

07. Spring boot

컴퓨터공학부 201911228 홍지우

01. Security - Security Config

@Override

```
@RestController
@RequestMapping("api")
public class GetController extends WebSecurityConfigurerAdapter {

    @GetMapping("/v1/hello")
    public String getParameter(){
        return "Hello World!";
    }

    @PostMapping("/v1/string")
    public String postMethod(@RequestBody String str){
        return str;
    }

    //스프링 시큐리티 룰을 무시하게 만드는 URI 규칙
    @Override
    public void configure(WebSecurity web) throws Exception {
        web.ignoring()
            .antMatchers("/api/**");
    }

    //스프링 시큐리티 규칙
    @Override
    protected void configure(HttpSecurity http) throws Exception {
        http.authorizeRequests()
            .antMatchers("/login/**").permitAll(); //전체 접근 허용
    }
}
```

01. Security - Security Config


GET ⌵ http://localhost:8080/api/v1/hello

Params Authorization Headers (8) Body Pre-request Script

Query Params

	KEY	VALUE
	Key	Value

Body Cookies (1) Headers (13) Test Results

Pretty Raw Preview Visualize JSON ⌵ 

```
1 {
2   "timestamp": "2021-05-18T11:56:41.779+00:00",
3   "status": 401,
4   "error": "Unauthorized",
5   "message": "",
6   "path": "/api/v1/hello"
7 }
```

 Console

GET ⌵ http://localhost:8080/api/v1/hello

Params Authorization Headers (8) Body Pre-request Script Tests Settings

Query Params

	KEY	VALUE
	Key	Value

Body Cookies (1) Headers (5) Test Results

Pretty Raw Preview Visualize Text ⌵ 

1 Hello World!

02. JWT Token + Authorization

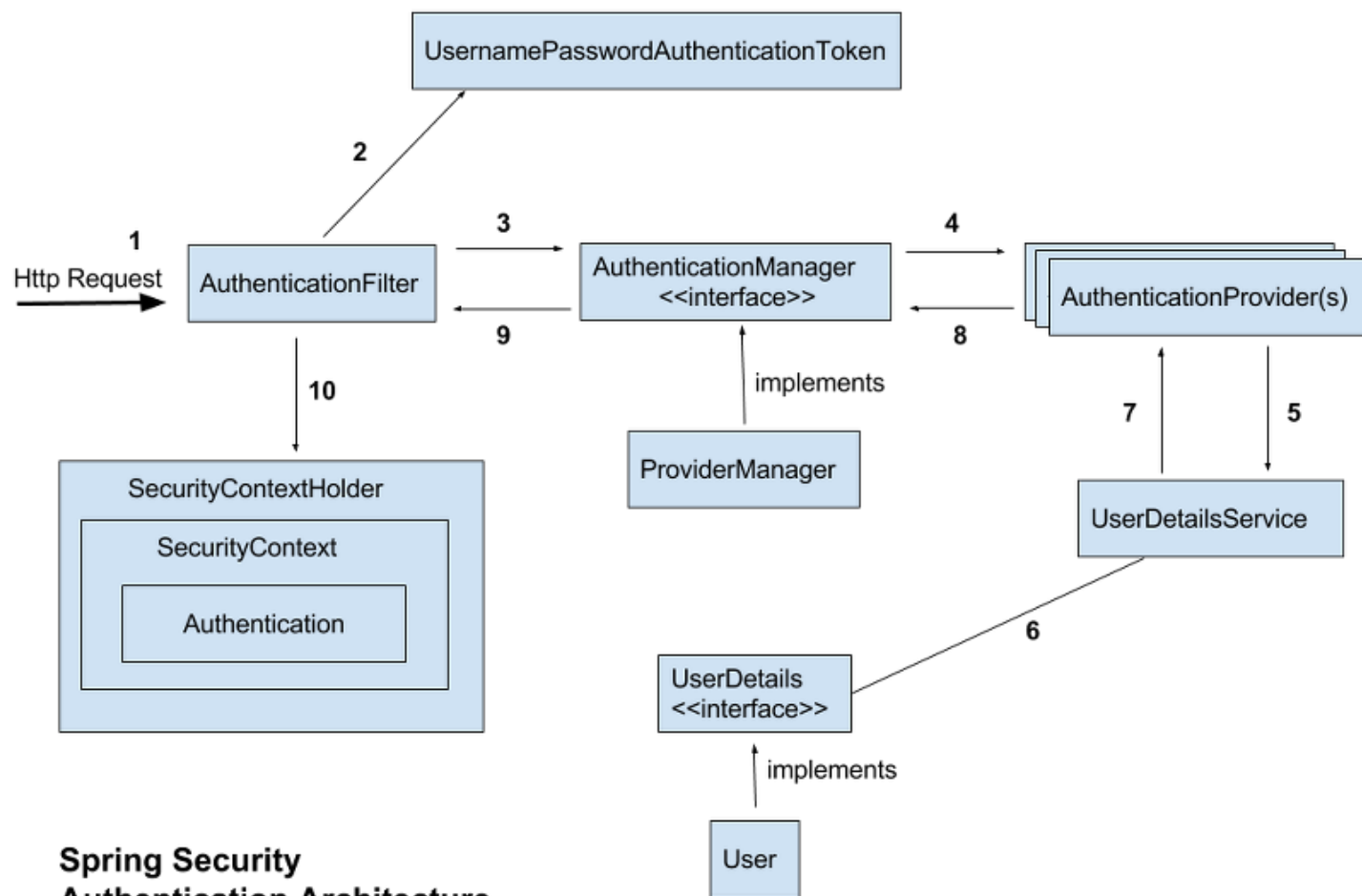
Spring Security

- 인증과 권한등 보안에 관한 기능을 제공하는 프레임워크

인증(Authentication)

- 사용자에게 적절한 접근 권한이 있는지 확인하는 일련의 과정
 - 접근 주체(Principal): 보호된 리소스에 접근하는 사용자
 - 권한(Authorization): 인증절차가 끝난 접근 주체가 보호된 리소스에 접근 가능한지를 결정하는 것
 - 인가(Authorize): 권한을 부여하는 작업
-
- 로그인, 웹 요청, 메소드 호출, 도메인 인스턴스에 대한 접근 등 상당히 깊은 수준의 권한 부여를 제공

02. JWT Token + Authorization



Spring Security Authentication Architecture

Chathuranga Tennakoon
www.springbootdev.com

1. 사용자가 로그인 정보와 함께 인증 요청(Http Request)
2. AuthenticationFilter가 이 요청을 가로칩니다. 이 때 가로챈 정보를 통해 UsernamePasswordAuthenticationToken이라는 인증용 객체를 생성합니다.
3. AuthenticationManager의 구현체인 ProviderManager에게 UsernamePasswordAuthenticationToken 객체를 전달합니다.
4. 다시 AuthenticationProvider에 UsernamePasswordAuthenticationToken 객체를 전달합니다
5. 실제 데이터베이스에서 사용자 인증정보를 가져오는 UserDetailsService에 사용자 정보(아이디)를 넘겨줍니다.
6. 넘겨받은 사용자 정보를 통해 DB에서 찾은 사용자 정보인 UserDetails 객체를 만듭니다. 이 때 UserDetails는 인증용 객체와 도메인용 객체를 분리하지 않고 인증용 객체에 상속해서 사용하기도 합니다.
7. AuthenticationProvider는 UserDetails를 넘겨받고 사용자 정보를 비교합니다.
8. 인증이 완료되면 권한 등의 사용자 정보를 담은 Authentication 객체를 반환합니다.
9. 다시 최초의 AuthenticationFilter에 Authentication 객체가 반환됩니다.
10. Authentication 객체를 SecurityContext에 저장합니다.

최종적으로 SecurityContextHolder는 세션 영역에 있는 SecurityContext에 Authentication 객체를 저장합니다. 세션에 사용자정보를 저장한다는 것은 스프링 시큐리티가 전통적인 세션-쿠키 기반의 인증 방식을 사용한다는 것을 의미합니다.

02. JWT Token + Authorization

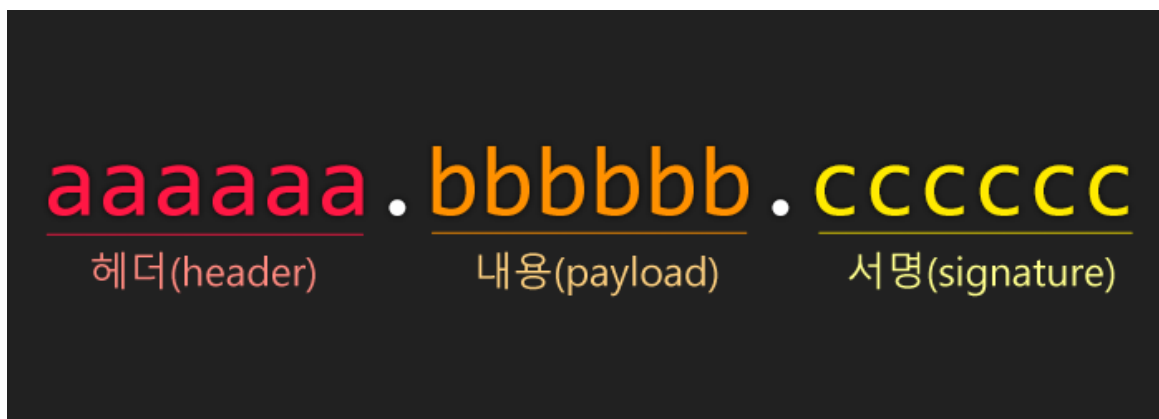
Spring Security Filter

다양한 필터체인을 사용하여 다양한 커스터마이징을 할 수 있도록 돕습니다

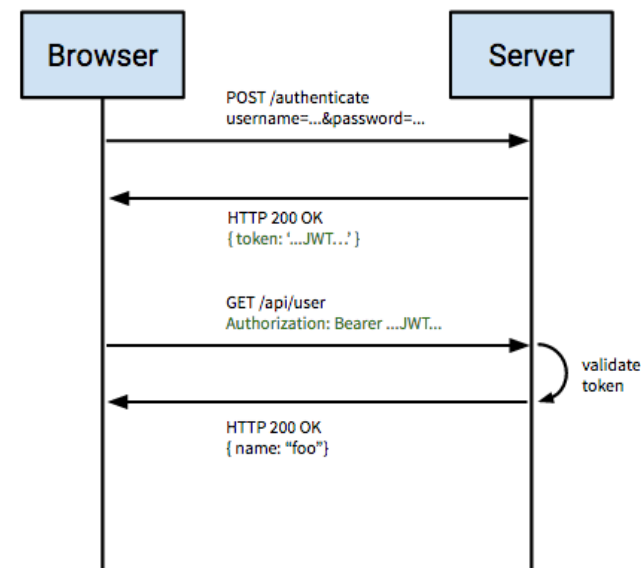
- SecurityContextPersistentFilter : SecurityContextRepository에서 SecurityContext를 가져와서 SecurityContextHolder에 주입하거나 반대로 저장하는 역할을 합니다.
- LogoutFilter : logout 요청을 감시하며, 요청시 인증 주체(Principal)를 로그아웃 시킵니다.
- UsernamePasswordAuthenticationFilter : login 요청을 감시하며, 인증 과정을 진행합니다.
- DefaultLoginPageGenerationFilter : 사용자가 별도의 로그인 페이지를 구현하지 않은 경우, 스프링에서 기본적으로 설정한 로그인 페이지로 넘어가게 합니다.
- BasicAuthenticationFilter : HTTP 요청의 (BASIC)인증 헤더를 처리하여 결과를 SecurityContextHolder에 저장합니다.
- RememberMeAuthenticationFilter : SecurityContext에 인증(Authentication) 객체가 있는지 확인하고 RememberMeServices를 구현한 객체 요청이 있을 경우, RememberMe를 인증 토큰으로 컨텍스트에 주입합니다.
- AnonymousAuthenticationFilter : 이 필터가 호출되는 시점까지 사용자 정보가 인증되지 않았다면 익명 사용자로 취급합니다.
- SessionManagementFilter : 요청이 시작된 이후 인증된 사용자인지 확인하고, 인증된 사용자일 경우 SessionAuthenticationStrategy를 호출하여 세션 고정 보호 매커니즘을 활성화 하거나 여러 동시 로그인을 확인하는 것과 같은 세션 관련 활동을 수행합니다.
- ExceptionTranslationFilter : 필터체인 내에서 발생하는 모든 예외를 처리합니다.
- FilterSecurityInterceptor : AccessDecisionManager로 권한부여처리를 위임하고 HTTP 리소스의 보안 처리를 수행합니다.

02. JWT Token + Authorization

JWT(Json Web Token)



Modern Token-Based Auth



- JSON 객체를 통해 안전하게 정보를 전송할 수 있는 웹표준(RFC7519)
- '.'을 구분자로 세 부분으로 구분되어 있는 문자열
- 각각 헤더는 토큰 타입과 해싱 알고리즘을 저장하고, 내용은 실제로 전달할 정보, 서명에는 위변조를 방지하기위한 값이 들어가게 됩니다.
- 토큰 자체에 데이터를 가지고 있다
- 기존의 세션-쿠키 기반의 로그인인 아니라 JWT같은 토큰 기반의 로그인을 하게 되면 세션이 유지되지 않는 다중 서버 환경에서도 로그인을 유지할 수 있게 되고 한 번의 로그인으로 유저정보를 공유하는 여러 도메인에서 사용할 수 있다는 장점
- 회원을 구분할 수 있는 정보가 담기는 곳이 바로 JWT의 payload 부분이고 이곳에 담기는 정보의 한 '조각'을 Claim 이라고 합니다. Claim은 name / value 한 쌍으로 이루어져 있으며 당연히 여러개의 Claim들을 넣을 수 있습니다.

02. JWT Token + Authorization

build.gradle

```
dependencies {  
    implementation 'org.springframework.boot:spring-boot-starter-data-jpa'  
    implementation 'org.springframework.boot:spring-boot-starter-web'  
    implementation 'org.springframework.boot:spring-boot-starter-security'  
    implementation 'io.jsonwebtoken:jjwt:0.9.1'  
  
    compileOnly 'org.projectlombok:lombok'  
    runtimeOnly 'com.h2database:h2'  
    annotationProcessor 'org.projectlombok:lombok'  
    testImplementation('org.springframework.boot:spring-boot-starter-test') {  
        exclude group: 'org.junit.vintage', module: 'junit-vintage-engine'  
    }  
    testImplementation 'org.springframework.security:spring-security-test'  
}
```

User Repository

```
package com.example.jwtProject2.User;  
  
import org.springframework.data.jpa.repository.JpaRepository;  
  
import java.util.Optional;  
  
public interface UserRepository extends JpaRepository<User, Long> {  
    Optional<User> findByEmail(String email);  
}
```


02. JWT Token + Authorization

User Entity 객체

```
package com.example.jwtProject2.User;

import lombok.AllArgsConstructor;
import lombok.Builder;
import lombok.Getter;
import lombok.NoArgsConstructor;
import org.springframework.security.core.GrantedAuthority;
import org.springframework.security.core.authority.SimpleGrantedAuthority;
import org.springframework.security.core.userdetails.UserDetails;

import javax.persistence.*;
import java.util.ArrayList;
import java.util.Collection;
import java.util.List;
import java.util.stream.Collectors;

@Getter
@NoArgsConstructor
@AllArgsConstructor
@Builder
@Entity
public class User implements UserDetails {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @Column(length = 100, nullable = false, unique = true)
    private String email;

    @Column(length = 300, nullable = false)
    private String password;

}
```

```
//UserDetails으로 객체의 권한 정보를 관리하기 때문에, 추가 정보를 재정의해야함
@ElementCollection(fetch = FetchType.EAGER)
@Builder.Default
private List<String> roles = new ArrayList<>();

@Override
public Collection<? extends GrantedAuthority> getAuthorities() {
    return this.roles.stream()
        .map(SimpleGrantedAuthority::new)
        .collect(Collectors.toList());
}

@Override
public String getUsername() {
    return email; //이메일이 ID 역할
}

@Override
public boolean isAccountNonExpired() {
    return true;
}

@Override
public boolean isAccountNonLocked() {
    return true;
}

@Override
public boolean isCredentialsNonExpired() {
    return true;
}

@Override
public boolean isEnabled() {
    return true;
}
```

02. JWT Token + Authorization

Security Config (Spring Security Filter Chain 사용 명시)

```
//필터 등록, 인증 요구
@RequiredArgsConstructor
@EnableWebSecurity
public class WebSecurityConfig extends WebSecurityConfigurerAdapter {

    private final JwtTokenProvider jwtTokenProvider;

    // 암호화에 필요한 PasswordEncoder 를 Bean 등록합니다.
    @Bean
    public PasswordEncoder passwordEncoder() {
        return PasswordEncoderFactories.createDelegatingPasswordEncoder();
    }

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

    @Override
    protected void configure(HttpSecurity http) throws Exception {
        http
            .httpBasic().disable() // rest api 만을 고려하여 기본 설정은 해제하겠습니다.
            .csrf().disable() // csrf 보안 토큰 disable처리.
            .sessionManagement().sessionCreationPolicy(SessionCreationPolicy.STATELESS) // 토큰 기반 인증이므로 세션 역시 사용하지 않습니다.
            .and()
            .authorizeRequests() // 요청에 대한 사용권한 체크
                .antMatchers("/admin/**").hasRole("ADMIN")
                .antMatchers("/user/**").hasRole("USER")
                .anyRequest().permitAll() // 그외 나머지 요청은 누구나 접근 가능
                .and()
            .addFilterBefore(new JwtAuthenticationFilter(jwtTokenProvider),
                UsernamePasswordAuthenticationFilter.class);
        // JwtAuthenticationFilter를 UsernamePasswordAuthenticationFilter 전에 넣는다
    }
}
```

```
package com.example.jwtProject2.config;

import com.example.jwtProject2.jwt.JwtAuthenticationFilter;
import com.example.jwtProject2.jwt.JwtTokenProvider;
import lombok.RequiredArgsConstructor;
import org.springframework.context.annotation.Bean;
import org.springframework.security.authentication.AuthenticationManager;
import org.springframework.security.config.annotation.web.builders.HttpSecurity;
import org.springframework.security.config.annotation.web.configuration.EnableWebSecurity;
import org.springframework.security.config.annotation.web.configuration.WebSecurityConfigurerAdapter;
import org.springframework.security.config.http.SessionCreationPolicy;
import org.springframework.security.crypto.factory.PasswordEncoderFactories;
import org.springframework.security.crypto.password.PasswordEncoder;
import org.springframework.security.web.authentication.UsernamePasswordAuthenticationFilter;
```

02. JWT Token + Authorization

JWT 토큰 Provider (JWT 생성, 검증)

//JWT에는 토큰 만료 시간, 회원 권한 정보를 저장할 수 있음

//토큰을 생성하고 검증하는 컴포넌트

@RequiredArgsConstructor

@Component

public class JwtTokenProvider {

private String secretKey = "webfirewood";

// 토큰 유효시간 30분

private long tokenValidTime = 30 * 60 * 1000L;

private final UserDetailsService userDetailsService;

// 객체 초기화, secretKey를 Base64로 인코딩한다.

@PostConstruct

protected void init() {

secretKey = Base64.getEncoder().encodeToString(secretKey.getBytes());

}

// JWT 토큰 생성

public String createToken(String userPk, List<String> roles) {

Claims claims = Jwts.claims().setSubject(userPk); // JWT payload 에 저장되는 정보단위
claims.put("roles", roles); // 정보는 key / value 쌍으로 저장된다.

Date now = new Date();

return Jwts.builder()

.setClaims(claims) // 정보 저장

.setIssuedAt(now) // 토큰 발행 시간 정보

.setExpiration(new Date(now.getTime() + tokenValidTime)) // set Expire Time

.signWith(SignatureAlgorithm.HS256, secretKey) // 사용할 암호화 알고리즘과

// signature 에 들어갈 secret값 세팅

.compact();

}

package com.example.jwtProject2.jwt;

import lombok.RequiredArgsConstructor;

import org.springframework.stereotype.Component;

import io.jsonwebtoken.Claims;

import io.jsonwebtoken.Jws;

import io.jsonwebtoken.Jwts;

import io.jsonwebtoken.SignatureAlgorithm;

import lombok.RequiredArgsConstructor;

import org.springframework.security.authentication.UsernamePasswordAuthenticationToken;

import org.springframework.security.core.Authentication;

import org.springframework.security.core.userdetails.UserDetails;

import org.springframework.security.core.userdetails.UserDetailsService;

import org.springframework.stereotype.Component;

import javax.annotation.PostConstruct;

import javax.servlet.http.HttpServletRequest;

import java.util.Base64;

import java.util.Date;

import java.util.List;

// JWT 토큰에서 인증 정보 조회

public Authentication getAuthentication(String token) {

UserDetails userDetails = userDetailsService.loadUserByUsername(this.getUserPk(token));

return new UsernamePasswordAuthenticationToken(userDetails, "", userDetails.getAuthorities());

}

// 토큰에서 회원 정보 추출

public String getUserPk(String token) {

return Jwts.parser().setSigningKey(secretKey).parseClaimsJws(token).getBody().getSubject();

}

// Request의 Header에서 token 값을 가져옵니다. "X-AUTH-TOKEN" : "TOKEN값"

public String resolveToken(HttpServletRequest request) {

return request.getHeader("X-AUTH-TOKEN");

}

// 토큰의 유효성 + 만료일자 확인

public boolean validateToken(String jwtToken) {

try {

Jws<Claims> claims = Jwts.parser().setSigningKey(secretKey).parseClaimsJws(jwtToken);

return !claims.getBody().getExpiration().before(new Date());

} catch (Exception e) {

return false;

}

}

}

02. JWT Token + Authorization

JWT Filter

```
package com.example.jwtProject2.jwt;

import lombok.RequiredArgsConstructor;
import org.springframework.security.core.Authentication;
import org.springframework.security.core.context.SecurityContextHolder;
import org.springframework.web.filter.GenericFilterBean;

import javax.servlet.FilterChain;
import javax.servlet.ServletException;
import javax.servlet.ServletRequest;
import javax.servlet.ServletResponse;
import javax.servlet.http.HttpServletRequest;
import java.io.IOException;

//컴포넌트를 이용하여 인증 작업을 진행
//검증이 끝난 JWT로부터 유저 정보를 받아와서 UsernamePasswordAuthehenicationFilter로 전달
@RequiredArgsConstructor
public class JwtAuthenticationFilter extends GenericFilterBean {
    private final JwtTokenProvider jwtTokenProvider;

    @Override
    public void doFilter(ServletRequest request, ServletResponse response, FilterChain chain) throws IOException, ServletException {
        // 헤더에서 JWT 를 받아옵니다.
        String token = jwtTokenProvider.resolveToken((HttpServletRequest) request);
        // 유효한 토큰인지 확인합니다.
        if (token != null && jwtTokenProvider.validateToken(token)) {
            // 토큰이 유효하면 토큰으로부터 유저 정보를 받아옵니다.
            Authentication authentication = jwtTokenProvider.getAuthentication(token);
            // SecurityContext 에 Authentication 객체를 저장합니다.
            SecurityContextHolder.getContext().setAuthentication(authentication);
        }
        chain.doFilter(request, response);
    }
}
```

02. JWT Token + Authorization

CustomUserDetail

```
package com.example.jwtProject2.User;

import lombok.RequiredArgsConstructor;
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;

//유저 정보를 재정의
@RequiredArgsConstructor
@Service
public class CustomUserDetailService implements UserDetailsService {

    private final UserRepository userRepository;

    @Override
    public UserDetails loadUserByUsername(String username) throws UsernameNotFoundException {
        return (UserDetails) userRepository.findByEmail(username)
            .orElseThrow(() -> new UsernameNotFoundException("사용자를 찾을 수 없습니다."));
    }
}
```

02. JWT Token + Authorization

POST

▼

http://localhost:8080/login?email=a@b.com&password=aaaa

Params ●

Authorization

Headers (10)

Body ●

Pre-request Script

Tests

Settings

☐ none

☐ form-data

☐ x-www-form-urlencoded

☒ raw

☐ binary

☐ GraphQL

JSON ▼

```
1 {  
2   ... "email" : "a@b.com",  
3   ... "password" : "aaaa"  
4 }
```

Body

Cookies (1)

Headers (11)

Test Results

⌚ Status: 200 OK Time: 425 ms Size

Pretty

Raw

Preview

Visualize

```
eyJhbGciOiJIUzI1NiJ9.eyJzdWIiOiJhQGCIuY29tIiwicm9sZXMiOiJsIiwiaWF0IjoxNjMzOTQyODUwLCJpc2lwZXhwIjozMjE0MDAwMDAwfQ.eyJhbGciOiJIUzI1NiJ9.eyJzdWIiOiJhQGCIuY29tIiwicm9sZXMiOiJsIiwiaWF0IjoxNjMzOTQyODUwLCJpc2lwZXhwIjozMjE0MDAwMDAwfQ.
```