웹개발 기술면접 질문리스트

질문사항	모범답안
React와 Spring Boot를 조합해 사용한 기술적 이유는 무엇인가요?	React는 가상 DOM 기반 렌더링으로 빠르고, 컴포넌트 단위 개발로 재사용성과 생산성이 높습니다. 반면 Spring Boot는 REST API를 쉽게 구축할 수 있고, JPA와의 통합으로 DB 연동이 간단합니다. 이 둘을 분리형으로 구성하면 비동기 통신 기반의 SPA 구조를 구현할 수 있고, 배포 시에도 React는 S3, Spring Boot는 EC2에 각각 올릴 수 있어 확장성과 독립성이 큽니다. 또한 이 구조는 실제 기업에서 많이 사용하는 아키텍처라 실무 적합성이 높았습니다.
프론트엔드와 백엔드 간 데이터 통신 구조를 구체적으로 설명해 주세요.	데이터 통신은 Axios를 이용한 RESTful API 기반으로 구성했습니다. 요청 시 Content-Type: application/json 헤더를 지정하고, 백엔드에서는 @RequestBody로 JSON 데이터를 수신합니다. 응답은 ResponseEntity를 이용해 JSON 형태로 반환하며, 상태 코드(200, 400, 500 등)를 명시했습니다. 예를 들어 로그인 시 Axios에서 /member/login에 POST 요청을 보내면, 백엔드는 DB에서 아이디와 비밀번호를 검증 후, 결과를 JSON으로 반환합니다.
CORS문제를 어떻게 인식하고 해결하셨나요?	로컬 개발 시 React(3000번 포트)와 Spring Boot(8080번 포트)가 다르기 때문에 CORS 오류가 발생했습니다. 이를 해결하기 위해 Controller 레벨에서는 @CrossOrigin(origins = "http://localhost:3000")을 추가했고, 전역 설정에서는 WebMvcConfigurer를 구현해 addCorsMappings 메서드로 모든 경로(/**)에 대해 React 서버를 허용했습니다. 배포 환경에서는 S3와 CloudFront 도메인을 추가로 허용하도록 수정했습니다.
React에서 상태관리는 어떤 방식으로 구현했나요?	상태 관리는 전역 상태는 Redux Toolkit, 로컬 컴포넌트 상태는 useState와 useReducer를 병행해서 사용했습니다. Redux에서는 로그인 상태(isLogin), 사용자 정보, 장바구니 목록 등을 전역적으로 관리했고, API 호출 시 createAsyncThunk를 사용해 비동기 액션을 처리했습니다. 이렇게 하면 로그인 여부나 장바구니 정보가 페이지 전환 간에도 유지되므로 UX가 좋아졌습니다.
React에서 API호출시 비동기 처리를 어떻게 제어했나요?	useEffect 내부에서 async/await를 사용하여 데이터를 불러왔습니다. 요청 전후에 isLoading 상태를 설정해 로딩 스피너를 표시했고, 에러가 발생하면 try-catch 문에서 에러 메시지를 화면에 노출했습니다. 또한 Redux Toolkit의 createAsyncThunk를 사용하면 액션 상태(pending, fulfilled, rejected)에 따라 자동으로 로딩/에러 처리가 가능했습니다.
컴퍼넌트 간 데이터 전달 구조는 어떻게 설계했나요?	기본적으로 상위 컴포넌트에서 하위 컴포넌트로는 props를 사용했습니다. 다만, props drilling이 발생하는 경우에는 Context API나 Redux를 이용했습니다. 예를 들어 로그인 상태나 사용자 이름 같은 공통 데이터는 AuthContext로 관리해 Header, Cart, MyPage 등 어디서든 쉽게 접근할 수 있게 했습니다.
Spring Boot REST API 설계시 가장 중요하게 고려한 점은 무엇인가요?	RESTful 원칙에 따라 리소스 중심 URL 설계를 지켰습니다. 예: /api/products, /api/products/{id}, /api/members/{id}/orders HTTP 메서드에 따라 의미를 분리했습니다. 이로 인해 협업 시 API 문서만으로도 직관적으로 이해가 가능했습니다.
JWT 토큰 인증 절차를 자세히 설명해 주세요.	로그인 성공 시 사용자 정보를 기반으로 JWT 토큰을 생성합니다. 이 토큰에는 사용자 ID와 유효기간이 포함되어 있으며, 클라이언트는 이후 요청 시 Authorization: Bearer<token>헤더로 전송합니다. Spring Boot의 OncePerRequestFilter를 상속한 JwtFilter에서 모든 요청의 헤더를 검사하여 토큰의 유효성을 검증하고, 문제가 없으면 SecurityContext에 인증 객체를 등록해 Controller까지 전달합니다.
성능개선을 위해 어떤 기술 또는 방법을 사용했나요?	React에서는 useMemo, useCallback, React.memo를 통해 불필요한 리렌더링을 방지했고, 이미지는 Lazy Loading 처리했습니다. Spring Boot에서는 JPA의 fetch join을 활용해 N+1 문제를 해결했고, DB 인덱스를 추가하여 조회 속도를 개선했습니다.

	또한 프론트엔드 정적 파일은 gzip 압축과 캐싱을 적용해 초기 로딩 속도를 단축했습니다.
프로젝트 진행 중 가장 어려웠던 문제와 해결과정을 구체적으로 말해주세요.	WT 만료 후 자동 로그아웃 문제와 CORS 예외 처리가 가장 어려웠습니다. 초기에는 토큰 만료시 백엔드에서 에러만 던져 사용자 경험이 좋지 않았습니다. 이를 개선하기 위해 Axios 인터셉터에서 401 응답을 감지해 자동 로그아웃 및 재로그인 페이지로 이동하도록 처리했습니다. CORS는 개발 단계에서는 @CrossOrigin으로 해결했지만, 운영 환경에서는 nginx reverse proxy 설정으로 제어했습니다.
Git 브랜치 전략과 협업방식은 어떻게 관리했나요?	GitHub Flow를 기반으로, main은 배포용으로 유지하고 기능별 브랜치를 feature/login, feature/cart 형식으로 생성했습니다. 작업 완료 후 Pull Request를 통해 코드 리뷰를 진행했고, merge 후 GitHub Actions로 자동 빌드 테스트를 실행했습니다. 또한,gitignore로 IDE 설정이나 build 파일을 제외하여 충돌을 최소화했습니다.
AWS배포 과정과 설정을 구체적으로 설명해 주세요.	React는 npm run build 후 생성된 정적 파일을 S3 버킷에 업로드하고, CloudFront로 CDN 캐시를 적용했습니다. Spring Boot는 EC2에 JAR 파일을 배포하고 nohup java -jar app.jar &로 실행했습니다. DB는 AWS RDS(MySQL)를 사용했고, EC2 보안그룹에서 3306 포트를 열어 RDS와 통신하며 S3, EC2, RDS를 IAM 역할로 연동하여 보안도 강화했습니다.
Spring과 Spring Boot의 차이	pring은 필요한 설정을 개발자가 직접 구성해야 하는 설정 중심 프레임워크입니다. 반면 Spring Boot는 Starter 의존성과 AutoConfiguration 덕분에 기본 설정이 자동 적용되어 훨씬 빠르게 개발을 시작할 수 있습니다. 또한 내장 Tomcat이 있어 별도 배포 없이 실행할 수 있고, application.yml을 이용해 환경별 프로파일 관리도 간단하게 할 수 있습니다.
의존성 주입이란? or IoC와 DI의 관계를 설명해보세요	의존성 주입(DI)은 객체를 직접 생성하지 않고, 필요한 객체를 외부에서 주입받는 방식입니다. Spring의 IoC 컨테이너가 객체를 만들고 관리하기 때문에 개발자는 생성 방식에 신경 쓰지 않고 비즈니스 로직에 집중할 수 있습니다. 주입 방식은 생성자 주입이 가장 많이 쓰이는데, 순환 참조를 방지하고 테스트하기 좋기 때문입니다.
Spring Bean이 생성되고 소멸되는 과정을 설명해보세요	Spring Bean은 컨테이너가 생성하고 필요한 의존성을 주입한 뒤, @PostConstruct 메서드를 먼저 호출해서 초기화 작업을 실행합니다. 애플리케이션이 종료될 때는 @PreDestroy가 실행되며 커넥션 정리 같은 정리 작업을 수행합니다. 상황에 따라 InitializingBean 같은 인터페이스로도 초기화 로직을 넣을 수 있습니다.
AOP를 실제로 적용해본 경험이 있나요	AOP는 여러 곳에서 반복되는 공통 기능을 분리할 때 사용했습니다. 예를 들어 로깅이나 서비스 단의 실행 시간을 측정하는 기능을 Aspect로 만들고, Pointcut으로 어떤 메서드에 적용할지 지정했습니다. 실제 실행될 코드는 Advice에 정의해서 핵심 비즈니스 코드를 건드리지 않고 공통 로직만 분리할 수 있었습니다.
API란 무엇인가요?	API는 서로 다른 프로그램이나 시스템이 데이터를 주고받을 수 있게 만든 인터페이스입니다. 보통 웹 개발에서는 클라이언트가 서버에 요청을 보내고, 서버가 JSON 같은 형태로 응답을 주는 방식으로 사용됩니다. 즉, 정해진 규칙에 따라 통신할 수 있게 해주는 약속이라고 이해하고 있습니다.
RESTful이란 무엇인가요?	RESTful은 HTTP 프로토콜을 기반으로 자원을 URI로 표현하고, HTTP 메서드(GET, POST 등)를 통해 자원을 조작하는 API 설계 방식입니다. REST의 원칙을 잘 지킨 API를 RESTful API라고 합니다.
@Transactional은 내부적으로 어떻게 동작하나요	@Transactional은 메서드 실행 전 트랜잭션을 시작하고, 정상 종료되면 자동으로 커밋, 예외가 발생하면 롤백하는 방식으로 동작합니다. 내부적으로 AOP 프록시가 메서드를 감싸며 실행 흐름을 제어합니다. 또한 런타임 예외는 자동 롤백되지만, 체크 예외는 기본적으로 롤백되지 않기 때문에 rollbackFor로 지정해야 합니다. 상황에 따라 전파 옵션(Propagation) 과 격리 수준(Isolation) 을 설정하여 트랜잭션 경계를 세밀하게 조절할 수 있습니다.
클라이언트 요청이 Controller로 전달되는 과정을 설명해보세요	클라이언트 요청은 먼저 DispatcherServlet에 전달됩니다. DispatcherServlet은 HandlerMapping을 통해 어떤 Controller가 요청을 처리할지 찾고, 해당 Controller에서 비즈니스 로직을 수행한 뒤 Model과 View 정보를 반환합니다. 이후 DispatcherServlet은 ViewResolver로 적절한 View를 선택해 최종 응답을 클라이언트에게 전달합니다. 즉, DispatcherServlet을 중심으로 요청 → Controller → View로 처리 흐름이 진행됩니다.
RESTful하게 설계하려면 어떤 점을 주의해야 하나요?	RESTful 설계를 위해서는 몇 가지 원칙을 지켜야 합니다. 첫째, URI는 동사가 아닌 명사 중심의 자원 표현을 사용하고, 계층 구조를 명확하게 드러내야 합니다. 둘째, 자원에 대한 동작은 HTTP 메서드인 GET·POST·PUT·DELETE로 구분하여

	일관된 패턴을 유지하는 것이 중요합니다. 셋째, 모든 응답은 JSON 형태로 통일하고, 상황에 맞는 HTTP 상태 코드(200, 201, 400, 404 등)를 사용해 API의 예측 가능성을 높여야 합니다. 넷째, /api/v1처럼 버전 관리를 통해 확장성과 유지보수성을 확보해야 합니다. 또한 REST의 핵심인 무상태성(Stateless) 을 지켜, 요청마다 필요한 정보를 모두 포함하도록 설계해야 합니다. 마지막으로 Swagger나 OpenAPI로 문서화를 진행해 협업과 유지보수를 용이하게 하는 것이 RESTful 설계의 중요한 요소입니다.
JPA의 N+1 문제는 왜 발생하나요?	JPA의 N+1 문제는 연관된 엔티티를 지연 로딩(Lazy Loading) 으로 가져올 때, 처음 1번의 조회 이후 연관 데이터마다 추가로 쿼리가 실행되면서 발생합니다. 즉, 하나의 쿼리가 N개의 추가 쿼리를 만들어내는 구조입니다. 이를 해결하려면 fetch join, EntityGraph, 또는 배치 사이즈 설정(@BatchSize) 을 사용해 연관 데이터를 미리 한 번에 조회하도록 설정합니다.
로그인 요청이 들어왔을 때 Spring Security는 어떤 절차로 인증하나요?	로그인 요청이 들어오면 UsernamePasswordAuthenticationFilter가 ID·비밀번호를 추출해 AuthenticationManager로 인증을 위임합니다. AuthenticationManager는 UserDetailsService와 PasswordEncoder를 통해 사용자를 검증하고, 성공하면 Authentication 객체를 SecurityContext에 저장합니다. 이후 요청마다 SecurityContext를 기반으로 인증 정보를 사용하며, 필요하면 JWT 인증 필터나 OAuth2 로직으로 구조를 커스터마이징할 수 있습니다.
접근성 높은 웹을 만들기 위해 어떤 점을 고려해야 하나요	접근성 높은 웹을 만들기 위해서는 먼저 모바일에서도 잘 보이도록 반응형 디자인(Media Query) 을 적용해야 합니다. 또한 HTML 구조는 시멘틱 태그를 사용하고, 보조기기 사용자를 위해 ARIA 속성을 올바르게 설정해야 합니다. WCAG 접근성 가이드라인을 준수하고, 키보드만으로도 모든 기능을 사용할 수 있는지, 스크린리더에서 올바르게 읽히는지를 실제로 테스트하는 것이 중요합니다. 마지막으로 디자인 시스템을 적용해 접근성 기준을 일관되게 유지합니다.
프로세스와 스레드의 차이는 무엇인가요	프로세스는 실행 중인 프로그램으로 자체 메모리 공간을 독립적으로 사용합니다. 반면 스레드는 하나의 프로세스 안에서 실행되는 작업 흐름으로, 프로세스의 메모리를 함께 공유합니다. 그래서 스레드는 생성 비용이 적고 빠르지만, 공유 자원 때문에 동시성 문제에 주의해야 하고, 프로세스는 안정적이지만 상대적으로 비용이 큽니다.
서버 포트 충돌 시 어떻게 해결하는가?	서버 포트 충돌이 발생하면 먼저 netstat -tulpn이나 lsof -i :포트번호를 사용해 어떤 프로세스가 해당 포트를 점유하고 있는지 확인합니다. 이후 필요 없는 프로세스라면 종료하거나(kill), 서비스 환경에 따라 포트를 변경해 재실행합니다. Docker나 Spring Boot처럼 포트 설정이 가능한 경우 설정 파일에서 다른 포트로 변경해 문제를 해결할 수도 있습니다.
배포 자동화를 해본 경험이 있나요? 어떤 툴을 사용했나요?	배포 자동화를 완전히 구축한 경험은 아직 많지 않지만, 실제 프로젝트에서는 주로 직접 서버에 접속해 파일을 업로드하고 서비스를 재시작하는 방식으로 배포해왔습니다. 이 과정에서 배포 시점 관리, 에러 로그 확인, 서버 자원 점검 등을 직접 경험할 수 있었습니다. 최근에는 GitHub Actions와 Jenkins를 활용해 CI/CD 파이프라인을 구성하는 방법을 학습하고 있으며, 자동 테스트·자동 배포까지 확장하는 과정을 익히고 있습니다. 앞으로는 실제 환경에서도 자동화된 배포 프로세스를 구축해보고 싶습니다.
장애 발생 시 로그는 어떻게 확인하고 분석하나요?	장애가 발생하면 먼저 애플리케이션 로그를 tail -f로 실시간 확인하고, 에러 시간대 기준으로 grep을 사용해 핵심 예외 메시지를 빠르게 추립니다. 그다음 스택 트레이스를 보고 어떤 계층(Controller, Service, Repository)에서 문제가 발생했는지 흐름을 파악합니다. 애플리케이션 로그만으로 부족하면 WAS(Nginx/Tomcat) 로그나 DB Slow Query 로그도 같이 확인해 원인을 좁혀갑니다. 마지막으로 동일 조건을 로컬에서 재현해 문제를 다시 검증합니다.
SSH 접근이나 인증 관리 방법을 설명해주세요 or 서버 보안 강화하려면 어떤 설정을 해야하나요?	SSH 보안을 강화하기 위해 먼저 비밀번호 인증을 비활성화하고 SSH 키 기반 인증을 사용합니다. 기본 포트(22)를 변경하거나, UFW 같은 방화벽으로 허용된 IP만 접속하도록 제한합니다. 또한 Fail2ban을 적용해 반복 로그인 시도를 차단하고, 루트 계정 직접 로그인을 막아 공격 표면을 줄입니다. 마지막으로 서버 패키지를 주기적으로 업데이트해 보안 취약점을 최소화합니다.
OOP란 무엇인가요?	OOP는 데이터를 객체 단위로 묶어서 관리하는 프로그래밍 방식입니다. 객체마다 역할과 책임을 명확히 나눠 개발할 수 있어서 유지보수에 유리합니다. 핵심 특징은 캡슐화, 상속, 다형성, 추상화이고, 이 원칙을 지키면 코드 재사용성과 확장성이 좋아집니다.
객체 지향 프로그래밍(OOP)에서 AOP가 필요한 이유를 비교하여	OOP는 기능을 객체 단위로 나눠서 유지보수성을 높이지만, 여러 객체에서 반복되는 공통 코드(로깅·트랜잭션·보안)는 중복이 생기기 쉽습니다.

설명해보세요	AOP는 이런 공통 관심사(Cross-cutting)를 분리해서 별도 모듈로 관리하고, 실제 비즈니스 로직에는 핵심 코드만 남기면 전체 코드 구조가 더 깔끔해지고 유지보수가 쉬워집니다.
웹개발에서 MVC 패턴의 전체적인 흐름을 설명해보세요.	MVC 패턴은 요청 처리 과정을 역할별로 분리하는 구조입니다. 사용자가 요청을 보내면 Controller가 가장 먼저 받아 라우팅 역할을 하고, 필요한 비즈니스 로직은 Model에 위임해 데이터 처리가 이루어집니다. 처리가 끝나면 Controller는 결과를 View에 전달해 화면으로 렌더링하고, 이를 사용자에게 반환하는 흐름입니다. 이 구조 덕분에 화면·로직·제어가 분리되어 유지보수성이 좋아집니다.
트리 구조에서 자식의 데이터를 부모에게 전달하려면 어떻게 해야 하나요?	트리 구조에서는 데이터가 위로 자연스럽게 올라가지 않기 때문에, 부모가 콜백 함수를 내려주고 자식이 그 함수를 호출하는 방식으로 값을 전달합니다. 상황에 따라 이벤트 기반 구조를 쓰기도 하지만, 기본적으로는 콜백 호출로 부모에게 데이터를 전달합니다.
웹에서 테이블 형식 데이터를 직접 전달할 수 없을 때, 데이터를 전달하는 방식은 무엇인가요?	웹에서는 테이블 같은 구조 데이터를 그대로 보낼 수 없어서 JSON 형태로 변환해서 전달하는 게 가장 일반적입니다. 상황에 따라 FormData, Query String, GET/POST 바디 같은 방식으로 텍스트 기반으로 넘기고, 백엔드는 이것을 다시 객체나 리스트 형태로 파싱해 사용합니다.
Request, Response, API란 무엇인가요?	Request는 클라이언트가 서버에 보내는 요청이고, Response는 서버가 그 요청에 대한 결과를 돌려주는 응답입니다. API는 이런 Request/Response를 통해 클라이언트와 서버가 어떻게 데이터를 주고받을지 정해놓은 규칙으로 보통 REST API 방식으로 JSON 데이터를 주고받습니다
Redux의 핵심 구성요소인 Action, State, Reducer의 역할을 설명해보세요.	Redux에서는 State가 화면에 필요한 현재 상태를 저장하고, Action은 “상태를 이렇게 바꿔주세요”라고 전달하는 요청 객체입니다. Reducer는 이전 State와 Action을 받아서 새로운 State를 만들어주는 순수 함수로, Redux의 상태 변경 흐름을 담당합니다.
데이터베이스 스키마를 어떻게 구성하고 설계하나요?	데이터베이스 스키마는 테이블 구조와 관계를 설계하는 과정입니다. 먼저 요구사항을 분석해서 필요한 엔티티와 속성을 정리하고, 엔티티 간 관계(1:N, N:M)를 정의해서 테이블로 구체화합니다. 그다음 정규화를 통해 중복을 줄이고 기본키·외래키 같은 제약조건을 설정해서 무결성을 확보합니다. 이렇게 전체 구조를 설계한 것이 스키마가 됩니다.
데이터를 백엔드로 넘길 때 조인(Join)으로 가상 뷰(View)를 만드는 이유는 무엇인가요?	여러 테이블을 조인해서 자주 쓰는 형태로 묶어두면, 백엔드에서 데이터를 가져올 때 한 번의 조회로 필요한 정보를 바로 받을 수 있어서 효율적입니다. 또 복잡한 조인 쿼리를 매번 작성할 필요가 없고, 뷰에서 민감한 컬럼을 제외할 수도 있어서 보안에도 도움이 됩니다. 그래서 조회 중심 API에서는 뷰를 자주 활용합니다.
의존적인 테이블과 독립적인 테이블의 차이를 설명해주세요.	독립 테이블은 다른 테이블을 참조하지 않는 기본 데이터 테이블이고, 회원·상품처럼 기준이 되는 정보를 저장합니다. 반면 의존 테이블은 외래키를 통해 다른 테이블에 의존하는 테이블입니다. 예를 들어 주문 테이블은 회원 ID나 상품 ID를 참조해서 상세 데이터를 구성합니다. 그래서 독립 테이블이 먼저 설계되고, 그 위에 의존 테이블을 연결해 전체 구조를 만듭니다.
주쿼리와 서브쿼리의 차이는 무엇인가요?	주쿼리는 DB에서 실제로 실행되는 메인 쿼리이고, 서브쿼리는 그 안에서 조건이나 계산을 도와주는 보조 쿼리입니다. 서브쿼리는 가독성은 좋지만 느릴 수 있어서, 상황에 따라 JOIN으로 바꾸기도 합니다.
개시와 세션 공통점과 차이점은?	캐시와 세션은 모두 데이터를 저장해 요청을 빠르게 처리하기 위한 기술입니다. 캐시는 클라이언트나 브라우저에 저장되고 속도는 빠르지만 보안성이 낮습니다. 반면 세션은 서버에 저장되고 로그인처럼 사용자의 상태를 유지하는 데 사용되며, 보안성이 높습니다.
Session과 Cookie 사용 이유	HTTP는 요청마다 연결이 끊어지는 비연결형 프로토콜이라, 사용자가 누구인지 기억할 수 없습니다. 그래서 로그인 상태처럼 사용자 정보를 유지하기 위해 클라이언트에는 쿠키, 서버에는 세션을 저장해서 요청이 여러 번 오더라도 같은 사용자라는 걸 확인할 수 있도록 합니다
멀티스레드 환경에서 안전하게 코드를 작성하려면 어떻게 해야 하나요?	멀티스레드에서는 여러 스레드가 동시에 같은 자원에 접근하기 때문에 동시성 문제가 생기지 않도록 공유 자원을 보호해야 합니다. 그래서 보통 synchronized 로 임계 영역을 잠그거나, 필요한 경우 ReentrantLock 같은 Lock을 사용합니다. 또 단순한 연산은 AtomicInteger 같은 Atomic 클래스로 처리해 락 없이도 안전하게 동작하도록 합니다.
INNER JOIN과 LEFT JOIN의 차	INNER JOIN은 두 테이블 모두에서 매칭되는 데이터가 있을 때만 결과를 가져옵니다.

이를 설명해보세요	반면 LEFT JOIN은 왼쪽 테이블을 기준으로 하고, 오른쪽에 매칭되는 데이터가 없어도 NULL 포함해서 모두 반환합니다. 그래서 LEFT JOIN이 결과 범위가 더 넓습니다. 또한 JOIN은 비용이 큰 작업이라 조인 컬럼에 인덱스가 없으면 성능이 떨어질 수 있기 때문에 인덱스 관리가 중요합니다.
인덱스는 왜 필요하고, 언제 사용하는 게 좋나요	인덱스는 테이블에서 원하는 데이터를 전체 스캔하지 않고 빠르게 찾기 위한 자료구조로, 보통 B-Tree 기반으로 만들어집니다. 조회 속도가 크게 빨라지기 때문에 조회가 자주 일어나는 컬럼, 특히 WHERE, JOIN, ORDER BY에 자주 사용되는 컬럼에 인덱스를 두는 것이 좋습니다. 다만 인덱스가 많으면 INSERT, UPDATE, DELETE 시 인덱스도 같이 수정돼야 해서 쓰기 성능이 떨어질 수 있기 때문에 필요한 컬럼에만 적절히 설정하는 것이 중요합니다.
트랜잭션의 ACID 속성을 설명해보세요.	ACID는 트랜잭션의 기본 성질로, 원자성은 전체가 한 번에 처리되는 것, 일관성은 데이터 규칙이 깨지지 않는 것, 격리성은 동시에 실행돼도 서로 간섭하지 않게 하는 것, 지속성은 커밋된 데이터가 장애가 나도 유지되는 걸 말합니다. MySQL은 MVCC로 격리성을 보장합니다
View와 Materialized View의 차이를 설명해보세요	View는 데이터를 저장하지 않는 가상 테이블이라 조회할 때마다 원본 테이블에 쿼리가 다시 실행됩니다. 반대로 Materialized View는 조회 결과를 실제로 저장해두기 때문에 조회는 빠르지만, 갱신해줘야 해서 최신 데이터가 아닐 수 있습니다
서브쿼리와 조인의 차이는 무엇인가요?	서브쿼리는 쿼리 안에서 다른 쿼리 결과를 참조하는 방식이고, 조인은 여러 테이블을 한 번에 합쳐서 조회하는 방식입니다. 서브쿼리는 직관적이지만 성능이 떨어질 수 있고, 조인은 보통 더 최적화돼 있어 대용량에 유리합니다
데이터 정규화란 무엇이며, 언제 비정규화를 고려하는가?	정규화는 중복을 줄이고 데이터 이상 현상을 막기 위해 테이블을 구조화하는 과정입니다. 보통 1·2·3정규형을 적용해 무결성을 높입니다. 하지만 조인이 너무 많아져 성능이 떨어질 때는, 읽기 속도를 높이기 위해 의도적으로 데이터를 중복시키는 비정규화를 고려합니다
쿼리가 느릴 때 어떻게 문제를 찾고 개선하나요?	쿼리가 느리면 먼저 EXPLAIN으로 실행 계획을 확인해서 풀스캔 여부나 인덱스 사용 여부를 봅니다. 그걸 기준으로 필요한 컬럼에 인덱스를 추가하고, 조인이나 조건절을 정리해서 쿼리를 최적화합니다. 상황에 따라 통계 갱신이나 쿼리 구조 조정도 함께 고려합니다.
JPA와 MyBatis의 차이를 설명해보세요	JPA는 객체 중심 ORM이라 SQL을 직접 작성하지 않아도 돼서 생산성과 유지보수가 좋습니다. 반대로 MyBatis는 SQL을 직접 작성하는 방식이라 복잡한 쿼리나 세밀한 제어가 필요할 때 유리합니다. 자동화가 강점이면 JPA, 제어력이 필요하면 MyBatis를 선택합니다.
브라우저가 HTML을 렌더링하는 과정을 설명해보세요	브라우저는 먼저 HTML을 파싱해 DOM 트리를 만들고, CSS를 파싱해 CSSOM을 생성한 뒤 이를 합쳐 Render Tree를 구성합니다. 이후 Layout 단계에서 요소의 위치와 크기를 계산하고, Paint 단계에서 화면에 실제로 그립니다. 이 과정에서 Reflow와 Repaint를 최소화하는 것이 성능에 중요하며, 이미지나 스크립트는 lazy-loading 등을 활용해 렌더링 속도를 최적화할 수 있습니다.
React/Vue의 핵심 개념은 무엇인가요?	React와 Vue의 핵심은 화면을 컴포넌트 단위로 구조화하고, 데이터는 상태(state)와 속성(props) 으로 분리해 관리한다는 점입니다. 상태 변화가 발생하면 가상 DOM(Virtual DOM) 기반으로 필요한 부분만 효율적으로 업데이트하며, 단방향 데이터 흐름(React) 또는 반응형(Reactivity, Vue) 시스템을 통해 UI를 안정적으로 렌더링합니다.
CORS 오류가 왜 발생하나요? 그리고 이를 해결하는 방법을 설명해보세요	CORS 오류는 브라우저의 보안 정책 때문에 다른 도메인으로 요청할 때 서버가 이를 허용하지 않아서 발생합니다. 브라우저는 먼저 preflight(OPTIONS) 요청을 보내 서버가 이 요청을 받아들일지 확인하는데, 이때 서버에서 Access-Control-Allow-Origin 등 필요한 CORS 헤더를 반환하지 않으면 오류가 발생합니다. 해결 방법은 서버에서 허용할 출처(Origin), 메서드, 헤더를 명확히 설정해 브라우저가 요청을 신뢰하도록 해주는 것입니다.
비동기 처리(AJAX, Fetch, Promise, async/await)의 차이	비동기 처리는 페이지를 새로고침하지 않고 서버와 데이터를 주고받기 위한 방식입니다. 초기에는 AJAX로 XMLHTTPRequest를 사용해 비동기 요청을 보냈고, 이후 Promise가 등장해 콜백 지옥을 해결했습니다. async/await은 Promise 기반으로 작성되며 동기 코드처럼 읽혀 가독성이 좋아집니다. 최근에는 Fetch API가 표준으로 사용되어 더 간단한 문법으로 비동기 요청을 처리할 수 있습니다.
브라우저 캐시를 어떻게 제어하나	브라우저 캐시는 Cache-Control, ETag, Last-Modified 같은 HTTP 헤더를 통해 제어할 수

요?	있습니다. Cache-Control로 캐시 기간·정책을 설정하고, ETag나 Last-Modified로 변경 여부를 판단해 불필요한 재다운로드를 줄입니다. 정적 파일은 버전 해싱을 사용해 캐시를 무효화하고, 필요하면 CDN의 Edge 캐싱을 함께 사용해 응답 속도를 최적화합니다.
JAVA의 특징	Java는 JVM 위에서 실행되기 때문에 운영체제에 독립적이라는 점이 가장 큰 특징입니다. 한 번만 작성하면 어떤 OS에서도 실행할 수 있고, 객체지향 언어라 구조화와 유지보수가 쉽다는 장점도 있습니다
자바를 만든 사람은?	“제임스 고슬링”
main 메서드는 왜 static 인가요?	static은 메모리 선언을 사용하지 않아도 사용할 수 있습니다. main메서드는 자바가상머신(JVM)에 의해 호출되는 것이므로 반드시 static으로 선언되어 미리 올라가 있어야 합니다. 만일 메모리에 있지 않다면 시작점인 main메서드를 호출하려고 할 때 메모리에 없기 때문에 실행되지 않습니다.
캡슐화란 무엇인가요?	관련된 데이터와 메서드를 하나의 단위로 묶는 원리입니다. 그로 인해 캡슐내부와 외부를 구별하게 됩니다. 캡슐화를하게 되면 클래스의 필드 값에 권한을 설정할 수 있습니다. 또한 사용자는 데이터가 클래스에 어떻게 저장되는지 알 수 없습니다. 그리고 클래스의 결합도가 낮아져 재사용이 용이하게 됩니다.
직렬화란 무엇인가요?	메모리에 있는 객체를 보조기억장치에 저장할 수 있도록 바이트 형태로 변환하는 것을 말합니다. 객체가 생성되어 데이터가 적재되는 메모리는 순간적이기 때문에 영구적으로 보관하기 위해 직렬화를 사용합니다.
제너릭이란 무엇인가요	클래스를 선언할 때 타입을 결정하지 않고 객체를 생성할 때 유동적인 타입으로 재사용하기 위한 것을 말합니다
리플렉션이란 무엇인가요	리플렉션은 컴파일러를 무시하고 런타임 상황에서 메모리에 올라간 클래스나 메서드등의 정의를 동적으로 찾아서 조작할 수 있는 일련의 행위를 말합니다. 즉 동적인 언어의 특징이라 말 할 수 있습니다. 프레임워크에서 유연성이 있는 동작을 위해 자주 사용하기도 합니다.
객체 와 클래스의 차이점	클래스는 객체를 만들기 위한 설계도이고, 객체는 그 설계도로 실제 메모리에 생성된 실체입니다. 클래스는 정의만 있고, 객체는 실제로 동작하는 상태입니다
객체지향 이란? 그 특징은	객체지향은 프로그램을 객체 단위로 나눠서 설계하는 방식입니다. 현실 세계처럼 ‘속성(필드)’과 ‘동작(메서드)’을 가진 객체들로 기능을 구성해 유지보수성을 높입니다. 주요 특징은 캡슐화, 상속, 다형성이고, 이 덕분에 재사용성과 확장성이 좋아집니다
JDK 란 무엇인가요	JDK는 자바 개발할 때 필요한 전체 도구 세트입니다. 컴파일러나 실행 도구, 표준 라이브러리가 다 포함돼 있어서 자바 코드를 작성하고 실행하는 데 필요한 기본 개발 환경입니다.
프레임워크란 무엇인가요?	프레임워크는 개발할 때 자주 쓰는 구조나 기능을 미리 만들어둔 ‘개발용 틀’입니다. 개발자는 이 틀 위에 비즈니스 로직만 올리면 돼서 개발 속도도 빨라지고, 코드 구조도 일정하게 유지할 수 있습니다.
IoC란 무엇인가요?	IoC는 객체 생성과 의존성 관리 같은 제어권을 개발자가 아닌 프레임워크가 대신 가져가는 개념입니다. Spring에서는 IoC 컨테이너가 필요한 객체를 알아서 만들고 주입해줘서, 개발자는 비즈니스 로직에만 집중할 수 있어요.
DI란 무엇인가요	DI는 객체가 필요한 의존성을 직접 생성하지 않고 외부에서 주입받는 방식입니다. Spring 컨테이너가 객체 생명주기를 대신 관리해서 결합도를 낮추고 테스트하기도 편해집니다. 실제로 저는 생성자 주입을 주로 사용해 순환 참조를 방지하고 구조를 더 명확하게 했습니다
React Hook이란 무엇인가요	React Hook은 함수형 컴포넌트에서도 상태 관리나 생명주기 로직을 사용할 수 있게 하는 기능입니다. useState, useEffect 같은 기본 Hook을 많이 쓰고, 필요하면 커스텀 Hook으로 로직을 재사용하기도 합니다
DML이란 무엇인가요?	DML은 DB에서 데이터를 조회·추가·수정·삭제하는 명령어를 말합니다. SELECT, INSERT, UPDATE, DELETE가 포함되고 실제 서비스 로직에서 가장 자주 쓰는 영역입니다
서브쿼리란 무엇인가요?	서브쿼리는 쿼리 안에서 조건이나 필터링을 도와주는 보조 쿼리입니다. 가독성은 좋지만 성능이 떨어질 수 있어 JOIN으로 대체 가능한지 함께 고려하는 편입니다
JOIN이란 무엇인가요?	OIN은 여러 테이블을 관계 기반으로 묶어서 한 번에 조회하는 방식입니다. INNER JOIN은 공통되는 데이터만, LEFT JOIN은 왼쪽 기준 전체를 가져오는 식으로 선택해 사용합니다. 조인 컬럼에 인덱스가 없으면 성능 저하가 생길 수 있어 인덱스 관리도 중요합니다.