Spring Boot 기반의 요구사항정의서 수행 미션형 프로젝트 01

강경미(carami@nate.com), 김성박(urstory@gmail.com)

Blog 플랫폼 만들기

- Spring Boot 를 이용한 웹 애플리케이션 개발 프로젝트
- https://velog.io/ 와 유사한 플랫폼을 직접 개발합니다.
    - 참고로 velog 사이트는 프론트엔드 서버와 백엔드 서버로 나뉩니다. 우리는 하나의 서버가 Thymeleaf 를 이용해 뷰를 만들고 API 를 모두 제공하도록 만듭니다.
- Spring Boot MVC 의 Template 엔진(Thymeleaf)을 이용해 View 를 구성하고 바닐라 JavaScript 로 API 를 호출하여 동적으로 동작하는 기능을 구현합니다.
    - 자바스크립트로 API 를 호출하여 목록을 보여주거나, email 이 존재하는지 검사하거나 등을 구현할 수 있습니다.
- Spring Data JPA, Spring MVC 를 이용해 구현한다.

요구사항

- https://velog.io/ 사이트를 보면서 어떻게 유사한 플랫폼을 만들 수 있는지 고민합니다. 동일하게 만들기에는 프론트엔드 기술이 필요합니다. 우리가 아는 기술들을 이용해 최대한 유사하게 플랫폼을 만들어 보는 것이 이번 프로젝트의 목적입니다.

요구사항 (블로그)

- 회원 1 명은 1 개의 Blog 를 가질 수 있습니다.
- 회원가입, 로그인 기능을 제공합니다.
- http://도메인 는 사용자들이 작성한 최신 글, 인기 글을 볼 수 있습니다. (정렬방식에 따라 최신, 인기글을 볼 수 있습니다.)
- http://도메인에서 보여주는 블로그 글에는 제목, 내용일부, 작성자사진, 작성자 아이디, 좋아요 수가 보여집니다.
- http://도메인/@carami 는 carami 아이디의 사용자의 블로그 페이지입니다.
- http://도메인/@아이디 를 가면 좌측에 tag 목록이 나오고 tag 에 글의 수가 보여집니다.
- http://도메인/@아이디 를 가면 글 목록, 시리즈 목록, 소개를 볼 수 있습니다.

요구사항 (블로그 포스트)

- 사용자는 임시 글을 작성할 수 있습니다.
- 사용자는 글을 작성하자 마자 즉시 출간할 수 있습니다.
- 사용자는 임시 글 목록을 볼 수 있습니다.
- 글을 작성할 때 이미지, URL 등을 포함시킬 수 있습니다.

- 글은 제목, 내용, 태그들을 작성할 수 있고 자동으로 작성일이 지정됩니다. (제목, 내용은 필수 입니다.)
- 임시 글 목록에서 임시 글을 삭제할 수 있습니다.
- 임시 글은 수정할 수 있습니다. 수정 후 바로 출간할 수 있습니다.

요구사항 (좋아요, 팔로우)

- 사용자는 블로그에서 “좋아하기”를 할 수 있습니다.
- 좋아하기를 선택한 블로그 글들만 모아서 볼 수 있습니다.
- 사용자가 읽은 글들만 모아서 볼 수 있습니다.
- 팔로우 한 사용자 목록을 볼 수 있습니다. 언팔로우 할 수 있습니다.

요구사항 (개인정보)

- 사용자는 본인의 프로필 이미지를 등록할 수 있습니다.
- 본인이 등록한 프로필을 삭제할 수 있습니다.
- 블로그 제목을 설정할 수 있습니다. 설정하지 않으면 기본적으로 사용자 아이디가 됩니다.
- 이메일 주소를 변경할 수 있습니다. (회원 가입시 등록할 수 있습니다.)
- 이메일 수신 설정을 변경할 수 있습니다. (댓글 알림, 업데이트 소식 알림)
- 회원 탈퇴 기능을 제공합니다.

요구사항 (출간)

- 임시 글을 출간하거나, 바로 출간하기를 하게 되면 포스트 미리보기 이미지를 등록할 수 있습니다.
- 임시 글을 출간하거나, 바로 출간하기를 하게 될 때 글의 제목과 내용을 일부 보여줍니다. 전체공개로 올릴지 비공개로 올릴지를 결정할 수 있습니다.
- 해당 글의 URL 은 /@아이디/posts/제목이 됩니다. 제목은 URLEncoding 으로 인코딩 되어 있어야 합니다.

- 임시 글을 출간하거나, 바로 출간하기를 하게 될 때 시리즈에 추가할 수 있습니다.

요구사항 (블로그 글보기)

- 블로그 글 보기 기능을 제공합니다.
- 비공개 글의 경우 비공개 표시를 제공합니다.
- 내 글이 아닌 경우 좋아요를 할 수 있습니다.
- 내 글의 경우 통계정보를 볼 수 있습니다.
- 내 글의 경우 수정할 수 있습니다.
- 내 글의 경우 삭제할 수 있습니다.
- 블로그를 작성한 사람의 프로필 이미지와 아이디를 하단에 보여줍니다.
- 다른 사람의 글의 경우 팔로우를 할 수 있습니다.
- 해당 사용자의 이전 블로그글, 이후 블로그글에 대한 링크를 볼 수 있습니다.

요구사항 (댓글)

- 블로그에 사용자는 댓글을 작성할 수 있습니다.
- 블로그 글보기를 하면 댓글의 수가 표시됩니다.
- 댓글에 답글을 작성할 수 있습니다.
- 댓글을 삭제할 수 있습니다.

요구사항 (관리자)

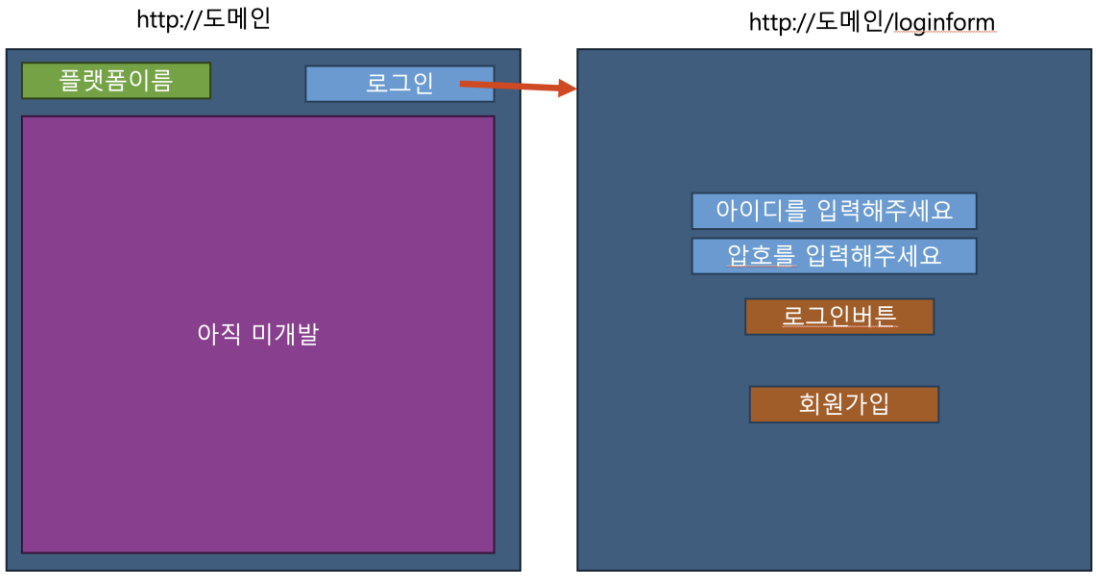
- 관리자 페이지로 가면 모든 포스팅된 글 목록을 볼 수 있습니다.
- 관리자는 어떤 글이든 삭제할 수 있습니다.

개발 - 회원 가입, 로그인

---

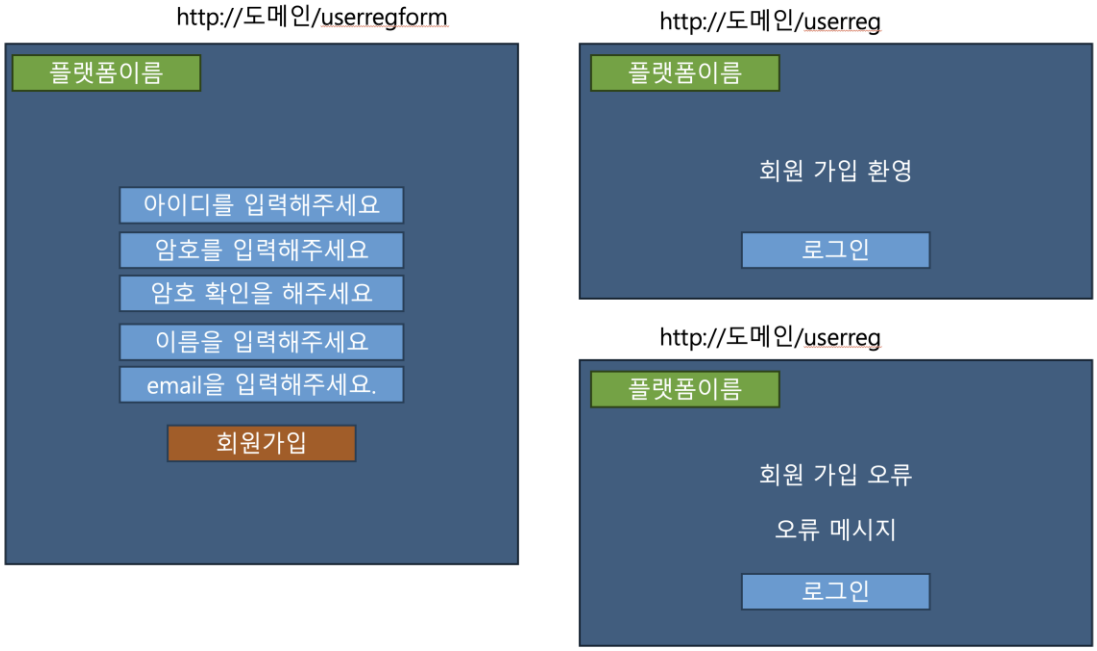
1. 메인 페이지 작성

- 메인 페이지에 로그인 링크를 클릭하면 http://도메인/loginform 이 보여진다.
- /loginform 에서는 아이디와 암호를 입력해 로그인 하거나 회원가입 링크를 클릭해서 회원가입을 진행한다.
- 이 부분은 velog 와는 다르다.



height:500

- 회원 가입을 누르게 되면 회원 정보를 입력합니다.
- Javascript 로 값에 대한 검증을 합니다.
- 아이디 중복, email 중복 여부를 Javascript 와 API 연동을 통해 검사를 해줄 수도 있습니다.
- 회원 정보는 POST 방식으로 /userreg 에게 전달됩니다.
- /userreg 는 항상 값에 대한 검증, 로직 실행 순서로 진행합니다.



height:500

- /loginform 에서 아이디와 암호를 입력하고 로그인버튼을 클릭하면 로그인 됩니다.
- 로그인을 하면 Cookie 정보가 브라우저에 전달됩니다.
- 로그인 이후에는 api 를 호출하거나 URL 이 호출될 때마다 로그인 쿠키가 전달됩니다.
- 로그인 쿠키 정보를 참고로 사용자를 검증합니다.
- 서버가 2 대 라운드로빈 방식으로 ALB 에 위해 제공된다고 생각합시다. 이 경우 세션을 사용하면 어떻게 될까요? 쿠키를 사용할때와 세션을 이용할 때 어떤 문제가 있을까요? 그리고 이런 문제를 해결하려면 어떻게 해야할까요? 해결방법이 아키텍처의 복잡도를 높이진 않을까요? 아키텍처의 복잡도가 높아진다면 어떤 문제가 발생할까요?

회원 정보를 위한 테이블 생성

- 사용자와 권한은 다대다 관계입니다.
- 권한 테이블에는 미리 권한정보를 insert 합니다. 권한 정보는 일반사용자와 관리자 2 개를 가집니다.

---

1. 테이블 생성 스크립트

users 테이블

```
CREATE TABLE users (
    id BIGINT AUTO_INCREMENT PRIMARY KEY,
    username VARCHAR(50) NOT NULL UNIQUE,
    password VARCHAR(100) NOT NULL,
    name VARCHAR(100) NOT NULL,
    email VARCHAR(100) NOT NULL,
    registration_date TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

- id : 정수값
- username : 사용자 로그인 id
- password : 암호화된 암호
- name : 사용자 이름
- email : 사용자 email
- registration_date : 등록일

---

roles 테이블

```
CREATE TABLE roles (
    id BIGINT AUTO_INCREMENT PRIMARY KEY,
    name VARCHAR(50) NOT NULL UNIQUE
);
```

---

user_roles 테이블(다대다 관계)

```
CREATE TABLE user_roles (
    user_id BIGINT NOT NULL,
    role_id BIGINT NOT NULL,
    PRIMARY KEY (user_id, role_id),
    FOREIGN KEY (user_id) REFERENCES users(id) ON DELETE CASCADE,
    FOREIGN KEY (role_id) REFERENCES roles(id) ON DELETE CASCADE
);
```

---

2. 데이터 삽입 스크립트

권한 데이터 삽입(roles 테이블)

```
INSERT INTO roles (name) VALUES ('ROLE_USER'), ('ROLE_ADMIN');
```

---

3. 회원 가입시 실행되야 할 SQL

- 사용자 정보 등록

```
INSERT INTO users (username, password, name, email, registration_date) VALUES
 (?, ?, ?, ?, ?);
```

insert 가 되면서 자동 생성된 id 가 권한 정보 등록될 때 사용되야 한다.

- 권한 정보 등록

```
-- Retrieve role ID for ROLE_USER
SELECT id INTO @ROLE_USER_ID FROM roles WHERE name = 'ROLE_USER';

-- Insert user role relationship
INSERT INTO user_roles (user_id, role_id) VALUES (?, @ROLE_USER_ID);
```

- 회원 가입 정보를 등록할 때 사용자 아이디, 이메일이 중복되는지 확인하기 위해선 @RestController 가 사용됩니다.
- 회원 정보를 입력받아 저장하기 위해선 @Controller 가 사용됩니다.
- Spring Data JPA 를 이용해 엔티티를 작성하고, 이를 이용하는 Repository, Service, Controller 를 작성한다.

API 명세

UserRestController API 명세

- **GET /api/users/check-username**
  - **설명**: 사용자 아이디가 중복되는지 확인합니다.
  - **요청 파라미터**:
    - `username` (String, 필수)
  - **응답**
    - `200 OK`
      - `application/json`

```
{
  "exists": true | false
}
```

- **GET /api/usres/check-email**
  - **설명**: 이메일이 중복되는지 확인합니다.
  - **요청 파라미터**:
    - `email` (String, 필수)
  - **응답**
    - `200 OK`
      - `application/json`

```
{
  "exists": true | false
}
```

UserController API 명세

POST /userreg

- **설명**: 회원 가입 정보를 받아들여 회원 등록을 수행하고, 성공하면 환영 메시지를 보여주는 화면으로 리다이렉트합니다. 오류가 발생하면 오류 메시지를 포함한 화면을 보여줍니다.

- **요청 바디**:
  - **폼 파라미터**
    - `username` (String): 사용자 아이디
    - `password` (String): 사용자 비밀번호
    - `name` (String): 사용자 이름
    - `email` (String): 사용자 이메일
- **응답**
  - **성공 시**:
    - **상태 코드**: `302 Found`
    - **리다이렉트 URL**: `/welcome`
    - **설명**: 회원 가입이 성공하면, 회원 가입 환영 메시지를 보여주는 화면으로 리다이렉트됩니다.

- **실패 시**:
  - **상태 코드**: `400 Bad Request`
  - **설명**: 회원 가입 요청이 유효성 검사를 통과하지 못하거나, 중복된 아이디 또는 이메일이 있는 경우, 오류 메시지를 포함한 화면을 보여줍니다.
  - **오류 메시지 예시**:
    - “Username is already taken”
    - “Email is already taken”
    - “Validation errors”

GET /welcome

- **설명**: 회원 가입 환영 메시지를 보여주는 화면입니다.
- **요청 파라미터**: 없음
- **응답**
  - **상태 코드**: `200 OK`
  - **뷰 템플릿**: `welcome.html`

    - **설명**: 회원 가입이 성공적으로 완료된 후, 환영 메시지를 포함한 화면을 보여줍니다.

뷰 템플릿

welcome.html

- **설명**: 회원 가입이 성공적으로 완료된 후, 환영 메시지를 포함한 화면을 표시합니다.

---

error.html

- **설명**: 회원 가입 시 발생한 오류 메시지를 포함한 화면을 표시합니다.

Cookie 예제

로그인을 하면 서버에 위해 생성된 쿠키가 브라우저에 저장됩니다. 그리고 브라우저는 자동으로 쿠키를 전송한 서버에 쿠키를 다시 전송하게 됩니다. 이를 이용해 해당 쿠키에 있는 인증 정보를 이용하여 어떤 사용자인지 판단할 수 있습니다.

---

build.gradle

- thymeleaf 라이브러리를 추가한다.

```
implementation 'org.springframework.boot:spring-boot-starter-thymeleaf'
```

- /createCookie 를 요청할 경우 쿠키(이름 : auth, 값 :carami)를 응답으로 보내고 createCookie.html 화면이 보여집니다.

```
package com.example.apirdsdemo.controller;

import jakarta.servlet.http.Cookie;
import jakarta.servlet.http.HttpServletResponse;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.GetMapping;

@Controller
public class CookieTestController {
    @GetMapping("/createCookie")
    public String createCookie(HttpServletResponse response) {
        Cookie cookie = new Cookie("auth", "carami");
        cookie.setPath("/");

        response.addCookie(cookie);
        return "createCookie";
    }
}
```

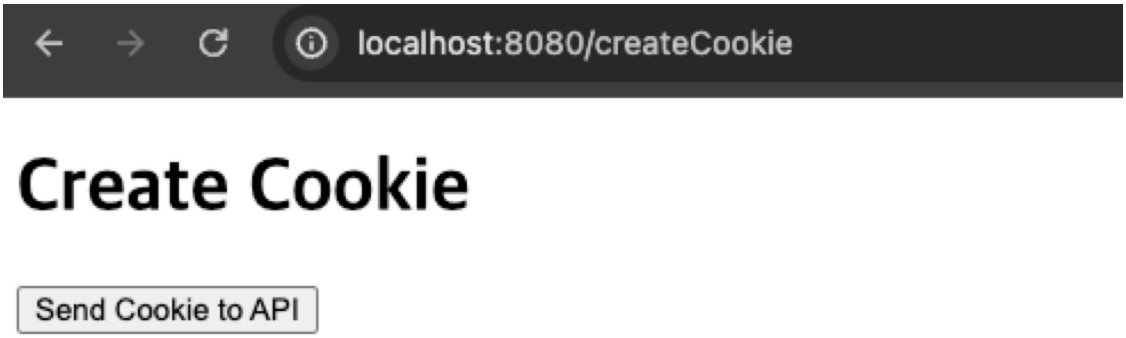
- 해당 화면이 보여질땐 응답으로 Cookie 가 전송받아 브라우저에 저장됩니다. 버튼을 클릭하면 /api/sendCookie API 로 쿠키가 전달되고 문자열을 받아 화면에 저장합니다.

```html
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
  <head>
    <title>Create Cookie</title>
    <script src="https://code.jquery.com/jquery-3.6.0.min.js"></script>
  </head>
  <body>
    <h1>Create Cookie</h1>
    <button id="sendCookieButton">Send Cookie to API</button>
    <p id="responseText"></p>

    <script>
      $(document).ready(function () {
        $("#sendCookieButton").click(function () {
          $.ajax({
            url: "/api/sendCookie",
            type: "GET",
            success: function (response) {
              $("#responseText").text("Response from API: " + response);
            },
            error: function () {
              $("#responseText").text("Failed to send cookie to API.");
            },
          });
        });
      });
    </script>
  </body>
</html>
```

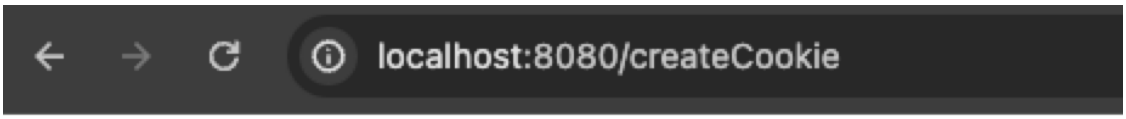
- /api/sendCookie 는 브라우저가 보내준 쿠키중에 쿠키이름이 “auth”인 쿠키를 찾아 해당 쿠키의 값을 응답합니다. “auth”라는 이름의 쿠키가 없을 경우 “no cookie”를 응답합니다.

```java
package com.example.apirdsdemo.controller;

import jakarta.servlet.http.Cookie;
import jakarta.servlet.http.HttpServletRequest;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RestController;

@RestController
public class CookieAPITestController {
    @GetMapping("/api/sendCookie")
    public String createCookie(HttpServletRequest request) {
```

```
        // request 로부터 auth 라는 이름의 쿠키를 구한다.
        Cookie[] cookies = request.getCookies();
        if (cookies != null) {
            for (Cookie cookie : cookies) {
                if (cookie.getName().equals("auth")) {
                    return cookie.getValue();
                }
            }
        }
        return "no cookie";
    }
}
```



- 버튼을 눌렀을 때 그 결과를 보인다.



Create Cookie

Send Cookie to API

Response from API: carami

로그인과 사용자 블로그 페이지 보여주기

UserController API 명세

GET /loginform

- **설명**: 로그인 폼을 보여주는 화면입니다.
- **요청 파라미터**: 없음
- **응답**
    - **상태 코드**: `200 OK`
    - **뷰 템플릿**: `login.html`
    - **설명**: 로그인 폼을 포함한 화면을 보여줍니다.

POST /login

- **설명**: 아이디와 암호를 받아서 로그인 처리를 합니다. 로그인 성공 시 사용자 정보 페이지로 리다이렉트하며, 로그인 실패 시 로그인 에러 메시지를 보여줍니다.
- **요청 바디**:
    - **폼 파라미터**
        - `username` (String): 사용자 아이디
        - `password` (String): 사용자 비밀번호

- **응답**
    - **성공 시**:
        - **상태 코드**: `302 Found`
        - **리다이렉트 URL**: `http://도메인/@{username}`
        - **설명**: 로그인 성공 시 사용자 정보 페이지로 리다이렉트합니다.
        - **쿠키**: 사용자 인증 쿠키를 클라이언트에게 전송합니다.
    - **실패 시**:

        - **상태 코드**: `400 Bad Request`
        - **설명**: 로그인 실패 시, 오류 메시지를 포함한 화면을 보여줍니다.

GET /@username

- **설명**: 사용자 정보를 간단하게 보여주는 화면입니다. 사용자 인증 쿠키를 검증하여 인증된 사용자만 접근할 수 있습니다.
- **요청 파라미터**: 없음
- **응답**
    - **상태 코드**: `200 OK`
    - **뷰 템플릿**: `blog.html`
    - **설명**: 사용자 정보를 포함한 화면을 보여줍니다.

뷰 템플릿

login.html

- **설명**: 로그인 폼을 포함한 화면을 표시합니다.

---

blog.html

- **설명**: 사용자 정보를 포함한 화면을 표시합니다.

개발 - 인증/인가

LoginCheckFilter

Http Request 의 요청에서 Cookie 를 꺼내고, 이 쿠키를 검사해서 특정 권한이 있는 사용자만 특정 URL,특정 HTTP Method 에서 접근할 수 있는지 검사해야 합니다.

이를 위해서 서블릿 필터를 사용하려고 합니다. 서블릿 필터에서 검사를 하고 접근할 수 없는 요청을 했을 경우 접근할 수 없다고 응답하도록 합니다.

접근할 수 있을 경우엔 사용자 정보를 ThreadLocal 에 저장하여 같은 Thread 상에서 값을 전달하게 됩니다. HTTP 요청 하나당 Spring MVC 는 하나의 Thread 가 기본적으로 동작합니다.

서블릿필터와 ThreadLocal 을 이용해서 인증 처리를 어떻게 해야할까?를 팀원들과 함께 논의를 하고 문제해결을 해보세요. 이를 통해 뒤에서 학습하게 될 Spring Security 의 원리를 이해 할 수 있게 됩니다.

- HTTP 요청 메소드(GET, POST 등)와 URL 경로에 따라서 권한을 체크하여 권한이 있을 경우 사용자 정보와 사용자 권한 정보를 ThreadLocal 에 담는 필터입니다.
- ThreadLocal 에 담긴 사용자 정보와 권한 정보는 Controller 등에서 사용할 수 있습니다.
- 이를 위해 ThreadLocal 에 대해 학습하세요.
- 이를 위해 Spring MVC 에서 필터를 등록하는 방법을 학습하세요.

Step 1: ThreadLocal 을 사용한 회원 정보 저장

먼저, `ThreadLocal` 을 사용하여 요청 스레드에서 회원 정보를 저장하고 액세스할 수 있도록 합니다.

```
package com.example.apirdsdemo.util;


public class UserContext {
    private static final ThreadLocal<String> userHolder = new ThreadLocal<>
();

    public static void setUser(String user) {
        userHolder.set(user);
    }

    public static String getUser() {
        return userHolder.get();
    }

    public static void clear() {
        userHolder.remove();
    }
}
```

Step 2: 서블릿 필터 작성 및 회원 정보 설정

`AuthenticationFilter` 를 수정하여 쿠키가 유효한 경우 회원 정보를 `ThreadLocal` 에 저장합니다.

```
package com.example.apirdsdemo.filter;


import com.example.apirdsdemo.service.UserService;
import com.example.apirdsdemo.util.UserContext;


import jakarta.servlet.Filter;
import jakarta.servlet.FilterChain;
import jakarta.servlet.FilterConfig;
import jakarta.servlet.ServletException;
import jakarta.servlet.ServletRequest;
import jakarta.servlet.ServletResponse;
```

```
import jakarta.servlet.http.Cookie;
import jakarta.servlet.http.HttpServletRequest;
import jakarta.servlet.http.HttpServletResponse;
import java.io.IOException;
```

```
public class AuthenticationFilter implements Filter {

    private UserService userService;

    public void setUserService(UserService userService) {
        this.userService = userService;
    }


    @Override
    public void init(FilterConfig filterConfig) throws ServletException {
        // 초기화 코드 (필요한 경우)
    }
```

```
    @Override
    public void doFilter(ServletRequest request, ServletResponse response, Fi
lterChain chain)
            throws IOException, ServletException {
        HttpServletRequest httpRequest = (HttpServletRequest) request;
        HttpServletResponse httpResponse = (HttpServletResponse) response;

        System.out.println("AuthenticationFilter: " + httpRequest.getRequestU
RI());
        System.out.println("AuthenticationFilter: " + httpRequest.getMethod
());

        // URI, Method 를 쿠키의 사용자가 이용할 수 있는지 확인한 후 문제가 없다면
 chain.doFilter(request, response); 를 실행하고 권한이 없을 경우엔 특정 경로로 r
edirect 한다.

        chain.doFilter(request, response);
    }
```

```
    @Override
    public void destroy() {
```

```
        // 정리 코드 (필요한 경우)
    }

    private String validateTokenAndGetUserId(String token) {
        // 토큰 검증 및 사용자 ID 추출 로직 (예: JWT 검증)
        // 유효한 경우 사용자 ID를 반환, 그렇지 않으면 null 반환
        return null;
    }
}
```

Step 3: 필터 등록

필터를 등록하여 모든 경로에 대해 필터를 적용합니다.

```
package com.example.apirdsdemo.config;

import com.example.apirdsdemo.filter.AuthenticationFilter;
import com.example.apirdsdemo.service.UserService;
import org.springframework.boot.web.servlet.FilterRegistrationBean;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;

@Configuration
public class FilterConfig {


    @Bean
    public FilterRegistrationBean<AuthenticationFilter> authenticationFilter
(UserService userService) {
        FilterRegistrationBean<AuthenticationFilter> registrationBean = new F
ilterRegistrationBean<>();
        AuthenticationFilter authenticationFilter = new AuthenticationFilter
();
        authenticationFilter.setUserService(userService); // UserService 주입
        registrationBean.setFilter(authenticationFilter);
        registrationBean.addUrlPatterns("/*"); // 모든 경로에 필터 적용
        registrationBean.setOrder(1); // 필터 실행 순서 설정 (낮을수록 먼저 실
행)
        return registrationBean;
    }
}
```

Step 4: Controller 에서 회원 정보 액세스

`Controller` 에서 `ThreadLocal` 에 저장된 회원 정보를 액세스합니다.

```java
package com.example.apirdsdemo.controller;

import com.example.demo.util.UserContext;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RestController;

@RestController
public class UserController {

    @GetMapping("/user-info")
    public String getUserInfo() {
        String userId = UserContext.getUser();
        if (userId != null) {
            return "User ID: " + userId;
        } else {
            return "No user authenticated";
        }
    }
}
```

개발 - 글 작성하기

---

1. 이미지를 업로드 할 경우 로컬의 디스크에 저장됩니다.
2. 이미지를 다운로드 할 수 있습니다.
3. 업로드한 이미지 정보는 DB 에 저장됩니다.
4. 글의 제목, 태그들, 내용을 저장합니다.
5. 3)에서 자장한 이미지에 대한 정보와 4)에서 저장하는 글의 정보는 같은 트랜잭션으로 저장되지 않습니다. 4)에서 저장하는 글과 3)에서 저장되는 이미지에 대한 정보가 관련을 맺으려면 어떻게 해야할까요?

- 글쓰기 폼은 다음과 같습니다. 바닐라 자바스크립트로 구현합니다.
- 하단에서 이미지를 추가할 경우 본문의 커서 위치에 img 태그가 추가됩니다.
- 이미지를 여러개 추가할 수 있습니다.

/postform

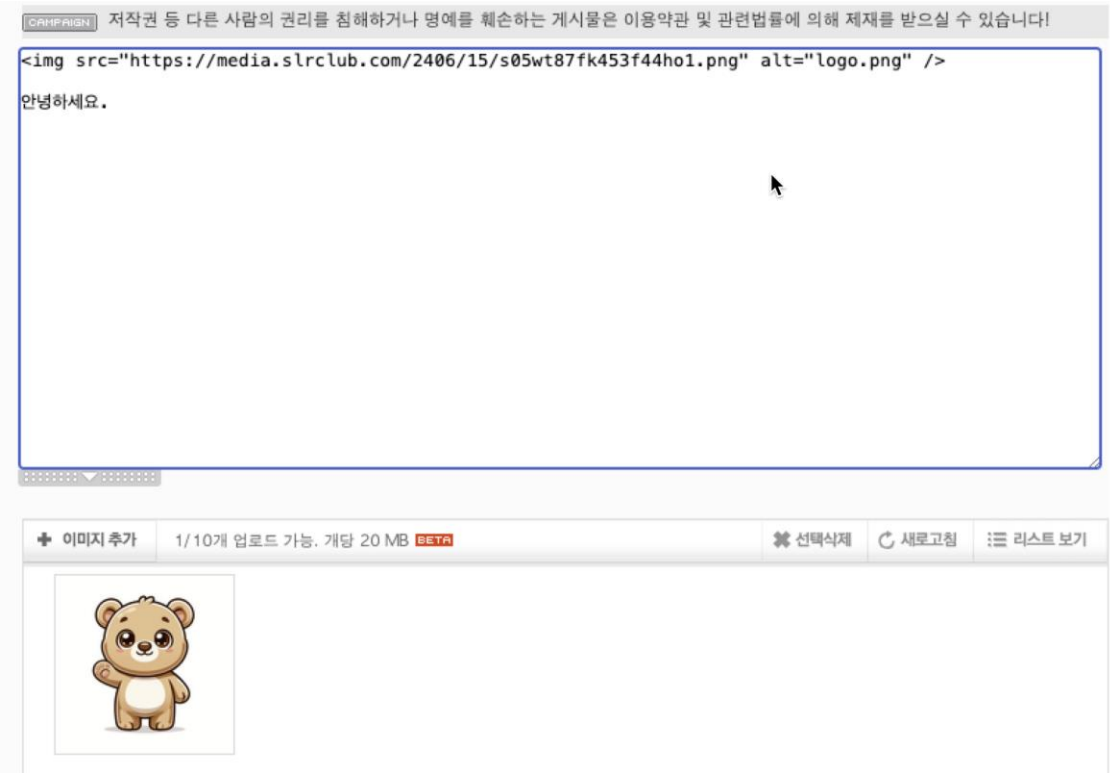
- 에디터의 구현은 굉장히 어려운 기술입니다. 자주 사용되는 에디터를 가져다 사용하기 위해서는 프론트 관련 기술에 대한 이해가 필요합니다. 이 예제에서는 이미지를 업로드를 할 경우 img 태그가 본문에 들어가도록 자바스크립트로 구현합니다.

제목을 입력하세요

태그 목록을 입력하세요.

임시저장 출간하기

height:500



height:500

/postform 에서 임시저장을 하게 되면 /post 로 사용자가 작성한 정보가 전달되어 등록됩니다. 출간하기를 누르게 되면 /post 로 사용자가 작성한 정보가 전달되어 등록합니다. 단 임시저장과 다른 점은 hidden 속성으로 숨겨져 있는 publishStatus 값이 true 로 바뀐후 전달됩니다.

예제)

```
<!DOCTYPE html>
<html lang="ko">
  <head>
    <meta charset="UTF-8" />
    <title>Post Form</title>
    <style>
      .button {
        background-color: #8b4513; /* 갈색 */
        color: white;
        padding: 10px 20px;
```

```
        border: none;
        cursor: pointer;
      }
      .button:hover {
        background-color: #a0522d; /* 좀 더 밝은 갈색 */
      }
    </style>
  </head>
  <body>
    <form id="postForm" action="/post" method="post">
      <div>
        <label for="title">제목을 입력하세요</label>
        <input type="text" id="title" name="title" required />
      </div>
      <div>
        <label for="tags">태그 목록을 입력하세요</label>
        <input type="text" id="tags" name="tags" required />
      </div>
      <div>
        <textarea id="content" name="content" rows="10" cols="50" required>
                <img src="https://media.slrclub.com/2406/15/s05wt87fk453f44ho
1.png" alt="logo.png" />
                안녕하세요.
            </textarea
        >
      </div>
      <div>
        <label for="image">이미지 추가</label>
        <input type="file" id="image" name="image" />
        <img
          src="https://example.com/path/to/image.png"
          alt="uploaded image"
          id="uploadedImage"
        />
      </div>
      <input
        type="hidden"
        id="publishStatus"
        name="publishStatus"
        value="false"
      />
      <div>
        <button type="button" class="button" id="saveDraft">임시저장</button>
        <button type="button" class="button" id="publishPost">출간하기</button
>
```

```
      </div>
    </form>

    <script>
      document
        .getElementById("publishPost")
        .addEventListener("click", function () {
          document.getElementById("publishStatus").value = "true";
          document.getElementById("postForm").submit();
        });

      document
        .getElementById("saveDraft")
        .addEventListener("click", function () {
          document.getElementById("publishStatus").value = "false";
          document.getElementById("postForm").submit();
        });
    </script>
  </body>
</html>
```

/post 는 publishStatus 파라미터의 값이 false 이면 임시저장으로 저장하고 publishStatus 값이 true 일 경우 출간형태로 저장합니다.

아이디어 : 미리 글을 저장하고, 이때 만들어진 PK 를 사용한다.

1. 폼이 보여질때 이미 임시로 글이 저장됩니다.
2. 임시로 저장된 글 번호가 form 에 hidden 값으로 저장되고 폼이 보여집니다.
3. 해당 폼에서 이미지를 등록할 때 마다 이미지는 임시로 저장된 글 번호를 참조하여 저장됩니다.
4. 해당 글을 저장하면 임시로 저장된 글이 수정되고 임시글이나 출간글로 바뀌게 됩니다.

View Template 과 API 를 적절히 활용하기

예: main.html - Controller 는 Model 에 로그인한 사용자 정보를 담아서 template 에게 전송합니다. template 에서는 Javascript 로 API 를 호출해서 그 결과를 화면에 출력합니다.

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
  <head>
    <meta charset="UTF-8" />
    <title>블로그 메인</title>
    <script src="https://code.jquery.com/jquery-3.6.0.min.js"></script>
    <script>
      let page = 0;
      const size = 10;
      let sort = "latest";

      $(document).ready(function () {
        loadPosts();

        $(window).scroll(function () {
          if (
            $(window).scrollTop() + $(window).height() >=
            $(document).height()
          ) {
            loadPosts();
          }
        });
      });

      function loadPosts() {
        $.ajax({
          url: `/api/posts?sort=${sort}&page=${page}&size=${size}`,
          method: "GET",
          success: function (response) {
            const posts = response.content;
            page++;
            posts.forEach((post) => {
              $("#posts").append(`
                            <div class="post">
                                <h2>${post.title}</h2>
                                <p>${post.excerpt}</p>
                                <div class="author">
```

```
                                    <img src="${post.author.profileImage}" al
t="profile" />
                                    <span>${post.author.username}</span>
                                </div>
                                <span>좋아요: ${post.likes}</span>
                            </div>
                        `);
            });
          },
          error: function (error) {
            console.error("Error loading posts", error);
          },
        });
      }
    </script>
    <style>
      .post {
        border: 1px solid #ccc;
        padding: 10px;
        margin: 10px 0;
      }
      .author img {
        width: 50px;
        height: 50px;
        border-radius: 50%;
      }
    </style>
  </head>
  <body>
    <h1>블로그 메인</h1>
    <div id="posts"></div>
  </body>
</html>
```

API 만들기

- API 를 호출하지 않고 Controller 와 Thymeleaf 를 이용해 구현할 수도 있습니다.

회원 탈퇴

- **URL**: `/api/users/{userId}`
- **Method**: DELETE
- **Request**:

```
{
  "password": "string"
}
```

- **Response**:
  - 성공:

```
{
  "message": "회원 탈퇴가 완료되었습니다."
}
```

  - 실패:

```
{
  "error": "비밀번호가 잘못되었습니다."
}
```

회원 정보 수정

- **URL**: `/api/users/{userId}`
- **Method**: PUT
- **Request**:

```
{
  "name": "string",
  "email": "string",
  "profileImage": "url",
  "blogTitle": "string",
  "emailNotification": {
    "comment": "boolean",
    "update": "boolean"
  }
}
```

- **Response**:
  - 성공:

    ```
    {
      "message": "회원 정보가 수정되었습니다."
    }
    ```

  - 실패:

    ```
    {
      "error": "잘못된 요청입니다."
    }
    ```

블로그 API

최신 글/인기 글 목록 조회

- **URL**: `/api/posts`
- **Method**: GET
- **Query Parameters**:
  - `sort`: `latest` or `popular`

- **Response**:
  - 성공:

    ```
    [
      {
    ```

```
    "postId": "string",
    "title": "string",
    "excerpt": "string",
    "author": {
      "username": "string",
      "profileImage": "url"
    },
    "likes": "integer"
  }
]
```

특정 사용자 블로그 페이지 조회

- **URL**: `/api/users/{username}/posts`
- **Method**: GET

- **Response**:
  - 성공:

```
{
  "tags": [
    {
      "tag": "string",
      "count": "integer"
    }
  ],
  "posts": [
    {
      "postId": "string",
      "title": "string",
      "excerpt": "string",
      "createdAt": "datetime",
      "series": "string"
    }
  ],
  "series": [
    {
      "seriesId": "string",
      "title": "string",
      "posts": [
        {
          "postId": "string",
          "title": "string"
        }
      ]
```

```
    }
  ],
  "introduction": "string"
}
```

글 삭제

- **URL**: /api/posts/{postId}
- **Method**: DELETE
- **Response**:
  - 성공:

```
{
  "message": "글 삭제가 완료되었습니다."
}
```

  - 실패:

```
{
  "error": "잘못된 요청입니다."
}
```

글 목록 조회 (임시 글 포함)

- **URL**: /api/posts/drafts
- **Method**: GET
- **Response**:
  - 성공:

```
[
  {
    "postId": "string",
    "title": "string",
    "createdAt": "datetime"
  }
]
```

좋아요 기능

- **URL**: `/api/posts/{postId}/like`
- **Method**: POST
- **Response**:
    - 성공:

      ```
      {
        "message": "좋아요가 추가되었습니다."
      }
      ```

    - 실패:

      ```
      {
        "error": "잘못된 요청입니다."
      }
      ```

댓글

댓글 작성

- **URL**: `/api/posts/{postId}/comments`
- **Method**: POST
- **Request**:

  ```
  {
    "content": "string"
  }
  ```

- **Response**:
    - 성공:

      ```
      {
        "message": "댓글 작성이 완료되었습니다.",
        "commentId": "string"
      }
      ```

    - 실패:

```
{
  "error": "잘못된 요청입니다."
}
```

댓글 삭제

- **URL**: `/api/posts/{postId}/comments/{commentId}`
- **Method**: DELETE
- **Response**:
    - 성공:

```
{
  "message": "댓글 삭제가 완료되었습니다."
}
```

    - 실패:

```
{
  "error": "잘못된 요청입니다."
}
```

팔로우

사용자 팔로우

- **URL**: `/api/users/{username}/follow`
- **Method**: POST
- **Response**:
    - 성공:

```
{
  "message": "팔로우가 완료되었습니다."
}
```

    - 실패:

```
{
  "error": "잘못된 요청입니다."
}
```

팔로우 취소

- **URL**: `/api/users/{username}/unfollow`
- **Method**: POST
- **Response**:
  - 성공:

    ```
    {
      "message": "언팔로우가 완료되었습니다."
    }
    ```

  - 실패:

    ```
    {
      "error": "잘못된 요청입니다."
    }
    ```

팔로우 목록 조회

- **URL**: `/api/users/followers`
- **Method**: GET
- **Response**:
  - 성공:

    ```
    [
      {
        "userId": "string",
        "username": "string",
        "profileImage": "url"
      }
    ]
    ```

관리자

모든 글 목록 조회

- **URL**: `/api/admin/posts`
- **Method**: GET
- **Response**:
  - 성공:

```
[
  {
    "postId": "string",
    "title": "string",
    "author": {
      "username": "string",
      "profileImage": "url"
    },
    "isPublished": "boolean",
    "createdAt": "datetime"
  }
]
```

글 삭제 (관리자 권한)

- **URL**: `/api/admin/posts/{postId}`
- **Method**: DELETE
- **Response**:
  - 성공:

```
{
  "message": "글 삭제가 완료되었습니다."
}
```

  - 실패:

```
{
  "error": "잘못된 요청입니다."
}
```

#끝