   @GetMapping(path = "userinfo")
   public ResponseEntity<String> getInfo(){
      return new ResponseEntity<>("info for user", HttpStatus.OK);
   }

   @GetMapping(path = "admininfo")
   public ResponseEntity<String> getInfoA(){
      return new ResponseEntity<>("info for admin", HttpStatus.OK);
   }
}
```

Отже, якщо користувач входить до застосування і система визначає його роль як ROLE_USER, то він може витягнути “info for user”, але не може витягнути “info for admin”, бо система не визначила його як адміністратора. Якщо до системи увійде адміністратор, то для нього усе навпаки: “info for admin” він бачить, а от “info for user” ні.

На даному рисунку (рисунок. 3.1) зображено, що звичайний користувач не може отримати інформацію для адміністраторів. Прийшов статус код 403, який означає Forbidden, тобто заборонено [15].

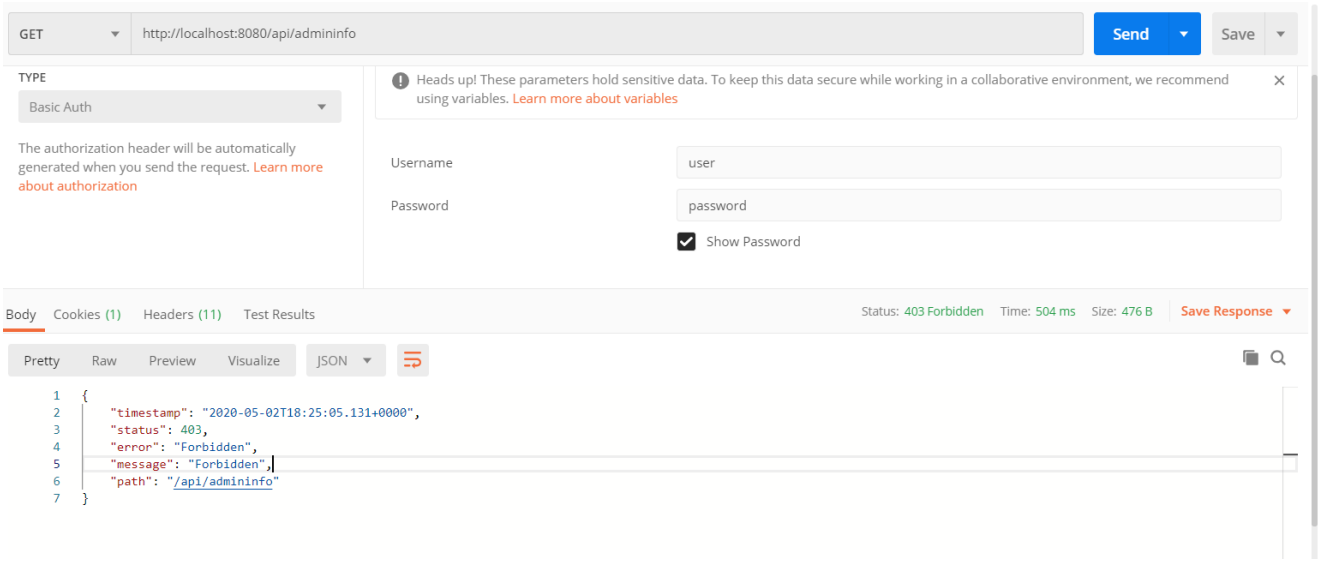


*Рисунок 3.1 – отриманий статус 403 у відповідь*

Далі, на іншому рисунку (рисунок 3.2) можна побачити, що доступаючись до дозволеного URL, користувач отримує запитану інформацію.

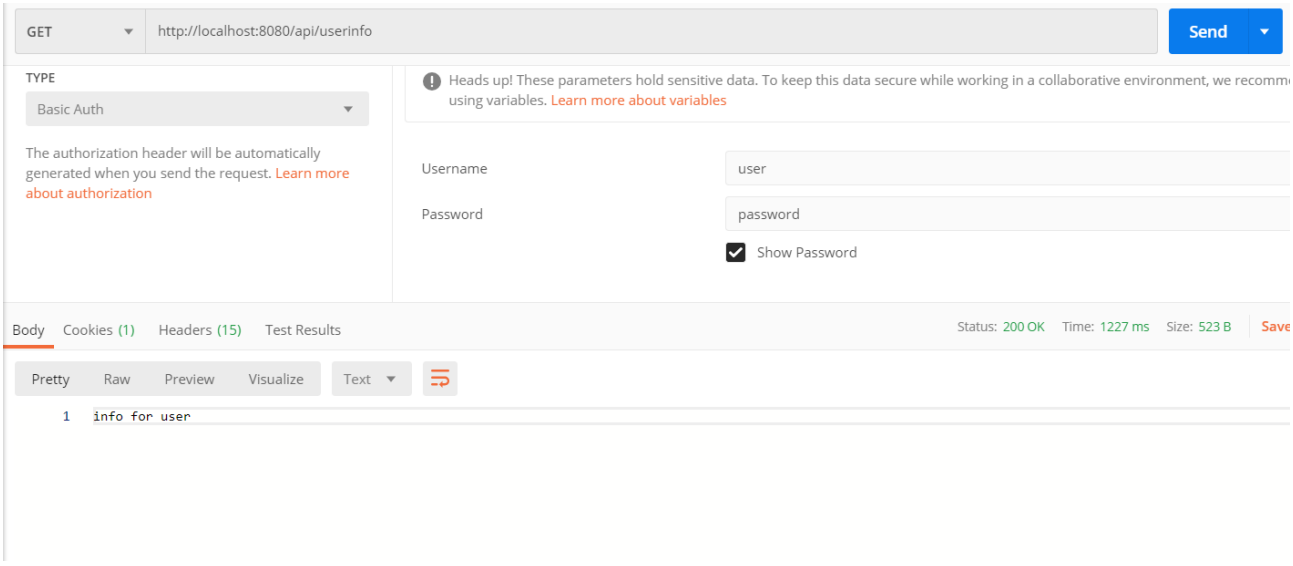


*Рисунок 3.2 – отримана потрібна інформація*

## 3.3 Digest authentication

Схожий на метод Basic authentication, але в якості передачі логіна та пароля використано не Base64, а MD5 [11].

Але зараз називати MD5 надійним також не можна, зараз у мережі можна знайти бази даних за якими можна декодити хешування [13], прикладом може бути сайт https://www.md5online.org/md5-decrypt.html, на наступному рисунку (рисунок 3.3) я закодував складний рядок, з неповторюваними символами.


*Рисунок 3.3 – закодований MD5*

Далі я спробував його декодувати на іншому сайті https://md5hashing.net/hash/md5/6f0693f3a19372a15e5a561d25bcdc40 і результат можна побачити на наступному рисунку (рисунок 3.4)

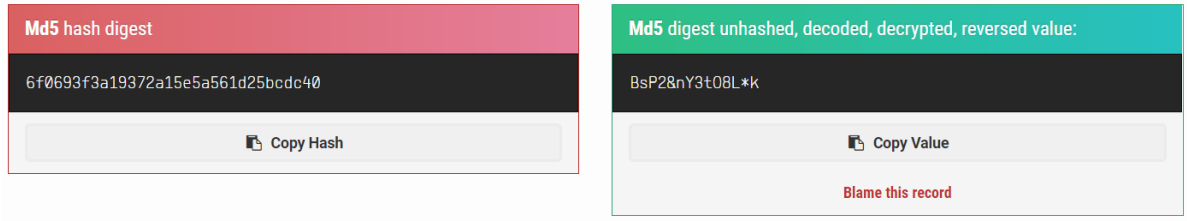


*Рисунок 3.4 – розшифрований MD5*

Також MD5 властиві таки речі як колізії (отримання однакових гешів, для різних вихідних рядків) [14]

Що до серверної частини, вона остається майже незмінною. Треба змінювати клієнт, щоб він зміг відправити закодовані дані на сервер[11], зміна клієнта буде залежити від системи його написання.

## 3.4 Token authentication

Такий спосіб аутентифікації є більш реальним для використання у справжньому застосуванні. Аутентифікація через токени може гарно поєднувати у собі різні типи аутентифікації, тобто можлива комбінація JWT + OAuth [16], про який піде мова у іншому підрозділі . Спосіб аутентифікації через токени полягає у тому, що користувач виконує аутентифікацію та отримує токен для використання ресурсів [11]. Тут потрібно сказати, що для зв'язку з сервісом, який буде видавати користувачу токени потрібно обов'язково використовувати більш безпечне https з'єднання, щоб інформація не була перехоплена[11].

В токен закладається інформація потрібна для клієнтського додатку: термін придатності токена, ім'я, ролі тощо [11]. Цей токен кидається через header кожного запиту що відправляється на сервер, там він може бути розкодований та визначити до яких ресурсів може доступатися наш користувач (як і зазвичай за роллю, яка міститься у закодованому токені) і тільки ваш сервер повинен мати можливість розшифровувати токен.

Для написання прикладу сервера, який використовує токен я обрав JWT (JSON Web Token). Розглянемо конфігурацію нашого застосунку.

```
@Override
public void configure(WebSecurity webSecurity) {
   webSecurity
         .ignoring()
         .antMatchers("/api/login");
}
```

У цьому методі я даю через WebSecurity зрозуміти, що за URL /api/login система повинна пропускати усіх користувачів [2]. Очевидно, що до аутентифікації повинен мати доступ кожен.

Далі конфігурація HttpSecurity

```
@Override
protected void configure(HttpSecurity http) throws Exception {
  http
      .httpBasic().disable()
      .csrf().disable()
      .exceptionHandling().authenticationEntryPoint(unauthorizedHandler)
      .and()
      .sessionManagement()
      .sessionCreationPolicy(SessionCreationPolicy.STATELESS)
      .and()
      .authorizeRequests()
      .anyRequest().authenticated()
      .and()
      .anonymous()
      .and()
      .apply(new JwtConfigurer(tokenProvider));


}
```

Тут так само як і в Basic authentication authenticationEntryPoint відповідає за обробку exception неправильної аутентифікації. [2]

Далі, sessionManagement() відповідає за те, що ми тепер можемо конфігурувати сесії, а у sessionCreationPolicy(SessionCreationPolicy.*STATELESS*)
ми визначаємо, що в нас Spring Security ніколи не створить HttpSession, і він ніколи не використовуватиме його для отримання SecurityContext. [2]

У .authorizeRequests().anyRequest().authenticated() конфігурація означає те, що будь-який запит має бути аутентифікованим (користувач отримав токен і його термін дії не минув). [2]

.anonymous() дозволяє конфігурацію неаутентифіікованого користувача.

І у кінці .apply(new TokenConfigurer(tokenProvider)) показуємо що ми використовуємо JWT

Так виглядає розширений клас TokenConfigurer:

```
public class TokenConfigurer extends
SecurityConfigurerAdapter<DefaultSecurityFilterChain, HttpSecurity> {

   private final TokenProvider tokenProvider;

   @Autowired
   public TokenConfigurer(TokenProvider tokenProvider) {
      this.tokenProvider = tokenProvider;
   }

   @Override
   public void configure(HttpSecurity httpSecurity) {
      httpSecurity.addFilterBefore(new
TokenAuthenticationFilter(tokenProvider),
            UsernamePasswordAuthenticationFilter.class);
   }
}
```

Тут можна побачити конфігурацію фільтра JWT токена у override методі configure, там використано свій TokenAuthenticationFilter, в якому можна побачити систему для фільтрування запитів тільки з виданими токенами, виглядає це так:

```
try {
   String jwtToken = getToken(httpServletRequest);
   if (StringUtils.hasText(jwtToken) &&
tokenProvider.validateToken(jwtToken)) {
      authenticationValidation(jwtToken);
   }
} catch (Exception ex) {
   System.err.println("Could not set authentication in context: "+ex);
}
filterChain.doFilter(httpServletRequest, httpServletResponse);
```

Тут витягнуто токен з header та спочатку tokenProvider валідує токен порівнюючи його із заданим Secret ключем, закладеним усередину, і якщо все гаразд повертає true. Далі перевіряємо чи пройшла аутентифікація у методі authenticationValidation.

Також, повертаючись до конфігурації, я по аналогії з Basic authentication використовую свій UserDetailsService з BCryptPasswordEncoder.

Перейдемо до контролерів.LoginController має єдиний метод:

```
@PostMapping("/login")
public ResponseEntity<String> login(@RequestBody LoginRequest
loginRequest) {
   User loggedUser =
userService.getUser(loginRequest.getUsername(),loginRequest.getPassword());

   return new ResponseEntity<>(tokenService.tokenProvider(loggedUser),
HttpStatus.OK);
}
```

Тут ми через dto (Data Transfer Object) LoginRequest отримуємо login та password користувача і перевіряємо, чи є він зареєстрованим. Якщо так, то повертаємо токен за допомогою методу tokenProvider через сервіс, якщо ні то буде показана помилка. При надсиланні запиту це виглядає так (рисунок 3.5)

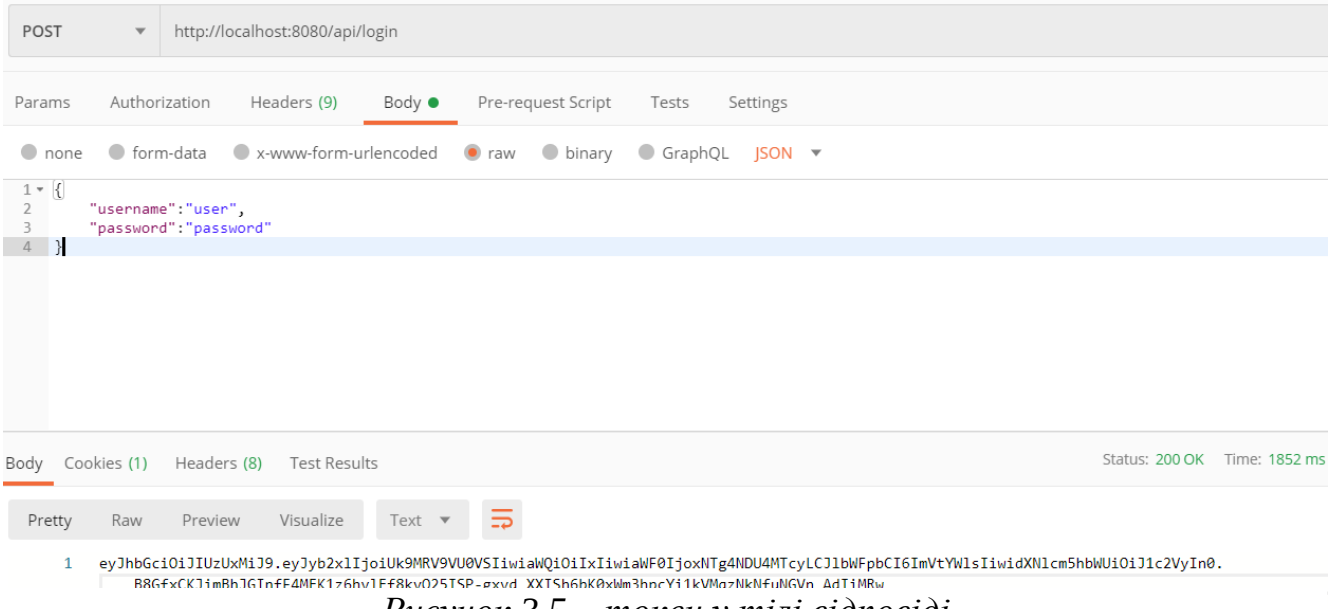


*Рисунок 3.5 – токен у тілі відповіді*

Ми пройшли аутентифікацію і отримали токен, тепер наш умовний клієнтський застосунок покладе цей токен у header наступного запиту інформації (рисунок 3.6)

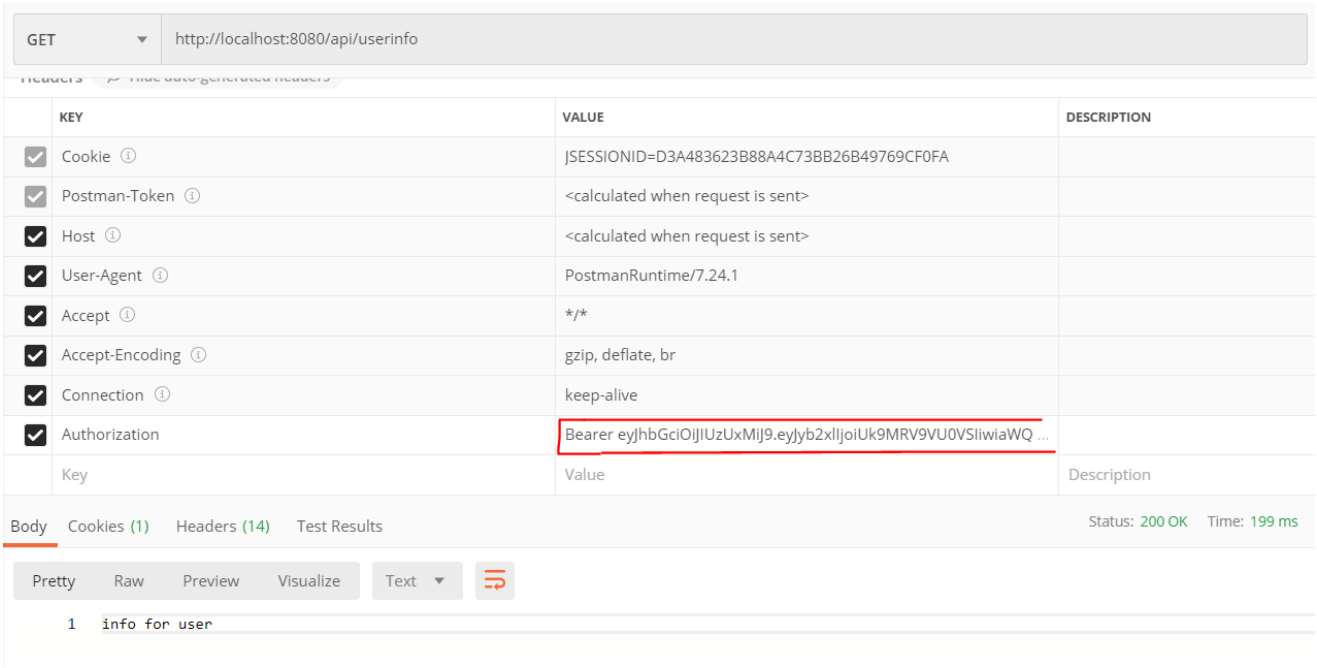


*Рисунок 3.6 – отримання інформації за виданим токеном*

У результаті отримуємо інформацію для користувача, бо також налаштовано, що за своєю роллю він її може отримати, InfoController:

```
@PreAuthorize("hasRole('USER')")
@GetMapping(path = "userinfo")
public ResponseEntity<String> getInfo(){
   return new ResponseEntity<>("info for user", HttpStatus.OK);
}

@PreAuthorize("hasRole('ADMIN')")
@GetMapping(path = "admininfo")
public ResponseEntity<String> getInfoA(){
   return new ResponseEntity<>("info for admin", HttpStatus.OK);
}
```

Тоді постає питання а на основі чого видається токен? Про це можна дізнатися з класу TokenProvider

```
public String provideToken(User user) {
return Jwts.builder()
      .setSubject(String.valueOf(user.getId()))
      .setClaims(
            new HashMap<String, Object>() {{
               put("id", user.getId());
               put("username", user.getUsername());
               put("email", user.getEmail());
               put("role", user.getRole().name());
            }})
      .setIssuedAt(new Date())
      .signWith(SignatureAlgorithm.HS512, SECRET)
      .compact();

}
```

Тут формується гешмапа для атрибутів сутності користувача, і сюди ми можемо додати будь-яку інформацію, яка може бути потрібна для опрацювання клієнтом, також налаштовується Secret ключем з підписним алгоритмом і встановлюється expiration date токена.

Отже у аутентифікації токенами можна виділити такі переваги преваги:

- Токен має усю потрібну інформацію для перевірки і не заважає масштабуванню
- Токен може містити у собі будь-яку інформацію, яку можна використати на клієнті без нових зайвих запитів і знижує тим самим навантаження на БД.
- Також токени простіше реалізувати для IoT, мобільних пристроїв та додатків де не має можливості використати cookies.
- Поєднання з іншими методами аутентифікації

## 3.5 Certificate authentication

Цей метод побудований на сертифікатах користувачів, які вони надають серверу щоб він міг по ньому визначити та надіслати відповідь про те чи підходить йому цей сертифікат. [11]

Для реалізації цього методу я використав стандарт X.509. X.509 - це стандартний формат для сертифікатів відкритого ключа, цифрових документів, які надійно асоціюють криптографічні пари ключів з особами, такими як веб-сайти, особи або організації[17].

Щоб реалізувати цей спосіб я описав просте клієнт-серверне застосування. Спочатку треба було згенерувати keystore застосунку (власне наші сертифікати), це було зроблено за допомогою keytool Java через командну стрічку.

Для сервера я визначив такі налаштування properties:

```
server.port = 8080
server.ssl.key-store=/STUDIES/CoursalMatSpring/WORK/Project
part/ProjectPartX509/keystores/serverkeystore.p12
server.ssl.key-store-password=password

server.ssl.trust-store=/STUDIES/CoursalMatSpring/WORK/Project
part/ProjectPartX509/keystores/servertruststore.p12
server.ssl.trust-store-password=password
server.ssl.client-auth=need
```

Тобто налаштував сервер для роботи на порті 8080 та визначив йому шляхи до keystore та truststore (ключі шо ідентифікують нас та інших відповідно [18]). Тепер лише trusted клієнт з дійсним сертифікатом може отримати доступ до сервера.

Також конфігуруємо Security:

```
@Override
protected void configure(HttpSecurity http) throws Exception {
   http.x509().subjectPrincipalRegex("CN=(.*?)(?:,|$)").
         and().authorizeRequests().anyRequest().authenticated().
         and().userDetailsService(userDetailsService());
}
```

Від конфігурацій які я наводив у попередніх розділах, в неї тільки є відмінність у тому, що ми вказуємо x509() після HttpSecurity. Це зроблено для того, щоб сервер зрозумів, що ми використовуємо саме X.509.

API сервера через контролер виглядає так:

```
@GetMapping(path = "info")
public String helloController()
{
   String loggedUser;
   Object principal =
SecurityContextHolder.getContext().getAuthentication().getPrincipal();

   if (principal instanceof UserDetails) {
      loggedUser = ((UserDetails) principal).getUsername();
   } else {
      loggedUser = principal.toString();
      System.out.println(principal);
   }
   return loggedUser;
}
```

Тобто, якщо користувач пройде аутентифікацію він зможе подивитись інформацію про себе.

Перейдемо до клієнтської частини. Для нього я описав інтерфейс сервісу з єдиним методом для отримання інформації з сервера:

```
public interface ServerService {
    String getInfo();
}
```

У реалізація метода також конфігурую систему наших сертифікатів себе та сервера:

```
System.setProperty("javax.net.ssl.keyStore", "/STUDIES/CoursalMatSpring/Project
part" +
      "/ProjectPartX509/keystores/clientkeystore.p12");
System.setProperty("javax.net.ssl.keyStorePassword", "password");
System.setProperty("javax.net.ssl.trustStore", "/STUDIES/CoursalMatSpring/Project
part/" +
      "ProjectPartX509/keystores/clienttruststore.p12");
System.setProperty("javax.net.ssl.trustStorePassword", "password");
```

Для виводу інформації використовуємо такий контролер:

```
@GetMapping(path = "info")
public String userInfo()
{
   return "User info: " + serverService.getInfo();
}
```

Після конфігурації запустимо сервер та клієнт і подивимось результат по localhost у браузері (рисунок 3.7):

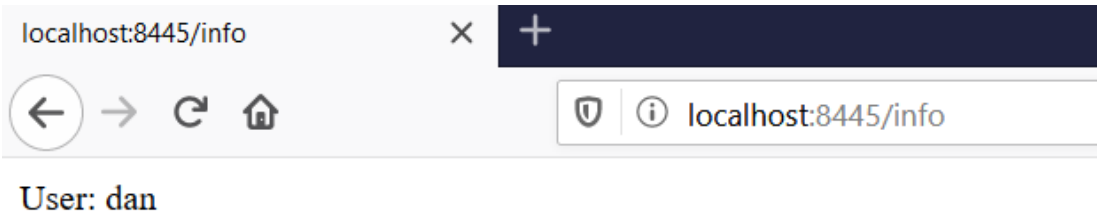


*Рисунок 3.7 – інформація про користувача за сертефікатом*

Як і було зазначено, ми отримуємо інформацію про себе, яку конфігурували для наших сертифікатів.

Отже, перейдемо до аналізування метода

Переваги:

- Забезпечення перевірки ідентичності через секретні приватні ключі [19]
- Неінтерактивний вхід можливий через агент аутентифікації. [19]
- Сертифікати можна використовувати для багатьох цілей, таких як вхід, доступ до файлових серверів та захист електронної пошти. [19]

Недоліки:

- Потрібна інфраструктура відкритого ключа (PKI). Це може збільшити витрати на початкове розгортання в деяких середовищах порівняно з іншими методами. [19]

## 3.6 OAuth2 authentication

Цей спосіб є достатньо поширеним, коли ви заходите на якийсь сайт іноді вам пропонується увійти через акаунт іншого сервісу. Я вважаю що це дуже зручно, бо для реєстрації не потрібно вводити додаткові дані, а використати ті, які вже маються у якомусь сервісі. В цьому і полягає вирішення проблеми багатьох акаунтів для користувачів, також не треба створювати і запам’ятовувати нові складні паролі для акаунтів.

Для налаштування через OAuth2 я використав сервіс Google Cloud Platform, щоб використовувати акаунти Google для аутентифікації.

Я створив проект у сервісі, створив ідентифікатор клієнта OAuth, отримав ідентифікатор та секретний код, тепер ми зможемо налаштовувати наше застосування.

У propertis пропишемо конфігурацію OAuth2:

```
security.oauth2.client.clientId=${clientId}
security.oauth2.client.clientSecret=${clientSecret}
security.oauth2.client.accessTokenUri=https://www.googleapis.com/oauth2/v4/token
security.oauth2.client.userAuthorizationUri=https://accounts.google.com/o/oauth2/v2/auth
security.oauth2.client.clientAuthenticationScheme=form
security.oauth2.client.scope=openid,email,profile
security.oauth2.resource.userInfoUri=https://www.googleapis.com/oauth2/v3/userinfo
security.oauth2.resource.preferTokenInfo=true
```

Введемо отримані clientId та clientSecret з нашого сервісу. Також показуємо accessTokenUri та userAuthorizationUri для визначення сервісу, схемою аутентифікації AuthenticationScheme, scope обмеження області запиту клієнта, userInfoUri для налаштування SSO (продукт або технологія, що дозволяє не вдаватись до повторної аутентифікації користувачів [20]) і перевагу TokenInfo.

Також треба додати, що для цього способу я використав базу даних MySQL для зберігання користувачів і технологію JPA для роботи з ними. У конфігурації:

```
spring.jpa.hibernate.ddl-auto=update
spring.datasource.url=jdbc:mysql://localhost:3306/coursal
spring.datasource.username=root
spring.datasource.password=admin
```

Тепер до налаштувань безпеки, анотуємо клас конфігурації:

```
@Configuration
@EnableWebSecurity
@EnableOAuth2Sso
public class WebSecurityConfig extends WebSecurityConfigurerAdapter {
```

Визначаємо, що це конфігурація безпеки, та підключаємо OAuth2 SSO

Конфігурація HttpSecurity достатньо проста :

```
@Override
protected void configure(HttpSecurity http) throws Exception {
  http
      .authorizeRequests()
      .antMatchers(HttpMethod.GET, "/api/userinfo").hasRole("USER")
      .antMatchers(HttpMethod.GET, "/api/admininfo").hasRole("ADMIN")
      .anyRequest().authenticated();
}
```

Для користувачів авторизованих з роллю ROLE_USER доступ до інформації userinfo, з роллю ROLE_ADMIN – admininfo.

Для виокремлення Principal використано Bean:

```
@Bean
public PrincipalExtractor principalExtractor(UserDetailsRepository
userDetailsRepository){
  return map ->
  {
    String id = String.valueOf(map.get("sub"));
    User user = userDetailsRepository.findById(id)
        .orElseGet(() ->
```

Витягуємо id аутнетифікованого клієнта, і якщо в базі даних він збережений, то авторизуємо його, якщо ні, то реєструємо як нового і видаємо роль ROLE_USER.

І остання бачимо знайомий контролер, який видає інформацію про користувача в системі:

```
@RestController
@CrossOrigin
@RequestMapping("/api")
public class InfoController {

   @GetMapping(path = "userinfo")
   public ResponseEntity<User> getInfo(@AuthenticationPrincipal User user){
      return new ResponseEntity<>(user, HttpStatus.OK);
   }

   @GetMapping(path = "admininfo")
   public ResponseEntity<User> getInfoA(@AuthenticationPrincipal User admin){
      return new ResponseEntity<>(admin, HttpStatus.OK);
   }
}
```

Запустимо програму і перейдемо по URL нашого застосунку, нас зустріне така сторінка (рисунок 3.8):

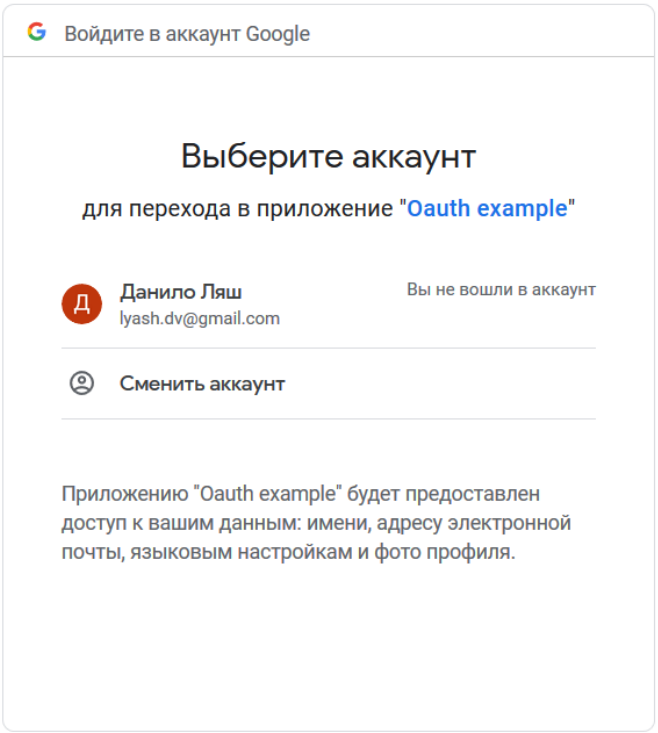


*Рисунок 3.8 – обрання акаунту Google для аутентифікації*

Застосунок знаходить Google акаунти у які було проведено вхід у даному браузері і пропонує мені використати їх для входу або змінити акаунт.

Після того як я обираю потрібний акаунт, я можу побачити інформацію про себе:

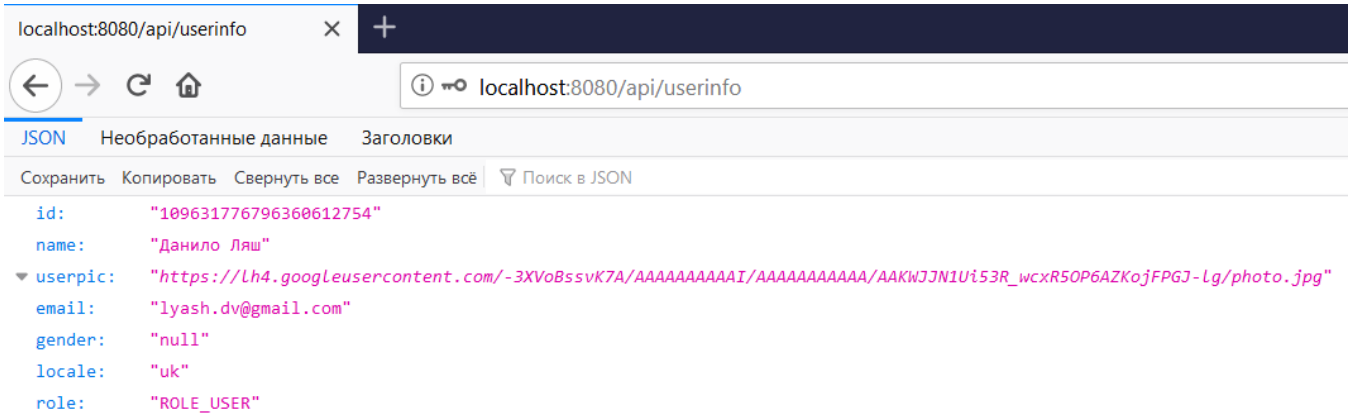


*Рисунок 3.9 – інформація отримана за допомогою OAuth2*

Отже, які переваги можна виокремити у цього метода:

- Можна отримувати інформацію про користувача з іншого застосунку (У моєму випадку Google).
- При отриманні доступу до ресурсів, ми не використовуємо прямий доступ до даних аутентифікації користувача.
- Це дозволяє обмежений доступ до даних користувача та дозволяє отримати доступ, коли термін дії авторизації закінчується. [21]

І недоліки:

- Якщо ваш акаунт захоплюють зловмисники, то сайти з OAuth стають під загрозу можливості несанкціонованого доступу.
- Якщо ви описуєте багато можливостей для аутентифікації таким чином, то можуть виникнути конфлікти і доведеться писати під кожне розширення писати свою частину коду. [21]

## 3.7 Авторизація

У Spring Security для авторизації як і описано у моїх реалізаціях використовується метод ролей.

Налаштування може проводитись різним чином, як через Java код, так і через XML. У випадку моїх реалізацій я використовую Java код, я вважаю його більш зручнішим і зрозумілішим для написання як розробнику.

Для визначення доступу до ресурсів можна вказувати як і у конфігурації HttpSecurity:

```
.antMatchers(HttpMethod.GET, "/api/userinfo").hasRole("USER")
.antMatchers(HttpMethod.GET, "/api/admininfo").hasRole("ADMIN")
```

Так і у контролерах у анотаціях до методів:

```
@PreAuthorize "hasRole('USER')"
@GetMapping(path = "userinfo")
public ResponseEntity<String> getInfo(){
   return new ResponseEntity<>("info for user", HttpStatus.OK);
}

@PreAuthorize "hasRole('ADMIN')"
@GetMapping(path = "admininfo")
public ResponseEntity<String> getInfoA(){
   return new ResponseEntity<>("info for admin", HttpStatus.OK);
}
```

Також, якщо використовувати JSP + Thymeleaf, то можна визначити доступ таким чином[22]:

```
<security:authorize access="hasRole('ROLE_USER')">
   Інформація для звичайного користувача
</security:authorize>
<security:authorize access="hasRole('ROLE_ADMIN')">
   Інформація для адміністратора
</security:authorize>
```

# Результати аналізу

Отже у результаті аналізів вище перерахованих способів аутентифікації та спроб їх реалізації можна зазначити, що кожен спосіб є унікальним та підходить під різні цілі. Усі ці методи є унікальними зі своїми перевагами та недоліками, для обирання свого метода треба розуміти що вам потрібно для вашого застосунку.

Також треба сказати, що це не всі можливі методи, бо їх реалізацій може бути дуже багато, для аналізу я виокремлював ті методи, про які написано вище через те що вони є популярними або базовими для розуміння загальної конфігурації вашого застосунку.

Що до методів авторизації, то можна сказати що для них зручно використовувати ролі. Ролі дуже добре визначають хто є користувач, також ролі можуть пересікатися, тобто користувач може бути як і звичайним користувачем так і адміністратором одразу, тому такий метод найбільше використаний для написання авторизації за допомогою платформи Spring + Spring Security.

# Висновки

Отже, аналіз методів був виконаний. У ході курсової роботи проаналізовані і визначені недоліки та переваги популярних методів авторизації та аутентифікації Spring Security, у ході аналізу яких було спроба по-різному виконати конфігурацію, визначити авторизацію та зрозуміти механізм роботи аутентифікації та авторизації.

У ході технічної сторони проекту були розроблені чотири проекти, які можуть допомогти зрозуміти як працюють методи авторизації та аутентифікації на обраних цікавих та популярних [23] прикладах їх реалізації, та можливо дати змогу зрозуміти який засіб підходить розробнику для розробки його застосування.

# Список літератури


1. Steve Shapiro. The World's Authentication Problem. 2017. [Електронний ресурс] – URL: https://www.infosecurity-magazine.com/opinions/worlds-authentication-problem/
2. Ben Alex, Luke Taylor, Rob Winch, Gunnar Hillert, Joe Grandja, Jay Bryant,Eddú Meléndez, Josh Cummings, Dave Syer. Spring Security Reference [Електронний ресурс] – URL: https://docs.spring.io/spring-security/site/docs/current/reference/html5/ (Дата звернення: 10 квітня 2020)
3. Rod Johnson, Juergen Hoeller, Keith Donald, Colin Sampaleanu, Rob Harrop and other. Spring Framework Documentation. [Елекрониий ресурс]. URL: https://docs.spring.io/spring/docs/current/spring-framework-reference/index.html (Дата звернення 09 квітня 2020)
4. Martin Fowler. InversionOfControl. 2005 [Електронний ресурс]. URL: https://martinfowler.com/bliki/InversionOfControl.html
5. Spring tutorial [Електронний ресурс]. URL: https://www.javatpoint.com/spring-tutorial (Дата звернення 5 березня 2020)
6. Santosh Kumar. Spring and Hibernate. 2009. [Електронний ресурс] URL:https://books.google.com.ua/books?id=jQyeAAAAQBAJ&pg=PA6&lpg=PA6&dq=spring+framework+modules
7. Ben Alex, Luke Taylor. Spring Security: Reference Documentation 2012 [Електронний ресурс] URL: https://docs.spring.io/spring-security/site/docs/3.0.x/reference/springsecurity.pdf
8. What is Spring Security? August 13, 2018 [Електронний ресурс] URL: https://www.developer.com/java/ent/what-is-spring-security.html

9. Сергій Бондаренко. Идентификация, аутентификация, авторизация — в чем разница?. [Електронний ресурс] URL: http://it-uroki.ru/uroki/bezopasnost/identifikaciya-autentifikaciya-avtorizaciya.html
(Дата звернення 1 березня 2020)
10. Spring Security authentication. October 30, 2018. [Електронний ресурс] URL: http://shazsterblog.blogspot.com/2018/10/spring-security-authentication-security.html
11. Користувач: OlegWock. Що таке RESTful API? [Електронний ресурс] URL: https://codeguida.com/post/601
(Дата звернення 15 квітня 2020)
12. Користувач: lehaalexx. 6 способов: как добавить security для Rest сервиса в Java. [Електронний ресурс] URL: https://habr.com/ru/post/245415/
(Дата звернення 1 квітня 2020)
13. Tom Tromey. Кодирование и декодирование в формате Base64 2018. [Електронний ресурс] URL: https://developer.mozilla.org/en-US/docs/Glossary/Base64
14. Користувач: мистер Олимпия. 6 способов: Хэш-функция MD5 2011. [Електронний ресурс] URL: https://habr.com/ru/sandbox/26876/
15. HTTP - Status Codes. [Електронний ресурс] URL: https://www.tutorialspoint.com/http/http_status_codes.htm
(Дата звернення 29 квітня 2020)
16. Користувач: AloneCoder. Как ты реализуешь аутентификацию, приятель? 2017. [Електронний ресурс] URL: https://habr.com/ru/company/mailru/blog/343288/
17. RFC 7519: JSON Web Token. [Електронний ресурс] URL: https://oauth.net/2/jwt/ (Дата звернення 24 квітня 2020)

18. Aaron Russell. What Is an X.509 Certificate? 2019 [Електронний ресурс] URL: https://www.ssl.com/faqs/what-is-an-x-509-certificate/

19. Denis Szczukock. Difference Between a Java Keystore and a Truststore 2019. [Електронний ресурс] URL: https://www.baeldung.com/java-keystore-truststore-difference

20. Advantages and Disadvantages of Certificate Authentication. [Електронний ресурс] URL: https://www.ssh.com/manuals/server-zos-product/55/ch06s03s05.html (Дата звернення 29 квітня 2020)

21. OAuth 2.0 – Overview. [Електронний ресурс] URL: https://www.tutorialspoint.com/oauth2.0/oauth2.0_overview.htm

22. Eugen Paraschiv, Spring Security Expressions – hasRole Example 2020. [Електронний ресурс] URL: https://www.baeldung.com/spring-security-expressions-basic

23. 4 Most Used REST API Authentication Methods. 2019. [Електронний ресурс] URL: https://blog.restcase.com/4-most-used-rest-api-authentication-methods/