6부.세부사항

30장. 데이터베이스는 세부사항이다.

소프트웨어 시스템의 아키텍처와 데이터베이스의 관계를 건물로 비교하면 건물의 아키텍처와 문 손잡이의 관계와 같다.

> 결국 집 밖의 사정일 뿐이다.

뛰어난 아키텍트라면 저수준의 메커니즘이 시스템 아키텍처를 오염시키는 일을 용납하지 않는다.

관계형 데이터베이스

데이터 접근 프레임워크가 테이블과 행이 객체 형태로 시스템 아키텍처에서 돌아다니게 허용하는데, 아키텍처적으로 잘못된 설계이다.

데이터베이스 시스템은 왜 이렇게 널리 사용되는가?

데이터베이스 시스템이 소프트웨어 시스템과 소프트웨어의 기업을 장악할 수 있었을까? 한마디로 답하자면 바로 디스크 때문이다.

안정적으로 데이터를 영구 보관할 수 있으면서도 메모리를 거쳐 효율적으로 읽어드리기 때문이다. 디스크 때문에 피해 갈 수 없는 시간 지연이라는 점을 완화하기 위해, 색인, 캐시, 쿼리 계획 최적화가 필요해졌다. 이러한 색인, 캐시, 쿼리 계획 최적화를 하려면 정확히 어떤 데이터인지를 판별할 수 있어야 했고 이를 위해 데이터 접근 및 관리 시스템이 필요했다.

세부사항

데이터베이스는 그저 메커니즘에 불과하며, 디스크 표면과 RAM 사이에서 데이터를 이리저리 옮길 때 사용할 뿐이다.

하지만 성능은?

성능은 완전히 캡슐화하여 업무 규칙과는 분리할 수 있는 관심사이다.

개인적인 일화

데이터 저장소 선별에 있어 저자 분은 파일 시스템 기반의 관리를 주장했고, 마케팅 담당자는 RDBMS를 주장했다. 하지만 결과적으로 마케팅 담당자의 RDBMS가 승리했다. 현재는 기본적으로 영속성이 필요한 데이터는 RDBMS에 저장하는 것이 일반적이고 대다수가 이러한 선택을 할 것이다. 하지만 저자는 이러한 선입견에 얽매이지 않고 오버 엔지니어링 없이 현재의 환경에 최적화된 데이터 저장소인 파일 시스템을 선택한 것이다. 다만 간과했던 부분은 선택에 있어 가장 높은 영향력이 있는 고객의 니즈가 RDBMS 였다는 점이다. 이 일화를 통해 무조건적인 유행을 따르는 것에 대한 위험성과 때론 논리적으로 정답일지라도 넓은 관점에서는 정답이 아닐 수도 있다는 점이다.

31장.웹은 세부사항이다

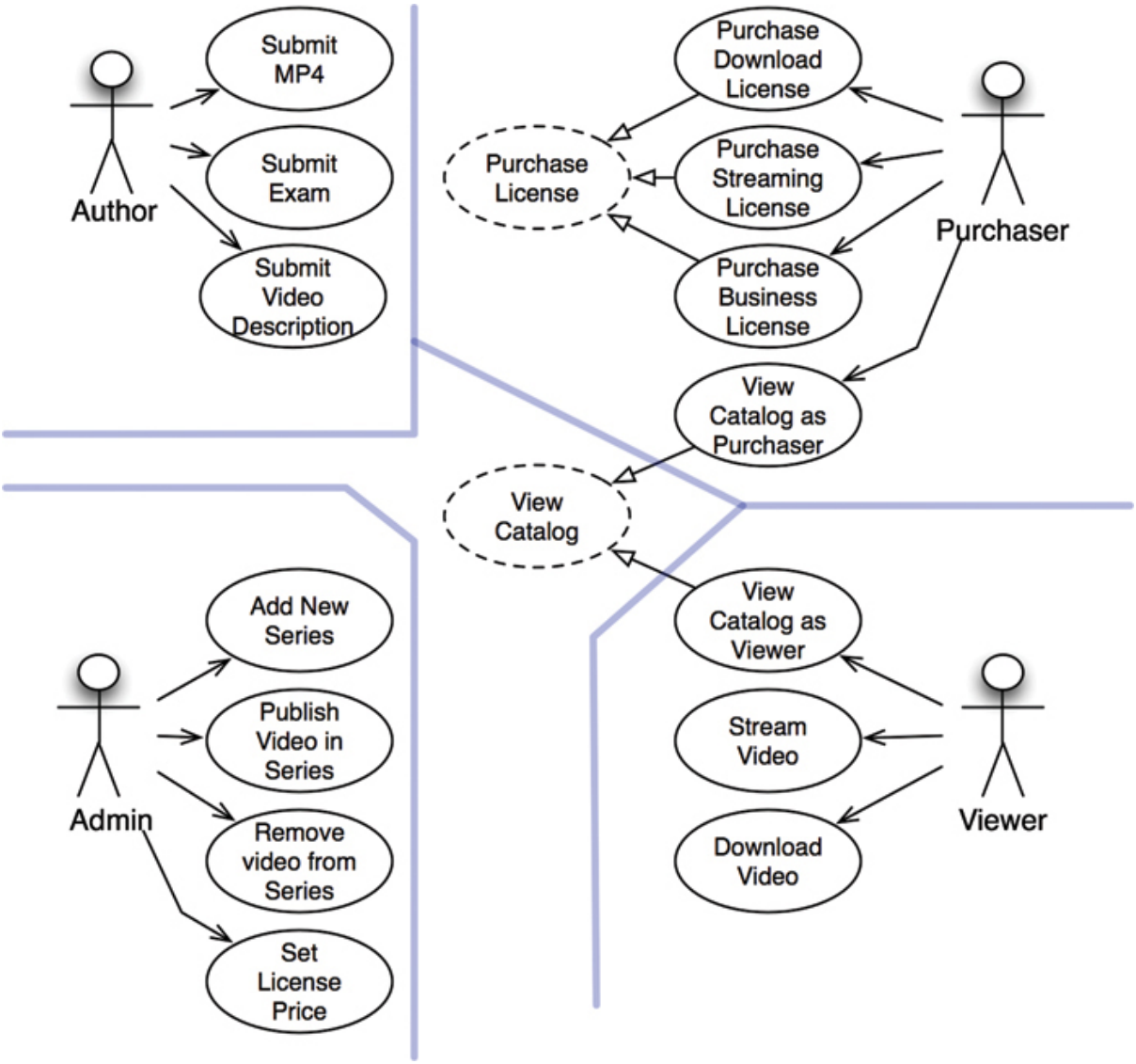
웹 == GUI == 세부사항 == 웹

32장.프레임워크는 세부사항

- 프레임워크가 있으면 애플리케이션 개발에 도움이 된다.
- 다만, 프레임워크의 발전 방향이나 기능 추가가 더뎌 향후 다른 프레임워크로 변경이 필요할 수 있다.
- 프레임워크를 이용하되 적절한 거리두기를 해서 너무 강결합이 되어서는 안된다.

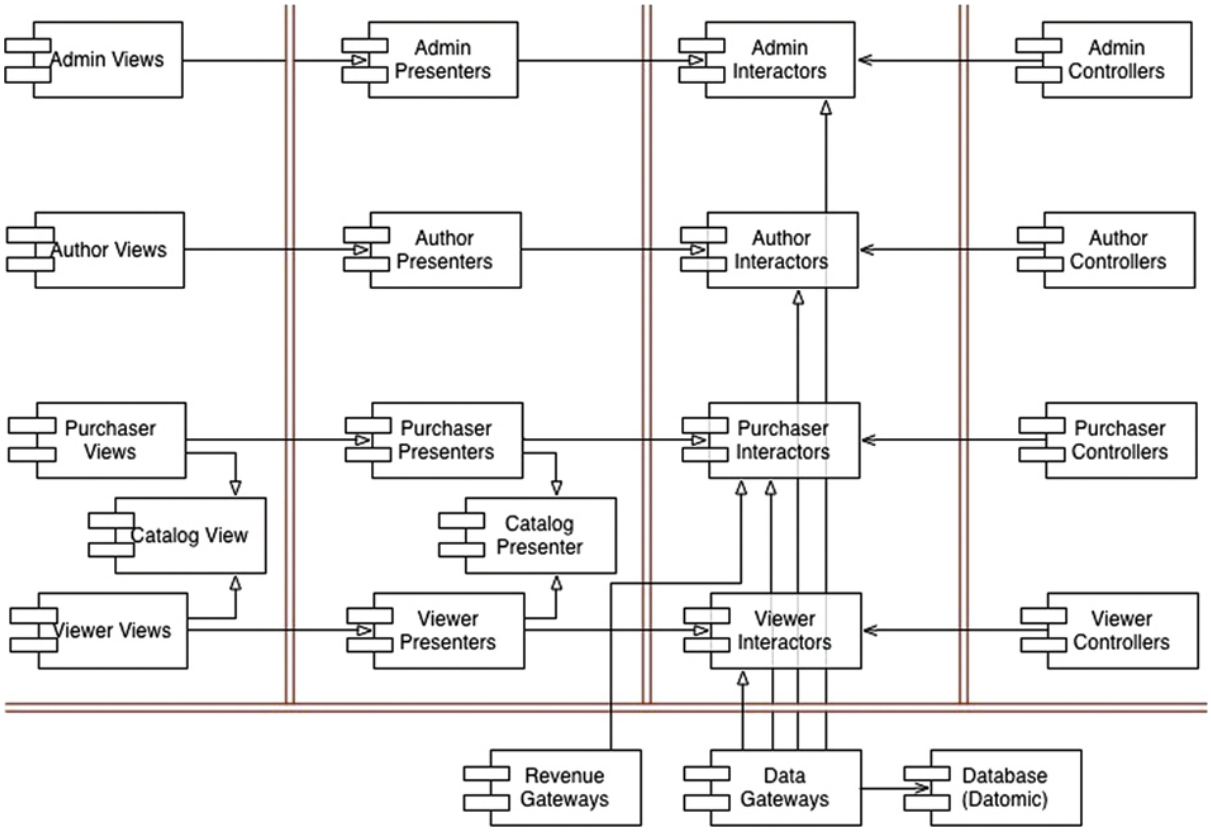
33장.사례 연구:비디오 판매

아래 그림은 비디오 판매를 주제로 유스케이스 다이어그램을 그린 사진이다.



위 유스케이스는 일부 유스케이스를 점선으로 표현했는데 이를 추상 유스케이스라 부르며 다른 유스케이스에서 이를 구체화한다. 보다시피 시청자 입장에서 카탈로그 조회하기와 구매자 입장에서 카탈로그 조회하기를 상속받는다.

하지만, 이 추상 유스케이스를 꼭 생성할 필요는 없었는데 이 추상 유스케이스를 없애더라도 관련 구현체들은 영향을 받지 않는다. 하지만 이들 두 유스케이스는 너무 비슷하기 때문에, 유사성을 식별해서 분석 초기에 통합하는 방법을 찾는 편이 더 현명하다고 판단했다.



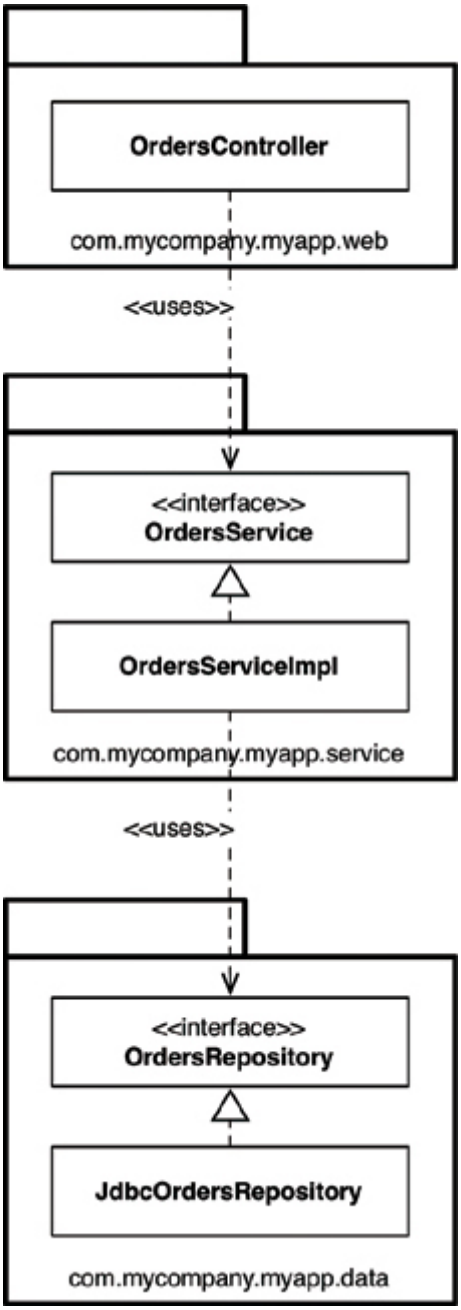
위 사진은 이전 사진의 유스케이스를 기반으로 컴포넌트 아키텍처를 그린 것인데 뷰, 프레젠터, 인터랙터, 컨트롤러, 데이터 게이트웨이로 나눠져있다. 5개의 jar로 배포하는 것이 가능하며 상황에 따라 뷰 하나 프레젠터, 인터렉터 컨트롤러 묶어서 하나, 데이터 게이트웨이로 하나 총 세개로 묶어서 배포하는 것도 가능하다. 배포 방식 조정이 가능한 것이다.

위 사진에서 열린 화살표(->)는 사용 관계이며 닫힌 화살표(-▷)는 상속 관계이다. 개방 폐쇄 원칙을 준수하는 좋은 컴포넌트 아키텍처인 샘이다.

34장.빠져 있는 장

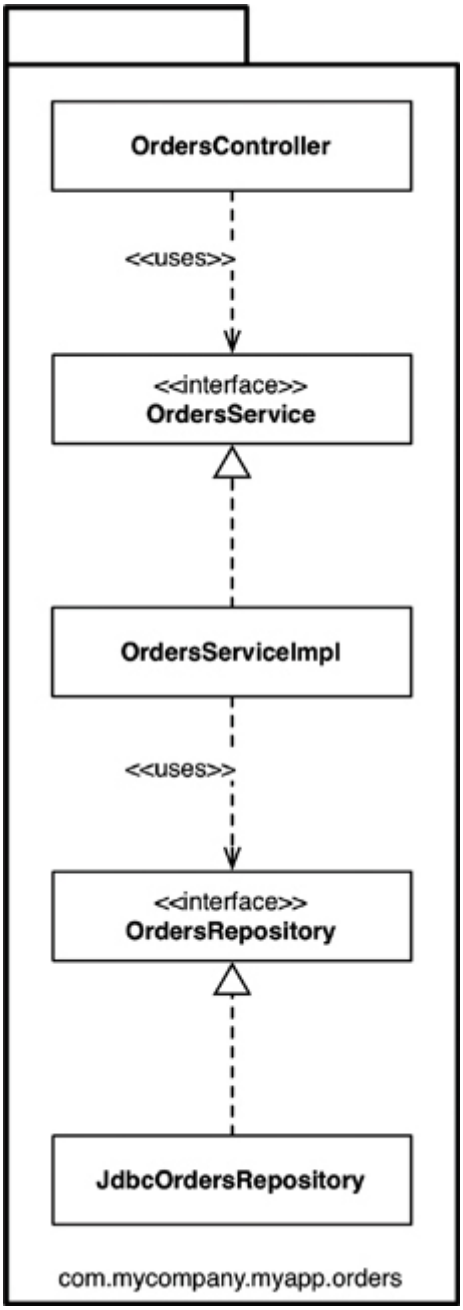
설계, 코드 조직화와 관련된 몇 가지 접근법을 알아보자

계층 기반 패키지



컨트롤러, 서비스, 리포지토리에 모두 담는 방식이다. 초기에 애플리케이션이 단순할 때는 문제가 없으나 복잡해지기 시작하면 세 개에 담아내기가 힘들어짐을 알 수 있다.

기능 기반 패키지



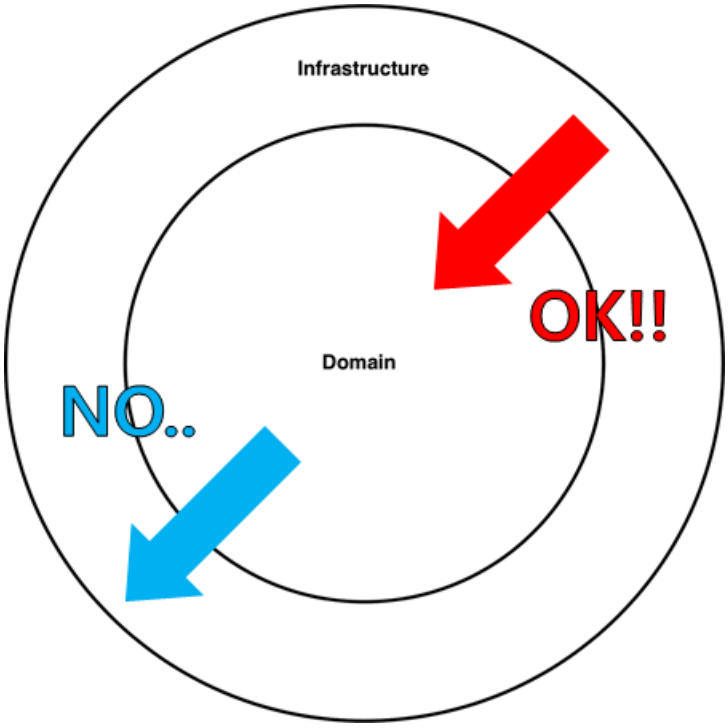
이는 서로 연관된 기능, 도메인 개념 또는 어그리거트 루트에 기반하여 수직의 얇은 조각으로 코드를 나누는 방식이다.

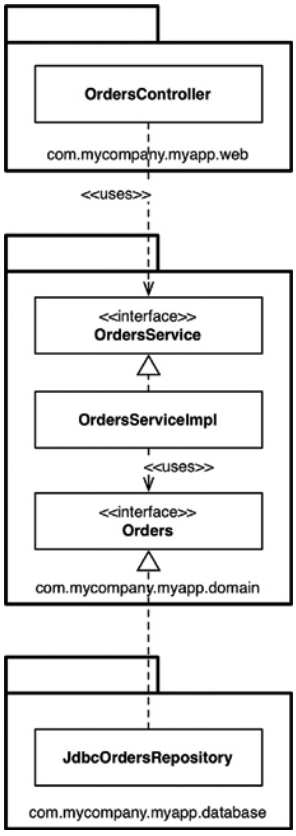
단점은 코드의 저수준이 고수준에 영향을 주게 된다.

포트와 어댑터

엉클 밥에 따르면 포트와 어댑터 혹은 헥사고날 아키텍처 등의 방식으로 접근하는 이유는 업무/도메인에 초점을 둔 코드가 프레임워크나 데이터베이스와 같은 기술적인 세부 구현과 독

립적이며 분리된 아키텍처를 만들기 위해서다. 그런 코드베이스는 주로 내부(도메인)과 외부(인프라)로 구성됨을 알 수 있다.





외부에는 UI, 데이터베이스, 서드파티, 대외계 등 많은 것이 폴함된다. 이 방식으로 유스케이스를 구현하면 오른쪽 사진과 같아진다.

내부에 존재하는 모든 것의 이름은 반드시 유비쿼터스 도메인 관점에서 기술하라고 조언한다. 바꿔 말하면 도메인에 대해 논의할 때 우리는 주문에 대해 말하는 것이지 주문 리포지토리에 대해 말하는 것이 아니다.

컴포넌트 기반 패키지

SOLID, REP, CCP, CRP 그리고 이 책에 나온 대다수의 조언에 대해서 나는 전적으로 동감하지만, 코드를 조직화하는 방법에 대해서는 다소 다른 결론에 이르렀다. 그래서 또 다른 선택지를 제시하려고 하는데, 나는 이 방법을 **컴포넌트 기반 패키지**라고 부른다.

컴포넌트 기반 패키지는 큰 단위의 단일 컴포넌트와 관련된 모든 책임을 하나의 자바 패키지로 묶는데 주안점을 둔다. 이 접근법은 서비스 중심적인 시각으로 소프트웨어 시스템을 바라보며 MSA가 가진 시각과도 동일하다. 사용자 인터페이스를 큰 단위의 컴포넌트로부터 분리해서 유지한다.

본질적으로 이 접근법에서는 업무 로직과 영속성 관련 코드를 하나로 묶는데 이 묶음을 나는 컴포넌트라고 부른다. 엉클 밥은 이 책 앞부분에서 컴포넌트에 대한 정의를 아래와 같이 정의했다.

> *컴포넌트는 배포 단위다. 컴포넌트는 시스템의 구성 요소로, 배포할 수 있는 가장 작은 단위다. 자바의 경우 JAR 파일이 컴포넌트다.*

저자의 정의는 다음과 같다.

> *컴포넌트는 멋지고 깔끔한 인터페이스로 감싸진 연관된 기능들의 묶음으로, 애플리케이션과 같은 실행 환경 내부에 존재한다.*

컴포넌트 기반 패키지 접근법의 주된 이점은 주문과 관련된 무언가를 코딩해야할 때 오직 한 곳, 즉 Orders Component만 보면 된다는 점이다. 이 컴포넌트 내부에서 관심사의 분리는 여전히 유효하며, 따라서 업무 로직은 데이터 영속성과 분리되어 있다. 하지만 이는 컴포넌트 구현과 관련된 세부 사항으로, 사용자는 알 필요가 없다. 이는 MSA를 적용 했을 때 얻는 이점과 같다.

구현 세부 사항에는 항상 문제가 있다