





# Spring Security

Урок 7



## Что будет на уроке сегодня

-  Важность информационной безопасности
-  Нарушение безопасности
-  Принципы безопасности
-  Что такое Spring Security
-  JSON Web Token
-  Настройка Spring Security
-  Практика



# Важность информационной безопасности

Если данные попадут в неправильные руки из-за недостаточной безопасности нашего серверного приложения, последствия могут быть катастрофическими.





## Примеры нарушения безопасности

Один из самых известных примеров – это компания Yahoo.  
В 2013 году они были жертвой крупнейшего в истории нарушения безопасности, которое затронуло около 3 миллиардов пользовательских аккаунтов.

**yahoo!**





## Примеры нарушения безопасности

В 2017 году злоумышленники получили доступ к личным данным около 147 миллионов людей, включая имена, номера социального страхования, даты рождения, адреса и номера водительских удостоверений. Это нарушение безопасности обошлось Equifax в \$575 миллионов штрафов и выплат по искам.

# ***EQUIFAX***





# Аутентификация

## Примеры аутентификации:

1. Вход в систему с использованием имени пользователя и пароля.
2. Вход с помощью отпечатка пальца или распознавания лица на вашем смартфоне.
3. Ввод PIN-кода или ответ на секретный вопрос.





## Аутентификация

1. В компьютерной игре пользователь может быть аутентифицирован как игрок, но авторизован только для выполнения определенных действий.
2. В банковской системе клиент может быть аутентифицирован как владелец счета, но авторизован только для выполнения операций.
3. В интернет-магазине покупатель может быть аутентифицирован как зарегистрированный пользователь, но авторизован только для просмотра товаров, добавления их в корзину и тд.



# Принципы безопасности

1. Минимальные привилегии

2. Защита данных

3. Аудит и мониторинг

4. Обработка ошибок и исключений

5. Обновления и патчи

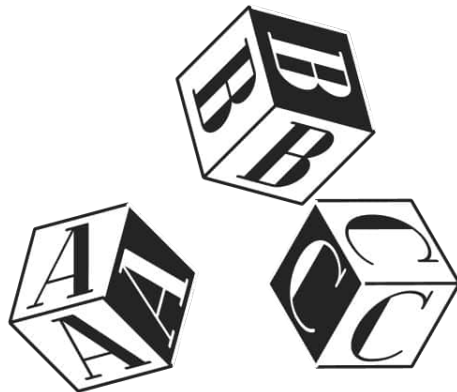




# Что такое Spring Security

## Основные компоненты Spring Security:

1. Аутентификация
2. Авторизация
3. Защита от атак
4. Шифрование паролей





## JSON Web Token (JWT)

**JWT** – это стандарт, который определяет способ безопасной передачи информации между двумя сторонами в виде JSON-объекта.





# JSON Web Token (JWT)

**В Spring Security JWT обычно используется вместе с OAuth 2.0 для аутентификации и авторизации:**

1. Пользователь аутентифицируется с помощью своих учетных данных.
2. После успешной аутентификации сервер создает JWT, который содержит уникальные данные пользователя, и отправляет его обратно пользователю.
3. Пользователь сохраняет этот JWT и отправляет его в заголовке Authorization с каждым последующим запросом.
4. Сервер проверяет JWT в каждом запросе, чтобы аутентифицировать пользователя и определить его права доступа.



# Настройка Spring Security

Шаг 1: Добавление зависимости

```
1 <dependency>
2     <groupId>org.springframework.boot</groupId>
3     <artifactId>spring-boot-starter-security</artifactId>
4 </dependency>
```



# Настройка Spring Security

Шаг 2: Создание класса конфигурации безопасности

```
1 @EnableWebSecurity
2 public class SecurityConfig extends WebSecurityConfigurerAdapter {
3
4     @Override
5     protected void configure(HttpSecurity http) throws Exception {
6         http
7             .authorizeRequests()
8                 .anyRequest().authenticated()
9                 .and()
10                .httpBasic();
11    }
12
```



# Настройка Spring Security

Шаг 2: Создание класса конфигурации безопасности

```
13     @Autowired
14     public void configureGlobal(AuthenticationManagerBuilder auth) throws Exception {
15         auth
16             .inMemoryAuthentication()
17             .withUser("user").password(passwordEncoder().encode("password")).roles("USER");
18     }
19
20     @Bean
21     public PasswordEncoder passwordEncoder() {
22         return new BCryptPasswordEncoder();
23     }
24 }
```



## Настройка Spring Security

Шаг 3: Тестирование нашей конфигурации безопасности

```
1 @RestController
2 public class HelloController {
3
4     @GetMapping("/hello")
5     public String hello() {
6         return "Hello, world!";
7     }
8 }
```



## Виды атак

1. CSRF (межсайтовая подделка запроса).

2. XSS (межсайтовый скриптинг).

3. SQL-инъекции.

4. Атаки с перехватом сессии (Session Hijacking).





# Создание проекта

## Шаг 1: Создание проекта в Spring Initializr

1. Откройте **Spring Initializr**.
2. Выберите “Maven Project” в качестве типа проекта.
3. В качестве языка программирования выберите “Java”.
4. Выберите последнюю версию Spring Boot.
5. Введите детали проекта. В качестве Group, введите что-то вроде “com.yourname”.  
В Artifact, введите что-то вроде “jwt-demo”.



# Создание проекта

## Шаг 1: Создание проекта в Spring Initializr

6. В качестве упаковки выберите “Jar”.
7. Выберите версию Java, которую вы используете.
8. В разделе “Dependencies” добавьте следующие зависимости: “Spring Web”, “Spring Security”, “Spring Data JPA” (для работы с базой данных) и “JJwt” (библиотека для работы с JWT).
9. Нажмите “Generate”, чтобы скачать проект.



# Создание проекта

## Шаг 2: Открытие проекта в IDE

Теперь, когда у вас есть сгенерированный проект, вы можете открыть его в вашей любимой IDE (например, IntelliJ IDEA, Eclipse и др.). Все настройки уже выполнены автоматически, и вы можете начать разработку вашего приложения.



## Создание проекта

### Шаг 3: Добавление зависимости для JWT

```
1 <dependency>
2     <groupId>io.jsonwebtoken</groupId>
3     <artifactId>jjwt</artifactId>
4     <version>0.9.1</version>
5 </dependency>|
```



# Настройка Spring Security

## Шаг 1: Создание класса конфигурации безопасности

```
1 @Configuration
2 @EnableWebSecurity
3 public class SecurityConfig extends WebSecurityConfigurerAdapter {
4     // Детали настройки будут добавлены позже
5 }
```



# Настройка Spring Security

## Шаг 2: Настройка правил безопасности

```
1 @Override
2 protected void configure(HttpSecurity http) throws Exception {
3     http
4         .csrf().disable() // Отключаем защиту CSRF, так как будем использовать JWT
5         .authorizeRequests()
6             .antMatchers("/login").permitAll() // Позволяем всем пользователям доступ к эндпоинту "/login"
7             .anyRequest().authenticated() // Все остальные эндпоинты требуют аутентификации
8         .and()
9         .addFilter(new JwtAuthenticationFilter(authenticationManager())) // Добавляем наш фильтр аутентификации
10        .sessionManagement()
11        .sessionCreationPolicy(SessionCreationPolicy.STATELESS); // Не создаем сессию, так как будем использовать JWT
12 }
```



# Настройка Spring Security

## Шаг 3: Создание фильтра аутентификации

```
1 public class JwtAuthenticationFilter extends UsernamePasswordAuthenticationFilter {  
2     // Детали реализации будут добавлены позже  
3 }
```



# Обработка JWT

## Шаг 1: Генерация JWT

```
1 @Override
2 protected void successfulAuthentication(HttpServletRequest request,
3                                         HttpServletResponse response,
4                                         FilterChain chain,
5                                         Authentication authResult) throws IOException, ServletException {
6
7     UserDetails principal = (UserDetails) authResult.getPrincipal();
8 }
```





# Обработка JWT

## Шаг 1: Генерация JWT

```
9  String token = Jwts.builder()
10      .setSubject(principal.getUsername())
11      .setExpiration(new Date(System.currentTimeMillis() + JwtProperties.EXPIRATION_TIME)) // Устанавливаем срок действия токена
12      .signWith(SignatureAlgorithm.HS512, JwtProperties.SECRET) // Подписываем токен нашим серверным секретом
13      .compact();
14
15  response.addHeader(JwtProperties.HEADER_STRING, JwtProperties.TOKEN_PREFIX + token); // Добавляем токен в заголовок ответа
16 }
```



# Обработка JWT

## Шаг 2: Проверка JWT

```
1 public class JwtAuthorizationFilter extends BasicAuthenticationFilter {  
2  
3     public JwtAuthorizationFilter(AuthenticationManager authenticationManager) {  
4         super(authenticationManager);  
5     }  
6  
7     @Override  
8     protected void doFilterInternal(HttpServletRequest request,  
9                                     HttpServletResponse response,  
10                                    FilterChain chain) throws IOException, ServletException {  
11         // Здесь будет код для проверки JWT  
12     }  
13 }
```



## Обработка JWT

Шаг 3: Добавление фильтра авторизации в конфигурацию безопасности

```
1 @Override
2 protected void configure(HttpSecurity http) throws Exception {
3     http
4         .addFilter(new JwtAuthenticationFilter(authenticationManager()))
5         .addFilterBefore(new JwtAuthorizationFilter(authenticationManager()),
6                         UsernamePasswordAuthenticationFilter.class);
7     // Остальной код конфигурации
8 }
```



## Защита от атак

- **CSRF (Cross-Site Request Forgery)** – это атака, которая заставляет пользователя выполнить действие, которого он не собирался делать.
- **XSS (Cross-Site Scripting)** – это атака, при которой злоумышленник вставляет злонамеренные скрипты в веб-страницы, просматриваемые другими пользователями.



## Защита от атак

- **SQL Injection** – это атака, при которой злоумышленник может вставлять злонамеренные SQL запросы в поля ввода формы.
- **Session Fixation** – это атака, при которой злоумышленник использует сессию, которую пользователь уже открыл, для получения несанкционированного доступа.



## Проверка аутентификации и обработки JWT

1. Регистрация нового пользователя

2. Вход в систему

3. Доступ к защищенному ресурсу



## Проверка защиты от атак

1. CSRF-атака

2. XSS-атака

3. SQL-инъекция

4. Атака Session Fixation



**Спасибо за внимание**

