6. 로그인 처리1 - 쿠키, 세션

https://s3-us-west-2.amazonaws.com/secure.notion-static.com/41d34066-6989-4d20-bb85-6c4e935a86a0/6._%EB%A1%9C%EA%B7%B8%EC%9D%B8_%EC%B2%98%EB%A6%AC1_-_%EC%BF%A0%ED%82%A4_%EC%84%B8%EC%85%98_(1).pdf

로그인 요구사항

- 홈 화면 - 로그인 전
    - 회원 가입
    - 로그인
- 홈 화면 - 로그인 후
    - 본인 이름
    - 상품 관리
    - 로그아웃
- 보안 요구사항
    - 로그인 사용자만 상품에 접근하고, 관리할 수 있음
    - 로그인하지 않은 사용자가 상품 관리에 접근하면 로그인 화면으로 이동
- 회원 가입, 상품 관리

프로젝트 생성

패키지의 구조

- domain
    - item

    - member
    - login
- web
    - item
    - member
    - login

도메인: 화면, UI, 기술 인프라 등등의 영역은 제외한 시스템이 구현해야 하는 핵심 비즈니스 업무 영역

홈 화면

회원 가입

1. 회원가입 클릭 시 members/add 페이지로 이동

2. 회원 정보를 입력 후 회원가입 버튼을 클릭 시 members/add로 데이터 POST 후 첫 화면으로 Redirect

- 아래는 Member의 코드이다.

```
@Data
public class Member {
    private Long id;
    @NotEmpty
    private String loginId; //로그인 ID
    @NotEmpty
    private String name; //사용자 이름
    @NotEmpty
    private String password;
}
```

- 아래는 MemberRepository의 코드이다.

```
@Slf4j
@Repository
public class MemberRepository {
    private static Map<Long, Member> store = new HashMap<>(); //static사용
    private static long sequence = 0L;
    public Member save(Member member){
        member.setId(++sequence);
        log.info("save: member={}", member);
        store.put(member.getId(), member);
        return member;
    }
    public Member findById(Long id){
        return store.get(id);
    }

    //LoginId로 회원 찾기
    public Optional<Member> findByLogInId(String loginId){
//        List<Member> all = findAll();
//        for(Member m: all){
//            if (m.getLoginId().equals(loginId)) {
//                return Optional.of(m);
//            }
//        }
//        return Optional.empty();

        return findAll().stream()
                .filter(m -> m.getLoginId().equals(loginId))
                .findFirst();
    }
    public List<Member> findAll(){
        return new ArrayList<>(store.values());
    }

    public void clearStore(){
        store.clear();
    }
}
```

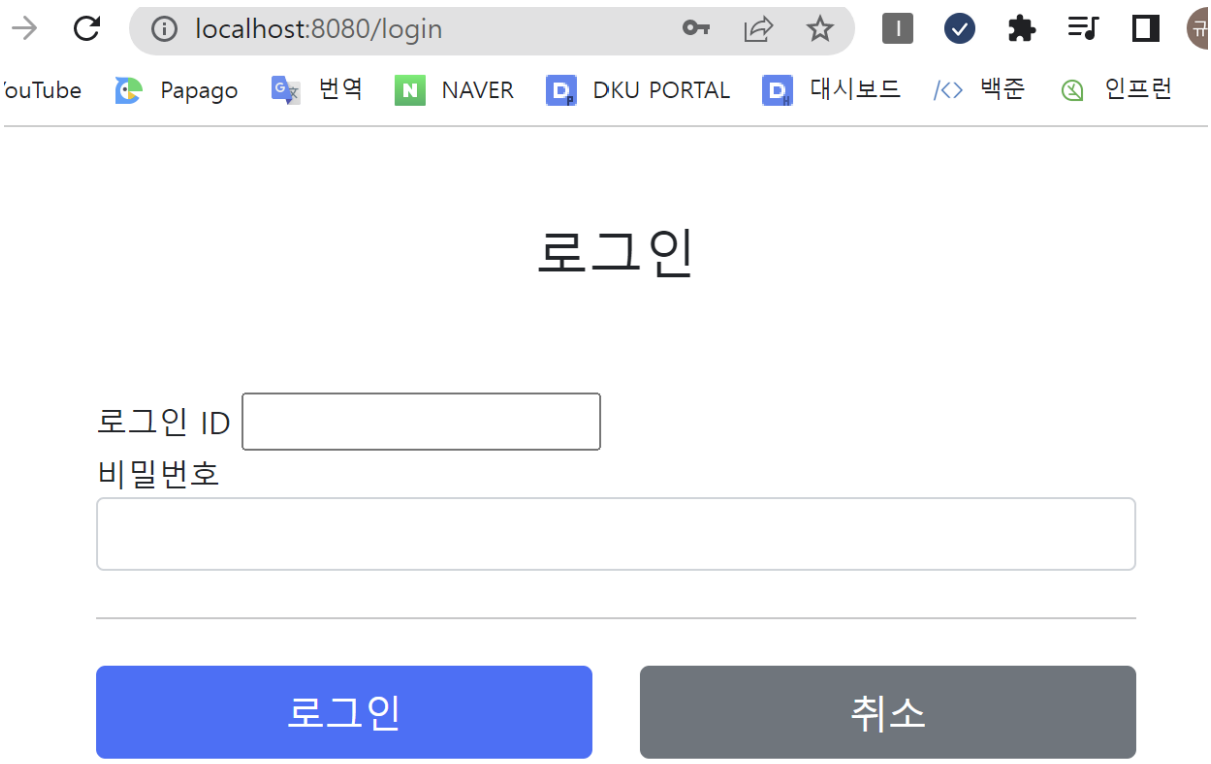
- 아래는 MemberController의 코드이다.

```
@Controller
@RequiredArgsConstructor
@RequestMapping("/members")
public class MemberController {
    private final MemberRepository memberRepository;

    //아래의 @ModelAttribute에서 "member"인자는 사라져도 된다.
    // 가져오는 객체의 첫 번째글자를 소문자로 바꾼 것을 default로 가져오기 때문!
    @GetMapping("/add")
    public String addForm(@ModelAttribute("member") Member member) {
```

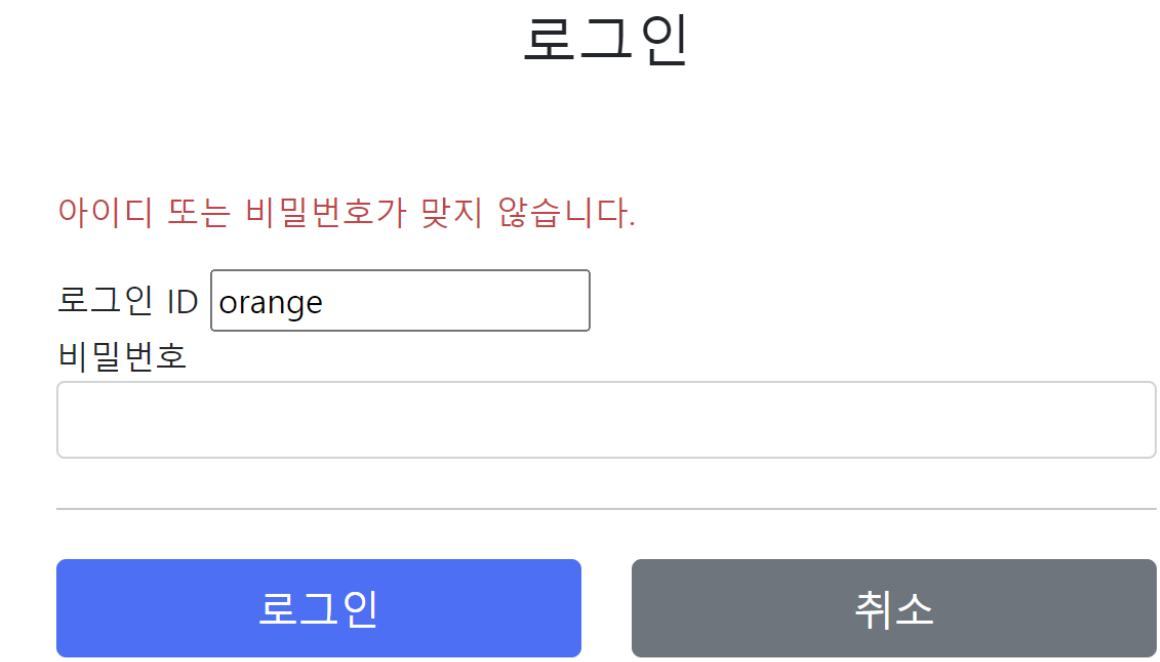
```
        return "members/addMemberForm";
    }

    @PostMapping("/add")
    public String save(@Valid @ModelAttribute Member member, BindingResult result){
        if (result.hasErrors()){
            return "members/addMemberForm";
        }
        memberRepository.save(member);
        return "redirect:/"; //회원을 저장하고 나서 메인 화면으로 redirect시킨다.
    }
}
```

로그인 기능

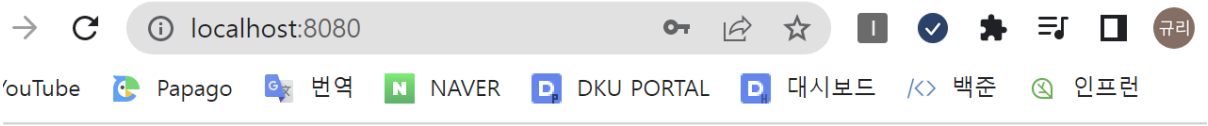
1. 로그인 페이지



2. 회원 id와 비밀번호를 입력하여 로그인 버튼을 눌렀을 때 아이디에 대한 비밀번호가 다른 경우 아래와 같이 알림 메시지를 띄우고, login페이지로 다시 redirect



3. 일치하는 경우 홈 화면으로 redirect



- 아래는 사용자에게 입력받을 폼(LoginForm)의 형태이다.

```
@Data
public class LoginForm {
    @NotEmpty
    private String loginId;
    @NotEmpty
    private String password;
}
```

- 아래는 LoginController의 코드이다.

```
@Controller
@Slf4j
@RequiredArgsConstructor
public class LoginController {
    private final LoginService loginService;

    @GetMapping("/login")
    public String loginForm(@ModelAttribute("loginForm") LoginForm form) {
        return "login/loginForm";
    }

    @PostMapping("/login")
    public String login(@Valid @ModelAttribute LoginForm form, BindingResult result){
        if (result.hasErrors()){
            return "login/loginForm";
        }
        Member loginMember = loginService.login(form.getLoginId(), form.getPassword());

        // 로그인 과정을 거치다가 오류가 난 경우
        // 즉, 아이디에 대한 비번이 맞지 않은 경우
        if (loginMember==null){
            result.reject("loginFail", "아이디 또는 비밀번호가 맞지 않습니다.");
            return "login/loginForm";
        }

        //로그인 성공 처리 -> 홈 화면으로 redirect
        return "redirect:/";

    }
}
```

- 아래는 LoginService의 코드이다.

```
@Service
@RequiredArgsConstructor
public class LoginService {
    private final MemberRepository memberRepository;

    /**
     * @return null 이면 로그인 실패
     */
    public Member login(String loginId, String password) {
//        Optional<Member> findMember = memberRepository.findByLogInId(loginId);
//        Member member = findMember.get();
//        //회원이 입력한 아이디와 비번이 일치하면 member return
//        if (member.getPassword().equals(password)){
//            return member;
//        }
//        //아닌 경우 null return
//        else{
//            return null;
//        }


        Optional<Member> byLoginId = memberRepository.findByLogInId(loginId);
        return byLoginId.filter(m -> m.getPassword().equals(password)) //Optional 객체는 filter기능을 사용할 수 있음
                .orElse(null); //이를 사용하여 filter조건을 만족하면 해당 객체를, 만족하지 않는다면 null를 return하도록 할 수 있다.
    }
}
```

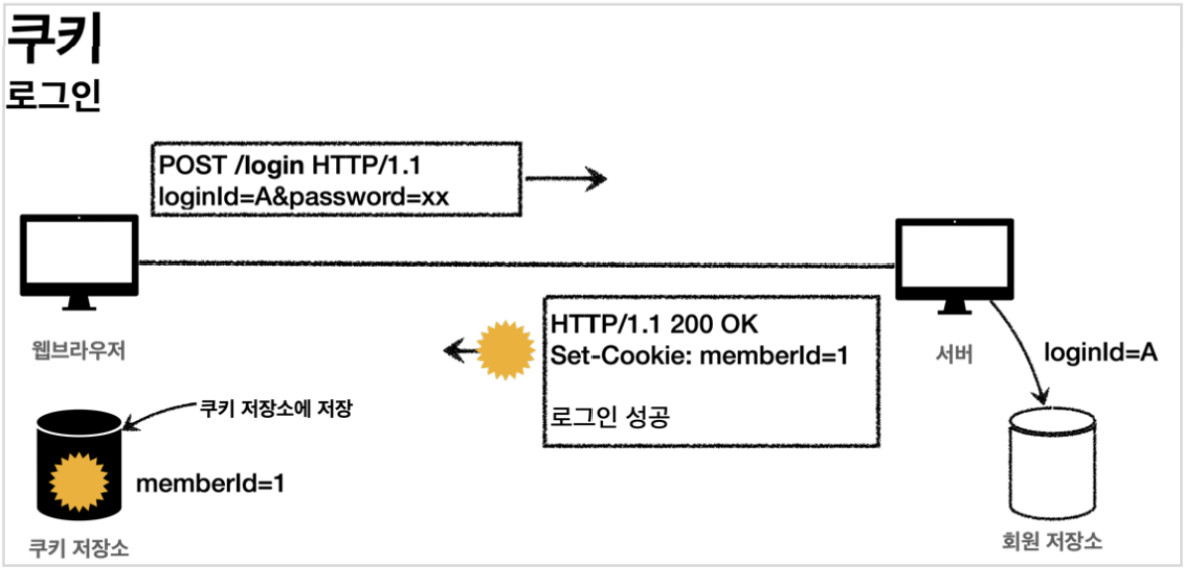
로그인 처리하기 - 쿠키 사용

로그인

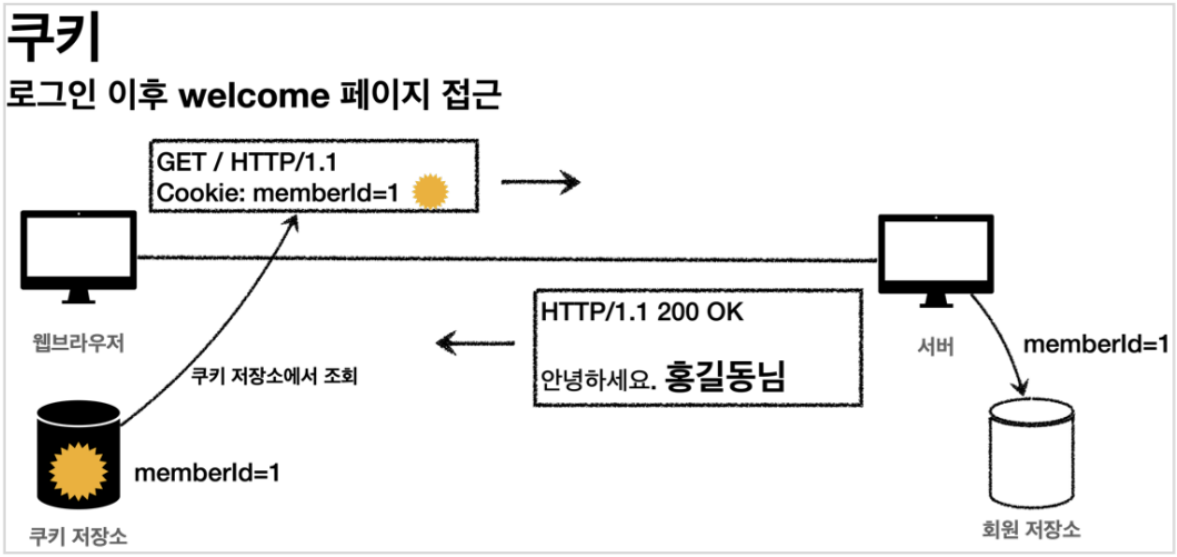
로그인 상태를 유지하는 방법

1. 쿼리 파라미터를 계속 유지하면서 주고 받기
2. 쿠키 사용 ⭐

쿠키 생성



클라이언트 쿠키 전달1



쿠키

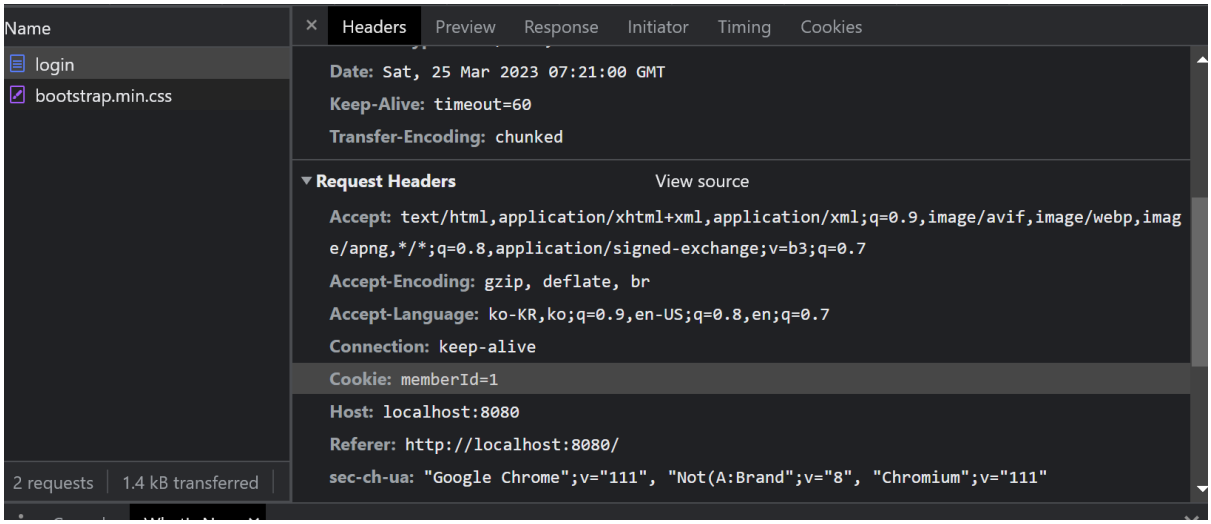
- 영속 쿠키 : 만료 날짜를 입력하면 해당 날짜까지 유지
- **세션 쿠키 : 만료 날짜를 생략하면 브라우저 종료시까지만 유지**

대부분 브라우저 종료시 로그아웃 되길 기대하므로, 우리에게 필요한 것은 세션 쿠키 ❗

> 💡 **쿠키 생성 방법**
> Controller에서 파라미터로 HttpServletResponse response객체를 받아와서 response.addCookie(쿠키 객체)를 통해 넣어준다.
> 코드는 아래와 같다.

```
//쿠키에 시간 정보를 주지 않으면 세션 쿠키(브라우저 종료시 모두 종료)
//cookie의 값은 항상 String 으로 들어가야 한다.
Cookie cookie = new Cookie("memberId", String.valueOf(loginMember.getId()));
response.addCookie(cookie);
```

이 코드를 LoginController의 `/login` 에 POST하는 메서드에 추가해주면 로그인시 아래와 같이 네트워크 상에서 memberId라는 쿠키가 계속해서 주어지는 것을 확인할 수 있다.



쿠키를 확인하는 방법

1. Network → Name에서 메서드명 클릭 → Cookie 찾기(위의 방식)
2. Application → Storage → Cookies에서 확인

이제 로그인 한 사용자에게 인사하는 페이지를 만들어보자.

```
@GetMapping("/")
public String homeLogin(@CookieValue(name="memberId", required = false) Long memberId, Model model){
    if (memberId==null){
        return "home";
    }
    // 로그인
    Member loginMember = memberRepository.findById(memberId);
    if (loginMember==null){
        return "home";
    }
    model.addAttribute("member", loginMember);
    return "loginHome";
}
```

- `@CookieValue`를 사용하여 쿠키 조회
- 로그인 하지 않은 사용자는 기존에 사용하던 홈으로 보내준다.

다음과 같이 로그인한 사용자의 홈 화면을 변경해줄 경우 로그인 후의 페이지는 아래와 같이 변경된다.

로직 분석

- 로그인 쿠키(memberId)가 없거나 로그인 쿠키가 있어도 회원이 없으면 `home`으로 보낸다.
- 로그인 쿠키(memberId)가 있는 사용자는 로그인 사용자 홈 화면인 `loginHome`으로 보낸다. 추가로 홈 화면에서 memeber의 이름을 띄워야 하므로 member데이터를 Model객체에 담아서 전달한다.

로그아웃

- 세션 쿠키이므로 웹 브라우저 종료시
- 서버에서 해당 쿠기의 종료날짜를 0으로 지정

아래와 같이 LoginController에 logout 메서드를 추가해주었다.

단순히 cookie의 수명을 줄여준 것 ❗

```
//로그아웃 로직 추가
@PostMapping("/logout")
public ring logout(HttpServletResponse response){
    expireCookie(response, "memberId");
    return "redirect:/";
}
private void expireCookie(HttpServletResponse response, String cookieName){
    Cookie cookie = new Cookie(cookieName, null);
    cookie.setMaxAge(0); // 쿠키의 종료날짜를 0으로 지정
    response.addCookie(cookie);
}
```

쿠키와 보안 문제

보안 문제

1. **쿠키 값은 임의로 변경할 수 있다.**

개발자 모드 → Application → Cookie 클릭 → 값 변경 → 새로고침 하면 해당 쿠키값에 해당하는 회원 계정으로 로그인되는 것 확인⚠️

2. 쿠키에 보관된 정보를 훔쳐갈 수 있다.
3. 해커가 쿠키를 한번 훔쳐가면 평생 사용할 수 있다.

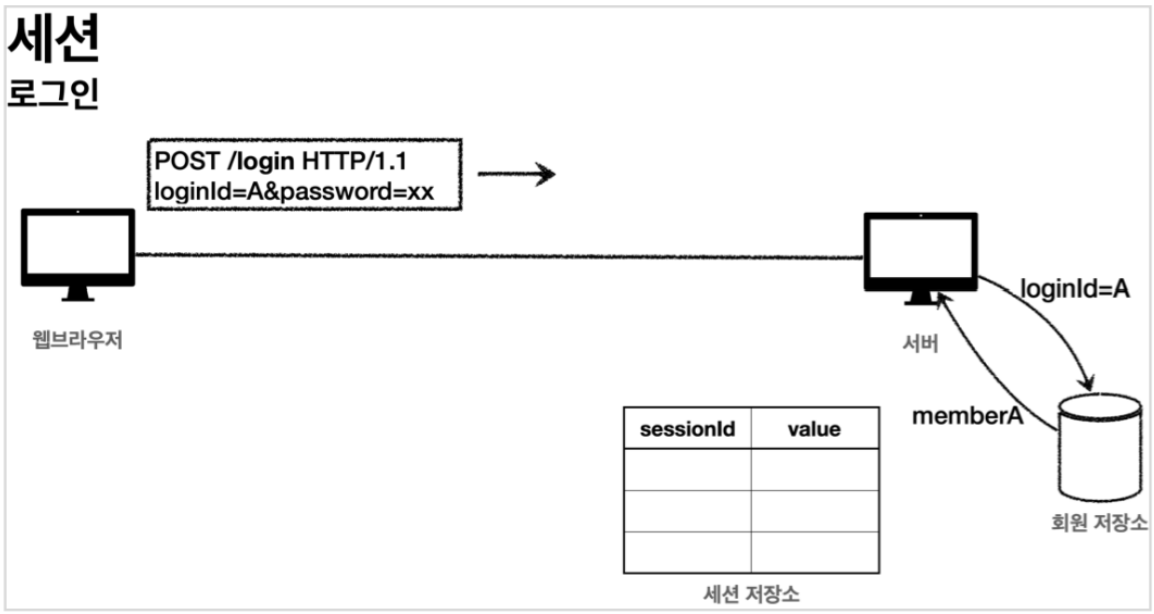
대안

1. 쿠키에 중요한 값을 노출하지 않고, 사용자별로 예측 불가능한 임의의 토큰을 노출하고, 서버에서 토큰과 사용자 id를 매핑해서 인식한다. 그리고 서버에서 토큰을 관리한다.
2. 토큰은 해커가 임의의 값을 넣어도 찾을 수 없도록 예상 불가능해야 한다.
3. 해커가 토큰을 털어가도 시간이 지나면 사용할 수 없도록 서버에서 해당 토큰의 만료시간을 짧게 유지해야 한다.

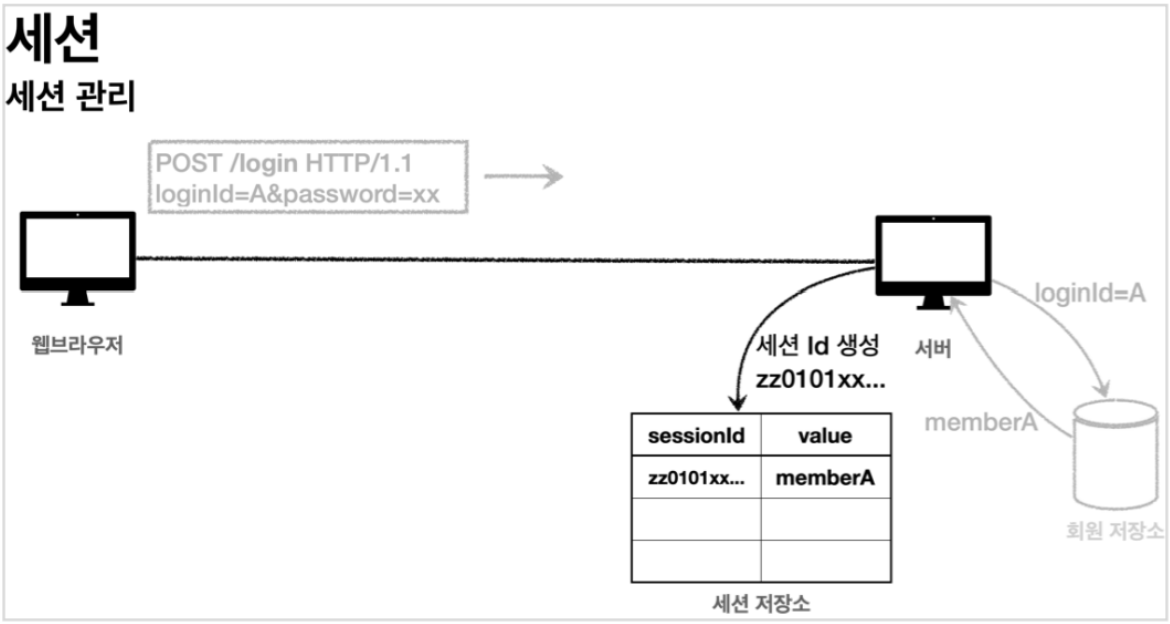
로그인 처리하기 - 세션 동작 방식

서버에 중요한 정보를 보관하고 연결을 유지하는 방법 → 세션 ❗

로그인

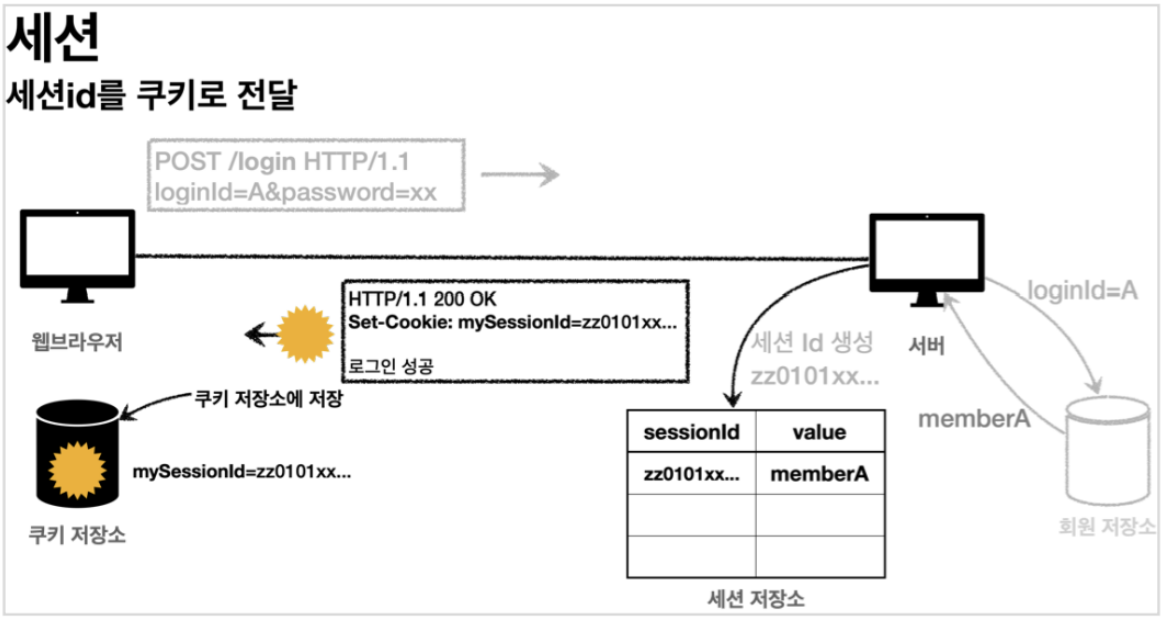


세션 생성



- 세션 ID를 생성하는데, 추정 불가능해야 한다.
- UUID는 추정이 불가능하다.
  - Cookie: mySessionId=xxxxxxxxxx
- 생성된 세션ID와 세션에 보관할 값(memberA)을 서버의 세션 저장소에 보관한다.

세션id를 응답 쿠키로 전달



클라이언트와 서버는 결국 쿠키로 연결이 되어야 한다.

- 서버는 클라이언트에 mySessionId라는 이름으로 세션 ID만 쿠키에 담아서 전달한다.
- 클라이언트는 쿠키 저장소에 mySessionId쿠키를 보관한다.
- 즉 **추정 불가능한 세션 ID만 쿠키를 통해 클라이언트에 전달된다.**

로그인 처리하기 - 세션 직접 만들기

세션 관리

1. **세션 생성**
   - sessionId 생성
   - 세션 저장소에 sessionId와 보관할 값 저장
   - sessionId로 응답 쿠키를 생성해서 클라이언트에 전달
2. **세션 조회**
   - 클라이언트가 요청한 sessionId쿠키의 값으로, 세션 저장소에 보관한 값 조회
3. **세션 만료**
   - 클라이언트가 요청한 sessionId쿠키의 값으로, 세션 저장소에 보관한 sessionId와 값 제거

```
/**
 * 세션 관리
 */
@Component
public class SessionManager {
    public static final String SESSION_COOKIE_NAME = "mySessionId";

    //동시성 문제가 있을 수 있는 경우 아래와 같은 HashMap을 사용하는 것이 안전
    private Map<String, Object> sessionStore = new ConcurrentHashMap<>();

    /**
     * 세션 생성
     */
    public void createSession(Object value, HttpServletResponse response){
        //session Id를 생성하고 값을 세션에 저장
        String sessionId = UUID.randomUUID().toString();
        sessionStore.put(sessionId, value);

        //쿠키 생성
        Cookie mySessionCookie = new Cookie(SESSION_COOKIE_NAME, sessionId);
        response.addCookie(mySessionCookie);
    }
    /**
     * 세션 조회
     */
    public Object getSession(HttpServletRequest request) {
        Cookie sessionCookie = findCookie(request, SESSION_COOKIE_NAME);
        if (sessionCookie == null) {
            return null;
        }
        return sessionStore.get(sessionCookie.getValue());
    }
    /**
     * 세션 만료
     */
    public void expire(HttpServletRequest request) {
        Cookie sessionCookie = findCookie(request, SESSION_COOKIE_NAME);
        if (sessionCookie != null) {
            sessionStore.remove(sessionCookie.getValue());
        }
    }

    private Cookie findCookie(HttpServletRequest request, String cookieName) {
        //쿠키값에 해당하는 쿠키가 없는 경우 null 반환
        if (request.getCookies() == null) {
            return null;
        }
        //찾고 싶은 쿠키명을 찾아 해당되는 쿠키를 반환
        return Arrays.stream(request.getCookies())
                .filter(cookie -> cookie.getName().equals(cookieName))
                .findAny()
                .orElse(null);
    }
}
```

로그인 처리하기 - 직접 만든 세션 적용

LoginController 내에서 `/login` 부분과 `/logout` 부분이 다음과 같이 변경되었다.

위에서 구현한 sessionManager을 주입받아 세션 매니저가 쿠키 생성부터 세션 id생성까지 후에 세션 만료 시간까지 설정하여 처리한다.

```
    // 세션과 쿠키를 사용한 로그인 구현
    @PostMapping("/login")
    public String loginV2(@Valid @ModelAttribute LoginForm form,
                          BindingResult result,
                          HttpServletResponse response) {
        if (result.hasErrors()) {
```

```
            return "login/loginForm";
        }
        Member loginMember = loginService.login(form.getLoginId(), form.getPassword());

        // 로그인 과정을 거치다가 오류가 난 경우
        // 즉, 아이디에 대한 비번이 맞지 않은 경우
        if (loginMember == null) {
            result.reject("loginFail", "아이디 또는 비밀번호가 맞지 않습니다.");
            return "login/loginForm";
        }

        //로그인 성공 처리
        //sessionManager에서 쿠키생성작업까지 다 해주기 때문에 기존 코드 삭제
        //세션 관리자를 통해 세션을 생성하고 회원 데이터 보관
        sessionManager.createSession(loginMember, response);

        return "redirect:/";

    }
    //쿠키와 세션을 사용한 로그아웃 구현
    @PostMapping("/logout")
    public String logoutV2(HttpServletRequest request){
        sessionManager.expire(request);
        return "redirect:/";
    }
```

로그인 처리하기 - 서블릿 HTTP 세션1

서블릿은 세션을 위해 HttpSession 이라는 기능을 제공한다.

💡 **HttpSession**
- 위에서 구현한 SessionManager와 같은 방식으로 동작한다.
- 쿠키 이름은 **JSESSIONID**로 생성된다.

세션 생성과 조회

session을 생성하려면 request.getSession()을 사용하면 된다.

- `request.getSession(true)`
  - 세션이 있으면 기존 세션을 반환
  - 없으면 새로운 세션을 생성해서 반환
  - 디폴트 값
- `request.getSession(false)`
  - 세션이 있으면 기존 세션을 반환
  - 없으면 null을 반환

따라서 위의 LoginController는 다음과 같이 변경된다.

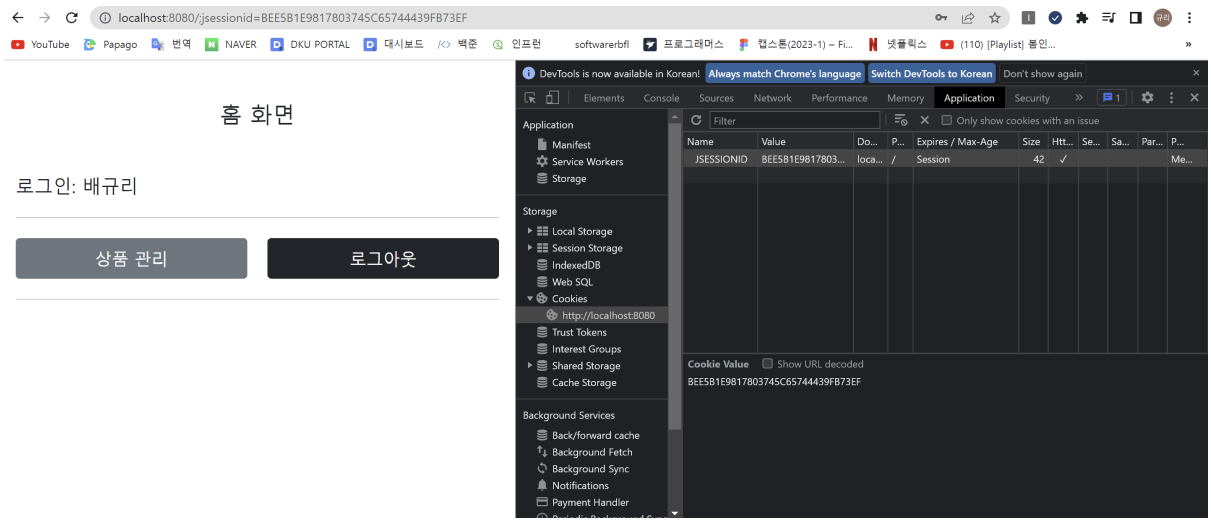
```
    @PostMapping("/login")
    public String loginV3(@Valid @ModelAttribute LoginForm form,
```

```
                        BindingResult result,
                        HttpServletRequest request) {
        if (result.hasErrors()) {
            return "login/loginForm";
        }
        Member loginMember = loginService.login(form.getLoginId(), form.getPassword());

        // 로그인 과정을 거치다가 오류가 난 경우
        // 즉, 아이디에 대한 비번이 맞지 않은 경우
        if (loginMember == null) {
            result.reject("loginFail", "아이디 또는 비밀번호가 맞지 않습니다.");
            return "login/loginForm";
        }

        //로그인 성공 처리
        //세션 있으면 있는 세션 반환, 없으면 신규 세션을 생성
        HttpSession session = request.getSession();

        //세션에 로그인 회원 정보 보관
        session.setAttribute(SessionConst.LOGIN_MEMBER, loginMember);


        return "redirect:/";

    }
```

HomeController도 아래와 같이 수정해주었다.

```
    @GetMapping("/")
    public String homeLoginV3(HttpServletRequest request, Model model) {
        //세션이 없으면 home
        HttpSession session = request.getSession(false);
        if (session==null){
            return "home";
        }

        //회원 정보가 잘못되었거나 없으면 home
        Member member = (Member) session.getAttribute(SessionConst.LOGIN_MEMBER);
        if (member==null){
            return "home";
        }

        //세션이 유지되었으면 로그인된 페이지로 이동
        model.addAttribute("member", member);
        return "loginHome";
    }
```

이와 같이 HttpSession을 사용하여 세션을 생성하게 되면 네트워크 상에서 다음과 같은 쿠키가 생성되는 것을 확인할 수 있다.



로그인 처리하기 - 서블릿 HTTP 세션2

더 편리하게 세션을 사용할 수 있도록 `@SessionAttribute` 사용

```
/**
 * 스프링에서 제공해주는 @SessionAttribute 애노테이션 사용
 * 세션을 가장 편리하게 사용할 수 있는 도구
 * name 인자에 해당하는 쿠키 값이 존재하면 그 객체를, 없다면 null을 반환한다.
 */
@GetMapping("/")
public String homeLoginV3Spring(
        @SessionAttribute(name = SessionConst.LOGIN_MEMBER, required = false)
        Member loginMember,
        Model model) {
    //세션에 회원 데이터가 없으면 home
    if (loginMember == null) {
        return "home";
    }
    //세션이 유지되면 로그인으로 이동
    model.addAttribute("member", loginMember);
    return "loginHome";
}
```

❗참고❗

application.properties에 다음과 같은 코드를 추가하면 URL에 JSESSIONID값이 보이는 것을 제거할 수 있다.

`server.servlet.session.tracking-modes= cookie`

세션 정보와 타임아웃 설정

세션이 제공하는 정보 확인하기

- sessionId : 세션 ID
- maxInactiveInterval : 세션의 유효 시간
- createonTime : 세션 생성 일시
- lastAccessedTime : 세션과 연결된 사용자가 최근에 서버에 접근한 시간
- isNew : 새로 생성된 세션인지, 아니면 이미 과거에 만들어졌고 클라이언트에서 서버로 요청해서 조회된 세션인지 여부

세션 타임아웃 설정

세션은 `session.invalidate()`가 호출되면 삭제된다.

application.properties에서 `server.servlet.session.timeout`값을 지정해주면 된다. → 해당 값은 초단위로 설정된다.!

ex) server.servlet.session.timeout=60 이라면 60초!