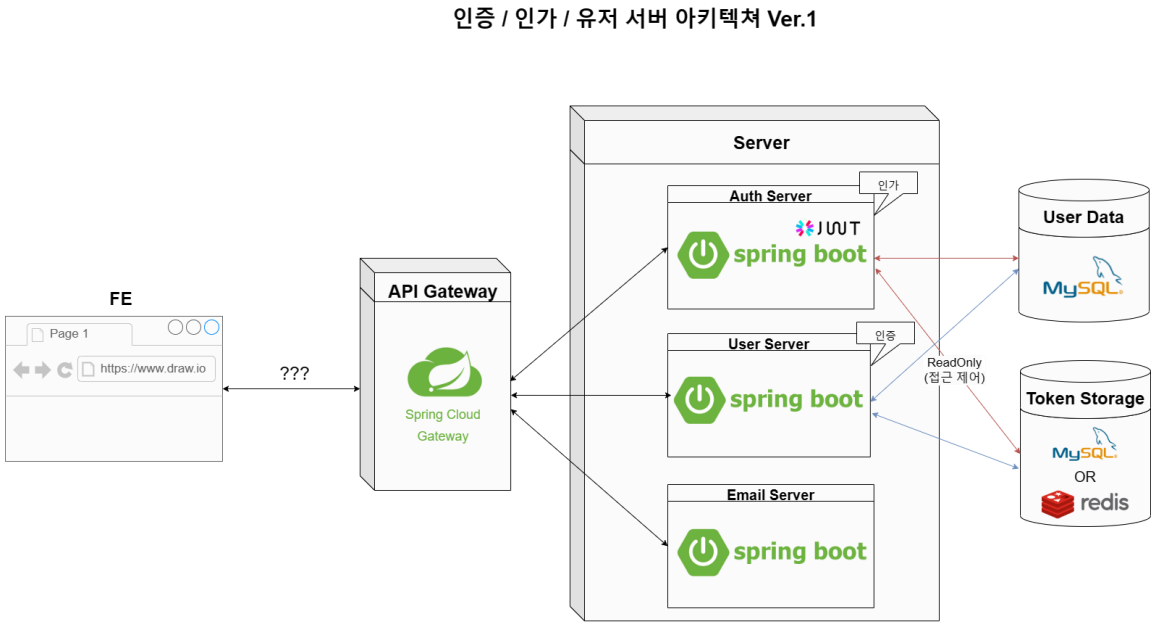
아키텍쳐 설계(23.01.09 회의 이전)

인증(로그인/회원가입)과 유저 관리 서버

- **기능 요구사항 분석**

회원	마이페이지
회원가입	개인정보 수정
로그인 / 로그아웃	비밀번호 변경

- **아키텍쳐(Ver.1)**



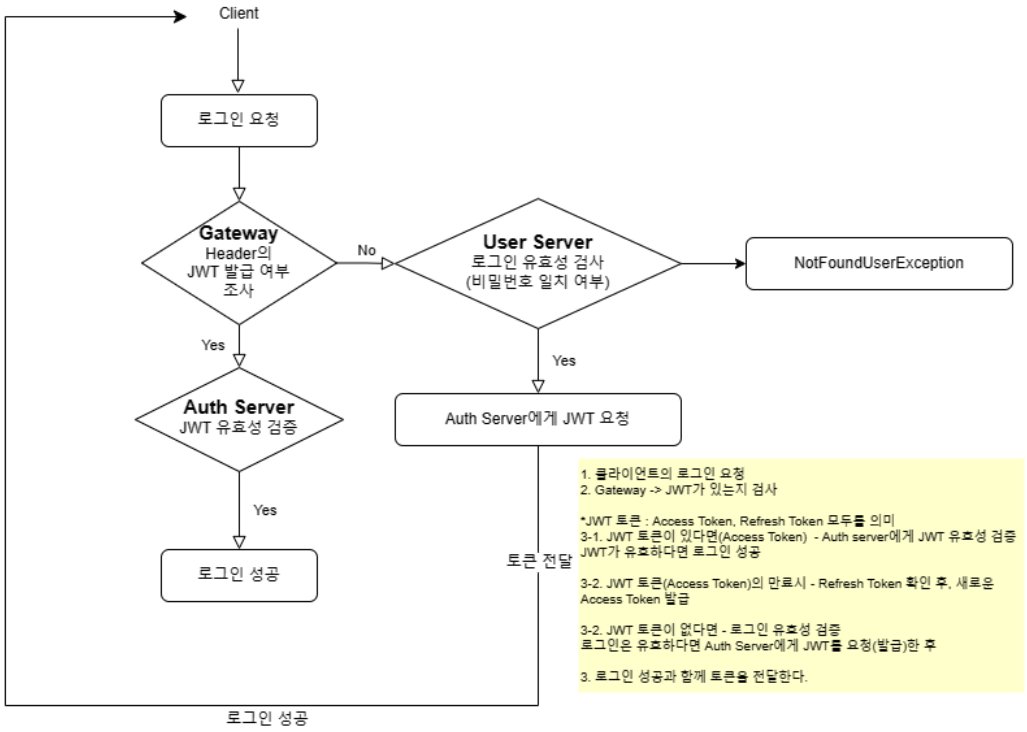
  - 유저 관리 서버 : 기존의 네이버 나우 → 네이버 자체 프로필 사용
    - 별도의 '마이페이지' 구현이 필요할 것으로 보임.
    - User Sever 내에서 → User Data에 접근해 가능할 것으로 보임.

- 예상 시나리오
  - (로그인이 되어 있을 때) → 본인 인증 →User Data가 있는 DB에 UPDATE 쿼리 작성

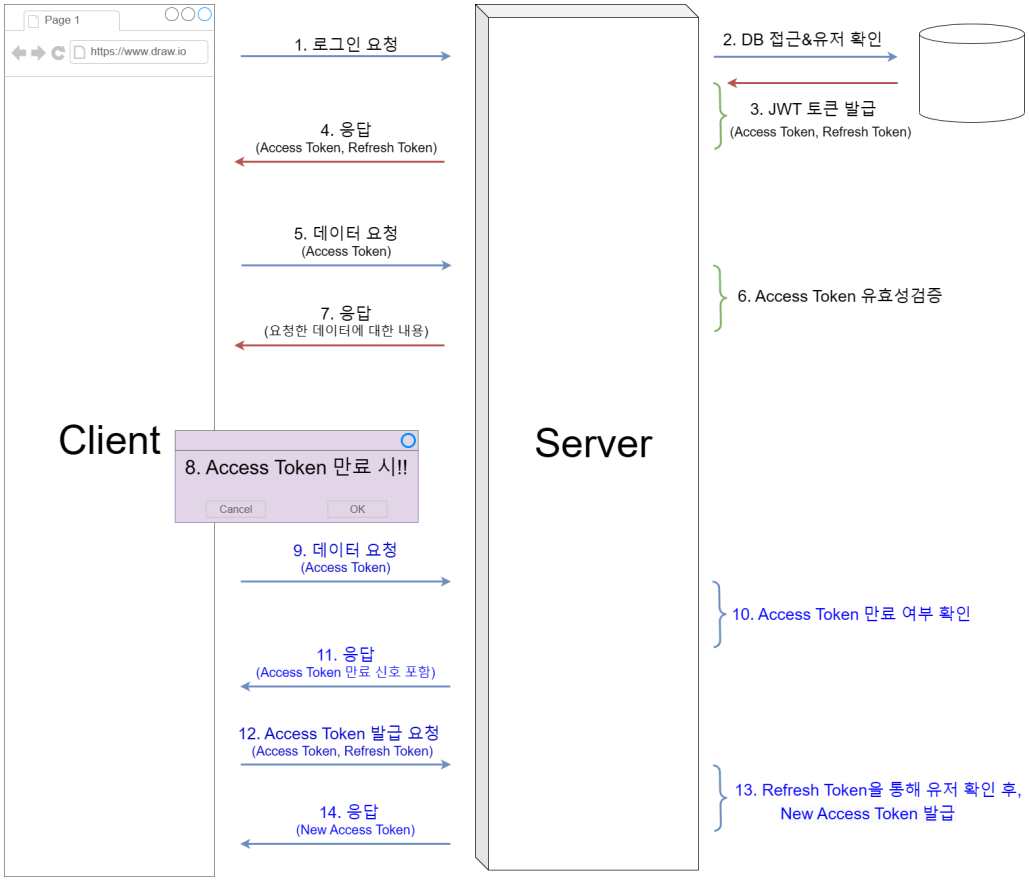
- **서버 플로우**

  ▼ **로그인**

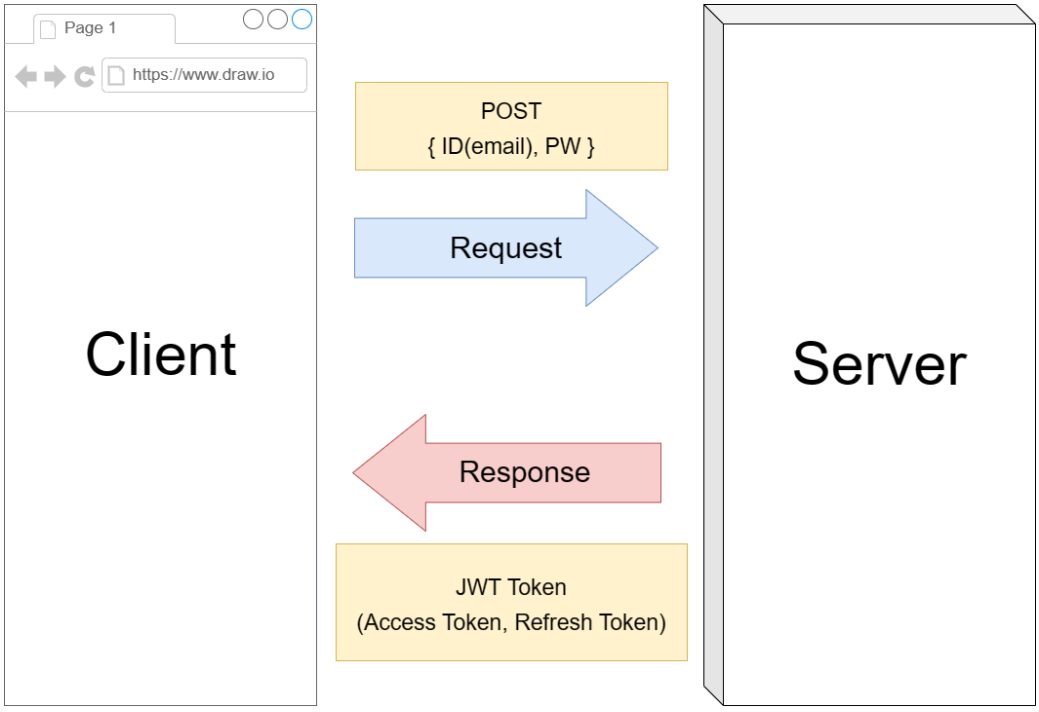
  - 러프한 순서



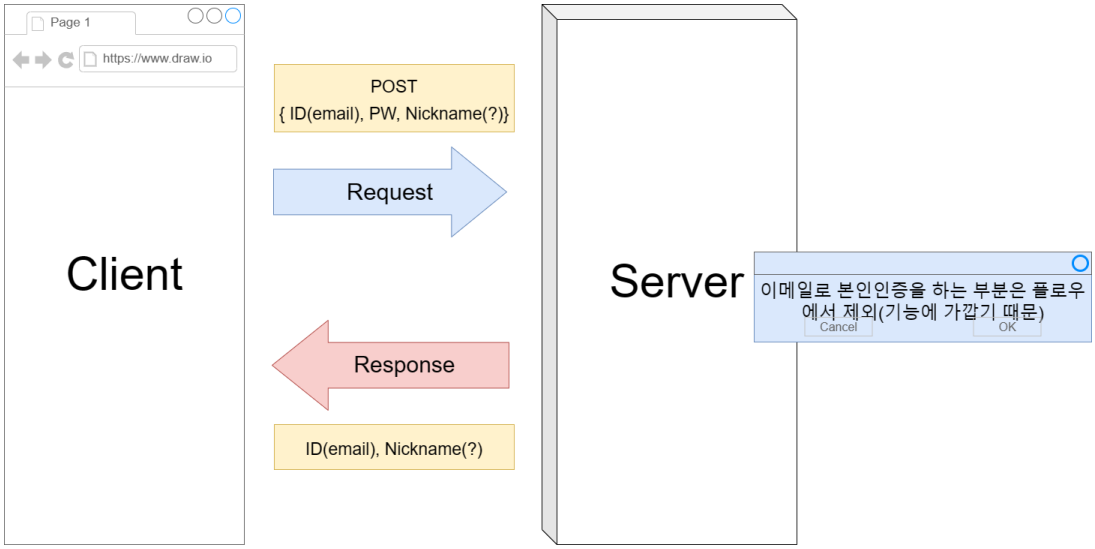
  - 상세 동작


Page 1
https://www.draw.io
1. 로그인 요청
2. DB 접근&유저 확인
3. JWT 토큰 발급
(Access Token, Refresh Token)
4. 응답
(Access Token, Refresh Token)
5. 데이터 요청
(Access Token)
6. Access Token 유효성검증
7. 응답
(요청한 데이터에 대한 내용)
Client
Server
8. Access Token 만료 시!!
Cancel
OK
9. 데이터 요청
(Access Token)
10. Access Token 만료 여부 확인
11. 응답
(Access Token 만료 신호 포함)
12. Access Token 발급 요청
(Access Token, Refresh Token)
13. Refresh Token을 통해 유저 확인 후,
New Access Token 발급
14. 응답
(New Access Token)

- Method 포함

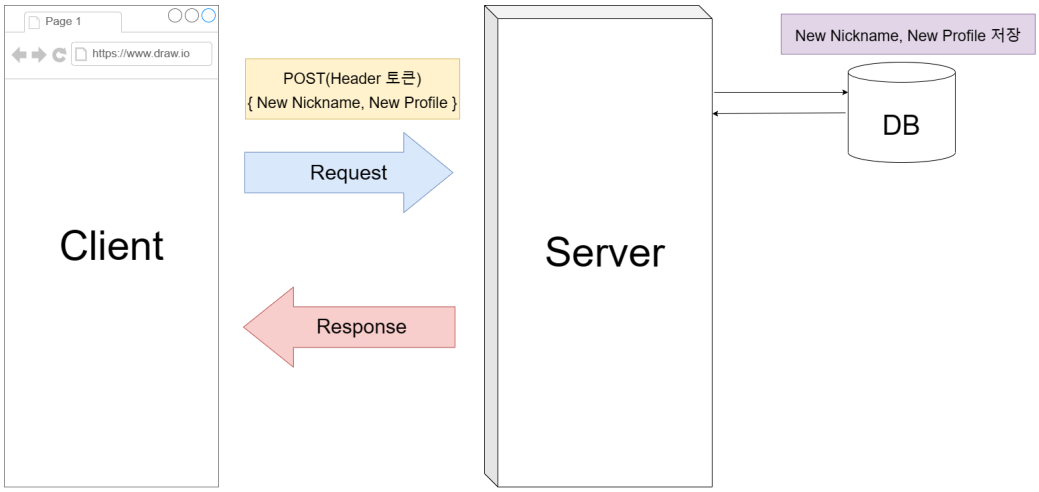

Page 1
https://www.draw.io
POST
{ ID(email), PW }
Request
Client
Server
Response
JWT Token
(Access Token, Refresh Token)

▼ 회원가입

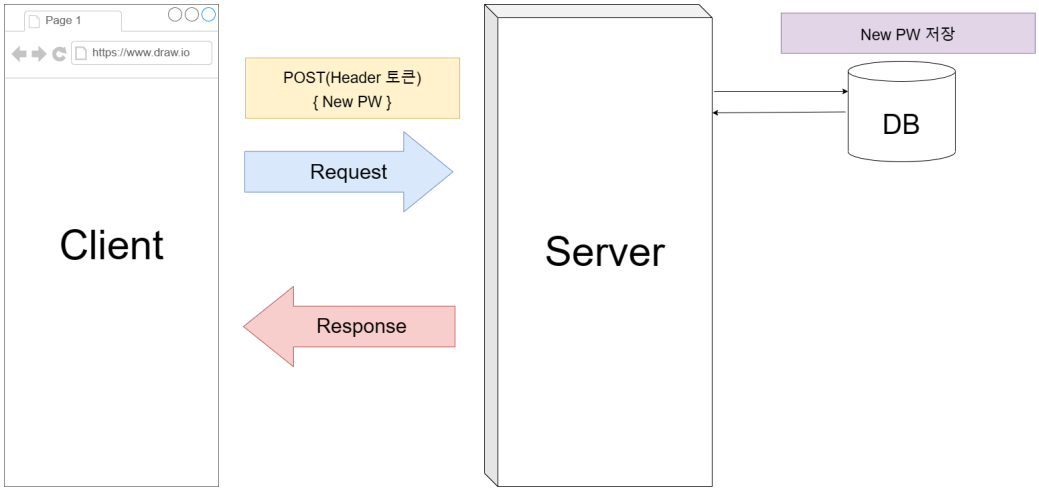


▼ 회원정보 수정

- 개인정보 수정



- 비밀번호 수정(변경)



- **URI & Response 설정**

  ▼ RESTful API의 URI 작성 7가지 규칙 & HTTP 프로토콜 Response 규칙에서 조금 더 구체화할 예정임.

URI

기능	METHOD	URI	ROLE
토큰 발급	POST	/api/v1/auth/token/access	USER / ADMIN
토큰 만료 시 재발급	POST	/api/v1/auth/token/refresh	USER / ADMIN
회원가입	POST	/api/v1/users/sign-up	PUBLIC
로그인	POST	/api/v1/users/sign-in	PUBLIC
로그아웃	POST	/api/v1/users/logout	USER / ADMIN
개인정보수정	POST(or PUT)	/api/v1/users/update (or /api/v1/users/{id})	USER / ADMIN
비밀번호 변경	POST(or PUT)	/api/v1/users/update (or /api/v1/users/{id})	USER / ADMIN
(회원조회)	(GET)	/api/v1/users	ADMIN

RESPONSE

→ 공통적으로 적용 예정

*출처 : HTTP 서버 응답 코드 (Response Code) 정리 (tistory.com)

Response Class Code	의미	설명	비고
1	Informational (정보)	리퀘스트를 받고, 처리 중에 있음.	
2	Success (성공)	리퀘스트를 정상적으로 처리함.	200번대
3	Redirection (리디렉션)	리퀘스트 완료를 위해 추가 동작이 필요함.	300번대
4	Client Error (클라이언트 오류)	클라이언트 요청을 처리할 수 없어 오류 발생.	400번대(클라이언트 측의 잘못된 요청)
5	Server Error (서버 오류)	서버에서 처리를 하지 못하여 오류 발생.	500번대(서버 처리중 문제)

200 300 400 500

200번대 응답 : Success(성공)

200	OK	- 요청 정상 처리
204	No Content	- 요청 정상 처리했으나, 돌려줄 리소스 없음. - 응답에 어떠한 엔티티 바디(Entity Body)도 포함하지 않음. - 서버에서 처리 후, 클라이언트에 정보를 보낼 필요가 없는 경우 사용.
206	Partial Content	- Range가 지정된 요청인 경우 -> 지정된 범위만큼의 요청을 받았다는 것을 알려줌.

200 300 400 500

300번대 응답 : Redirection(리디렉션)

301	Moved Rermanently	- 요청된 리소스에는 새로운 URI가 지정되어 있기 때문에 이후로는 새 URI를 사용해야 한다는 것(영구적인 URI 변경)
302	Found	- 요청된 리소스에는 새로운 URI가 지정되어 있기 때문에 이후로는 새 URI를 사용해야 한다는 것(일시적인 URI 이동)
303	See Other	- 이 응답은 요청에 대한 리소스는 다른 URI가 있기 때문에 GET 메서드를 사용해서 얻어야 한다는 것을 나타냄. - 302코드와 같지만, 303은 리디렉션 위치를 GET 메서드를 통해 얻어야 한다고 명확히 되어 있음.
304	Not Modified	- 요청한 리소스가 마지막 요청 이후 변경된 적이 없기 때문에 기존 클라이언트의 로컬 캐시 리소스를 사용하도록 알려줌. - 300번대로 분류되어 있지만, 리디렉션과는 관계없는 처리를 함.
307	Temporary Redirect	- 임시로 페이지를 리다이렉트함.

200 300 400 500

400번대 응답 : Client Error(클라이언트 에러)

400	Bad Request	- 클라이언트의 요청 구문이 잘못됨. - 브라우저는 이 응답을 200 OK 응답과 동일한 형태로 취급
401	Unauthorized	- 요청 처리를 위해 HTTP 인증(BASIC, DIGEST) 정보가 필요함을 알려줌. - 접근 허용을 차단 -> 최초 요청에는 인증 다이얼로그 표시, 두번째는 인증 실패 응답 보냄.
403	Forbidden	- 접근 금지 응답. Driectory Listing 요청(서버 파일 디렉토리 목록 표시) 및 관리자 페이지 접근 등을 차단하는 경우의 응답 (파일 시스템 퍼미션 거부, 허가되지 않은 IP 주소를 통한 액세스의 거부 등) - 서버는 엔티티 바디에 접근 거부에 대한 이유를 명시하여 보낼 수 있음.
404	Not Found	- 클라이언트가 요청한 리소스가 서버에 없음.
405	Mothod Not Allowed	- 허용되지 않는 HTTP 메서드를 사용함.

200 300 400 500

500번대 응답 : Server Error(서버 에러)

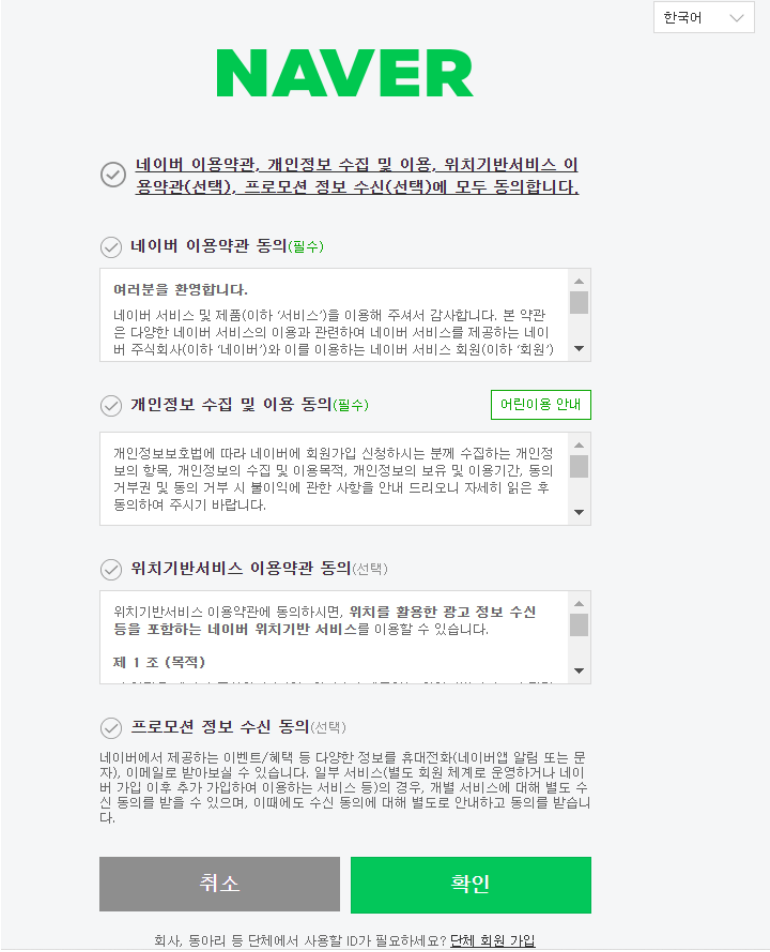
500	Internal Server Error	- 서버에서 클라이언트 요청을 처리 중에 에러가 발생함.
503	Service Unavailable	- 서버가 일시적으로 요청을 처리할 수 없음. - 서버가 과부하 상태이거나 점검중이므로 요청을 처리할 수 없음을 알려줌.
504	Gateway Timeout	- 서버를 통하는 게이트웨이에 문제가 발생하여 시간이 초과됨.
505	HTTP Version Not Supprted	- 해당 HTTP 버전에서는 지원되지 않는 요청임을 알려줌.

- **DB 설계**

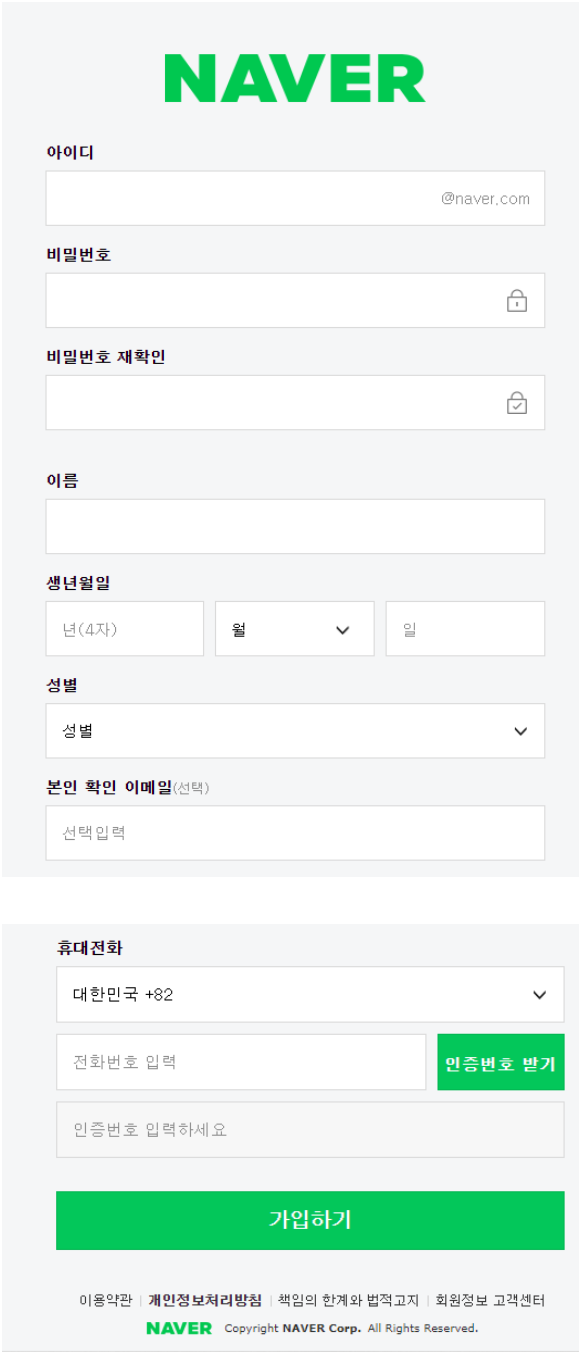
  ▼ 회원 정보 기준

  - 네이버 자체 정보

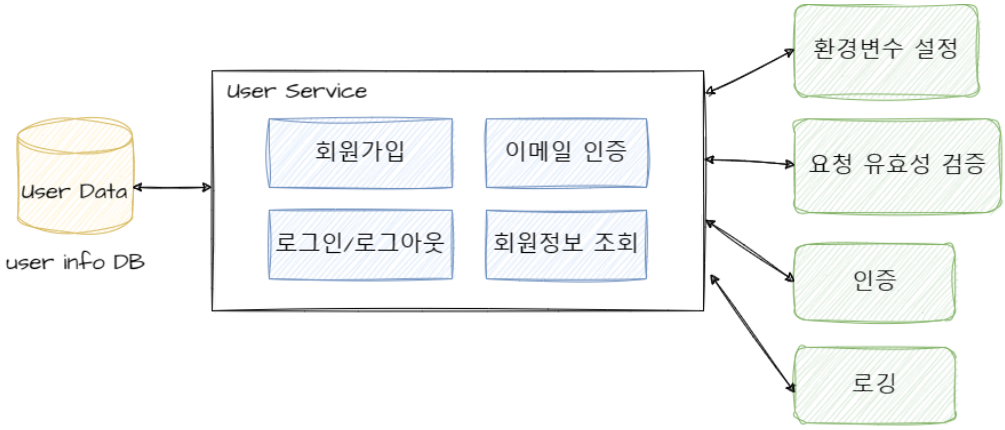
    1) 이용약관 동의

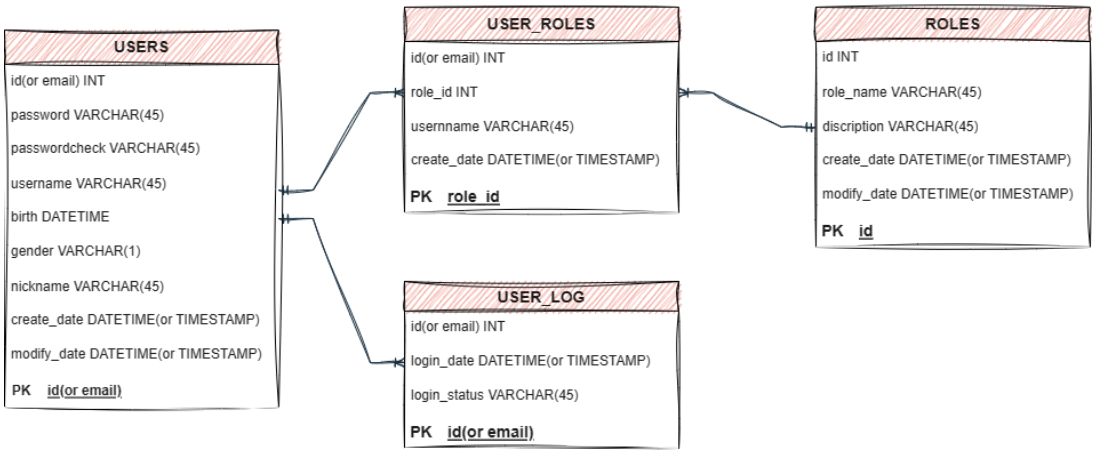


    2) 아이디 / 비밀번호, 비밀번호 재확인 / 이름 / 생년월일 : 년(4자), 월, 일 / 성별 / 본인확인 이메일(선택) / 휴대전화 인증 → 인증번호 입력


NAVER
아이디
@naver.com
비밀번호
비밀번호 재확인
이름
생년월일
년(4자)
월
일
성별
성별
본인 확인 이메일(선택)
선택입력
휴대전화
대한민국 +82
전화번호 입력
인증번호 받기
인증번호 입력하세요
가입하기
이용약관 | 개인정보처리방침 | 책임의 한계와 법적고지 | 회원정보 고객센터
NAVER Copyright NAVER Corp. All Rights Reserved.

▼ ERD

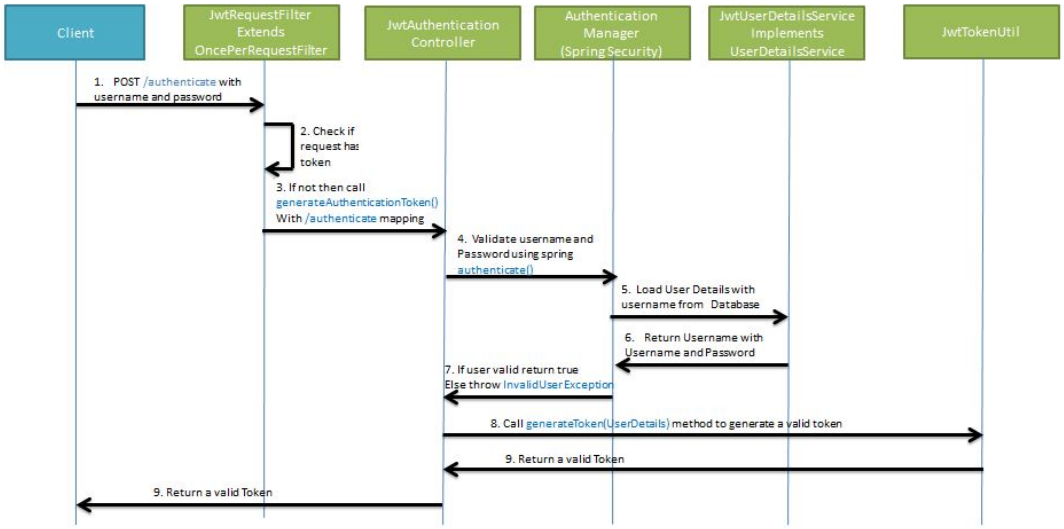




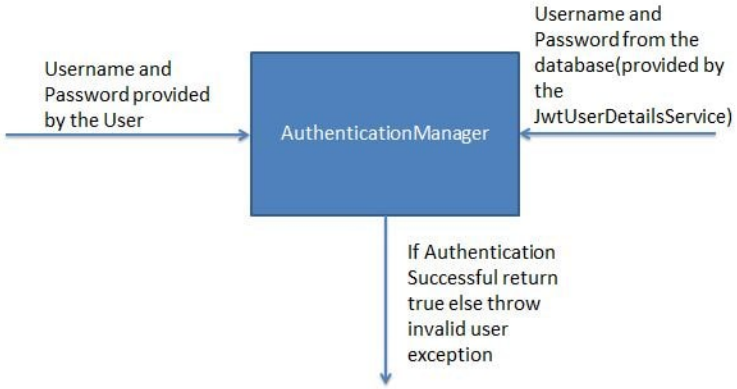
- **UML(클래스 다이어그램) 설계**
  - 기준으로 정해져 있는 표준을 사용할 예정임(출처 : https://github.com/devAon/SpringSecurity-JWT)

  ▼ 동작 과정

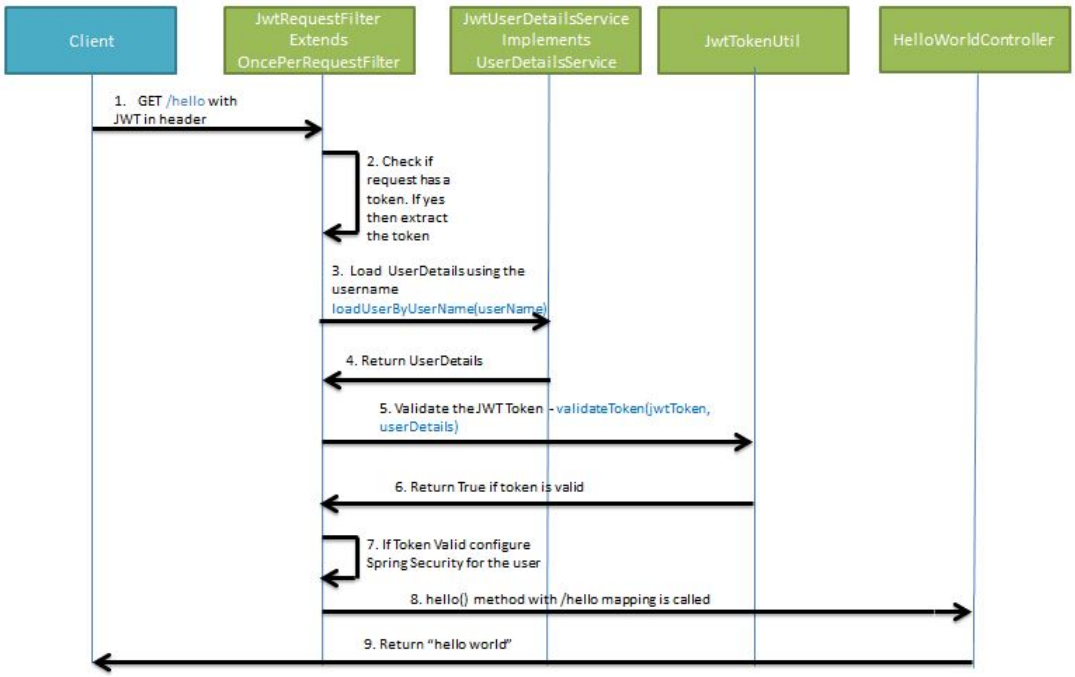
  - Generating JWT



1. Client : 로그인 요청 POST (id, pw)
2. Server : id, pw가 맞는지 확인 후 맞다면 JWT를 SecretKey로 생성
3. Client : Server에게 받은 JWT를 로컬 or 세션에 저장
4. Client : 서버에 요청할 때 항상 헤더에 Token을 포함시킴
5. Server : 요청을 받을 때마다 SecretKey를 이용해 Token이 유효한지 검증
    - 서버만이 SecretKey를 가지고 있기 때문에 검증 가능
    - Token이 검증되면 따로 username, pw를 검사하지 않아도 사용자 인증 가능
6. Server : response



▼ **Token 유효 검증**



1. 클라이언트의 요청 (Header : Token)
2. Spring의 Interceptor에 의해 요청이 Intercept됨
3. 클라이언트에게 제공되었던 Token과 클라이언트의 Header에 담긴 Token 일치 확인
4. `auth0 JWT`를 이용해 `issuer, expire` 검증

알림 서버

- 웹 푸시 알림 기능 사용 → 구독 알림에 사용
- 이전 조사 자료

  🕰️ 구독알림(푸시알림)

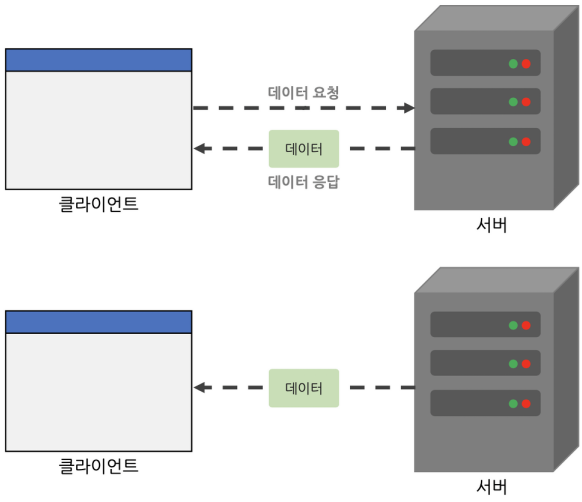
  ⏰ 알림 서버

  23.01.14. 다시 조사

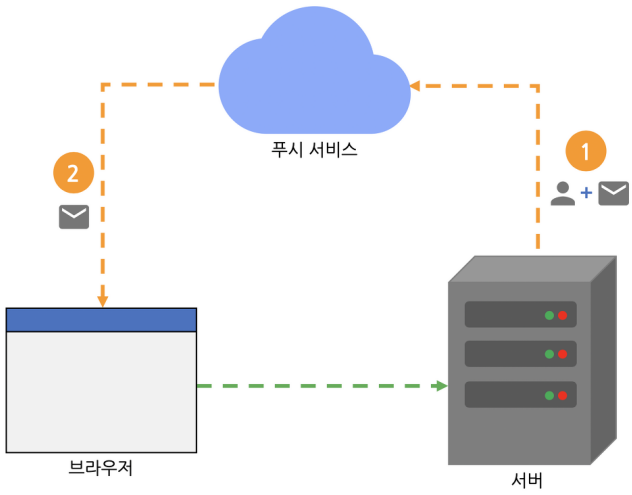
▼ 개념적인 내용

- 웹 푸시 알림
  - 출처(https://geundung.dev/114)

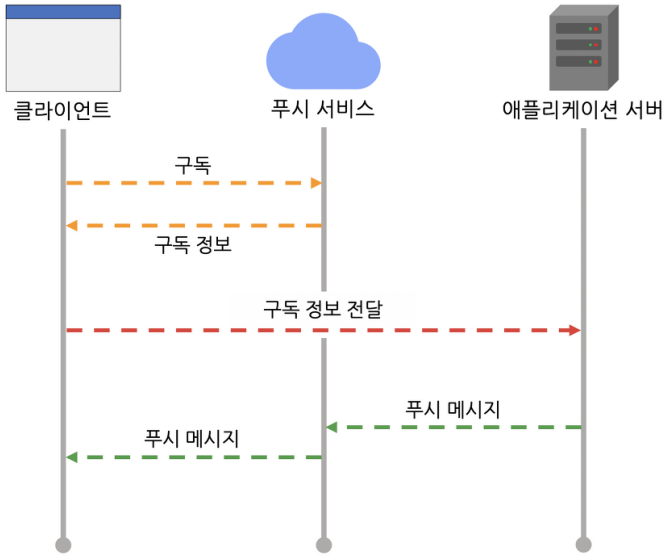
  - **푸시란?**
    - 서버에서 클라이언트로 데이터를 밀어넣는(push) 방식
    - 클라이언트가 먼저 요청을 보내지 않더라도 서버가 데이터를 밀어 넣어주는 형태



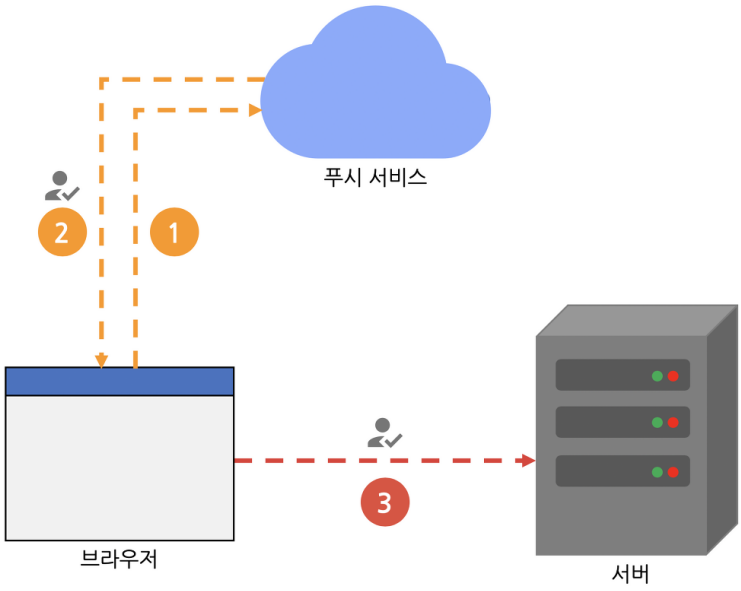
  - **웹 푸시 구성 요소**
    - 브라우저(사용자)
    - 푸시 알림을 발송할 서버
    - 푸시 알림을 사용자에게 전달할 푸시 서비스



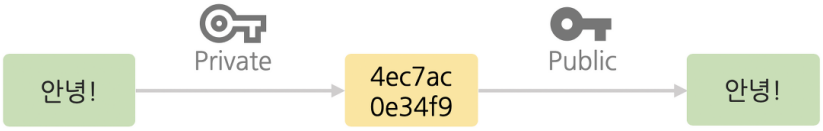
- **웹 푸시 프로토콜**
  - 푸시 알림을 수신하는 브라우저와 발송하는 서버가 푸시 서비스의 상호작용을 하기 위해 정해놓은 규약



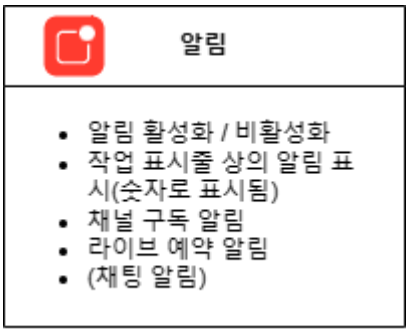
  - 푸시 서비스 구독



- 안전한 메시지를 위한 VAPID



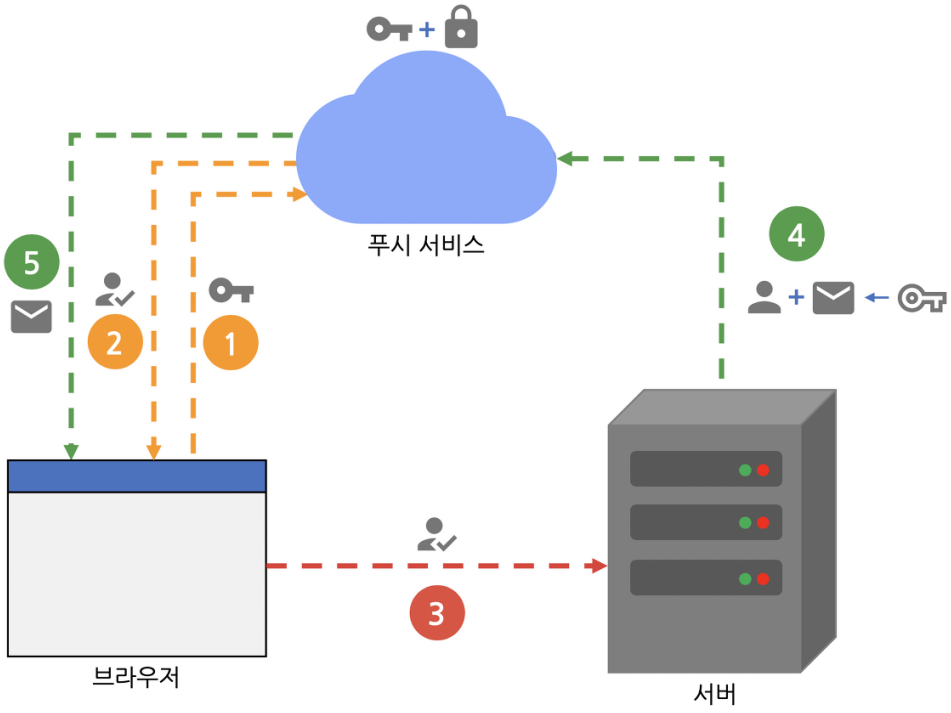
- **기능 요구사항 분석**



- **아키텍쳐(Ver.1) 및 서버 플로우**

→ 정해진 Notificatin API가 있어 그것에 따라서 만들예정(자세한 기술 및 기술 스택은 추후 확정)

- 출처 : https://geundung.dev/114



1) 브라우저에서 푸시 서비스 구독 + VAPID 공개 키 전달

2) 구독 정보 수신

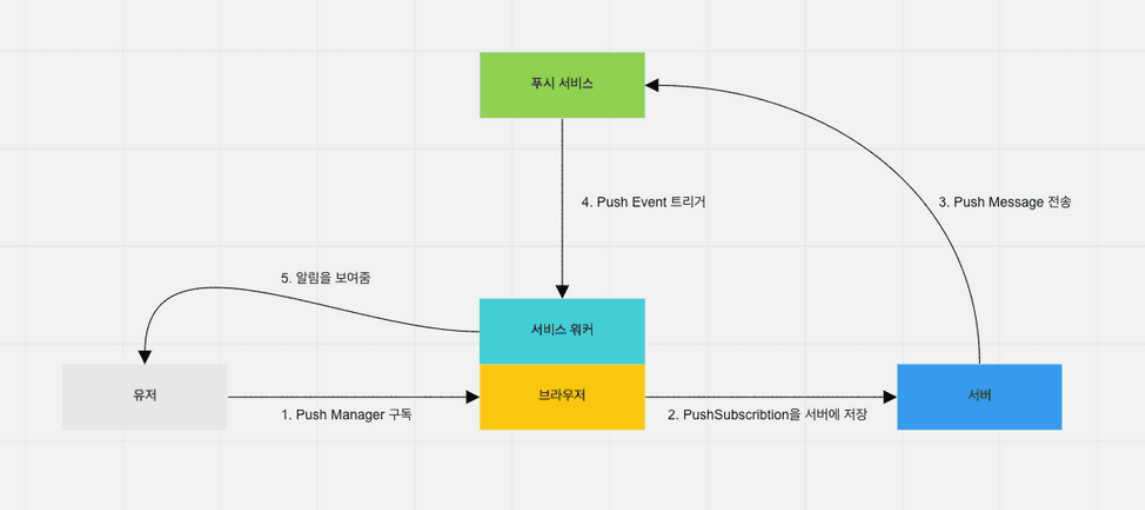
3) 구독 정보 서버로 전달

4) 구독 정보 + 메시지 + VAPID 비공개키로 암호화된 JWT

5) VAPID 공개 키로 유효성 검증 & 검증된 경우 구독 정보에 해당하는 브라우저로 푸시 메시지 발송

*4,5번 과정 : 데이터 위조, 키 불일치 → 푸시 메시지는 유효하지 않은 상태가 되어 브라우저까지 도달하지 못하고 서비스에서 폐기됨.

- 출처 : https://blog.hoseung.me/2021-11-28-web-push-notification/



- **URI & Response 설정**

  ▼ **1. Push Manager 구독**

  가장 첫 번째 단계는 유저가 푸시 매니저를 구독하게 만드는겁니다.

  우선 유저가 브라우저에게 알림을 표시할 수 있는 권한을 줘야 하는데, **Notification API**의 **requestPermission**을 사용해 유저에게 권한을 요청할 수 있습니다.

  유저에게 권한 받기에 성공하면, PushSubscription 이라는 객체를 얻어야 합니다.

  PushSubscription은 **Push API**를 사용해 **Push Manager를 구독**하면 얻을 수 있습니다.

  이 때, Push Manager를 구독하려면 application server keys, 다른 말로 VAPID keys 라는게 필요한데, VAPID keys는 유저에게 푸시 알림을 보내는 서버가 누군지 식별하는 역할을 합니다.

  PushSubscription에는 푸시 알림을 보낼 주소(endpoint)와, 암호화에 필요한 key들이 들어있습니다. endpoint는 유저 하나에 고유하고, key들은 **Web Push Protocal**에 맞춰서 Push Message를 암호화하기 위해 필요합니다.

  ▼ **2. PushSubscription을 서버에 저장**

  1번 단계에서 PushSubscription을 얻었다면, 그걸 서버에 저장하면 됩니다. 이 때, 웹 푸시를 보내기 위해선 VAPID keys를 서버가 갖고있어야 합니다. 환경 변수를 사용하거나, 하드코딩해서 저장해두면 됩니다.

  ▼ **3. Push Message 전송**

  이제 서비스 특성에 따라 자유롭게 알림을 전송해주면 됩니다. 예를 들어, 이커머스 서비스라면 포인트 적립이나 상품 재입고 등을 알림으로 전송할 수 있겠죠.

▼ **4. Push Event 트리거**

푸시 서비스로 Push Message가 전달되면, 서비스 워커에 **Push Event**가 트리거 됩니다.

서비스 워커 스크립트에 Push Event의 핸들러를 작성하여 적절한 처리를 해줄 수 있습니다. 유저가 알림을 클릭했을 때 특정 URL이 열리도록 할 수도 있고, 푸시 알림을 보여주지 않을 수도 있습니다.

▼ **5. 알림을 보여줌**

위의 모든 단계를 거쳐서 최종적으로는 유저에게 푸시 알림이 보여지게 됩니다.

▼ **설계**

웹 푸시에 대한 전반적인 개념을 익혔으니, 개발에 들어가 봅시다.

푸시 메시지 형식 설계

우선 푸시 메시지를 어떤 형식으로 보낼지 정해야 합니다.

제가 필요한건 단순히 알림을 보내는 것도 있었지만, 알림을 클릭했을 때 지정된 페이지가 열리게 만들고 싶었습니다.

따라서 간단하게 아래의 형식으로 정해주었습니다.

```
interface Payload {  title: string;  body: string;  link: string;}
```

title, body는 알림의 내용이고, link는 유저가 알림을 클릭했을 때 띄울 페이지의 링크입니다.

이 때, 푸시 메시지는 string으로 보내야 하는데, Javascript에서 JSON은 자유롭게 stringify/parse가 가능하므로 괜찮습니다.

서비스 워커 스크립트 작성

이제 서비스 워커 스크립트를 작성해봅시다.

서비스 워커는 Javascript 파일을 따로 다운로드 받는 방식입니다. 따라서 React를 쓰고계신 분들은 public 디렉토리에 파일을 생성하시거나, CRA를 사용하시는 분들은 **공식 가이드**를 참고해서 파일을 생성해주시면 됩니다.

```
self.addEventListener("push", (event) => {  const payload = JSON.parse(event.
data.text());  event.waitUntil(    registration.showNotification(payload.titl
e, {      body: payload.body,      data: { link: payload.link },    })  );});
self.addEventListener("notificationclick", (event) => {  clients.openWindow(e
vent.notification.data.link);});self.addEventListener("install", () => {  sel
f.skipWaiting();});
```

3가지 이벤트에 대해 핸들러를 등록했는데, 각각을 설명드리면,

- push
  - 위에서 언급했던 푸시 서비스가 메시지를 받았을 때 트리거하는 이벤트입니다. data를 JSON으로 다시 바꾼 후, showNotification으로 유저에게 보내줬습니다.
- notificationclick
  - 유저가 푸시 알림을 클릭했을 때 발생하는 이벤트입니다. push 이벤트의 핸들러에서 showNotification을 실행할 때 data로 link를 넘겨주는걸 볼 수 있는데, notificationclick 핸들러에서 그 data를 그대로 사용할 수 있습니다.
- install
  - 서비스 워커가 설치됐을 때 발생하는 이벤트입니다. 이미 서비스 워커가 실행되고 있고, 수정된 서비스 워커가 새롭게 다운로드 된 경우, 새로운 서비스 워커는 웹 사이트가 닫혀야 활성화되는데, 이 때 skipWaiting을 실행하면 새로운 서비스 워커를 설치 직후 즉시 활성화 할 수 있습니다.

서비스 워커 등록

이제 브라우저에 서비스 워커를 등록해줘야 하는데, 우선 브라우저가 서비스 워커와 웹 푸시를 지원하는지 확인해야 합니다.

아래의 조건으로 확인할 수 있습니다.

```
if ("serviceWorker" in navigator && "PushManager" in window) {  /* ... */}
```

그리고 아래와 같이 서비스 워커를 등록해주면 됩니다.

```
async function registerServiceWorker(): Promise<ServiceWorkerRegistration> {
return await navigator.serviceWorker.register("/service-worker.js");}
```

저는 서비스 워커 스크립트를 public/service-worker.js에 위치시켰기 때문에 register 메소드의 인자를 저렇게 넘겨주었는데, 본인 상황에 따라 바꿔주시면 됩니다.

Push Manager 구독

다음은 Push Manager를 구독해서 PushSubscription을 얻어올 차례입니다.

위에서 registerServiceWorker를 실행하고 얻은 ServiceWorkerRegistration을 사용해 Push Manager를 구독하면 됩니다.

```
async function subscribePushManager(  serviceWorkerRegistration: ServiceWorke
rRegistration): PushSubscription {  return await serviceWorkerRegistration.pu
shManager.subscribe({    userVisibleOnly: true,    applicationServerKey: /* P
ublic VAPID key */,  });}
```

서버에서 푸시 메시지 전송

이제 모든 준비가 끝났고, 푸시 메시지를 전송하기만 하면 유저에게 알림이 보여질 겁니다.

위에서 Push Message를 보낼 때 Web Push Protocol에 따라 데이터를 암호화해서 전송해야 한다고 언급했었습니다.

하지만 암호화를 구현하기는 복잡하니, npm 패키지인 **web-push**를 사용하면 편합니다.

web-push를 사용해 VAPID keys도 생성할 수 있습니다.

```
webpush.generateVAPIDKeys();
```

우선 setVapidDetails를 사용해 초기 세팅을 해줍시다. 이걸 한번 해주면 나중에 sendNotifcation을 실행할 때 계속해서 VAPID keys 등을 넘겨줄 필요가 없어집니다.

```
webpush.setVapidDetails(  "mailto:hsjang.dev@gmail.com",  /* Public VAPID Key
*/,  /* Private VAPID Key */,);
```

1번째 인자는 이 VAPID키의 소유자의 연락처가 담긴 페이지의 URL이나, mailto 주소가 들어가야 합니다. 전 mailto 주소로 제 이메일을 남겼습니다.

그리고 2, 3번째 인자는 각각 Public / Private VAPID Key를 넘겨주면 됩니다.

설정이 끝났으면, 필요할 때 sendNotification을 실행하면 됩니다.

```
webpush.sendNotification(  pushSubscription,  JSON.stringify({    title: "안녕
하세요",    body: "제가 누군지 궁금하시면 눌러보세요!",    link: "https://about.hoseun
g.me",  }),  {    TTL: 3600 * 12,  });
```

1번째 인자는 PushSubscription입니다. 알림을 보내려는 유저의 PushSubscription을 넘겨주면 됩니다.

2번째 인자는 message string입니다. 위에서 정한 JSON 형식대로 payload를 작성하고, JSON.stringify 해서 넘겨주면 됩니다.

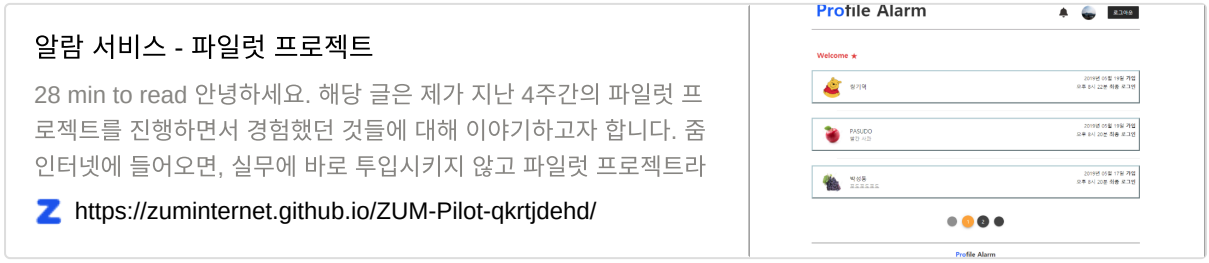
3번째 인자는 필수는 아니고 옵션입니다. 웹 푸시는 생각보다 많은 기능을 지원하는데, 그 중 대표적인게 TTL입니다.

유저의 디바이스가 오프라인이거나, 브라우저가 꺼져있는 등의 상황에서는 알림이 전달되지 못하고 대기 큐에 들어가게 됩니다. 이 때, TTL이 설정되있으면 TTL 내에 메시지가 전달되지 못할 경우 그냥 큐에서 메시지가 삭제됩니다.

전 12시간 내에 전달되지 못할 경우 삭제하도록 설정했습니다.

- **DB 설계(미정)**
- **UML(클래스 다이어그램) 설계(미정)**

→ 위 2가지는 해당 자료를 기본으로 다른 자료들을 조금 더 학습 한 후, 설계할 예정임.



기타 부가적인 기능

- 시간 상 아직 알아보지는 못함.
- 조사 과정에서 기능적인 부분에 과하게 치중됨이 느껴진다면 충분히 고려한 후, 채승님과 이야기해서 프론트, 백엔드를 조금 더 명확하게 나눌 생각.
- 필요한 정보를 어떻게 불러와줄것인지 정도로 설계하고, 기능 구현해볼 생각임.