

06

OAuth2.0 인증서버 구현



6. OAuth2.0 인증서버 구현

● 인증 서버 구현 방법

- Spring Security 적용
 - Spring Boot + Security
 - 다양한 Token Store
 - ✓ InMemoryTokenStore
 - ✓ JdbcTokenStore
 - ✓ JwtTokenStore
- 라이브러리의 사용보다 인증 흐름을 이해하는 것이 중요함

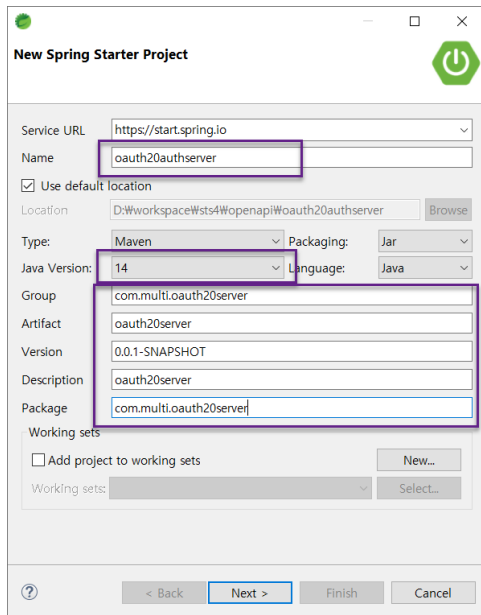
● OAuth 2.0 인증서버, 리소스 서버 작성

- 사용 도구 설정
 - postman 도구
 - curl : 윈도우는 <https://gs.saro.me/dev?tn=507> 을 참조하여 설치
 - Spring Tool Suites



6.1 프로젝트 초기화

● Spring Starter 프로젝트 생성



New Spring Starter Project

Service URL:

Name:

☒ Use default location

Location:

Type: Packaging:

Java Version: Language:

Group:

Artifact:

Version:

Description:

Package:

Working sets

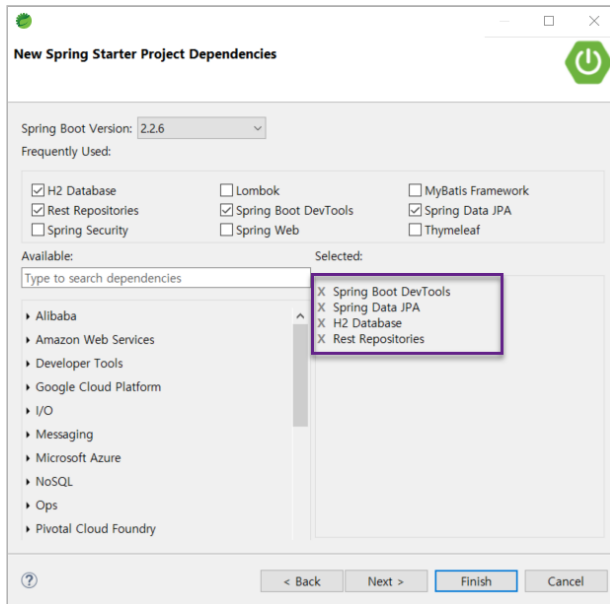
☐ Add project to working sets

Working sets:



6.1 프로젝트 초기화

● Spring Starter 프로젝트 생성



6.1 프로젝트 초기화

● DataSource 설정

- src/main/resources/application.properties

```
#data source settings
spring.datasource.url=jdbc:h2:tcp://localhost/~ /test
spring.datasource.username=sa
spring.datasource.password=
spring.datasource.driver-class-name=org.h2.Driver

#JPA settings
spring.jpa.database-platform=org.hibernate.dialect.H2Dialect
spring.jpa.hibernate.ddl-auto=update
spring.jpa.show-sql=true
spring.jpa.properties.hibernate.format_sql=true

#server settings
server.port=80
```

- DB Table 생성이 필요하다면 완성된 예제에서 다음 파일 참조
 - src/main/resources/ddl.sql.txt



6.1 프로젝트 초기화

- OAuth 2.0 액세스 토큰을 이용해 접근할 리소스

- 경로 : /contacts
- Domain 클래스 : Contact.java

```
package com.multi.oauth20server.domain;
.....
@Entity
public class Contact {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private long no;
    private String name;
    private String tel;
    private String address;

    .....
    //생성자, getter, setter 생략
}
```



6.1 프로젝트 초기화

- ContactRepository.java

```
package com.multi.oauth20server.dao;

import org.springframework.data.jpa.repository.JpaRepository;
import com.multi.oauth20server.domain.Contact;

public interface ContactRepository extends JpaRepository<Contact, Long>{
}
```



6.1 프로젝트 초기화

- ContactRestController.java

```
package com.multi.oauth20server;
.....
@RestController
@RequestMapping(value="/api")
public class ContactRestController {
    @Autowired
    ContactRepository contactRepository;

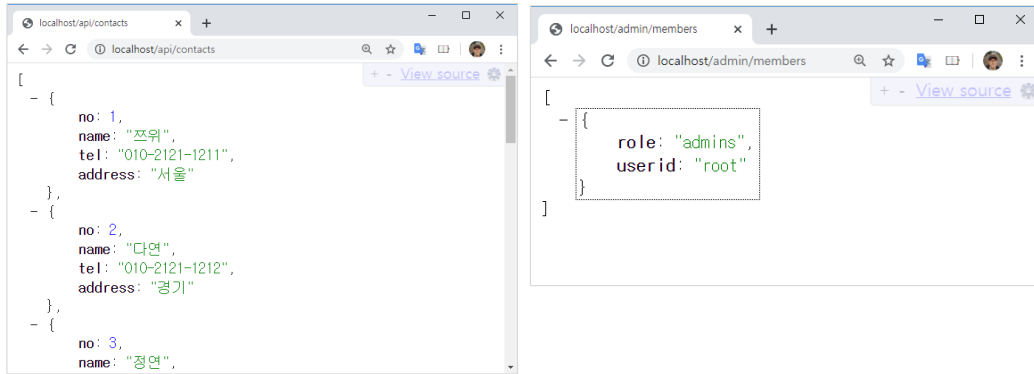
    @GetMapping(value="/contacts", produces = {"application/json"})
    public List<Contact> getContactList() {
        return contactRepository.findAll();
    }

    @GetMapping(value="/profiles", produces = {"application/json"})
    public HashMap<String, String> getProfile() {
        HashMap<String, String> profile = new HashMap<String, String>();
        profile.put("system", "Test Auth Server");
        profile.put("devenv", "Spring Boot 2.x");
        return profile;
    }
}
```



6.1 프로젝트 초기화

● 실행 결과



```

[
  {
    no: 1,
    name: "프워",
    tel: "010-2121-1211",
    address: "서울"
  },
  {
    no: 2,
    name: "다연",
    tel: "010-2121-1212",
    address: "경기"
  },
  {
    no: 3,
    name: "정연",
  }
]

```

```

[
  {
    role: "admins",
    userid: "root"
  }
]

```



6.2 JDBC Token Store 적용

● 의존성 추가

```
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-security</artifactId>
</dependency>
<dependency>
  <groupId>org.springframework.security.oauth.boot</groupId>
  <artifactId>spring-security-oauth2-autoconfigure</artifactId>
  <version>2.2.5.RELEASE</version>
</dependency>
```

● 사용자 인증 설정

- inmemory 사용자 계정 설정
- 각 경로별 접근 권한 설정
 - /api/contacts/**: contacts scope가 있어야 접근 허용
 - /api/profiles/**: profiles scope가 있어야 접근 허용
- Form Login 화면 사용



6.2 JDBC Token Store 적용

● JdbcTokenStore

- 토큰을 관계형 데이터베이스에 저장할 수 있도록...
 - <https://github.com/spring-projects/spring-security-oauth/blob/master/spring-security-oauth2/src/test/resources/schema.sql>
 - H2, Oracle 등에 사용 가능. 다른 데이터베이스에 적용하기 위해서는 구문 변경이 필요함

● BCryptPasswordEncoder Bean 등록

```
@SpringBootApplication
@Configuration
public class OAuth2authserverApplication {
    public static void main(String[] args) {
        SpringApplication.run(OAuth2authserverApplication.class, args);
    }
    @Bean
    public BCryptPasswordEncoder passwordEncoder() {
        return new BCryptPasswordEncoder();
    }
}
```

- BCryptPasswordEncoder는 패스워드와 같은 중요한 정보를 데이터베이스에 저장할 때 단순 해시가 아닌 salt가 적용된 해시를 만들어 저장할 수 있도록 하는 라이브러리이다.

6.2 JDBC Token Store 적용

● OAuth2SecurityConfig.java 추가

```
package com.multi.oauth20server;
.....
@Configuration
@EnableWebSecurity
public class OAuth2SecurityConfig extends WebSecurityConfigurerAdapter {
    @Autowired
    BCryptPasswordEncoder passwordEncoder;

    @Override
    protected void configure(HttpSecurity http) throws Exception {
        http.authorizeRequests()
            .antMatchers("/oauth/**", "/oauth/token", "/login**").permitAll()
            .anyRequest().authenticated();
        http.formLogin();
    }

    @Override
    protected void configure(AuthenticationManagerBuilder auth) throws Exception {
        auth.inMemoryAuthentication()
            .withUser("root").password(passwordEncoder.encode("1234")).roles("admins", "users")
            .and().withUser("user1").password(passwordEncoder.encode("1234")).roles("users")
            .and().withUser("user2").password(passwordEncoder.encode("1234")).roles("users");
    }
}
```

■ WebSecurityConfigurerAdapter 클래스를 상속받아 작성함

- 이 클래스는 일반적인 사용자 인증과 접근 제어를 위해 사용함.
- OAuth2 인증에서 authorize를 위해서는 사용자 인증이 선행되어야 함

■ 사용자 계정은 inmemory로 처리하였음

- 사용자 암호는 BcryptPasswordEncoder를 이용해 Memory에 저장하도록 함
- 계정 정보를 데이터베이스로 처리하는 방법은 "AuthenticationManagerBuilder UserDetailsService"와 같은 키워드로 검색하도록 함
- 이번 과정에서는 사용자 인증이 주 목적이 아니므로 inmemory로 처리하도록 간단히 구현하였음

■ 접근 권한은 다음과 같음

- /oauth로 시작하는 경로 /login으로 시작하는 경로는 모든 사용자 허용함
- 나머지 경로는 인증된 사용자만 접근함
- Basic 인증이 아닌 form 인증을 수행하도록 formLogin() 메서드 호출

6.2 JDBC Token Store 적용

● OAuth2AuthServerConfig.java

```
package com.multi.oauth20server;
.....
@Configuration
@EnableAuthorizationServer
public class OAuth2AuthServerConfig extends AuthorizationServerConfigurerAdapter {
    @Autowired
    DataSource dataSource;
    @Autowired
    BCryptPasswordEncoder passwordEncoder;

    public TokenStore tokenStore() {
        return new JdbcTokenStore(dataSource);
    }

    public ApprovalStore approvalStore() {
        return new InMemoryApprovalStore();
    }

    public AuthorizationCodeServices authorizationCodeServices() {
        return new InMemoryAuthorizationCodeServices();
    }
}
```

(다음페이지에 이어짐)

- 승인 처리와 authorization_code의 처리는 inmemory로 사용하도록 처리함
 - 일반적으로 이 두가지 작업은 짧은 시간안에 이루어지므로 굳이 데이터베이스를 이용하지 않아도 될 것이라 생각됨
 - 만일 인증서버를 여러 대를 배치하고 로드 밸런싱한다면 동일한 정보를 공유해야 한다. 이런 경우에는 둘 다 JdbcApprovalStore, JdbcAuthorizationCodeServices 클래스를 이용하도록 변경하면 됨

6.2 JDBC Token Store 적용

```

@Override
public void configure(AuthorizationServerEndpointsConfigurer endpoints) throws Exception {
    endpoints
        .tokenStore(tokenStore())
        .approvalStore(approvalStore())
        .authorizationCodeServices(authorizationCodeServices());
}

@Override
public void configure(AuthorizationServerSecurityConfigurer oauthServer) throws Exception {
    oauthServer.tokenKeyAccess("permitAll()")
        .checkTokenAccess("isAuthenticated()")
        .allowFormAuthenticationForClients();
}

@Override
public void configure(ClientDetailsServiceConfigurer clients) throws Exception {
    clients.inMemory().withClient("client1").secret(passwordEncoder.encode("1234"))
        .authorizedGrantTypes("authorization_code", "implicit", "password",
            "client_credentials", "refresh_token")
        .scopes("contacts", "profiles", "messages").authorities("TEST_CLIENT")
        .redirectUris("http://jcnor.com:8080/callback", "http://localhost:8080/callback")
        .accessTokenValiditySeconds(3600)
        .refreshTokenValiditySeconds(0);
}
}

```



6.2 JDBC Token Store 적용

● OAuth2ResourceServerConfig.java

```
package com.multi.oauth20server;
.....
@EnableResourceServer
@Configuration
public class OAuth2ResourceServerConfig extends ResourceServerConfigurerAdapter {
    @Autowired
    DataSource dataSource;

    public TokenStore tokenStore() {
        return new JdbcTokenStore(dataSource);
    }
    @Override
    public void configure(HttpSecurity http) throws Exception {
        http.csrf().disable()
            .requestMatchers()
            .antMatchers("/api/**")
            .and().authorizeRequests()
            .antMatchers("/api/contacts").access("#oauth2.hasScope('contacts')")
            .antMatchers("/api/profiles").access("#oauth2.hasScope('profiles')")
            .and().exceptionHandling()
            .accessDeniedHandler(new OAuth2AccessDeniedHandler());
    }
}
```

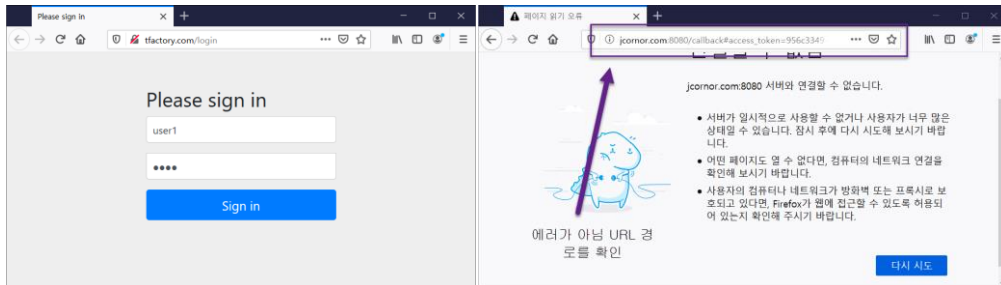
■ 리소스 서버의 설정 코드이다

- 이 코드는 추후 JWT Token이 적용되면 ContactRestController와 함께 다른 서버로 분리될 예정임
- 리소스 서버의 권한 적용은 /api로 시작하는 경로로 한정됨
- /api/contacts로 접근할 때는 contacts scope가 존재해야만 함
- /api/profiles로 접근할 때는 profiles scope가 존재해야만 함

6.2 JDBC Token Store 적용

● 실행 후 테스트

- implicit grant
 - 브라우저를 열고 다음의 경로로 요청
 - `http://tfactory.com/oauth/authorize?response_type=token&client_id=client1&scope=contacts%20messages&redirect_uri=http://jcnor.com:8080/callback`

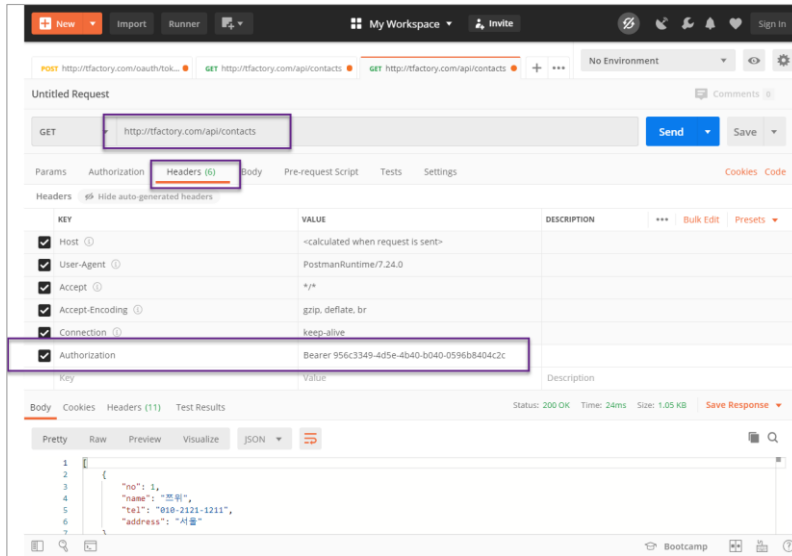


`http://jcnor.com:8080/callback#access_token=956c3349-4d5e-4b40-b040-0596b8404c2c&token_type=bearer&expires_in=3398`



6.2 JDBC Token Store 적용

- 발급받은 access_token으로 리소스 접근
 - postman 도구를 이용함

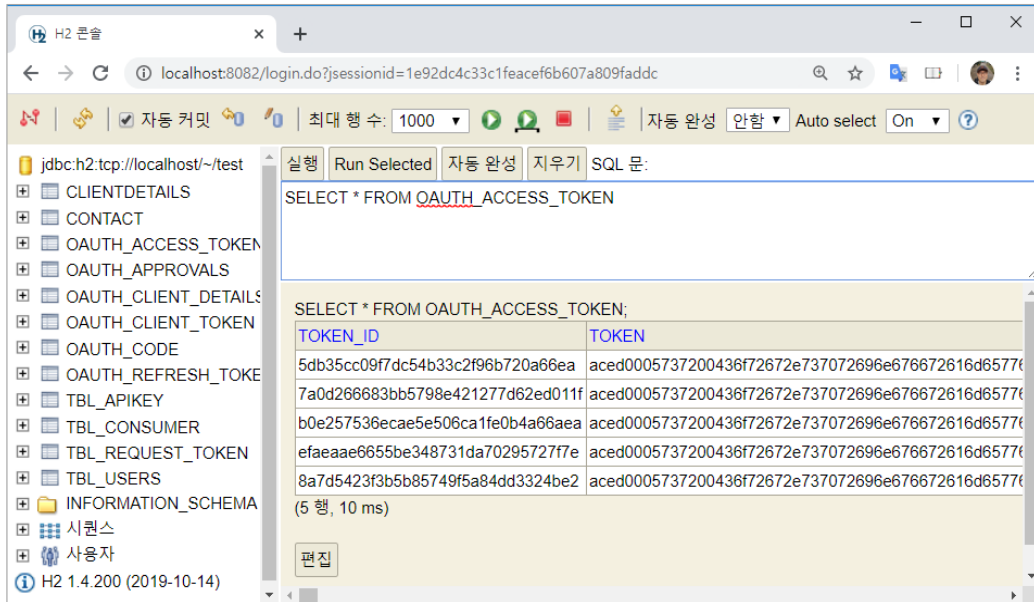


- 잘못된 token을 사용하면 다음과 같은 오류 메시지를 확인할 수 있다.

```
{
  "error": "invalid_token",
  "error_description": "Invalid access token: 956c3349-4d5e-4b40-b040-XXXXXXXXXX"
}
```

6.2 JDBC Token Store 적용

● 데이터베이스에서 생성된 access token 확인



The screenshot shows the H2 console interface. The left sidebar lists database schemas, including 'OAUTH_ACCESS_TOKEN'. The main area displays the SQL query 'SELECT * FROM OAUTH_ACCESS_TOKEN;' and its results. The results are shown in a table with two columns: 'TOKEN_ID' and 'TOKEN'.

TOKEN_ID	TOKEN
5db35cc09f7dc54b33c2f96b720a66ea	aced0005737200436f72672e737072696e676672616d65776
7a0d266683bb5798e421277d62ed011f	aced0005737200436f72672e737072696e676672616d65776
b0e257536ecae5e506ca1fe0b4a66aea	aced0005737200436f72672e737072696e676672616d65776
efaeaae6655be348731da70295727f7e	aced0005737200436f72672e737072696e676672616d65776
8a7d5423f3b5b85749f5a84dd3324be2	aced0005737200436f72672e737072696e676672616d65776

The query executed was: `SELECT * FROM OAUTH_ACCESS_TOKEN;`
 (5 행, 10 ms)



6.3 JWT TokenStore 적용

● 의존성 추가

```
<dependency>
  <groupId>org.springframework.security</groupId>
  <artifactId>spring-security-jwt</artifactId>
  <version>1.1.0.RELEASE</version>
</dependency>
```

● JWT는 Token을 데이터베이스 등에 저장하지 않음

- Converter를 이용해서 서명정보와 함께 생성함
- 이 과정에서는 HS-256 알고리즘을 적용함
 - 대칭키 방식: 동일한 비밀키를 인증서버와 리소스 서버가 공유해야 함
- 다음과 같은 경우라면 RS-256, RS-384, RS-512 방식을 권장함
 - 대규모 시스템: 인증서버 + 다수의 리소스 서버
 - 리소스 서버가 타사의 서버, 타 사업부의 서버가 될 가능성이 있는 경우



6.3 JWT TokenStore 적용

● OAuth2AuthServerConfig.java 변경

```
package com.multi.oauth2server;
.....
@Configuration
@EnableAuthorizationServer
public class OAuth2AuthServerConfig extends AuthorizationServerConfigurerAdapter {
    .....
    public JwtAccessTokenConverter accessTokenConverter() {
        JwtAccessTokenConverter converter = new JwtAccessTokenConverter();
        converter.setSigningKey("SECRET KEY");
        return converter;
    }
    public TokenStore tokenStore() {
        return new JwtTokenStore(accessTokenConverter());
    }
    .....
    @Override
    public void configure(AuthorizationServerEndpointsConfigurer endpoints) throws Exception {
        endpoints
            .tokenStore(tokenStore()).accessTokenConverter(accessTokenConverter())
            .approvalStore(approvalStore())
            .authorizationCodeServices(authorizationCodeServices());
    }
    .....
}
```

■ 다음과 같이 코드를 변경한다.

- dataSource 멤버 필드를 삭제한다.
- tokenStore() 메서드가 사용하는 accessTokenConverter() 메서드를 추가한다.
- tokenStore() 메서드를 변경한다. JwtTokenStore를 사용하도록 변경한다.
- configure 메서드에서 accessTokenConverter를 등록한다.

6.3 JWT TokenStore 적용

● OAuth2ResourceServerConfig.java

```
package com.multi.oauth20server;
.....
@EnableResourceServer
@Configuration
public class OAuth2ResourceServerConfig extends ResourceServerConfigurerAdapter {
    public TokenStore tokenStore() {
        return new JwtTokenStore(accessTokenConverter());
    }
    @Bean
    public JwtAccessTokenConverter accessTokenConverter() {
        JwtAccessTokenConverter converter = new JwtAccessTokenConverter();
        converter.setSigningKey("SECRET KEY");
        return converter;
    }
    @Bean
    public DefaultTokenServices tokenServices() {
        DefaultTokenServices defaultTokenServices = new DefaultTokenServices();
        defaultTokenServices.setTokenStore(tokenStore());
        return defaultTokenServices;
    }
    @Override
    public void configure(ResourceServerSecurityConfigurer config) {
        config.tokenServices(tokenServices());
    }
    .....
}
```

- 리소스 서버에서는 Token을 받아서 서명 검증을 할 것이며, 따라서 사용자 인증 정보 등을 이용할 필요가 없음
 - 인증서버와 동일한 비밀키를 사용함

6.3 JWT TokenStore 적용

- <https://jwt.io> 에서 토큰 확인

The screenshot shows the JWT.io website interface. On the left, under the 'Encoded' tab, a long string of characters is pasted into the 'PASTE A TOKEN HERE' field. A purple box highlights this string, and a purple arrow points from it to the 'Decoded' tab. On the right, under the 'Decoded' tab, the token is broken down into three parts: a header, a payload, and a signature. The header is an object with 'alg' set to 'HS256' and 'typ' set to 'JWT'. The payload is an object containing 'exp', 'user_name', 'authorities', 'jti', 'client_id', 'scope', 'messages', and 'contacts'. The signature is a long string of characters. Below the payload, the 'VERIFY SIGNATURE' section shows the formula for verifying the signature: `HMACSHA256(base64UrlEncode(header) + "." + base64UrlEncode(payload), your-256-bit-secret)`.



6.3 JWT TokenStore 적용

● 리소스 접근

■ 잘못된 토큰 사용시

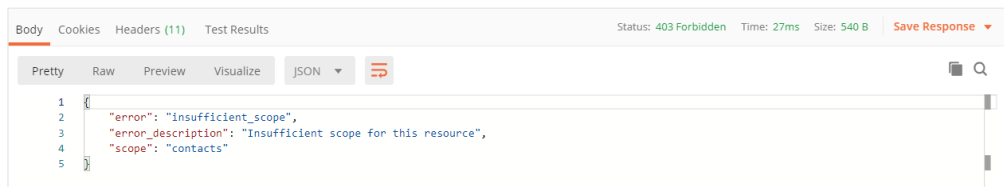


Body Cookies Headers (11) Test Results Status: 401 Unauthorized Time: 11ms Size: 519 B Save Response ▾

Pretty Raw Preview Visualize JSON ▾

```
1 {
2   "error": "invalid_token",
3   "error_description": "Cannot convert access token to JSON"
4 }
```

■ scope에 해당되지 않을 때



Body Cookies Headers (11) Test Results Status: 403 Forbidden Time: 27ms Size: 540 B Save Response ▾

Pretty Raw Preview Visualize JSON ▾

```
1 {
2   "error": "insufficient_scope",
3   "error_description": "Insufficient scope for this resource",
4   "scope": "contacts"
5 }
```



6.3 JWT TokenStore 적용

● 리소스 서버의 기능을 모두 별도의 프로젝트로 이동

- oauth20resourceserver 프로젝트
 - oauth20authserver 프로젝트의 pom.xml의 dependency를 복사
 - application.properties 파일을 복사 후 server.port를 8000으로 변경
- 다음의 요소를 새 프로젝트 이동
 - dao, domain 패키지
 - ContactRestController
 - OAuth2ResourceServerConfig

● 서버 주소

- 인증서버 : <http://tfactory.com>
- 리소스 서버 : <http://server.com:8000>

