**기술 면접 – 팀 프로젝트**

**// 분석 및 설계**

**1. UML 문서 작업에 사용한 툴은 무엇인가?**

- Visual Paradigm은 UML 및 다른 표준 모델링 언어를 통해 모델링할 수 있는 툴로써, 도메인 객체 분석 및 백엔드 클래스 다이어그램 작성에 사용하였고, 시퀀스 다이어그램으로 인터페이스 연동을 표현, 컴포넌트 다이어그램으로 서비스 간의 배포관계를 표현하는데 사용하였다.

\*\*클래스 다이어그램의 관계표현

Dependency(의존, -->): 메소드의 매개변수인 객체, 메소드의 리턴 객체, 인스턴스 생성 객체

Association(연관, ㅡ>): 클래스의 필드 또는 속성인 객체

Aggregation(집합, --◇): 연관관계이면서 논리적인 포함관계

Composition(합성, --◆): 집합관계이면서 하위객체의 생명주기를 관할

Generalization(일반화, ㅡ▷): 상위클래스의 필드나 메소드를 상속받아 사용하는 관계

Realization(실체화, --▷): 추상클래스나 인터페이스를 구현한 객체

**// 프론트엔드**

**1. react는 무엇이고 장단점은 무엇인가?**

- react는 facebook에서 개발 및 유지관리하여 인지도가 높으며 많은 커뮤니티 및 라이브러리를 보유하고 있는 프론트엔드 라이브러리이다.

- 변동 사항이 발생할 때마다 dom에 반영하는 방식 대신에 virtual dom 방식의 DOM 트리를 관리하여 렌더링 속도를 높이고 있으며, 렌더링하는 순서 및 로직을 작성하는 절차지향적/명령적 처리 방식 대신에 html과 같이 무엇을 구현하면 될지에 집중하는 선언형 프로그래밍을 사용하고 있다.

- 하지만, 라이브러리의 형태라서 여러 개의 라이브러리를 설치하여 프로젝트를 구성하는 방식은 다소번잡한 과정일 수 있고, 프로젝트 구조가 애플리케이션마다 달라질만한 여지가 많고 개발자 마다 다른 방법으로 개발할 가능성이 높아서 어느정도는 규칙을 정하여 개발하는 것이 유지보수면에서 좋다.

**2. react에서 사용하는 virtual DOM은 무엇인가?**

- html요소(DOM)의 변동사항을 브라우저에 바로 반영하는 대신에 별도의 virtual DOM 트리라고 하는 javascript객체를 가지고, html요소의 정보를 트리구조로 유지관리하며 렌더링 사이클에 따라서 변동된 사항을 일괄로 브라우저의 DOM에 반영하는 구조로 되어있다.

(하단 링크는 vue.js 예시이나 개념적으로 같음)

<https://medium.com/sjk5766/virtual-dom%EC%97%90-%EB%8C%80%ED%95%B4-7222d752ee65>

**4. react에서 props와 state의 차이는 무엇인가?**

- props와 state는 일반 javascript 객체이다. 두 객체 모두 **렌더링 결과물에 영향을 주는 변수**를 갖고 있는데, **props는 함수 매개변수처럼 컴포넌트에 전달**되는 반면 **state는 함수 내에 선언된 변수**처럼 컴포넌트 안에서 관리된다.

<https://ko.reactjs.org/docs/faq-state.html>

**5. function component와 class component의 차이는 무엇인가?**

- function component는 props를 받아 단순히 렌더링만 하는 컴포넌트이고 class component는 컴포넌트안에서 관리되는 state와 라이프 사이클이 가지고 컴포넌트이다. 하지만, react hooks를 통하여 function component에서도 컴포넌트 내부의 state관리나 라이프 사이클에 따른 처리를 추가하여 사용할 수 있다.

**6. react hooks 무엇이고 왜 사용하는 것인가? 장단점은 무엇인가?**

- react hooks는 **기존 function component에 state 관리나 라이프 사이클에 따른 처리 등을 사용할 수 있도록 해주는 확장 기능**으로 16.8버전에 처음 나왔다. 대표적으로 **useState**와 **useEffect**가 있다.

- function component는 간단한 함수와 jsx element만 정의하면 컴포넌트를 만들 수 있다. 그리고, 필요한 경우 react hooks를 통해 상태관리를 추가, 컴포넌트 라이프 사이클에 따른 로직 추가 등과 같이 점진적으로 컴포넌트의 기능을 확장시키는데 class component 보다 유리하다.

- useEffect와 같이 기존 class component의 라이프 사이클을 처리하는 것보다 사이드-이펙트 처리에 편리한 것도 있는 반면, 컴포넌트 내부에 함수 및 변수 선언 부분과 렌더링 부분이 따로 분리되지 않으므로, 렌더링 시에 특정 시점 외에 함수 재생성을 막는 useCallBack을 사용하여야 하고 state외의 변수가 렌더링 될 때마다 값을 유지하기 위해서는 useRef를 사용해야 하는 등의 번잡한 요소가 발생한다.

- 이러한 이유로 hooks를 처리하는 container component와 실제 rendering 처리하는 presentation component 두개로 나누어 구현하는 패턴이 일반적이다. <https://ko.reactjs.org/docs/hooks-intro.html>

**7. SPA란 무엇이고 장단점은 무엇인가?**

- **SPA는 Single Page Application**의 약자로 **페이지 파일**, 주로 index.html **파일 하나에** 자바스크립트로 작성된 **component를 로딩하면서 처리**하는 구조로 되어있는 애플리케이션을 말한다.

- CSR(Client Side Rendering) 방식의 **SPA경우 화면 이동시에** 서버에서 웹 페이지를 받아오지 않아 브라우저에서 웹페이지가 로딩되는 **딜레이가 적은 장점이** 있다. 따라서, 화면전환이 잦은 콘솔형 관리자 화면 구현에 적합하고, 모바일이나 데스크탑 애플리케이션과 유사한 사용자 경험을 제공할 수 있다. 또한, CSR SPA로 처리할 시에 데이터 연동을 위한 백엔드 API호출을 제외하고 렌더링 처리를 클라이언트에서 처리하므로 서버의 부담이 줄어드는 장점이 있다.

- 하지만, **최초 접속 시에 클라이언트에서 번들링(묶음)된 javascript 파일을 받아야** 하므로 **로딩 시간이 길어질 수 있어**, **lazy-loading**을 이용하여 화면 이동 시에 javascript 컴포넌트를 비동기로(XHR) 받아오는 구현도 가능하다. 그리고, 웹페이지를 javascript에서 렌더링하므로 **검색엔진에 최적화(SEO, Search Engine Optimization)가 되지 않을 수 있다는 단점**이 있어, SSR(Server Side Rendeing) 또는 Static Site Generation을 이용하여 최적화하는 방법도 존재한다. 또한 운영시에 웹 컴포넌트가 전부 빌드어야 배포가 가능하므로 특정 페이지의 문구나 이미지 같은 것을 간단하게 수정하려고 하여도 개발자가 빌드하고 배포하는 형태의 작업이 필요하다.

**8. 전역(global) 상태관리는 무엇이고 왜 필요한가? 또한 어떤 경우에 사용할 수 있는가?**

- 트리구조가 너무 깊거나 또는 다른 트리에 있는 컴포넌트에 데이터를 props로 전달하는 것은 힘든 작업이므로 각 컴포넌트간 데이터를 공유하거나 참조할 때 중앙집중적인 state를 관리하는 저장소를 통하여 공유할 수 있도록 하는 것이 전역 상태관리이다.

- 대다수의 컴포넌트 마다 공유되는 데이터들 예를 들어 프로필 정보, 사용자의 애플리케이션 테마 정보, 사용자의 애플리케이션 설정정보 등을 전역 상태관리를 통해 구현할 수 있다.

- 또한, 컴포넌트에서 입력 값을 입력하고 있던 중 다른 컴포넌트로 이동하였다가 원래의 컴포넌트로 돌아왔을 때 이전의 상태를 유지하는 형태 구현이 필요할 때도 사용할 수 있다.

- 그 외에 클라이언트의 애플리케이션의 데이터 처리로직을 집중화하여 로직이 파편화 되는 것을 막을 수 있으나 불필요한 전역 상태관리는 메모리 사용량을 많게하여 필요할 때만 사용하는 것이 좋다.

**9. flux 패턴은 무엇인가?**

- 데이터의 흐름을 단방향으로 흐르도록하여 처리 순서나 방향에 대한 예측가능성을 증가시키는 클라이언트 애플리케이션의 디자인 구조이다.

- 사용자의 이벤트나 특정 시점에 action creator를 통하여 action을 생성하고 단일화된 dispatcher에 action을 전달하여 store를 변경하고 변경된 부분이 view의 전달되는 구조로 되어있다.

<https://haruair.github.io/flux/docs/overview.html>

**10. redux는 왜 사용했으며 프로젝트의 어떤 부분에서 사용하여 구현하였는가?**

- 컴포넌트간 데이터를 공유하기 위하여 redux를 사용하였다.

- (본인에 맞게) 연락처 관리 세부 컴포넌트간 데이터를 공유하기 위해 사용하였다. useDispatcher 훅을 통하여 폼 컴포넌트에서 state에 추가/변경하고 목록 컴포넌트에서 useSelector 훅을 이용하여 상태를 구독하고 조회하는 형태로 구현하였다. 차트데이터는 백엔드에서 데이터만 조회하며 다른 컴포넌트와 데이터가 공유되는 기능이 아니여서 리덕스로 구현하지는 않음.

**11. redux의 action과 reducer는 무엇인가?**

- action은 action creator에서 생성하며, 수행할 액션의 타입과 액션을 처리할 때 사용할 매개변수를 가지는 구조로 되어 있다.

- 디자인 패턴 중에서 요청 객체를 메서드 이름, 매개변수 등을 캡슐화하여 전달하는 커맨드 패턴과 유사하다고 볼 수 있다.

- reducer는 dispatcher로부터 이전 state와 action객체를 매개변수로 받아 action type에 따라서 state를 변경하는 객체이다.

<https://ko.wikipedia.org/wiki/%EC%BB%A4%EB%A7%A8%EB%93%9C_%ED%8C%A8%ED%84%B4>

**12. redux-toolkit은 무엇이고 왜썼는지?**

- name, initialState, reducer 함수목록 등 reducer 구조를 정형화 할 수 있는 createSlice 함수를 제공하고, action creator를 별도로 생성하지 않아도 reducer 함수들 이름의 맞게 action 이 생성됨.

- createAction 처럼 액션을 페이로드 타입과 페이로드 타입만 주면 action creator 함수를 만들어주는 util 메서드도 존재하면, nanoid와 같은 client에서 랜덤한 id를 생성할 수 있도록 하는 메서드도 제공한다.

- 또한 기본적으로 redux-thunk 미들웨어를 제공하지만 프로젝트에서는 redux-saga를 사용하였다.

**13. saga 패턴은 무엇이고 javascript의 generator는 무엇인가?**

- saga패턴은 트랜잭션이나 메시지 처리의 순서를 보장하기 위한 아키텍처 패턴으로써 이전에 처리한 작업이 다음에 수행할 작업을 시작하게 하는 구조로 되어있다.

- javascript의 generator는 함수 내부에 여러 개의 분리된 로직을 필요한 시점에 반복자 객체의 next 함수를 이용하여 순차적으로 처리하도록 하는 방법을 제공하는 문법이다.

**14. redux-saga는 왜 쓰며 프로젝트에 어떤 부분에서 사용했는지 설명하시오.**

- 백엔드 API 서버와 연동하거나 로컬 스토리지에 저장하는 등의 **비동기작업과 redux의 상태변경 작업을 순차적으로 실행하기위해 사용**하는 미들웨어이다.

- (본인에 맞게) redux로 상태관리되는 데이터에 대하여 백엔드에 데이터를 axio로 페이징하여 조회한 후 상태를 업데이트 하고 데이터 추가/변경/삭제 작업에 대하여 백엔드 연동 후 redux 상태를 업데이트하는데 사용하였다. 백엔드에서 지속적으로 데이터를 조회만 하는 기능은 redux 상태관리를 사용하지 않으므로 axios 라이브러리를 이용하여 ajax를 통해 rest api를 호출하여 데이터를 컴포넌트에서 출력하였다.

**15. Next.js는 왜 썼으며 장점은 무엇이고 단점은 무엇인가?**

- Next.js는 프레임워크 기본적인 폴더와 파일구조를 가져 팀프로젝트를 통합하여 작업하기 수월하게 하였음. 또한 기본적으로 페이지 기반 라우팅을 제공하여 별도로 route map 파일을 작성할 필요없이 URL과 매칭되는 페이지파일을 생성하여 직관적이다.

- 또한 처음 페이지를 로딩할 때 일부 컨텐츠를 서버에서 렌더링하여 나오고 일부는 클라인트에서 렌더링하는 Hydration 기법을 사용하여 초기 페이지의 체감적인 로딩속도가 빠르도록 설계되었으며, Next 12버전부터는 빌드속도도 크게 향상시켰다.

- 하지만, 처음 페이지를 로딩할 때 SSR(Service Side Rendering)로 로딩하지만 모든 부분을 렌더링하지 않는데 예를 들어 next/image는 검색엔진 SEO에 필요한 alt속성과 1x1픽셀 base64url 이미지를 서버에서 렌더링한 후에 브라우저에 전달되고, 클라이언트에서 화면에 보이는 영역만 이미지URL이 서버로 호출되어 Lazy Loading한다.

- 그리고 사용자가 페이지 간 이동할 때 next/link 및 next/router는 CSR렌더링으로 처리하여 페이지 컴포넌트만 받아오는 Lazy Loading으로 처리한다. 하지만 next/link를 생성할 때 a태그에 href 속성에 주소가 들어가 있어 검색엔진이 링크를 수집해 갈 수 있으며 a태그의 링크로 직접 이동할 때는 SSR처리된다. CSR의 장점과 SSR 장점을 모두 수용할 수 있는 방법을 제공한다.

- 또한 API Routes를 제공하여 REST API 백엔드를 개발할 수 있고, Prisma와 같은 ORM 스타일의 데이터베이스 라이브러리를 이용하여 풀스택 개발도 가능하며, next-auth와 같은 인증 및 세션처리와 같은 라이브러리도 존재하여 생태계를 갖추고 있는 웹 프레임워크라고 할 수 있다.

- 하지만, SSR처리할 때 외부 라이브러리들이 window, document 등의 브라우저 객체 사용하면 오류가 있을 수 있어, 해당 라이브러리들을 사용할 때는 SSR처리를 하지 않도록 dynmic import를 사용하는 등의 전략이 필요하다.

**// 백엔드**

**1. Spring Boot의 특징은 무엇인가?**

- Spring Boot는 auto-configuration을 지원하여 특정 의존성 라이브러리만 삽입하고 설정에 필요한Bean 클래스를 선언하면 Framework에서 자동으로 Bean 객체를 생성하여 환경을 구성해주거나 또는properties, yml파일 등의 설정정보를 읽어 자동으로 Bean 객체를 생성하여 환경을 구성하여 주는 장점이 있다.

- 또한 UI를 이용한 Starter 도구를 이용하여 프로젝트에 필요한 기본적인 의존성 라이브러리 구조를 생성하여 주는 장점이 있다.

**2. 프로젝트에 사용한 Spring의 컴포넌트, 클래스들은 어떤 것들이 있는가?**

- Controller, 클라이언트에서 백엔드 API를 접근하는 Endpoint 역할을 하면서 요청에 대한 데이터를 검증(validation)하고 처리된 결과를 응답해주는 컴포넌트이다.

- Service, 데이터베이스 트랜잭션 처리 또는 외부 서비스 연동과 같은 비즈니스 로직이 처리되는 컴포넌트이다. 간단한 쿼리의 경우는 직접 컨트롤러에 작성하여 필요하지 않을 수도 있다.

- Entity, ORM의 사상에 따라서 데이터베이스의 물리적인 테이블과 맵핑되는 논리적인 클래스이다. 다른 객체를 참조하는 Relationship 구조나 하위 객체를 포함하는 Aggregate 구조로도 생성할 수 있다.

- Repository, 데이터베이스에 접근하여 CRUD를 처리하는 데이터 액세스 컴포넌트이다. JpaRepository를 상속받아 인터페이스로 작성하여 기본적인 CRUD처리가 가능하고 Spring Framework에서 실제 쿼리를 실행하는 구현객체를 생성하여 처리한다. 또한, Query Creation 패턴에 맞게 함수명을 작성하여 SELECT 문이 생성되도록 작성할 수도 있고, JPQL이라는 객체쿼리나 Native SQL 쿼리도 사용가능하다.

**3. REST는 무엇이고 특징에는 어떤 것이 있는가?**

- REST는 **HTTP 기반위에서 실행되는 프로토콜의 형식**으로써 **HTTP Methods로 행위(Method)를 표현**하고, **URL을 통하여 자원(Resource)를 표현**하는 형식을 가진다.

예) GET /boards - 게시글 목록을 조회한다.

예) GET /boards/paging?page=0&size=1 – 게시글을 페이징하여 조회한다.

예) POST /boards - 게시글 목록에 글 1건을 추가한다.

예) PUT 또는 PATCH /boards/1 - 게시글 목록에서 글 id가 1인 글에 대한 정보를 수정한다.

예) DELETE /boards/1 – 게시글 목록에서 id가 1인 글을 삭제한다.

- **클라이언트/서버(C-S) 구조**로 REST API로 요청하는 클라이언트와 API 요청에 대하여 처리하고 응답을 주는 서버로 나뉘어진다.

- **무상태(Stateless) 프로토콜 방식**으로 요청 간 클라이언트의 상태가 서버에 저장되지 않는다. 따라서 별도의 상태, 세션관리 등의 구현이 필요할 수도 있고, 클라이언트/서버간 연동할 때 클라이언트의 토큰정보에 상태를 저장하는 JWT(Json Web Token)를 이용하는 기법도 존재한다.

**4. ORM은 무엇이고 JPA는 무엇인가?**

- ORM은 **프로그램의 객체와 데이터베이스의 테이블** 또는 컬렉션 등을 **맵핑**하는 개념으로 객체지향 설계와 프로그래밍(OOD/OOP, Object Oriented Design/Programming), 도메인 기반 설계(DDD, Domain Driven Design) 밀접한 관련이 있다.

- 프로젝트 분석/설계 단계에서부터 데이터베이스를 기준으로 분석/설계하는 것이 아닌 **도메인 객체들을 분석하고** 개발 플랫폼에 맞게 객체들을 설계하여 도출된 객체들 중 Entity 타입의 객체를 데이터베이스 **테이블 또는 컬렉션 등과 맵핑**한다.

\*\*도메인 객체: 각 비즈니스 영역별로 주요한 데이터 및 비즈니스 로직을 가지는 객체

\*\*설계 단계에 도출된 객체의 유형

- Boundary Object: 주로 사용자와 시스템 또는 시스템간의 경계에 생성되는 객체로 사용자와 시스템간의 경계에

존재하는 객체는 UI객체(버튼, 리스트)가 될 수 있다.

- Entity Object: 실제 데이터를 가지고 데이터간 관계를 가지고 있는 객체이다.

- Control Object: Boundary Object와 Entity Object를 중계하여 데이터처리를 하거나 데이터 요청에 대하여

응답을 처리하는 객체이다.

- ORM으로 데이터를 액세스하고 처리할 때는 **추상화된 계층을 이용하여** 최대한 데이터베이스에 종속화된 구문을 작성하지 않는 방법을 사용하므로, **특정 DBMS에 종속화되는 것**을 막을 수 있어 애플리케이션의 **독립성이 보장될 수 있다**.

- 또한, **기본적인 CRUD를 추상화된 계층에서 제공하여 개발 편의성**을 제공한다.

- JPA(Java Persistence API)는 자바객체를 저장(영속화, Persist)하고자 할 때 사용할 수 있는 API로써 관계형 데이터베이스는 Hibernate라는 프레임워크로 구현된다. 실제로 구현되는 라이브러리나 프레임워크의 위에 API수준에서 추상화된 계층을 제공하여 관계형 DB 뿐만 아니라 문서형 DB, 키-값형 데이터베이스에 접속하더라고 특정 어노테이션들만 일부 변경될 뿐 구현에 있어서 유사하게 구현할 수 있도록 제공한다.

**5. QueryDSL은 무엇이고 왜 사용하였는가?**

- JPA의 메서드명을 통한 Query 생성 기법은 공학적으로 좋다고 생각되나, 복잡한 구문을 처리하려면 함수명이 너무 길어지고, JOIN쿼리를 사용할 수 없다는 단점이 존재한다. JPQL이나 Native SQL을 사용할 수 있으나, type-safe하지 않아 실행하기전에 오류를 찾기가 어려우며 Java IDE에서 자동완성을 할 수 없다는 단점이 존재한다.

- QueryDSL은 JPA와 잘 결합되어 gradle의 annotation processor를 통하여 엔티티클래스들에 대한 Q타입 클래스를 생성해주며, Q타입 클래스를 통해 type-safe한 쿼리를 작성할 수 있도록 지원한다. 또안 JPA와 같이 연관관계를 생성하지 않아도 join 구문을 처리할 수 있이면, 집계함수(aggregation)를 처리하고 반환 타입을 쉽게 변환할 수 있는 기법을 제공한다.

- (본인에 맞게) 프로젝트에서는 검색조건을 걸어 조회할 때와 [어떠어떠한] 데이터를 집계하여 차트로 보여줄 때 사용하였다.

**6. MSA는 무엇이고 어떤 어떤 장단점을 가지는지?**

- **독립적인 소프트웨어 기능 단위인 서비스**를 주로 공통적인 **네트워크 인터페이스를 통해 느슨하게 결합**되도록 구조화하는 **서비스 지향 아키텍처(SOA) 스타일의 일종**인 소프트웨어 개발 기법이다.

- 서비스간 연동에는 동기화된 처리는 주로 REST나 gRPC, 비동기화된 처리는 메시지 큐를 이용하여 처리한다.

<https://ko.wikipedia.org/wiki/%EB%A7%88%EC%9D%B4%ED%81%AC%EB%A1%9C%EC%84%9C%EB%B9%84%EC%8A%A4>

<https://www.redhat.com/ko/topics/cloud-native-apps/what-is-service-oriented-architecture>

- 각 서비스간 서로 빌드 시간이 공유되는 문제점을 제거하여 **개별적이고 지속적인 배포가 가능**하도록 하고, 서비스별로 단일환경이 아닌 서로 다른 배포환경에서 배포하여 운영할 수 있는 장점을 가진다. 또한, **단일 장애지점을 최대한 없애** 특정 서비스의 문제가 전체 애플리케이션의 문제로 전이되는 문제를 제거하고자 하는 아키텍처이다.

- 하지만, **네트워크를 통해 연동을 하므로 지연성의 문제**가 발생할 수 있으며, 데이터베이스 공유를 통한 손쉬운 **트랙잭션의 정합성 및 일관성 보장의 잇점**을 잃을 수 있으며, 분산된 서비스를 관리하기 힘들어진다는 단점이 생긴다. 또한, 도메인 영역 및 개발/운영 책임자에 대해서 잘못 분리된 서비스들은 불필요한 인터페이스가 발생하여 개발 생산성이 저하되는 문제가 발생하기도 한다.

**7. RabbitMQ는 무엇이고 왜 사용하였는가?**

- RabbitMQ는 AMQP라는 표준 메시지 큐 프로토콜을 사용하는 메시지 큐 브로커(Broker)이다. 직접 Queue로 전송하는 Direct, 여러 개의 Queue에 전송하는 Fanout, 특정 주제에 구독 중인 Queue에 전송하는 Topic 방식으로 전송할 수 있다.

- RabbitMQ를 팀원별로 개발하는 서비스간 연동하는 이벤트 버스 역할을 하는 미들웨어로 사용하였다. 팀원간 연동이 필요한 프로세스 중에서 비동기로 처리가 가능한 프로세스는 RabbitMQ를 이용하여 처리하였다.

**8. Kafka에 대해서 알고 있는지?**

- Kafka는 Hadoop 빅데이터 플랫폼의 한가지 프로젝트로 RabbitMQ와 마찬가지로 메시지 큐 기반의 브로커이다. 성능은 RabbitMQ 대비 더 좋은 편이나, 기본적인 시스템 리소스 사용량이 많아 좀더 좋은 성능의 인스턴스 필요하고 표준 프로토콜을 사용하지는 않으므로 다른 제품으로 대체하기가 어려우며, 주로 데이터나 이벤트 스트림을 수집 처리하는데 사용하므로 RabbitMQ와 같이 라우팅 방법은 다양하지는 않다. 그러나, 일반적인 MSA 애플리케이션의 이벤트 스트리밍 처리에 많이 사용하고 있다.

**9. Spring Cloud(API) Gateway는 무엇이고 왜 사용하는가?**

- Cloud(API) Gateway는 **백엔드 서비스 API들의 진입점이고 통합**적인 API 로깅, 모니터링 및 인증, 보안처리를 가능하게 한다.

- 클라이언트에서 백엔드 API 호출 시에 단일 호스트 주소를 이용하고, 요청에 대한 URL 패턴을 통하여 라우팅함으로써 백엔드 서비스의 인프라 환경의 변화(IP주소변경, 서비스 URL 패턴 변경)와 관련없이 클라이언트에서는 동일한 엔드포인트를 호출하도록 함으로써, 클라이언트와 백엔드간의 결합도를 낮출 수 있다.

**10. Eureka는 무엇이고 왜 사용하였는지?**

- Eureka는 **서비스들의 논리적인 이름과 물리적인 주소**를 맵핑하여 **목록으로 가지고 있는** **Service Registry**이다.

- 서비스의 이름 및 주소를 등록할 때 Eureka 서버에서 설정을 통하여 등록하는 것이 아닌, **Eureka 클라이언트에서 등록할 수 있도록** 하여 주소를 서버에서 수동으로 관리해야하는 관리 포인트를 줄이고, 주소 변경의 유연성을 부여하였다.

- 또한 Eureka 클라이언트들은 Registry의 주소를 통하여 다른 서비스로 접근할 때 클라이언트 사이드 로드밸런싱을 할 수 있는 Ribbon이 내장되어 있다.

**11. PostgreSQL을 사용한 이유는 무엇이고 SQL은 어느정도 작성가능한지?**

- AWS RDS 서비스로 손쉽게 프리티어 인스턴스를 생성하여 사용가능 하였고, 기본적인 ANSI SQL과 데이터베이스를 배우는데 크게 문제가 없으며 주로 데이터베이스 접근은 ORM 방식으로 사용하여 오라클 대신에 PostgreSQL로 강의를 진행하였음.

- 그리고, MSA환경에서 경량의 오픈소스 데이터베이스(MySQL, MariaDB, PostgresQL)를 좀 더 선호하는 경향이 있어 프로젝트에서는 PostgreSQL을 사용하였음.

- 기본적인 SQL DML 및 쿼리문 정도만 사용가능하며, 강의 및 프로젝트에서는 주로 JPA로 작성하여 복잡한 SQL문 작성은 익숙하지 않다.

**12. Redis는 왜 사용하였고 어떤 부분에 사용하였는가?**

- 자주 조회되지만 변동성이 적은 데이터를 로컬 서버에 존재하는 Redis에 캐싱하여 네트워크를 통한데이터베이스 접속에 따른 응답성능 저하를 개선하도록 구현하였다.

- 강의 간에는 인증처리 후 만료시간이 있는 **세션정보 저장 및 프로필 정보를 캐싱**하는데 사용하였으나 프로젝트에서는 시간관계상 적용하지 못하였다.

**13. 관계형 DBMS와 NoSQL DBMS의 차이는 무엇인가?**

- 관계형 DBMS는 데이터 간의 정합성 및 높은 일관성을 보장하도록 구현된 데이터베이스 엔진으로써 대표적으로 오라클, MySQL, MS SQL Server가 있다. 하지만 모든 데이터가 데이터간의 관계가 복잡하고 일관성의 순위가 꼭 높아야 되는 것만은 아니였고, 데이터변경/삭제보다 데이터의 추가/조회가 자주일어나는 상황이 발생하였다. 또한, SQL을 이용한 검색은 유연성은 좋으나 대용량 데이터를 처리하기에는 성능이 떨어졌다.

- 따라서, 데이터 관계는 단순하면서 검색이나 조회가 빈번하게 발생하는 경우, 관계형 DBMS에 추가/변경된 데이터를 별도의 NoSQL DBMS에 캐시하여 잠금현상을 방지하여 조회할 수 있도록 하거나, 시계열(시간의 흐름에 따라 쌓이는) 데이터들의 빠른 처리 등에 NoSQL 데이터베이스가 사용되고 있다.

**14. S3는 왜 사용하였는가? CloudFront는 왜 사용하는가?**

- 클라이언트에서 업로드하는 파일을 구현한 스프링 백엔드를 통하여 S3버킷에 저장하도록하였고, Cloudfront 배포URL을 통하여 파일을 접근할 수 있도록 하였다.

- S3는 저장소 사용량 및 요청 수, 데이터 전송 양에 따라서 과금이 되는 AWS 서버리스 인프라로 높은 가용성을 가지고 있고 동시처리량이 우수한 서비스이다.

- CloudFront는 CDN 서비스이며 컨텐츠를 요청하는 클라이언트와 가까운 지점에 컨텐츠 캐시를 생성하여 접근할 수 있도록 하는 서비스이다. 팀 프로젝트에서는 필수 사항은 아니었으나 S3로 배포한 애플리케이션을 https배포로 손쉽게 할 수 있어 사용하였다.