제5장

건축적 특징 파악하기

아키텍처를 설계하거나 기존 아키텍처의 타당성을 검증하기 위해 아키텍트는 아키텍처 특성과 도메인이라는 두 가지를 분석해야 합니다. 4 장 에서 배웠듯이 , 특정 문제나 애플리케이션에 적합한 아키텍처 특성("-ilities")을 파악하려면 도메인 문제를 이해하는 것뿐만 아니라 이해관계자들과 협력하여 도메인 관점에서 진정으로 중요한 것이 무엇인지 결정해야 합니다.

프로젝트에 필요한 아키텍처적 특성을 파악하는 데는 최소한 세 가지 요소가 있습니다. 바로 도메인 관련 문제, 프로젝트 요구사항, 그리고 암묵적인 도메인 지식입니다. 보안 및 모듈성 같은 일반적인 암묵적 아키텍처적 특성 외에도, 일부 도메인에는 암묵적인 아키텍처적 특성이 내재되어 있는 경우가 있습니다. 예를 들어, 진단 장비에서 데이터를 읽어오는 의료 소프트웨어를 개발하는 아키텍트는 데이터 무결성의 중요성과 메시지 손실로 인한 잠재적 결과를 이미 잘 알고 있어야 합니다. 이러한 도메인에서 일하는 아키텍트는 데이터 무결성을 내재화하고 있기 때문에, 설계하는 모든 솔루션에 데이터 무결성이 암묵적으로 반영됩니다.

도메인 관심사로부터 아키텍처적 특징 추출하기

대부분의 아키텍처 특성은 주요 도메인 이해관계자들의 의견을 경청하고 그들과 협력하여 비즈니스 관점에서 무엇이 중요한지 파악하는 과정에서 도출됩니다. 언뜻 보기에는 간단해 보이지만, 문제는 아키텍트와 도메인 이해관계자들이 서로 다른 언어를 사용한다는 점입니다. 아키텍트는 확장성, 상호 운용성, 내결함성, 학습 용이성, 가용성에 대해 이야기하는 반면, 도메인 이해관계자는 인수합병, 사용자 만족도, 출시 기간, 경쟁 우위에 대해 이야기합니다. 이로 인해 아키텍트와 도메인 이해관계자 간의 "번역 오류" 문제가 발생합니다.

이해관계자들이 서로를 이해하지 못한다. 아키텍트는 사용자 만족도를 지원하는 아키텍처를 어떻게 설계해야 할지 모르고, 도메인 이해관계자들은 아키텍트가 애플리케이션의 가용성, 상호 운용성, 학습 용이성, 내결함성에 왜 그렇게 집중하는지 이해하지 못한다.

다행히 도메인 관련 사항을 아키텍처 특성의 언어로 "변환"하는 것이 가능합니다. 표 5-1은 몇 가지 일반적인 도메인 관련 사항과 이를 뒷받침하는 "-ilities"(특성)를 보여줍니다. 핵심 도메인 목표를 이해하면 아키텍트는 해당 도메인 관련 사항을 "-ilities"(특성)로 변환할 수 있으며, 이는 올바르고 타당한 아키텍처 결정을 내리는 기반이 됩니다. 예를 들어, 출시 시간 단축이라는 도메인 관련 사항이 확장성보다 더 중요한가요, 아니면 아키텍트는 내결함성, 보안 또는 성능에 집중해야 할까요? 어쩌면 시스템에는 이러한 모든 특성이 복합적으로 필요할 수도 있습니다.

표 5-1. 도메인 관련 사항을 아키텍처 특성으로 변환

도메인 관심사	건축적 특징
인수합병	상호운용성, 확장성, 적응성, 확장성
시장 출시 기간	민첩성, 테스트 용이성, 배포 용이성
사용자 만족도	성능, 가용성, 내결함성, 테스트 용이성, 배포 용이성, 민첩성, 보안
경쟁 우위	민첩성, 테스트 용이성, 배포 용이성, 확장성, 가용성, 내결함성
시간 및 예산	단순성, 실현 가능성

복합적인 건축적 특징

도메인 관련 사항을 아키텍처 특성으로 변환할 때 흔히 저지르는 실수 중 하나는 애자일성을 단순히 출시 기간과 동일시하는 것과 같이 잘못된 등가 관계를 형성하는 것입니다.
애자일은 복합적인 아키텍처 특성의 한 예입니다. 즉, 단일한 객관적 정의가 있는 것이 아니라 측정 가능한 여러 요소로 구성됩니다. 애자일 자체를 측정할 수 있는 기준은 없으므로, 아키텍트는 "애자일은 무엇으로 구성되는가?"라는 질문을 던져야 합니다. 애자일에는 배포 용이성, 모듈성, 테스트 용이성 등 측정 가능한 요소들이 포함됩니다.

흔히 발생하는 안티패턴은 아키텍트가 편의상 복합적인 요소의 한 부분에만 집중하는 경우입니다. 이는 마치 케이크 반죽에 밀가루를 넣는 것을 잊는 것과 같습니다. 예를 들어, 해당 분야의 이해관계자가 "규제 요건 때문에 일일 펀드 가격 책정을 제시간에 완료하는 것이 절대적으로 필수적입니다."라고 말할 수 있습니다.

비효율적인 아키텍트는 해당 도메인의 주요 관심사가 성능인 것처럼 보이기 때문에 성능에만 집중하는 방식으로 대응할 수 있습니다. 그러나 이러한 접근 방식은 여러 가지 이유로 실패할 것입니다.

- 시스템 속도가 아무리 빨라도 필요할 때 사용할 수 없다면 소용없습니다.
- 도메인이 성장하고 자금이 더 많이 생성됨에 따라 시스템은 이에 대응할 수 있어야 합니다. 일일 마감 처리를 제시간에 완료할 수 있도록 규모를 확장합니다.
- 시스템은 단순히 사용 가능한 것뿐만 아니라, 일일 마감 펀드 가격 계산 중에 시스템이 다운되지 않도록 안정적이어야 합니다.
- 일일 마감 펀드 가격 계산이 85% 완료된 상태에서 시스템이 다운되면 어떻게 될까요? 시스템은 복구되어 중단된 부분부터 다시 계산을 시작할 수 있어야 합니다.
- 마지막으로, 시스템 속도가 빠르더라도 펀드 가격이 정확하게 계산되고 있는지 확인해야 합니다.

따라서 이 아키텍트는 성능뿐만 아니라 가용성, 확장성, 신뢰성, 복구 가능성 및 감사 가능성에도 동등하게 집중해야 합니다.

많은 비즈니스 목표는 복합적인 아키텍처 특성에서 시작됩니다. 이러한 특성을 분해하고, 결과적으로 얻어진 특성에 객관적인 정의를 부여하는 것이 아키텍트의 역할입니다. (이 점의 중요성은 6 장 에서 자세히 살펴보겠습니다.)

건축적 특징 추출

대부분의 아키텍처적 특징은 요구사항 문서에 명시적으로 기술된 내용에서 비롯됩니다. 예를 들어, 도메인 관련 고려 사항 목록에는 일반적으로 예상 사용자 수와 규모에 대한 구체적인 내용이 포함됩니다. 다른 특징들은 아키텍트의 도메인 지식에서 비롯됩니다 (모든 아키텍트가 자신의 도메인을 잘 알아야 하는 여러 이유 중 하나입니다).

예를 들어, 대학생들의 수강 신청을 처리하는 애플리케이션을 설계하는 아키텍트라고 가정해 봅시다. 계산을 쉽게 하기 위해 학교에 학생이 1,000명이고 수강 신청 기간이 10시간이라고 해봅시다. 이때 학생들이 10시간 동안 고르게 분산될 것이라고 가정하고, 그에 맞춰 시스템을 설계해야 할까요? 아니면 대학생들의 습관과 성향에 대한 당신의 지식을 바탕으로, 수강 신청 마지막 10분 동안 1,000명의 학생 모두가 몰리는 상황을 처리할 수 있도록 시스템을 설계해야 할까요?

학생들이 얼마나 미루는 습관이 있는지 아는 사람이라면 누구나 이 질문에 대한 답을 알 것입니다! 이런 세부 사항은 요구사항 문서에 거의 나타나지 않지만, 설계 결정에 중요한 영향을 미칩니다.

건축 카타의 기원 10여 년 전, 유명 건축가

테드 뉴어드는 건축 카타라는 독창적인 방법을 고안했습니다. 이는 신진 건축가들이 특정 분야에 대한 설명에서 건축적 특징을 도출하는 연습을 할 수 있도록 고안된 방법입니다. '카타'라는 단어는 일본어에서 유래했는데, 일본어에서는 정확한 자세와 기술을 강조하는 개인 무술 훈련 동작을 의미합니다.

> 어떻게 하면 훌륭한 디자이너를 얻을 수 있을까요? 훌륭한 디자이너는 당연히 디자인을 합니다.
> —프레드 브룩스

대규모 건축 프로젝트는 시간이 오래 걸리고, 건축가들이 경력 동안 설계하는 시스템은 기껏해야 6개 정도에 불과한 것이 일반적입니다. 그렇다면 우리는 어떻게 훌륭한 건축가를 확보할 수 있을까요?
핵심은 예비 건축가들에게 실무 경험을 쌓을 기회를 제공하는 것입니다. 이를 위해 테드는 최초의 건축 카타(Architecture Kata) 웹사이트를 만들었고, 저자인 닐과 마크는 이 웹사이트를 기반으로 최신 기술을 접목하여 이 책의 부록 웹사이트인 fundamental!sofsowarearchitecture.com에 게시했습니다. 건축 카타는 본래의 목적에 충실하게, 예비 건축가들에게 유용한 실습의 장을 제공합니다.

Katas와 함께 일하기

기본 전제는 각 카타 연습에서 도메인 용어로 명시된 문제, 요구 사항 세트, 그리고 추가적인 맥락 정보(요구 사항에는 나타나지 않지만 설계에 영향을 줄 수 있는 요소)가 제공된다는 것입니다. 소규모 팀은 정해진 시간 동안 아키텍처 특성 분석 및 다이어그램 작성을 포함한 설계를 진행합니다. 그런 다음 각 그룹은 결과를 공유하고 투표를 통해 가장 우수한 아키텍처를 설계한 팀을 결정합니다.

각 카타에는 미리 정의된 섹션이 있습니다.

설명: 시스템
: 이 해결하고자 하는 전반적인 도메인 문제.

사용자
: 시스템의 예상 사용자 수 및/또는 유형.

요구사항 도메
: 인 요구사항(도메인 사용자 및 전문가가 기대하는 바).

추가 설명: 저자들은 앞서
: 언급한 사이트의 카타 형식을 업데이트하여 중요한 고려 사항을 담은 추가 설명 섹션을 포함시켰으며, 이를 통해 연습을 더욱 현실적으로 만들었습니다.

저희는 독자들이 이 사이트를 활용하여 직접 카타 연습을 해보시기를 권장합니다. 누구나 점심시간을 이용해 예비 건축가들로 구성된 팀이 문제를 해결하는 모임을 주최할 수 있습니다.

경험이 풍부한 건축가는 설계 및 장단점 분석을 평가하고, 놓친 장단점과 대안 설계를 현장에서 또는 사후에 논의할 수 있습니다. 시간 제약이 있으므로 설계는 정교하지 않을 것입니다.

다음으로, 아키텍처 카타를 통해 아키텍트가 요구사항으로부터 아키텍처적 특징을 도출하는 방법을 설명하겠습니다. 실리콘 샌드위치 카타에 오신 것을 환영합니다.

카타: 실리콘 샌드위치

설명: 한 전국적
인 샌드위치 체인점이 현재의 전화 주문 서비스 외에 온라인 주문 시스템을 도입하고자 합니다.

사용자
수천 명, 어쩌면 언젠가는 수백만 명이 될지도 모릅니다.

요구 사항 • 사용자
가 주문을 할 수 있도록 하고, 상점에서 배달 서비스를 제공하는 경우 픽업 또는 배달을 선택할 수 있도록 해야 합니다. • 픽업
고객에게는 주문 픽업 시간
과 상점 위치 안내를 제공해야 합니다(이 안내는 교통 정보를 포함하는 여러 외부 지도 서비스와 연동되어야 합니다).

• 배달 서비스의 경우, 주문 상품을 가지고 운전기사를 사용자에게 전달하십시오. • 모바일 기기 접근성
을 제공하십시오. • 전국 단위의 일일 프로모션 및 특별
할인을 제공하십시오. • 지역별 일일 프로모션 및 특별 할인을 제공하십
시오. • 온라인, 매장 또는 배달 시 결제를 허용하십시오.

추가 정보 • 샌드위치 가게
는 프랜차이즈 형태로 운영되며, 각 가게마다 소유주가 다릅니다. • 모기업은 가까운 시일 내
에 해외 진출을 계획하고 있습니다. • 기업의 목표는 저렴한 노동력을 고용하여 이윤을 극대화하는
것입니다.

이러한 시나리오에서 아키텍처적 특징을 어떻게 도출하시겠습니까? 요구사항의 각 부분은 아키텍처의 하나 이상의 측면에 영향을 미칠 수 있으며(많은 부분이 그렇지 않을 수도 있습니다), 아키텍트는 시스템 전체를 설계하는 것이 아닙니다. 도메인 설계를 해결하기 위한 코드 작성에도 상당한 노력이 필요합니다(8 장 에서 다룹니다). 따라서 아키텍트는 특히 구조적인 측면에서 설계에 영향을 미치는 요소들을 찾아야 합니다.

먼저, 후보 아키텍처 특성을 명시적 특성과 암묵적 특성으로 구분합니다.

명시적 특징

명시적인 아키텍처적 특징은 필수 설계의 일부로서 요구사항 명세서에 나타납니다. 예를 들어, 쇼핑 웹사이트는 특정 수의 동시 접속 사용자를 지원해야 할 수 있으며, 도메인 분석가는 이를 요구사항에 명시합니다.
요구사항의 각 부분을 검토하여 아키텍처적 특성에 기여하는지 확인하십시오. 하지만 그 전에, 먼저 카타의 사용자 섹션에 제시된 예상 지표에 대한 도메인 수준의 예측을 고려하십시오.

가장 먼저 눈에 띄는 것은 사용자 수입니다. 현재는 수천 명이지만, 언젠가는 수백만 명에 이를 수도 있습니다(정말 야심찬 샌드위치 가게네요!). 따라서 확장성, 즉 심각한 성능 저하 없이 많은 동시 사용자를 처리할 수 있는 능력은 가장 중요한 아키텍처 특징 중 하나입니다. 문제 설명에서 확장성을 명시적으로 요구한 것이 아니라 예상 사용자 수로 표현했다는 점에 주목하세요. 아키텍트는 종종 도메인 용어를 엔지니어링 용어로 해독해야 합니다.

또한 요청 폭주를 처리할 수 있는 탄력성도 필요할 것입니다. 이 두 가지 특성은 종종 함께 언급되지만, 각각 다른 제약 조건을 가지고 있습니다.
확장성은 그림 5-1에 표시된 그래프와 같습니다 .

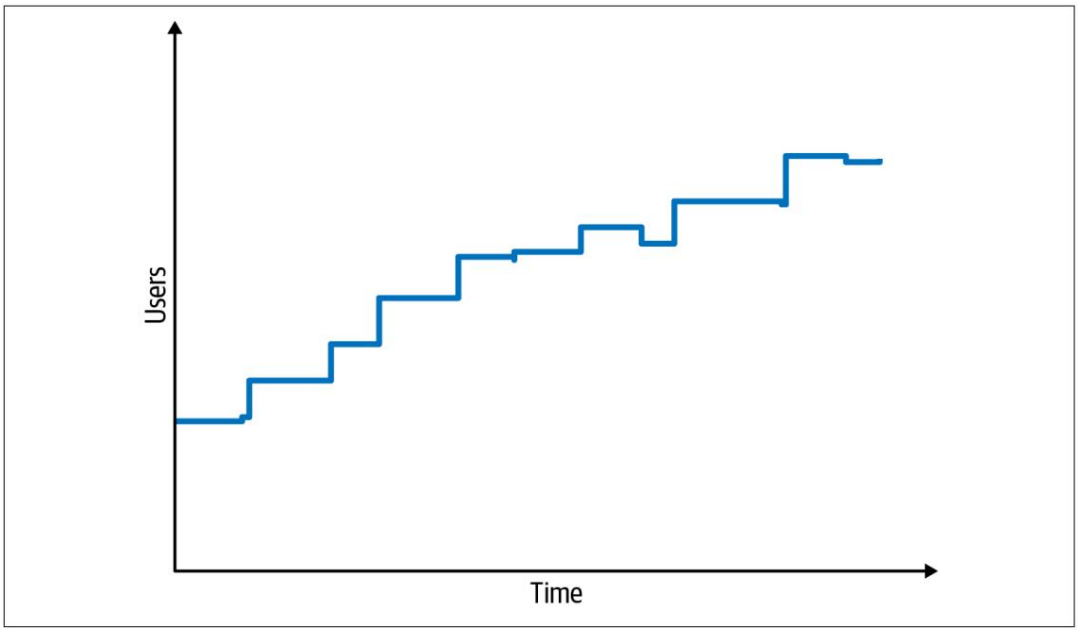

그림 5-1. 확장성은 시간에 따른 사용자 수 증가를 나타내는 척도입니다.

반면 탄력성은 그림 5-2에서 볼 수 있듯이 트래픽 급증을 측정합니다.

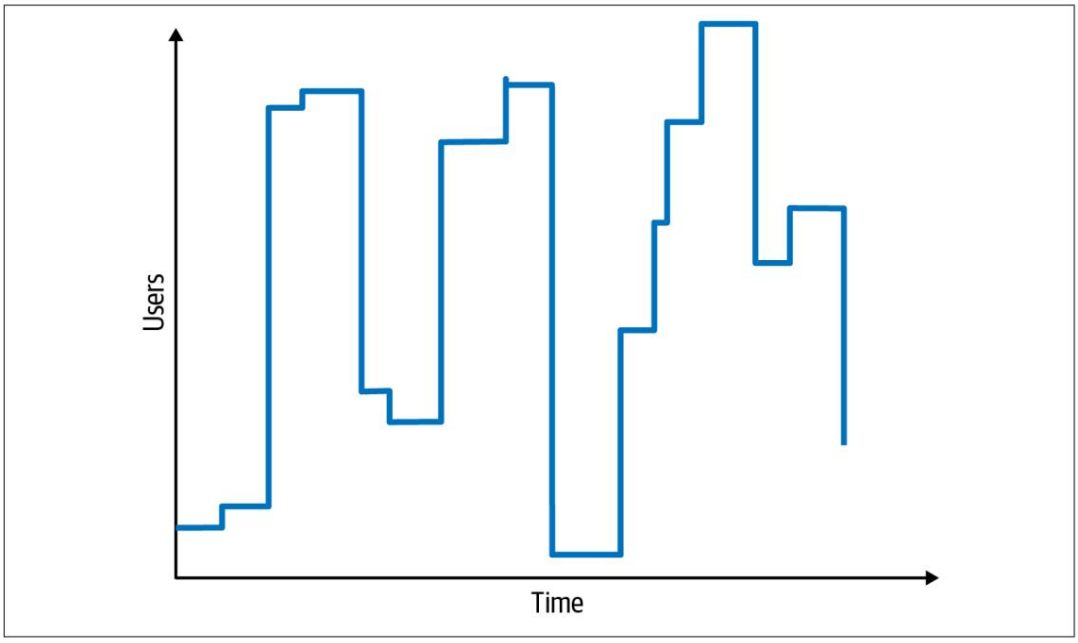

그림 5-2. 탄력적인 시스템은 사용자 폭증에 대응해야 합니다.

일부 시스템은 확장성은 뛰어나지만 탄력성은 떨어집니다. 예를 들어, 호텔 예약 시스템의 사용자 수는 특별 할인이나 이벤트가 없는 한 계절에 따라 예측 가능한 변동을 보일 것입니다. 반면 콘서트 티켓 예매 시스템을 생각해 보세요. 새로운 티켓이 판매되기 시작하면 열렬한 팬들이 사이트에 몰려들기 때문에 높은 수준의 탄력성이 요구됩니다. 탄력적인 시스템은 종종 확장성도 필요로 합니다. 즉, 사용자 폭증과 동시 접속자 수 급증을 처리할 수 있어야 합니다.

실리콘 샌드위치 요구사항에는 탄력성이 명시적으로 언급되어 있지 않지만, 중요한 고려 사항으로 파악해야 합니다. 요구사항에는 아키텍처적 특징이 직접적으로 명시되는 경우도 있지만, 문제 영역 내에 숨겨진 특징들이 있는 경우도 있습니다. 샌드위치 가게의 고객 유입량은 하루 종일 일정할까요, 아니면 식사 시간에 집중될까요? 거의 대부분 후자일 것이므로, 이러한 잠재적인 아키텍처적 특징을 파악해야 합니다.

다음으로, 이러한 비즈니스 요구사항들을 차례로 검토하여 아키텍처적 특성이 필요한지 살펴보십시오.

사용자는 주문을 하고, 상점에서 배달 서비스를 제공하는 경우 픽업 또는 배달을 선택합니다.
이러한 요구 사항을 충족하는 데 특별한 건축적 특징은 필요하지 않은 것 같습니다!
멘트.

픽업 고객에게는 샌드위치를 수령할 시간과 매장 위치 안내가 제공됩니다(매장 위치는 교통 정보가 포함된 외부 지도 서비스와 연동할 수 있는 옵션을 제공해야 합니다).

외부 매핑 서비스는 통합 지점을 의미하며, 이는 안정성과 같은 측면에 영향을 미칠 수 있습니다. 예를 들어, 개발자가 다음과 같은 서비스를 사용하는 시스템을 구축한다고 가정해 보겠습니다.

타사 시스템과의 통신이 실패할 경우, 해당 통신 시스템의 신뢰성에 영향을 미칩니다. 반대로, 아키텍처 특성을 지나치게 구체적으로 명시하는 것도 경계해야 합니다. 외부 트래픽 서비스가 중단되면 어떻게 될까요? 실리콘 샌드위치 사이트가 다운되어야 할까요, 아니면 트래픽 정보 없이 효율성이 약간 떨어지는 정도로만 처리해야 할까요? 설계자는 불필요한 취약성이나 불안정성을 설계에 반영하지 않도록 항상 주의해야 합니다.

배송 서비스의 경우, 주문 상품을 가지고 운전기사를 사용자에게 보내십시오.

이러한 요구 사항을 충족하는 데 특별한 건축적 특징은 필요하지 않은 것으로 보입니다.

모바일 기기 접근성을 제공하십시오.

이 요구사항은 주로 애플리케이션의 사용자 경험(UX) 디자인에 영향을 미치며, 모바일 친화적인 웹 애플리케이션 또는 여러 개의 네이티브 웹 애플리케이션 개발 방향을 제시합니다. 예산 제약과 애플리케이션의 단순성을 고려할 때, 여러 개의 애플리케이션을 개발하는 것은 과도할 가능성이 높으므로 모바일 최적화 웹 애플리케이션 개발이 유력합니다. 따라서 페이지 로딩 시간 및 기타 모바일 환경에 최적화된 성능을 구현하기 위해 특정 성능 관련 아키텍처 특성을 정의하는 것이 좋습니다.

이런 상황에서는 혼자서 행동해서는 안 됩니다. UX 디자이너, 도메인 이해관계자 및 기타 관련 당사자와 협력하여 결정을 검토해야 합니다. 예를 들어, 비즈니스 요구 사항에 따라 네이티브 애플리케이션 개발이 필수적인 동작이 있을 수 있습니다.

전국적인 일일 프로모션 및 특별 할인; 지역별 일일 프로모션 및 특별 할인.

이 두 가지 요구사항 모두 프로모션과 특별 행사에 걸쳐 맞춤 설정 기능을 명시합니다. 요구사항 1은 또한 사용자의 주소를 기반으로 한 맞춤형 교통 정보 제공을 의미합니다. 이러한 두 가지 요구사항을 바탕으로 맞춤 설정 기능을 아키텍처적 특징으로 고려할 수 있습니다. 예를 들어, 마이크로커널 아키텍처 스타일(13장에서 다룸) 은 플러그인 아키텍처를 정의함으로써 맞춤형 동작을 매우 효과적으로 지원합니다. 이 경우 기본 동작은 코어에 구현되고, 개발자는 위치 기반의 선택적 맞춤 설정 부분을 플러그인으로 작성합니다. 하지만 기존 설계 방식 또한 템플릿 방식과 같은 디자인 패턴을 통해 이러한 요구사항을 충족할 수 있습니다. 이러한 딜레마는 아키텍처 설계에서 흔히 발생하며, 설계자는 경쟁하는 옵션들 사이에서 끊임없이 장단점을 비교 검토해야 합니다. 특정 장단점에 대해서는 76페이지의 "설계 대 아키텍처 및 장단점"에서 더 자세히 논의합니다.

온라인, 매장 또는 배송 시 결제를 허용하세요.

온라인 결제에는 보안이 필요하지만, 이 요구사항 자체에는 암묵적인 수준 이상의 특별히 높은 보안을 요구하는 내용은 없습니다. 아키텍트는 설계 또는 아키텍처 단계에서 보안을 처리할 수 있으므로, 이 애플리케이션에서는 아키텍처적 특성으로서의 보안은 최소한의 고려 사항입니다.

샌드위치 가게는 프랜차이즈 형태로 운영되며, 각 가게마다 소유주가 다릅니다.

이러한 요구사항은 아키텍처에 비용 제약을 가할 수 있습니다. 비용, 시간, 직원 역량 등의 제약 조건을 적용하여 실현 가능성을 검토하고, 단순하거나 일부 기능을 희생하는 아키텍처가 타당한지 판단하십시오.

모회사는 가까운 시일 내에 해외 진출을 계획하고 있습니다.

이 요구사항은 국제화(흔히 "i18n"으로 약칭됨)를 의미합니다. 이러한 요구사항을 처리하기 위한 다양한 디자인 기법이 존재하며, 이를 수용하기 위해 특별한 구조를 만들 필요는 없습니다. 하지만 이는 분명 사용자 경험(UX) 관련 의사 결정에 영향을 미칠 것입니다.

기업의 목표는 저렴한 노동력을 고용하여 이윤을 극대화하는 것입니다.

이 요구사항은 사용성이 중요하다는 것을 시사하지만, 이 역시 건축적 특징보다는 디자인에 더 중점을 두고 있습니다.

앞서 제시된 요구사항에서 도출할 수 있는 세 번째 건축적 특징은 성능입니다. 누구도, 특히 피크 시간대에 성능이 떨어지는 샌드위치 가게에서 물건을 사고 싶어 하지 않습니다. 하지만 성능은 미묘한 차이가 있는 개념입니다. 어떤 종류의 성능을 염두에 두고 설계해야 할까요? (성능의 다양한 측면은 6 장 에서 다룹니다.)

성능 수치를 확장성 수치와 함께 정의하는 것도 중요합니다. 다시 말해, 특정 규모에 구애받지 않는 성능 기준선을 설정하고, 특정 사용자 수를 고려했을 때 허용 가능한 성능 수준을 결정해야 합니다. 아키텍처적 특성들은 서로 상호작용하는 경우가 많기 때문에, 설계자는 이러한 특성들을 서로 연관시켜 정의해야 합니다.

암묵적 특성

요구사항 문서에는 명시되지 않은 아키텍처적 특징들이 많지만, 이러한 특징들은 설계에 중요한 영향을 미칩니다. 시스템이 지원해야 할 암묵적인 아키텍처적 특징 중 하나는 가용성입니다. 즉, 사용자가 언제든 사이트에 접속할 수 있도록 보장하는 것입니다. 가용성과 밀접하게 관련된 또 다른 중요한 요소는 안정성입니다. 안정성은 사용자가 사이트를 이용하는 동안 사이트가 안정적으로 유지되도록 하는 것입니다. 연결이 끊겨 다시 로그인해야 하는 사이트에서 구매하고 싶어하는 사람은 아무도 없습니다.

보안은 모든 시스템에 내재된 특성입니다. 누구도 안전하지 않은 소프트웨어를 만들고 싶어하지 않기 때문입니다. 하지만 중요도에 따라 보안에 부여하는 우선순위는 달라질 수 있으며, 이는 보안 정의의 상호 연관성을 보여줍니다. 보안이 설계의 구조적 측면에 영향을 미치고 애플리케이션에 중요하거나 필수적인 경우, 이를 아키텍처적 특성으로 간주할 수 있습니다.

실리콘 샌드위치 구조에서는 결제 처리가 제3자에 의해 이루어져야 한다고 가정할 수 있습니다. 따라서 개발자가 일반적인 보안 수칙(신용카드 번호를 평문으로 전달하지 않거나, 과도한 정보를 저장하지 않는 등)을 준수한다면, 애플리케이션의 설계가 우수하기만 하면 충분하며, 보안을 고려한 특별한 구조적 설계는 필요하지 않습니다. 아키텍처적 특성을 염두에 두어야 합니다.

아키텍처 특성들은 서로 시너지 효과를 내며 상호 작용합니다. 이것이 바로 아키텍처 특성을 과도하게 명시하는 것이 흔히 발생하는 함정인 이유입니다. 특성을 과도하게 명시하는 것은 시스템 설계를 지나치게 복잡하게 만들기 때문에, 특성을 불충분하게 명시하는 것만큼이나 해롭습니다.

실리콘 샌드위치가 지원해야 하는 마지막 주요 아키텍처 특성인 사용자 정의 기능은 요구 사항의 여러 세부 사항으로 구성됩니다. 문제 영역의 여러 부분에서 레시피, 지역 판매, 길 안내와 같은 사용자 정의 동작을 제공하며, 이러한 동작은 로컬에서 재정의될 수 있습니다. 따라서 아키텍처는 사용자 정의 동작을 지원해야 합니다. 일반적으로 이는 애플리케이션 설계 영역에 속합니다. 그러나 정의에 명시된 바와 같이, 문제 영역의 일부가 사용자 정의 구조에 의존하는 경우 아키텍처 특성 영역으로 이동합니다. 하지만 이 설계 요소가 애플리케이션의 성공에 필수적인 것은 아닙니다. 아키텍처 특성을 선택하는 데 정답은 없으며, 잘못된 선택만 있을 뿐입니다.

> 건축에는 틀린 답이 없고, 비싼 답만 있을 뿐이다.
>
> 마크의 유명한 명언 중 하나

실리콘 샌드위치 카타에서 디자인과 아키텍처의 장단점,

그리고 설계와 아키텍처의 차이점을 생각해 보세요. 시스템의 일부로 맞춤 설정 기능을 떠올릴 수 있겠지만, 여기서 중요한 질문은 아키텍처인가 디자인인가입니다. 아키텍처는 구조적 요소를 의미하는 반면, 디자인은 아키텍처 내부에 존재합니다.
실리콘 샌드위치 아키텍처의 경우, 마이크로커널과 같은 아키텍처 스타일을 선택하고 사용자 정의를 위한 구조적 지원을 구축할 수 있습니다. 하지만 여러 가지 고려 사항 때문에 다른 스타일을 선택해야 하는 경우, 개발자는 부모 클래스에서 워크플로를 정의하고 자식 클래스에서 이를 재정의할 수 있는 템플릿 메서드 디자인 패턴을 사용하여 사용자 정의를 구현할 수 있습니다. 어떤 옵션이 더 나을까요?

건축의 모든 것과 마찬가지로, 상황에 따라 다릅니다. 첫째, 마이크로커널 아키텍처를 구현하지 않을 만한 타당한 이유(예: 성능 및 결합도)가 있는지 살펴봐야 합니다. 둘째, 바람직한 아키텍처적 특성을 구현하는 것이 설계 방식에서 더 어려운지, 아니면 다른 설계 방식에서 더 어려운지가 중요합니다. 셋째, 아키텍처 설계에서 모든 특성을 지원하는 데 드는 비용과 설계에서 지원하는 데 드는 비용을 비교해야 합니다. 이러한 절충 분석은 아키텍트의 중요한 역할 중 하나입니다.

무엇보다 중요한 것은 개발자, 프로젝트 관리자, 운영팀 및 소프트웨어 시스템 공동 구축자들과의 협업입니다. 아키텍처 관련 결정은 구현팀과 분리하여 내려서는 안 됩니다(이는 악명 높은 상아탑 아키텍처 안티패턴으로 이어집니다). 실리콘 샌드위치 방식의 경우, 아키텍트, 기술 책임자, 개발자 및 도메인 분석가가 협력하여 사용자 정의 기능을 구현하는 최적의 방법을 결정해야 합니다.

완벽한 아키텍처 특성 세트를 찾는 데 너무 스트레스를 받을 필요는 없습니다. 개발자는 다양한 방식으로 기능을 구현할 수 있습니다.
하지만 중요한 구조적 요소를 정확하게 파악하면 더 단순하거나 우아한 디자인을 구현할 수 있습니다. 예를 들어, 구조적으로 사용자 정의 기능을 지원하지 않는 아키텍처를 설계하고, 대신 애플리케이션 자체의 디자인에서 해당 기능을 지원하도록 할 수 있습니다(76 페이지의 "디자인과 아키텍처 및 장단점" 참조).

명심하세요: 건축에는 최고의 디자인이란 없고, 단지 최악의 상황을 차선으로 모아놓은 절충안만 있을 뿐입니다.

가장 간단한 필수 요소 세트를 찾을 때 이러한 아키텍처 특성의 우선순위를 정하십시오. 팀이 아키텍처 특성을 처음 파악한 후에는 가장 중요도가 낮은 특성을 파악하는 것이 유용합니다. 만약 하나를 제거해야 한다면 무엇을 제거하시겠습니까? 일반적으로 아키텍트는 명시적인 아키텍처 특성을 제거하는 경향이 더 강한데, 이는 암묵적인 특성들이 전반적인 성공을 뒷받침하기 때문입니다. 성공에 무엇이 중요하거나 필수적인지 정의하는 방식은 애플리케이션에 각 아키텍처 특성이 실제로 필요한지 판단하는 데 도움이 됩니다. 가장 필요성이 낮은 특성을 파악하는 것은 필수적인 요구 사항을 결정하는 데 도움이 될 수 있습니다.

실리콘 샌드위치 구조의 경우, 우리가 파악한 아키텍처 특성 중 가장 중요도가 낮은 것은 무엇일까요? 물론 절대적인 정답은 없습니다. 하지만 이 경우, 솔루션은 사용자 정의 기능이나 성능 중 하나를 포기해야 할 수도 있습니다. 사용자 정의 기능을 아키텍처 특성에서 제외하고 애플리케이션 설계 단계에서 해당 기능을 구현하는 방안을 고려할 수 있습니다. 운영 아키텍처 특성 중에서는 성능이 성공에 가장 덜 중요한 요소일 가능성이 높습니다. 물론 그렇다고 개발자들이 성능이 매우 떨어지는 애플리케이션을 만들려고 한다는 의미는 아닙니다. 단지 이 설계는 확장성이나 가용성과 같은 다른 특성보다 성능을 우선시하지 않는다는 것을 의미합니다.

건축적 특징의 제한 및 우선순위 설정

도메인 이해관계자들과 협력하여 핵심 아키텍처 특성을 정의할 때 한 가지 팁을 드리자면, 최종 목록을 최대한 간결하게 유지하는 데 집중해야 합니다. 흔히 저지르는 실수는 모든 아키텍처 특성을 지원하는 일반적인 아키텍처를 설계하려는 것입니다. 아키텍처가 지원하는 각 특성은 전체 시스템 설계를 복잡하게 만들기 때문에, 너무 많은 특성을 지원하려고 하면 복잡성이 더욱 커집니다. 이는 아키텍트와 개발자들이 소프트웨어 개발의 원래 동기인 문제 영역을 해결하기 시작하기도 전에 발생하는 문제입니다. 특성의 개수에 집착하지 말고, 설계의 본래 목적, 즉 단순성을 유지하는 데 집중해야 합니다.

사례 연구: 바사호 건축적 특

징을 지나치게 구체적으로 명시한 나머지 결국 프로젝트를 좌초시킨 대표적인 사례는 바로 바사호입니다. 바사호는 1626년부터 1628년까지 스웨덴의 한 왕이 역사상 가장 웅장한 배를 원하여 건조한 전함입니다. 당시까지 배는 병력 수송선이나 군함으로만 사용되었는데, 바사호는 두 가지 역할을 모두 수행하도록 설계되었습니다. 대부분의 배는 갑판이 하나였지만, 바사호는 두 개였습니다! 또한 모든 대포는 비슷한 크기의 다른 배들에 비해 두 배나 컸습니다. 숙련된 조선 기술자들은 우려를 표했지만, 아돌푸스 왕의 요구를 거절할 수 없었습니다.

건조 완료를 기념하기 위해 바사호는 스톡홀름 항구로 출항하여 한쪽 측면에서 대포를 발사하며 예포를 쏘았습니다. 하지만 배가 무게중심이 높아 전복되어 바닷속으로 가라앉았습니다. 1961년, 인양 작업자들이 바사호를 인양했고, 현재 스톡홀름 박물관에 전시되어 있습니다.

아키텍처 특성 분석 과정에서 아키텍트와 비즈니스 이해관계자 간의 협업은 매우 중요합니다. 하지만 아키텍트가 가능한 아키텍처 특성 목록을 광범위하게 제시하고 비즈니스 이해관계자에게 어떤 특성을 아키텍처에 포함시키고 싶은지 묻는다면, 매번 어떤 대답이 나올까요? "모두 다요!"

따라서 건축가는 구조적 결정에 영향을 미치고 성공에 필수적인 건축적 특성을 파악하는 기술이 필요합니다. 저자들은 이러한 노력을 용이하게 하기 위해 여러 가지 기법을 개발했는데, 그중 하나가 그림 5-3 에 나와 있는 건축적 특성 "워크시트"입니다 (https://developertoarchitect.com/resources.html 에서 다운로드 가능).

이 워크시트는 건축가가 진행하는 대화형 세션에서 사용하도록 고안되었으며, 건축가는 이해관계자들에게 필요한 건축적 특징의 수와 세부 사항에 대한 의견을 수렴합니다. 왼쪽에는 원하는 건축적 특징을 적을 수 있는 7개의 칸이 있습니다.

왜 하필 7개일까요? 안 될 이유가 있을까요? 진지하게 말하자면, 아키텍트는 목록을 적절한 개수로 제한해야 합니다. 6개나 8개도 괜찮겠지만, 7이라는 숫자에는 흥미로운 심리적 의미가 담겨 있습니다. 두 번째 열에는 암묵적인 아키텍처 특성이 포함됩니다. 이러한 특성은 대부분의 시스템에 나타나지만, 때때로 아키텍트는 이러한 특성을 애플리케이션의 핵심 고려 사항으로 간주하여 특별한 설계와 고려를 요구합니다. 이러한 경우, 아키텍트는 암묵적인 특성을 첫 번째 열로 옮깁니다. 첫 번째 열이 가득 차고 기존 항목을 대체할 더 나은 항목이 나타나면, 아키텍트는 해당 항목을 "고려 대상" 범주로 이동합니다.

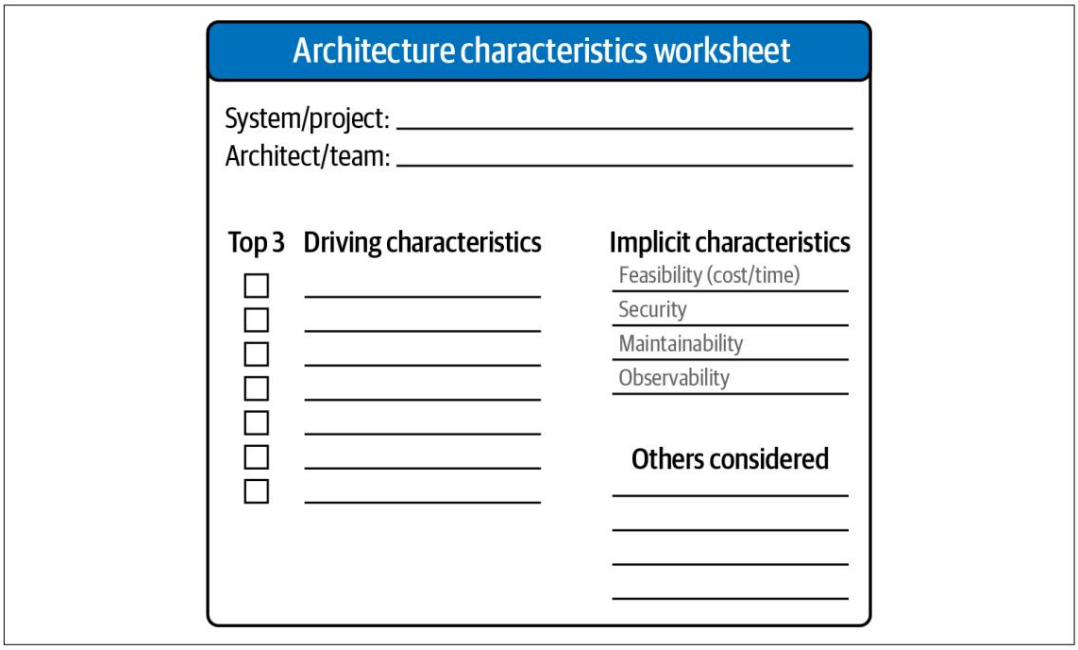

그림 5-3. 건축적 특징 워크시트

마지막 단계는 순서에 상관없이 가장 우선순위가 높은 건축적 특징 세 가지를 공동으로 선택하는 것입니다(각 항목 옆의 체크박스에 표시). 이 과정을 통해 건축가들은 설계 결정 및 절충 분석에 활용할 수 있는 핵심 요소 목록을 간결하고 우선순위에 따라 정리할 수 있습니다.

많은 아키텍트와 도메인 이해관계자들은 애플리케이션이나 시스템이 지원해야 하는 아키텍처 특성의 최종 목록을 만드는 데 우선순위를 두려고 합니다. 물론 이는 바람직한 결과이지만, 대부분의 경우 시간 낭비뿐 아니라 주요 이해관계자들과 불필요한 갈등과 의견 불일치를 야기하는 헛수고에 불과합니다. 모든 이해관계자가 모든 특성의 우선순위에 동의하는 경우는 드뭅니다. 더 나은 접근 방식은 도메인 이해관계자들이 최종 목록에서 가장 중요한 특성 세 가지를 (순서에 상관없이) 선택하도록 하는 것입니다. 이렇게 하면 합의를 도출하기가 훨씬 쉬워지고 무엇이 가장 중요한지에 대한 논의가 활발해집니다.

이 모든 것은 건축가가 중요한 건축적 결정을 내릴 때 절충점을 분석하는 데 도움이 됩니다.