- Java 기반의 애플리케이션 개발을 지원하는 프레임워크 (스프링 vs 스프링부트)
  - 스프링(Spring): **엔터프라이즈급** Java 애플리케이션 개발을 위한 경량 프레임워크로 설정이 유연하고 확장성이 뛰어나지만, 초기 설정이 복잡할 수 있습니다
    - XML 또는 Java Config를 사용하여 수동으로 설정, 설정 파일이 많아 복잡
    - 별도의 외부 서버(Tomcat, Jetty 등)를 설치하고 설정
    - 의존성을 직접 추가하고 관리해야 하며, 충돌이 발생할 가능
  - 스프링부트(Spring Boot): 스프링을 기반으로 한 프레임워크로, 설정을 간소화하고 빠른 애플리케이션 개발을 지원으로 개발 생산성을 높임
    - 대부분의 설정이 자동으로 이루어지며, **application.properties** 또는 application.yml 파일을 통해 간단히 관리
    - Tomcat, Jetty, Undertow와 같은 내장 서버를 제공하여 별도의 서버 설정 없이 애플리케이션을 실행
    - 스타터 의존성(spring-boot-starter-*)을 제공하여 의존성을 쉽게 추가하고 관리
  - 적용 사례
    - 스프링: 대규모 엔터프라이즈 애플리케이션이나 복잡한 비즈니스 로직이 필요한 프로젝트에 적합합니다.
    - 스프링부트: 빠른 프로토타이핑, 마이크로서비스 아키텍처, 클라우드 네이티브 애플리케이션 개발에 적합합니다 (Netflix, 금융서비스, IoT 애플리케이션 등)

- 스프링 부트의 장점
  - 높은 생산성: 코드 작성 시간 단축
  - 설정 간소화: 복잡한 설정을 자동화
  - 빠른 개발 및 배포: 기본 설정과 내장 서버 덕분에 신속한 작업 가능
  - 넓은 커뮤니티와 지원: 풍부한 자료와 활성화된 사용자 그룹
- 스프링 부트의 단점
  - 메모리 사용량 증가: 내장 서버로 인해 메모리 소비가 늘어남
  - 제한된 커스터마이징: 자동 설정이 제한적일 수 있음
  - 초반 학습 곡선: 프레임워크 구조와 철학 이해가 필요
- 스프링부트 현황 및 동향
  - 현재 많은 기업에서 클라우드 네이티브 애플리케이션과 마이크로서비스 개발에 표준으로 사용
    - Netfllix Java 사용 변천사
      >> https://www.integer.blog/evolution-of-java-usage-at-netflix/
  - **스프링 부트 3.x**는 Java 17 지원과 GraalVM 네이티브 이미지 지원을 포함해 최신 기술 트렌드와의 통합성을 지속적으로 강화
  - **스프링 부트 4.x**는 Java 21과 같은 최신 LTS(Long-Term Support) 버전 지원, Spring AI와의 통합 강화, Kubernetes 및 Docker와의 통합이 더욱 간소화되고, 클라우드 환경에서의 확장성과 유연성이 향상, Reactive Programming 개선되어 고성능 비동기 애플리케이션 개발이 용이, DevOps 및 CI/CD 파이프라인과의 통합이 강화되어 개발 및 배포 프로세스가 더욱 간소화
- 부가적인 기술
  - Spring Cloud: 마이크로서비스 아키텍처를 지원하며, 서비스 디스커버리, API 게이트웨이, 분산 트랜잭션 관리 등을 제공
  - Spring Data: 다양한 데이터베이스와의 통합을 지원하며, JPA, MongoDB, Redis와 같은 데이터 저장소를 쉽게 사용
  - Spring Security: OAuth2, JWT 등을 활용한 인증 및 권한 부여 기능을 제공
  - Spring Batch: 대규모 데이터 처리 및 배치 작업을 효율적으로 관리할 수 있는 도구

- 스프링부트 개발 환경 설정
  - 스프링부트 개발 환경 설정은 개발 도구 설치, 프로젝트 생성, 그리고 애플리케이션 실행을 포함
  - 개발 도구
    - Java JDK 설치: 스프링부트는 Java 기반이므로 JDK가 필요
    - IDE: IntelliJ IDEA, Eclipse 등 사용
    - Gradle 또는 Maven: 빌드 관리 도구
    - Git: 버전 관리 툴
    - Docker (선택): 컨테이너 환경 설정
  - 프로젝트 생성 : Spring Initializr 웹사이트나 IDE를 사용하여 새 스프링부트 프로젝트를 생성
    - 종속성 선택: Web, JPA, MySQL 등
    - 프로젝트 유형 선택: Maven 또는 Gradle
    - 패키지 구조 정의
  - 애플리케이션 실행
    - 내장 서버: Spring Boot는 Tomcat 서버를 내장하고 있어 실행 환경 설정이 간단
    - 명령어: ~~`mvn spring-boot:run` 또는 `gradle bootRun`~~
    - 실행 후 확인: 로컬호스트 URL에서 애플리케이션 확인
  - 빌드 및 배포
    - JAR 파일 생성: `mvn package` 또는 `gradle build`를 사용하여 실행 가능한 파일 생성
    - 클라우드 배포: AWS, Azure, GCP와 같은 클라우드 플랫폼을 활용
    - Docker 컨테이너화: `Dockerfile` 작성 후 이미지를 생성하여 배포

- 스프링부트 계층 구조
  - 스프링부트는 일반적으로 MVC(Model-View-Controller) 아키텍처를 기반으로 구성
    - Controller: 사용자 요청을 처리하고 응답을 생성하는 역할을 하며, HTTP 요청을 받아 비즈니스 로직을 호출합니다.
    - Service: 비즈니스 로직을 수행하며, 데이터 처리 및 트랜잭션 관리를 담당
    - Repository: 데이터베이스와 상호작용하며, CRUD(Create, Read, Update, Delete) 작업을 수행
    - Entity: 데이터베이스 테이블과 매핑되는 객체로, 영속적인 데이터를 관리
    - DTO(Data Transfer Object): 계층 간 데이터 전송을 위한 객체로, 데이터를 캡슐화하여 전달
- 스프링부트 애플리케이션과 데이터베이스 간의 상호작용
  - JDBC (Java Database Connectivity)
    - Java에서 데이터베이스와 직접 상호작용하기 위한 API입니다. SQL 쿼리를 작성하고 실행하며, 결과를 처리하는 데 사용
  - MyBatis
    - SQL 매핑 프레임워크로, JDBC의 단점을 해결하기 위해 설계되었습니다. SQL 쿼리와 자바 객체 간의 매핑을 간소화
    - Object와 SQL의 필드를 매핑하여 데이터를 객체화 하는 기술 (객체와 테이블 간의 관계를 매핑하는 것은 아님)
  - JPA
    - **ORM**(Object Relational Mapping) 기술, 대표적인 오픈소스로 Hibernate
    - CRUD 메소드 기본 제공, 쿼리를 만들지 않아도 됨
    - MyBatis는 쿼리가 수정되어 데이터 정보가 바뀌면 그에 사용 되고 있던 DTO와 함께 수정해주어야 하는 반면에, JPA 는 객체만 바꾸면 된다.
    - 즉, 객체 중심으로 개발 가능
    - but 복잡한 쿼리는 해결이 어려움
- 개발 환경 팁
  - Lombok 설정: Getter/Setter 자동 생성으로 개발 편의성 향상 ex) @Data

- MyBatis
  - 변경사항
    - iBatis(~2.3)에서 MyBatis(2.5~)로 변경
    - com.ibatis.*에서 org.apache.ibatis.*로 **패키지 구조 변경**
    - Dynamic Query Tag의 방식 변경
  - OGNL(Object-Graph Navigation Language)의 개념
    - 자바 객체의 속성에 접근하고 값을 설정하거나, 메서드를 호출할 수 있는 표현 언어
    - MyBatis에서 동적 SQL을 작성할 때 if, choose, when, otherwise와 같은 태그 안에서 사용
    - OGNL은 자바 문법을 기반으로 하며, 간단한 조건문부터 복잡한 논리 연산까지 지원
  - OGNL의 역할
    - 동적 SQL 처리: OGNL은 MyBatis의 <if> 태그와 함께 사용되어 조건에 따라 SQL 문을 동적으로 생성
    - 객체 탐색 : OGNL은 객체의 속성을 탐색할 수 있음, user.address.city와 같은 표현식을 통해 중첩된 객체의 속성에 접근
  - OGNL의 주요 표현식
    - 속성 접근: propertyName 또는 object.propertyName
    - 논리 연산 및 비교 연산
    - 메서드 호출: object.methodName()
    - 컬렉션 처리: list.size(), map.keySet()
  - 주의사항
    - OGNL 표현식이 복잡할수록 성능에 영향을 미칠 수 있으므로 간단하게 작성
    - OGNL 표현식에서 오류가 발생하면 디버깅이 어려울 수 있으므로 테스트를 철저히 해야함

- 데이터 전송
  - 웹 애플리케이션에서 MFE(Micro Frontend)와 MSA(Microservices Architecture)는 각각 프론트엔드와 백엔드의 모듈화를 통해 확장성과 유지보수성을 높이는 아키텍처
  - MFE (**Micro Frontend**) : 프론트엔드 애플리케이션을 여러 독립적인 마이크로 애플리케이션으로 나누어 개발 및 배포하는 방식, 아래와 같은 데이터 전송 방식
    - 브라우저 기반 통신 (각 MFE간 데이터를 공유하기 위해 브라우저의 localStorage, sessionStorage 또는 cookies를 사용)
    - 이벤트 기반 통신 (CustomEvent를 활용하여 MFE간 이벤트를 통해 데이터 전달)
    - API 호출 (**백엔드 API를 통해** 데이터를 가져오거나 parameters전송)
    - 프레임워크 사용 (Webpack Module Federation과 같은 도구를 사용하여 모듈 간 데이터 공유)
  - MSA (**Microservices Architecture**) : 애플리케이션을 여러 독립적인 서비스로 나누어 개발 및 배포하는 아키텍처, 각 서비스는 독립적인 데이터베이스와 비즈니스 로직을 가진다
    - 동기식 통신: REST API(HTTP를 사용하여 JSON 또는 XML 형식으로), gRPC (HTTP/2 기반의 고성능 원격 프로시저 호출로, Protocol Buffers를 사용), **GraphQL** (클라이언트가 필요한 데이터만 요청할 수 있는 효율적인 쿼리 언어)
    - 비동기식 통신: 메시지 큐(RabbitMQ, Kafka와 같은 메시지 브로커를 사용하여 서비스 간 데이터전달), 이벤트 기반 통신 (이벤트를 Publish하고 Subscribe하는 방식)

- Docker + MSA + Spring boot
  - 도커(Docker), MSA(Microservices Architecture), 그리고 스프링부트(Spring Boot)는 현대 애플리케이션 개발과 배포에서 필수적인 기술
  - 개념 및 특징 : 컨테이너 기술을 기반으로 애플리케이션 및 그 종속성을 격리된 환경에서 실행할 수 있도록 지원하는 플랫폼
    - 도커 이미지와 컨테이너를 사용하여 개발 환경과 실행 환경 간의 일관성을 유지
    - 컨테이너화: 애플리케이션, 라이브러리, 설정 등을 단일 패키지(이미지)로 배포
    - 환경 일관성: 개발, 테스트, 배포 환경이 동일한 컨테이너에서 실행
    - 오케스트레이션: Kubernetes와 같은 도구를 활용하여 수많은 컨테이너를 효율적으로 관리

- 도커 & MSA & 스프링부트의 통합
  : 이 세 가지 기술은 함께 사용될 때 강력한 시너지를 발휘
  - 도커
    - 각 마이크로서비스를 컨테이너로 실행하여 배포 환경을 표준화
    - 스프링부트 애플리케이션을 도커 컨테이너로 배포하여 독립적 실행 가능
  - MSA
    - 애플리케이션을 여러 마이크로서비스로 분리하여 각 서비스를 개별적으로 관리
    - 도커와 함께 사용하면 서비스의 확장성과 가용성을 극대화
  - 스프링부트
    - 각 마이크로서비스의 구현을 간소화하고 빠른 개발을 지원
    - 도커를 통해 스프링부트 기반 서비스의 배포를 효율적으로 수행