바이브 코딩 시대의 진로 이야기

MVC는 옛것인가? 백엔드 진로와 트렌드

TypeScript + Go, 그리고 Java Spring — 무엇이 바뀌었고 무엇을 배워야 하나

MVC | Spring Boot 4 | Java 25 | Kotlin | TypeScript | Go | 바이브 코딩

AI 애플리케이션 개발 과정 · 보충 강의

수강생의 질문

“어느 쪽을 배워야 진로에 좋을까요?”

선택지 A

TypeScript + Go

프런트엔드는 TypeScript(React · Next.js), 백엔드는 Go

vs

선택지 B

Java Spring

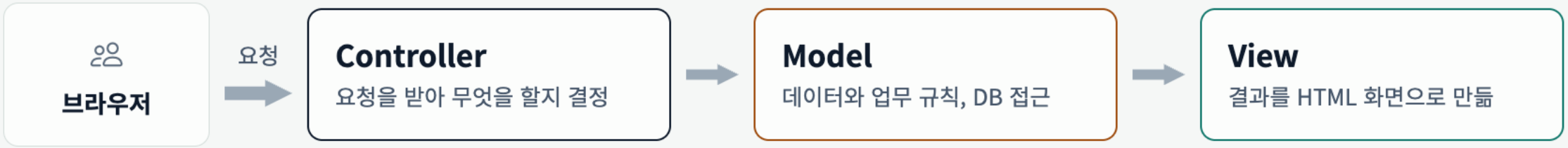
Spring Boot로 백엔드, 화면은 별도 프런트엔드 또는 서버 렌더링

오늘의 결론 미리 보기

옛것이 된 것은 **“서버가 화면까지 그리는 MVC 방식”**이지 Spring이 아니다. 둘 다 현역이며, 선택은 **가고 싶은 회사**가 정한다.

01 · MVC란

MVC: 역할을 셋으로 나누는 설계 패턴



기원

1979년 Xerox PARC의 Smalltalk에서 데스크톱 GUI를 위해 고안된 패턴

핵심 아이디어

관심사의 분리 — 화면을 바꿔도 업무 규칙은 그대로

지금도 살아 있다

Rails · Django · Laravel · ASP.NET Core · Spring 모두 이 구조를 쓴다

01 · 무엇이 옛것이 되었나

옛것이 된 것은 “서버가 화면을 그리는 방식”

2000년대

서버 렌더링 MVC

Struts · Spring MVC + JSP. View가 서버에서 HTML을 만들어 보낸다. 클릭마다 페이지 전체 새로고침

← 흔히 말하는 “예전 방식”

2010년대

API + SPA 분리

서버는 JSON API만, 화면은 React · Vue가 브라우저에서. Controller는 남고 View가 프런트엔드로 이사

← 지금의 주류 구조

2020년대

서버 렌더링의 귀환

Next.js 서버 컴포넌트, htmx. 속도 · SEO를 위해 서버가 다시 화면 일부를 그린다 — 단, 컴포넌트 단위로

← 유행은 돌고 돈다

기술은 사라지기보다 **역할이 옮겨 간다**. 지난 강의의 “프런트엔드 / 백엔드 분리”가 바로 2010년대의 변화다.

02 · Spring의 진화

Spring은 그 변화를 따라 API 서버가 되었다

2008년경 · Spring MVC + JSP

```
@Controller
public class OrderController {
  @RequestMapping("/orders")
  public String list(Model model) {
    model.addAttribute("orders",
      service.findAll());
    return "orders"; // → orders.jsp
  }
}
```

XML 설정, WAR 파일을 Tomcat에 배포, 결과는 **HTML 화면**

2026년 · Spring Boot 4 REST API

```
@RestController
@RequestMapping("/api/orders")
class OrderController(
  private val service: OrderService) {

  @GetMapping
  fun list(): List<OrderDto> =
    service.findAll() // → JSON
}
```

설정 자동화, 실행 가능한 JAR · 컨테이너, 결과는 **JSON 데이터** (예시는 Kotlin)

⚠ **이름의 함정:** 모듈 이름은 여전히 “Spring Web MVC”지만, 오늘날 대부분은 화면이 아니라 **API를 만드는 데** 쓴다.

02 · 지금의 Spring

2025–26년 Spring의 최신 모습

Framework 7 · Boot 4

2025년 11월 정식 출시. Java 17 이상, Java 25 LTS 대응, Jakarta EE 11

가상 스레드

Java 21부터. Go의 고루틴처럼 가벼운 스레드로 대량 동시 요청 처리

안정성 기본 탑재

JSpecify 널 안정성, API 버전 관리, @Retryable 재시도가 프레임워크 기본 기능으로

Kotlin 일급 지원

Kotlin 2.2 지원. 국내 스타트업은 Kotlin + Spring 조합이 흔하다

클라우드 네이티브

컨테이너 이미지, GraalVM 네이티브 빌드, Actuator · Micrometer 관측

Spring AI

LLM 호출 · RAG · 도구 호출을 Spring 방식으로. 기존 Java 시스템에 AI를 붙이는 통로

02 · 진짜 바뀐 것

언어보다 크게 바뀐 것은 아키텍처다

구분	예전 (2000년대 MVC 시대)	지금 (2020년대)
화면	서버가 HTML 생성 (JSP)	프런트엔드 분리 + API, 필요하면 서버 컴포넌트
구조	거대한 단일 애플리케이션	모듈형 모놀리스 또는 마이크로서비스
배포	WAR를 사내 서버에 수동 배포	컨테이너 + CI/CD + 쿠버네티스
통신	동기 호출 중심	REST · gRPC + 메시지 큐 · 이벤트
운영	로그 파일 확인	메트릭 · 트레이스 · 대시보드 (관측 가능성)
AI	없음	LLM · RAG가 백엔드의 한 부품, AI가 코드 작성 보조

Spring도 Go도 **이 오른쪽 열 위에서** 쓰인다 — 배워야 할 것은 이 구조다.

03 · 선택지 A

TypeScript + Go 스택은 어떤 모습인가

TypeScript · 프런트엔드

- React · Next.js가 사실상 표준
- JavaScript에 타입을 더해 큰 코드도 안전하게
- 웹 화면이 있는 회사라면 거의 필수
- Node · NestJS로 백엔드까지 한 언어도 가능

Go · 백엔드

- 문법이 작고 단순, 단일 실행 파일 배포
- 고루틴으로 높은 동시성, 적은 메모리
- 클라우드 인프라의 언어: Docker · Kubernetes · Prometheus · Terraform
- API 서버, 게이트웨이, 수집기 · 워커에 강함

```
// Go 표준 라이브러리만으로 만든 API (Go 1.22+ 라우팅)
http.HandleFunc("GET /api/orders", func(w http.ResponseWriter, r *http.Request) {
    json.NewEncoder(w).Encode(service.FindAll())
})
```

03 · 비교

Go 백엔드 vs Java · Kotlin Spring

항목	Go	Java · Kotlin + Spring Boot
철학	작고 단순, 필요한 것만 직접 조립	기능이 풍부, 프레임워크가 많은 것을 대신
학습 곡선	입문은 쉬움, 설계는 스스로	입문이 무거움(DI · JPA · 설정), 익히면 생산적
동시성	고루틴 · 채널	가상 스레드(Java 21+), Kotlin 코루틴
강한 분야	클라우드 인프라, 게이트웨이, 고성능 API	금융 · 커머스 · 대규모 업무 시스템
국내 채용 규모	성장 중, 아직 소수	가장 많음 (백엔드 기본값)
신입 경쟁	공고 적음, 지원자도 적음	공고 많음, 지원자도 많음
AI 코드 생성	코드가 짧아 검토하기 쉬움	반복 코드를 AI가 많이 덜어 줌

03 · 국내 시장

국내 백엔드는 지금 어떻게 쓰고 있나

Java · Spring	국내 백엔드의 기본값. 대규모 운영 경험과 인력 풀이 가장 두텁고, 교체보다 점진 개선이 일반적
Kotlin	Java와 호환성이 높아 JVM 위에서 점진적으로 도입. Spring과 함께 쓴다
Go	컨테이너 · 쿠버네티스 기반 인프라에서 성장 — API 서버, 메시지 워커, 로그 · 메트릭 수집기
Node · TS (NestJS)	프런트엔드 중심 팀이 한 언어로 통일할 때 선택
Python · FastAPI	AI 모델 서빙 · 데이터 파이프라인의 표준. 업무 로직은 Java · Kotlin과 나눠 맡는 경우가 많다

출처: 요즘IT “2026년 백엔드 개발자에게 찾아올 변화와 선택” (2025.12) — 필자의 업계 관찰 기반

03 · 앞으로의 전망

향후 3–5년, 무엇이 남고 무엇이 커질까

Java · Spring: 크고 느리게 변하는 기반

점유율은 장기적으로 완만히 줄지만 이미 운영 중인 시스템이 거대하다. 유지 · 현대화 수요는 오래 간다

Go: 클라우드와 함께 성장

쿠버네티스 · 플랫폼 엔지니어링이 커질수록 수요 증가. 인프라 지향 진로와 궁합이 좋다

TypeScript: 화면이 있는 곳 어디나

프런트엔드 표준 지위는 당분간 유지. 풀스택 · 1인 개발에서 특히 유리

가장 확실한 흐름: 여러 언어 + AI

한 회사가 여러 언어를 섞어 쓰고, AI가 코드 작성을 맡는 비중이 커진다 — 언어 하나에 올인할 이유가 줄어든다

전망은 현재 흐름에 근거한 판단이며 확정된 예측이 아니다.

04 · 바이브 코딩

코드를 쓰는 비용이 내려가면, 가치는 어디로 가나

AI가 빠르게 해 주는 일 ↓ 가치 하락

- 문법 · 보일러플레이트 작성
- CRUD API, DTO, 설정 파일
- 언어 간 코드 변환
- 프레임워크 사용법 암기

사람이 책임지는 일 ↑ 가치 상승

- 무엇을 만들지 · 구조 설계
- AI 코드 검토 · 테스트 · 보안
- 장애 대응 · 운영 · 관측
- 업무 도메인 이해

그래서 질문이 바뀐다: "어떤 언어를 쓸 줄 아나?" → **"AI가 만든 시스템을 이해하고 책임질 수 있나?"**

바이브 코딩 시대 백엔드 개발자의 역량 4층



언어는 ①을 배우는 도구

Spring으로 배우든 Go로 배우든 ①②는 그대로 옮겨 간다

아래가 비면 위가 무너진다

원리 없이 ④만 있으면 AI 코드가 맞는지 판단할 수 없다

면접도 이쪽으로

“왜 이렇게 설계했나”, “AI 코드의 문제를 어떻게 찾았나”를 묻는다

언어가 아니라 가고 싶은 곳에서 거꾸로 고른다

대기업 · 금융 · 커머스 · SI · 공공	**Java · Kotlin + Spring Boot**
클라우드 · 인프라 · 플랫폼 팀	**Go + 쿠버네티스 · 리눅스**
스타트업 풀스택 · 1인 창업	**TypeScript 풀스택 (Next.js) + Go 또는 Python**
AI 서비스 · 데이터	**Python(FastAPI) + Spring 또는 Go 중 하나**

숙제: 가고 싶은 회사 5곳의 백엔드 채용 공고를 열어 “필수 기술” 칸을 직접 세어 보자. 그것이 가장 정확한 통계다.

05 · 학습 로드맵

둘 중 하나가 아니라, 순서의 문제

1단계

주력 하나를 깊게

국내 취업이 목표면 Spring Boot(Java 또는 Kotlin)를 추천. HTTP · JPA · 트랜잭션 원리까지

2단계

TypeScript 기초

React로 내 API를 쓰는 화면 하나. 프런트엔드와 대화할 수 있을 만큼

3단계

운영까지 완주

Docker Compose · CI/CD · Grafana. 배포하고 관측한 프로젝트 하나

4단계

두 번째 언어 Go

같은 API를 Go로 다시. AI와 함께라면 몇 주면 충분하다

바이브 코딩 학습법 — AI를 "대신 짜는 도구"가 아니라 "과외 선생"으로

"이 Spring 코드를 Go로 바꾸고, 두 코드에서 트랜잭션과 동시성 처리가 어떻게 다른지 설명해 줘. 설명을 들은 뒤 내가 직접 테스트를 쓸게."

정리

1. 옛것이 된 것은 **서버가 화면을 그리던 방식**이다. MVC의 아이디어와 Spring은 진화해서 현역이다.
2. TS + Go와 Spring은 **우열이 아니라 가는 곳이 다르다**. 목표 회사에서 거꾸로 고른다.
3. 바이브 코딩 시대의 경쟁력은 언어가 아니라 **원리 · 설계 · 운영 · 검증**이다.