1. DI 추가

Setup new Spring Boot project

Use Spring web tool or your development tool (Spring Tool Suite, Eclipse, Intellij) to create a Spring Boot project.

Then open **pom.xml** and add these dependencies:

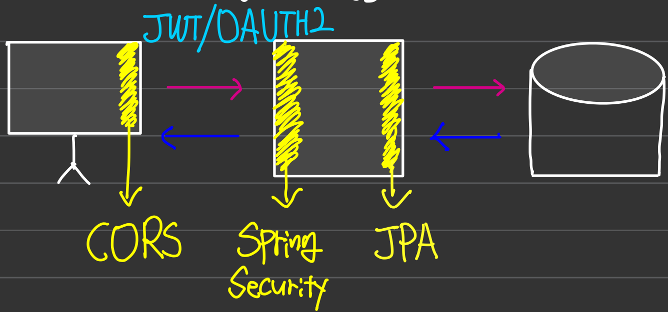
```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-data-jpa</artifactId>
</dependency>

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

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

<dependency>
    <groupId>io.jsonwebtoken</groupId>
    <artifactId>jjwt</artifactId>
    <version>0.9.1</version>
</dependency>
```

We also need to add one more dependency.
– If you want to use **PostgreSQL**:

DB에 따라 변경

```
<dependency>
    <groupId>org.postgresql</groupId>
    <artifactId>postgresql</artifactId>
    <scope>runtime</scope>
</dependency>
```

보안(로그인, 개인정보 등 민감사항) 위해 Spring Security + JWT 를 DI



기존방식 : 세션 + 쿠키 방식 (ex 공인인증서)
⇒ SOAP : Connectionless / Stateless / XML 기반 / HTTP(프로토콜)

↓ 변환

RESTful : JWT + Spring Security 방식 (ex 카카오뱅크)
· Stateful / JSON 기반

↳ Re Presentational State

대표상태 : '키' 값 Key : { [k{v}, k{v}, ...] ....} ⇒ JWT 방식

Key ↓ Token

value (보안된 상태로 전송)

데이터소스 / value (보안되는 부분) → Payload

* Payload vs Parameter

서버 ⇄ 클라이언트 (JAVA / JS) — 이거랑 시스템에서 전달

서버 ⇄ 서버 (J / J), 클라이언트 ⇄ 클라이언트 (JS / JS) — 얘은이랑 내에서 전달

Repository

```java
package com.example.demo.uss.repository;

import java.util.Optional;

import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.stereotype.Repository;

import com.example.demo.uss.domain.User;
import com.example.demo.uss.domain.UserDto;

interface UserCustomRepository{
        public UserDto login(String username, String password);
}
public interface UserRepository extends JpaRepository<User, Long>,
                                        UserCustomRepository{

        public UserDto findByUsername(String username);
}
```

component

```java
package com.example.demo.uss.repository;

import org.springframework.stereotype.Repository;

import com.example.demo.uss.domain.User;
import com.example.demo.uss.domain.UserDto;

import java.util.List;
import java.util.Optional;

import org.springframework.data.domain.Example;
import org.springframework.data.domain.Page;
import org.springframework.data.domain.Pageable;
import org.springframework.data.domain.Sort;
import org.springframework.data.jpa.repository.JpaRepository;
@Repository
public class UserRepositoryImpl implements UserCustomRepository{

        @Override
        public UserDto login(String username, String password) {
                return null;
        }



}
```

Service

```java
package com.example.demo.uss.service;

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

import com.example.demo.uss.domain.UserDto;



public interface UserService extends UserDetailsService {

        public UserDto login(String username, String password);
}
```

Controller

```java
package com.example.demo.sec.controller;

import java.util.HashMap;
import java.util.Map;

import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

import com.example.demo.uss.domain.UserDto;

@RestController
@RequestMapping("/auth")
public class AuthController {
        @PostMapping("/login")
        public void login(@RequestBody UserDto user) {
                Map<String, Object> map = new HashMap<>();

        }

}
```

```java
package com.example.demo.sec.config;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.context.annotation.Configuration;
import org.springframework.security.config.annotation.web.builders.HttpSecurity;
import org.springframework.security.config.annotation.web.configuration.WebSecurityConfigurerAdapter;
import org.springframework.security.config.http.SessionCreationPolicy;
import org.springframework.security.web.authentication.UsernamePasswordAuthenticationFilter;

import com.example.demo.sec.domain.JwtTokenProvider;
import com.example.demo.sec.filter.CustomAuthenticationEntryPoint;
import com.example.demo.sec.filter.JwtAuthenticationFilter;
@Configuration
public class WebSecurityConfigurer extends WebSecurityConfigurerAdapter{
        @Autowired JwtTokenProvider jwtTokenProvider;
        @Override
        protected void configure(HttpSecurity http) throws Exception {
                http.httpBasic().disable()
                .csrf().disable()
                .sessionManagement().sessionCreationPolicy(SessionCreationPolicy.STATELESS)
                .and()
                        .authorizeRequests()
                        .antMatchers("/*/login", "/*/signUp").permitAll()
                        .anyRequest().hasRole("USER")  // 그외 나머지 요청은 모두 인증된 회원만 접근 가능
                .and()
                        .exceptionHandling().authenticationEntryPoint(new CustomAuthenticationEntryPoint())
                .and()
                        .addFilterBefore(new JwtAuthenticationFilter(jwtTokenProvider), UsernamePasswordAuthenticationFilter.class);

        }
}
```

```java
package com.example.demo.sec.domain;

import java.util.Collection;

public class AbstractAuthenticationToken implements Authentication   {
// CredentialsContainer
        @Override
        public String getName() {
                // TODO Auto-generated method stub
                return null;
        }
/*
        @Override
        public Collection<? extends GrantedAuthority> getAuthorities() {

                return null;
        }
*/
        @Override
        public Object getCredentials() {
                // TODO Auto-generated method stub
                return null;
        }

        @Override
        public Object getDetails() {
                // TODO Auto-generated method stub
                return null;
        }

        @Override
        public Object getPrincipal() {
                // TODO Auto-generated method stub
                return null;
        @Override
        public boolean isAuthenticated() {
                // TODO Auto-generated method stub
                return false;
        }

        @Override
        public void setAuthenticated(boolean isAuthenticated) throws IllegalArgumentException {
                // TODO Auto-generated method stub

        }
}
```

```java
package com.example.demo.sec.domain;

import java.io.Serializable;
import java.security.Principal;
import java.util.Collection;

public interface Authentication extends Principal, Serializable{
        // 현재 사용자의 권한 목록을 가져옴
  //   Collection<? extends GrantedAuthority> getAuthorities();

    // credentials(주로 비밀번호)을 가져옴
    Object getCredentials();

    Object getDetails();

    // Principal 객체를 가져옴.
    Object getPrincipal();

    // 인증 여부를 가져옴
    boolean isAuthenticated();

    // 인증 여부를 설정함
    void setAuthenticated(boolean isAuthenticated) throws IllegalArgumentException;
}
```

→ 스트링

```java
@RequiredArgsConstructor
@Component
public class JwtTokenProvider {
        @Value("spring.jwt.secret")
    private String SECRET_KEY;
    private long tokenValidMilisecond = 1000L * 60 * 60; // 1시간만 토큰 유효
    private final UserDetailsService userDetailsService;
    @PostConstruct
    protected void init() {
        SECRET_KEY = Base64.getEncoder().encodeToString(SECRET_KEY.getBytes());
    }
    public String createToken(String userPk, Collection<? extends GrantedAuthority> roles)
        Claims claims = Jwts.claims().setSubject(userPk);
        claims.put("roles", roles);
        Date now = new Date();

        return Jwts.builder()
                .setClaims(claims)
                .setIssuedAt(now)
                .setExpiration(new Date(now.getTime() + tokenValidMilisecond))
                .signWith(SignatureAlgorithm.HS256, SECRET_KEY)
                .compact()
                ;

    }


}
```

→ 토큰 생성자

@Value는 application.property에서 조정

@PostConstruct 의존성 주입이 이루어진 후 초기화 수행 @

64진수의 키 값

Wild Card

Claims: Payload의 정보 한조각

builder(): 빌더패턴 : 복합객체 생성과정, 표현방법 분리 다른 표현 결과를 만드는 패턴

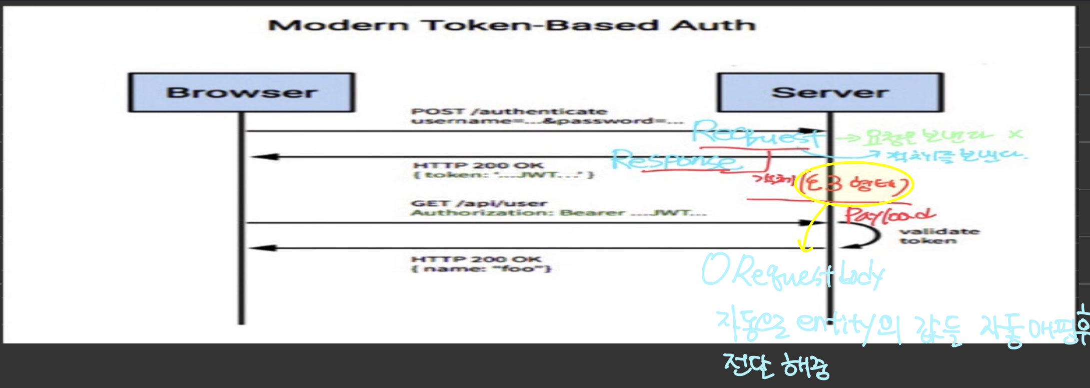
. 으로 연속적으로 다음메서드 사용가능

Jwt J = new Jwt() 대신 사용하는 것

```javascript
..
methods: {
  onSubmit (payload) {
    const { title, content } = payload
    console.log('BoardModifyPage payload: ' + payload)
    axios.put(`http://localhost:7777/boards/${this.boardNo}`, { title, content })
      .then(res => {
        alert('Modify Success')
        console.log('res: ' + res.data)
        this.$router.push({
          name: 'BoardReadPage',
          params: { boardNo: res.data.boardNo.toString() }
        })
      })
      .catch(err => {
        alert(err.response.data.message)
      })
  },
  ...mapActions([
    'fetchBoard'
  ])
}
}
```

구조분해 할당

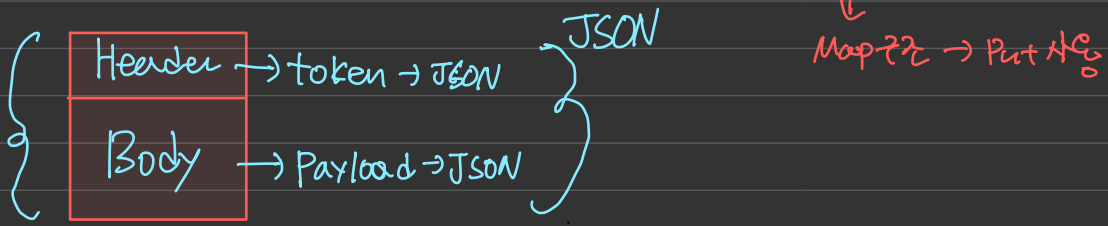
Entity 일치



API 서버는 로그인 요청이 완료되면 클라이언트에게 회원을 구분할 수 있는 정보를 담은 JWT를 생성하여 전달합니다. 그러면 클라이언트는 이 JWT를 헤더에 담아서 요청을 하게 됩니다. 권한이 필요한 요청이 있을 때 마다 API 서버는 헤더에 담긴 JWT 값을 확인하고 권한이 있는 사용자인지 확인하고 리소스를 제공하게 됩니다.

이렇게 기존의 세션-쿠키 기반의 로그인이 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 장점이 있습니다.

이 때 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.



추상팩토리 → @Component