**기획 문서**

**비즈니스 요구사항**

**누가 누가 서비스를 이용하는가**

* **스타일쉐어를 이용하는 유저가 서비스를 이용한다. 유저는 로그인한 상태일수도 있고, 비회원으로 서비스를 이용할 수도 있다.**

**왜 어떤 이유로 상품 리스트 페이지를 이용하는가**

* **(최소 요구사항) 유저가 마음에 드는 상품을 찾기 위해서, 다양한 상품들을 한눈에 보기 위해 상품 리스트를 이용한다.**
* **(심화 요구사항) 유저가 최신 트렌드를 알아보기 위해서, 인기 있는 상품과 리뷰들을 보기 위해서 상품 리스트를 이용한다.**

**판매자 입장에서는 유저가 상품 리스트 페이지를 목표하는 상품이 없이도 SNS 서비스를 제공해 소통의 장으로 만들어서 오래 머무르게 하는 것이 더 좋은 유저 경험을 제공하고 잠재적인 유저의 니즈를 구매 단계까지 끌어내는 전략이 될 수 있다. (스타일쉐어 앱을 사용해보고 찾았습니다.)**

**왜 어떤 이유로 장바구니 페이지를 이용하는가**

* **(최소 요구사항) 유저가 현재까지 마음에 든다고 선택했던 상품들을 조회, 구매, 제거하고 최종 결제 금액을 조회하기 위해 장바구니 페이지를 이용한다.**
* **(심화 요구사항) 장바구니 페이지는 결제 직전의 유일한 단계로서 유저가 번거로운 작업 없이 빠르게 원하는 상품을 구매하는 단계까지 안내하는 역할을 한다.**

**유저가 번거로운 작업 때문에 구매를 포기하지 않게 옵션 결정 과정을 직관적으로 만들고, 유저가 고른 상품의 장점을 부각시켜서 유저가 상품을 구매하는 이유를 깨닫게 도와준다.**

**무엇을 어떤 상품을 다룰 것인가**

* **일반적인 웹 쇼핑몰의 상품들을 담는다. 하지만, 주로 다룰 상품 종류에 맞게, 상품을 어떻게 유저에게 보여줄지에 따라서 집중할 정보가 다르다.  
  또한 상품들의 종류마다 어떤 정보를 담을 것인지, 얼마나 많은 깊이의 정보가 필요한지에 따라서 RDBMS와 NoSQL중 적합 여부가 결정된다.**
* **상품의 가치를 고객에게 효과적으로 보여주기 위해서는 우선 시각적인 증거인 상품 이미지가 있어야 한다. 또, 상품 자체의 특징에 대한 키워드, 다른 사람들의 리뷰를 직관적으로 함께 보여주면 고객에게 상품 가치를 효과적으로 전달할 수 있다.**

**어떻게 어떻게 상품 리스트 페이지를 보여줄 것인가**

* **(최소 요구사항) 유저에게 다양한 상품들을 리스트 방식으로 편하게 볼 수 있도록 안내한다. Infinite scroll(스크롤을 통해 리스트 로딩)방식을 사용해서 다 봤다는 인식 없이 유저가 계속 상품구경을 할 수 있도록 도와준다.**
* **(최소 요구사항) 상품 리스트 페이지에서 바로 상품 장바구니에 넣기 기능을 구현해서 유저가 구매 결정 단계까지 빠르게 진행할 수 있게 만든다.**
* **(심화 요구사항) 유저가 각 상품들의 첫인상을 잘 느낄 수 있도록 상품 사진이 차지하는 비중을 높여서, 상품 리스트를 볼 때 강한 첫인상이 남도록 도와준다. 상품의 평가나 키워드 등을 강조해서 유저가 쉽게 상품을 파악할 수 있게 도와준다.  
  또는 SNS 형식으로 유저가 상품 리스트가 아닌 SNS를 보고 있다는 인식을 줘서, 유저가 직접 참여함과 동시에 유저에게 더 많은 상품을 효과적으로 노출시킬 수 있다.**

**어떻게 어떻게 상품 장바구니 페이지를 보여줄 것인가**

* **(최소 요구사항) 유저가 장바구니의 상품들을 조회, 구매, 제거 및 최종 결제 금액을 조회할 수 있게 해준다.**
* **(심화 요구사항) 장바구니 페이지에서 상품 구매에 필수적인 옵션을 한눈에 보고 빠르게 선택할 수 있도록 인터페이스를 직관적으로 배치한다.**
* **(심화 요구사항) 유저가 상품의 장점을 쉽게 기억할 수 있도록 상품의 키워드 또는 추천수 등 긍정적인 지표를 부각한다.**
* **(심화 요구사항) 유저가 결제 금액에 부담을 느끼지 않도록 배려해서 디자인한다.**

**언제 언제 유저가 서비스를 이용하는가**

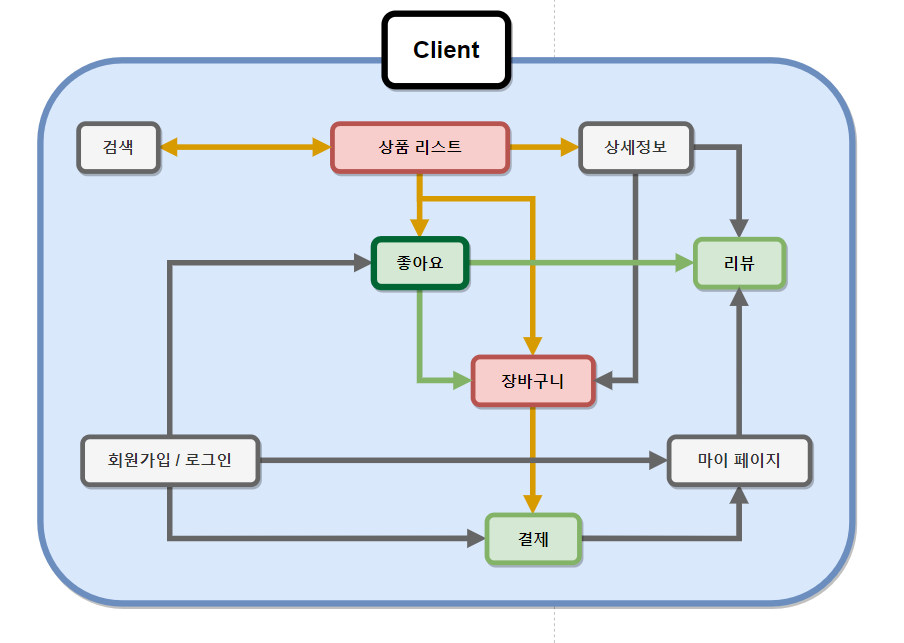
* **(최소 요구사항) 유저가 상품 정보를 알고, 구매하고 싶을 때 이 서비스를 이용한다.**
* **(심화 요구사항) 꼭 상품을 구매하려고 서비스를 이용하는 것이 아니라, 심심할 때 서비스를 이용할 수 있도록 SNS 기능을 만들면 좋을 것 같다. 자신이 올린 포스트의 좋아요 수에 따라 마일리지나 상품 할인쿠폰 등 인센티브를 제공하는 방식으로 많은 유저가 서비스에 접근하고 상품 정보와 가까워지게 유도하는 것도 좋을 것 같다.**

**어디서 어디서 유저가 이 서비스에 접근하는가**

* **(최소 요구사항) 웹 사이트를 통해서 서비스에 접근하고 이용한다.**
* **(심화 요구사항) 이 서비스를 전용 모바일 애플리케이션으로 접근하면 유저들이 이용하기 훨씬 편할 것이다.**

**소프트웨어 요구사항**

**Component Diagram**

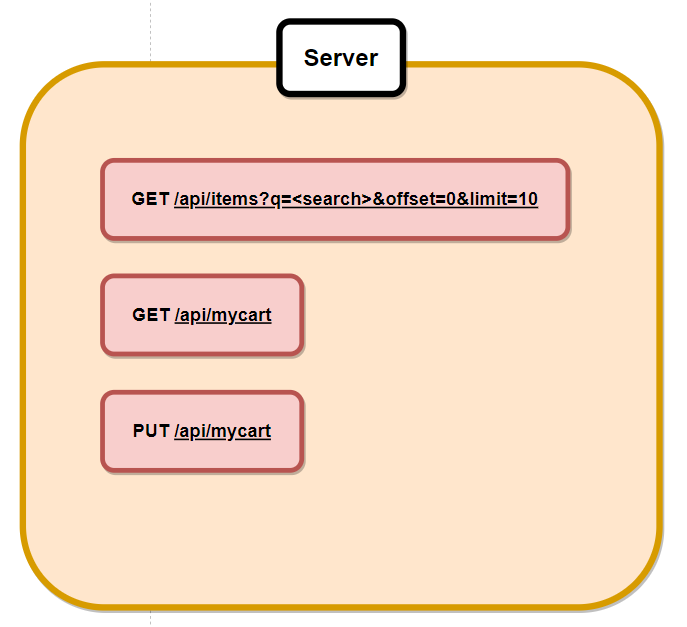


**Client의 기능적인 컴포넌트들을 나타낸 Component diagram이다.**

**실제 요구사항에서 나와있지 않지만, 과제의 요구사항인 “상품 리스트 페이지”와 “장바구니”가 정상적으로 웹 쇼핑몰 시나리오에서 기능하기 위해서는 위의 그림과 같이 검색, 상세정보, 회원가입/로그인, 마이 페이지, 결제 기능이 필요하다.**

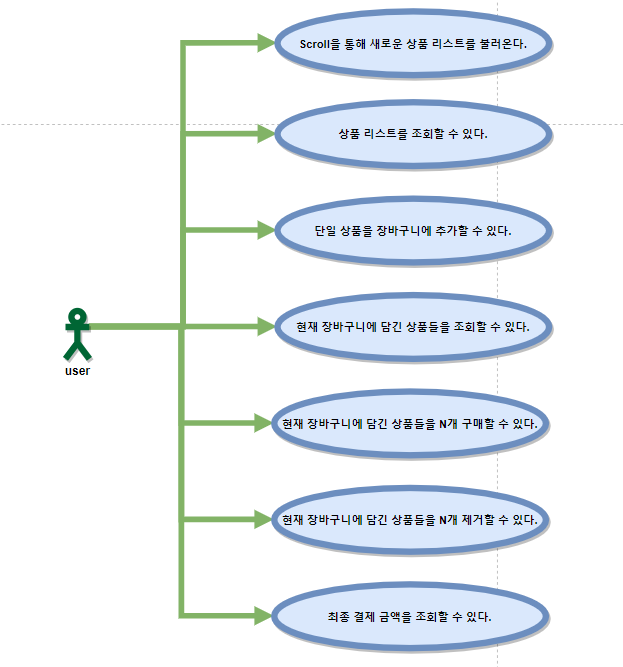
**또한 초록색으로 표시한 부분은 기획상 비즈니스 모델에 핵심이 되는 기능들이다. 실제로 비즈니스 요구사항에서 언급된 것처럼 기본적인 시나리오는 다음과 같은 프로세스와 같다.  
“유저가 상품리스트에서 상품 탐색” => ”상세정보 및 리뷰를 보며 장바구니 담기” => “결제”  
이외에도 유저에게 더 좋은 경험과 함께 소통과 관심사 공유라는 더 높은 차원의 서비스를 제공해서 유저의 잠재적인 구매 가능성을 이끌어내고 유저 그룹을 활성화하는 효과를 다음 프로세스를 강화시킴으로 얻을 수 있다.  
“유저가 일상적인 리뷰 작성”=>”다른 사람의 리뷰를 보며 소통”=>”활성화된 리뷰들을 통해 간접적인 상품 홍보”=>”받은 좋아요에 따라 피드백을 적용해서 리뷰 장려”=>”좋은 리뷰 작성”  
같은 선 순환 프로세스를 시도해볼 수 있다.**

**실제로 구현할 부분은 상품 리스트와 장바구니 기능이지만, 설계 과정에서 전체 흐름을 잡고 가는 것이 더 좋은 서비스를 제공하는 기능을 구현할 수 있기 때문에 먼저 전체 기획을 간단하게 언급했다.**



**다음은 클라이언트 기능에 대응되는 서버 API를 추상화해서 표현했다. 실제 배포환경에 가까운 nginx, docker같은 아키텍처보다 기능 자체에 초점을 맞춰서 표현했다.**

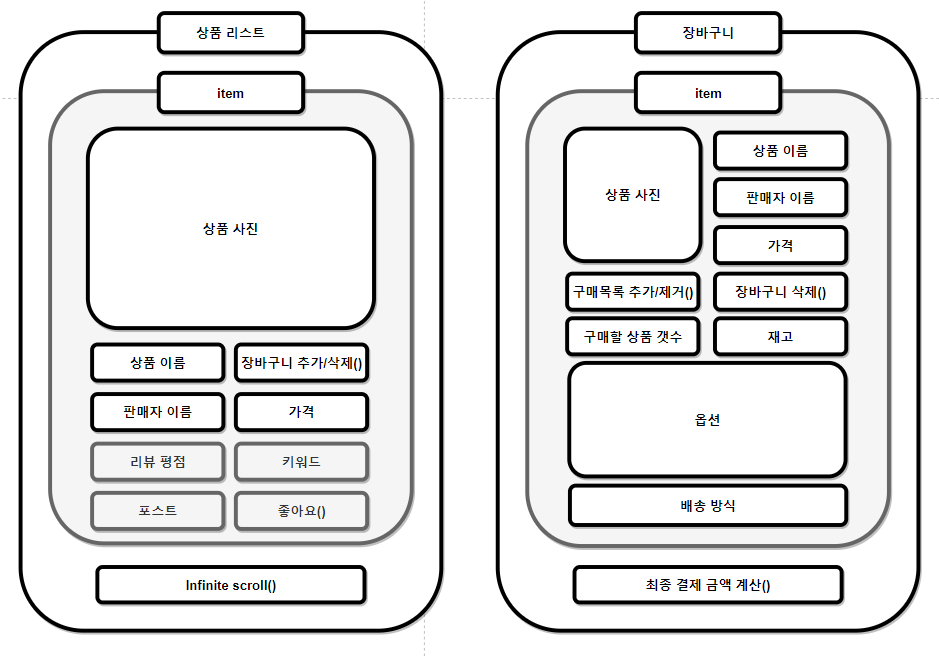
**Use Case Diagram**



**다음은 Use Case Diagram이다. 실제 구현해야 할 기능을 명시해뒀다. 다음과 같은 내용은 이후 validation과정에서 기준이 될 목록들이다.**

**전체 시스템에선 전체 쇼핑몰 관리자 또는 상품 판매자 역시 agent가 될 수 있고, user가 가질 수 있는 다른 use case들이 많지만, 여기서는 구현 범위를 벗어나기 때문에 생략했다.**

**Class Diagram**



**다음은 상품 리스트와 장바구니 페이지에 대한 Class Diagram이다.**

**상품 리스트 페이지에서는 infinite scroll 기능을 지원하고, 안에 item 정보에서는 상품 사진이 높은 비중을 가지며, 상품 이름, 판매자 이름, 가격 정보를 갖고 있다. 또 장바구니 추가/삭제 기능을 지원한다. 흐리게 처리된 항목들은 기획상 꼭 필요하다고 생각하지만 기능 구현에는 일단 제외된 항목들이다.**

**장바구니 페이지에서는 최종 결제 금액 계산을 지원하고, 안에 item 정보에서는 상품 사진, 상품이름, 판매자 이름, 가격, 구매할 상품 개수, 재고 정보를 갖고 있다. 또한 구매목록 추가/제거 기능, 장바구니 삭제 기능을 제공하며, 구매 옵션과 배송 방식에 매우 큰 비중을 두어 사용 시 장바구니에서 직관적인 옵션 선택을 통해 최종 결제 단계로 빠르게 도달할 수 있도록   
도와준다.**

**사용할 기술 Stack**

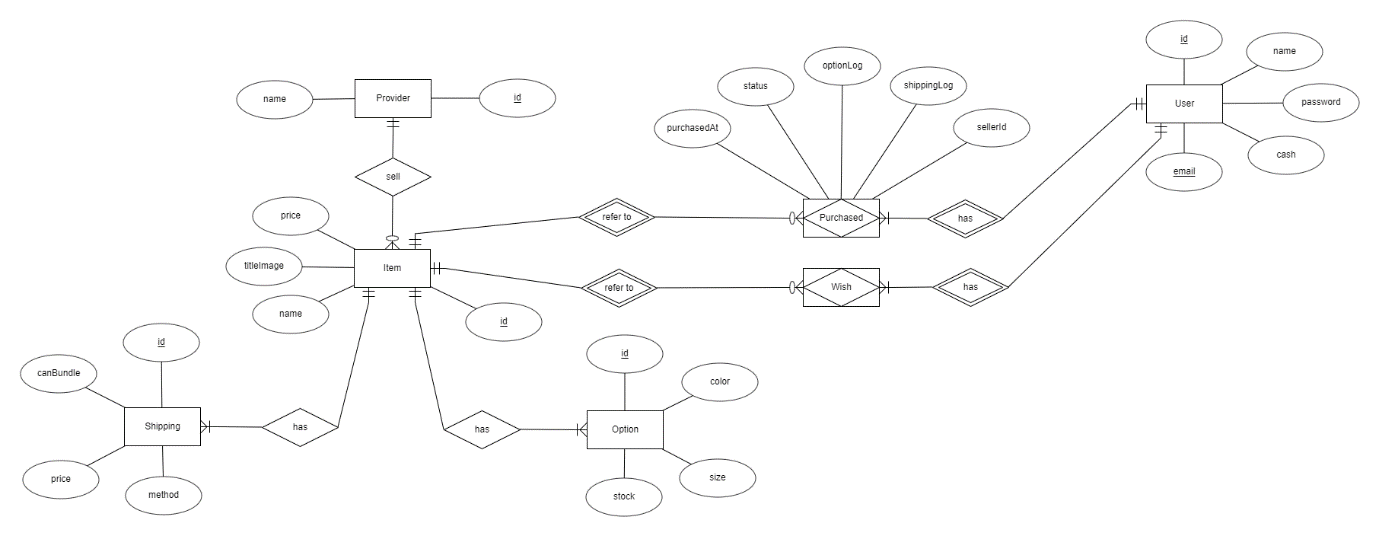
**Client를 만들기 위해 Node.js, React, Redux, Axios, AntDesign, TypeScript를 사용한다.**

**이미 익숙한 기술들이라서 큰 무리 없이 빠르게 구현을 할 수 있고, 성능상으로 효율적인 기술들이며, Scalability가 뛰어나고 모듈화가 잘 되어 있어서 완성도 높은 프로젝트를 만들기 좋다는 이유로 위와 같은 기술을 사용했다. 사실 구현할 사이즈가 작기 때문에 Redux와 TypeScript는 보일러플레이트를 만드는 오버헤드가 더 크지만, 완성도를 높이기 위해 사용한다.**

**Server를 만들기 위해 Node.js, Express, TypeScript, TypeORM, MySQL, Jest를 사용한다.**

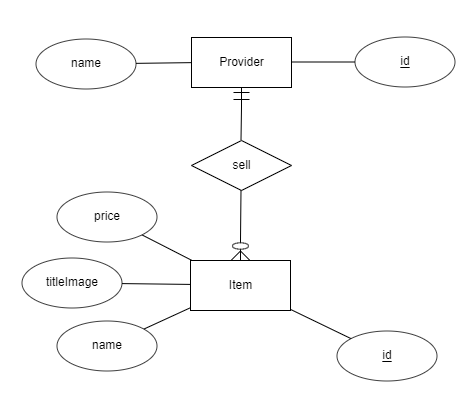
**이전에 경험했기 때문에 익숙한 기술들이고 성능상으로 효율적이며, Scalability가 뛰어나고 모듈화 하기 좋은(적극적으로 한다면) 기술이기 때문에 위와 같은 Stack을 사용한다. Backend에서는 API에 대해 Test를 해서 안정성을 보장할 것이다.**

**구현 양 자체는 적지만, 깔끔한 구조 및 각 layer에 맞게 정돈된 코드로 구현할 생각이다.**

**ER Diagram**

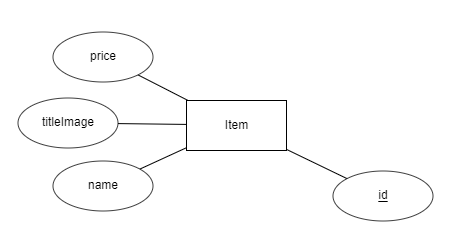
**기획 및 구현할 기능을 기반으로 ER Diagram을 만들었다.**

1. **Provider**



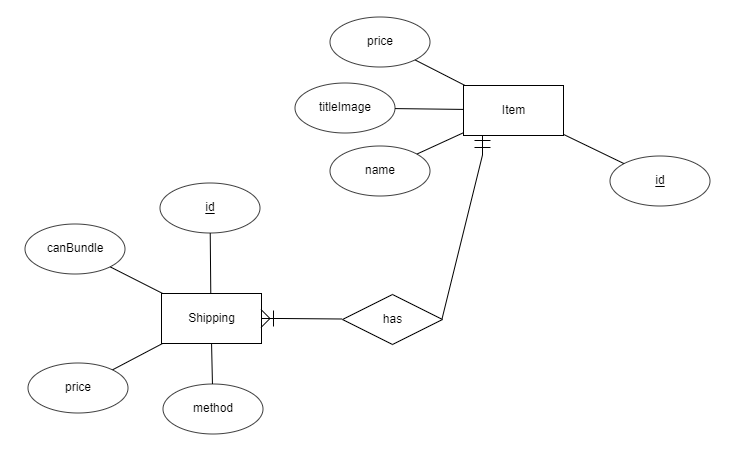
**상품을 등록하고, 판매하는 객체를 나타낸다. Provider : Item = 1 : N (sell) 관계이고, Item에서 Provider의 id를 저장하는 방식으로 관계를 구현한다.**

1. **Item**



**상품을 나타낸다. Provider, Shipping, Option, User(Purchased, Wish)와 모두 관계를 갖고 있다.**

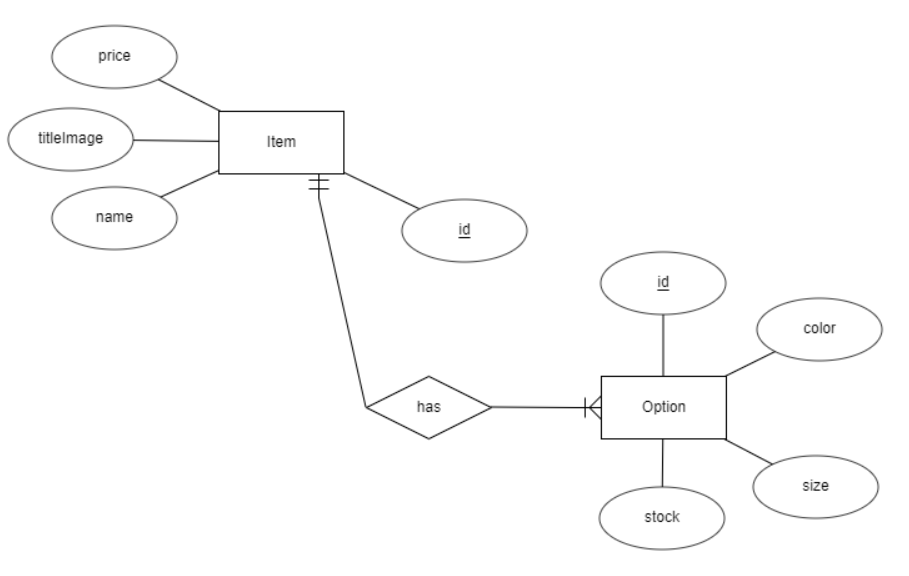
1. **Shipping**



**상품의 배송 방식을 저장하는 객체를 표현한다. Shipping : Item = N : 1(has)관계이고,**

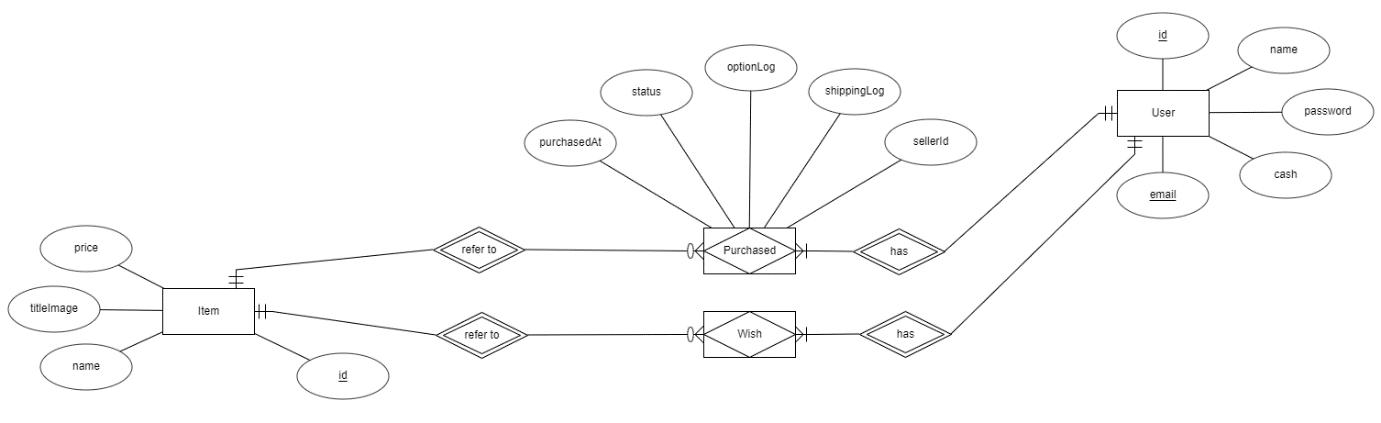
**Shipping에서 Item의 id를 저장하는 방식으로 관계를 구현한다.**

1. **Option**



**상품 구매 시 선택하는 option을 표현한다. Item : Option = 1 : N(has)관계이고,**

**Option에서 item의 id를 저장하는 방식으로 관계를 구현한다.**

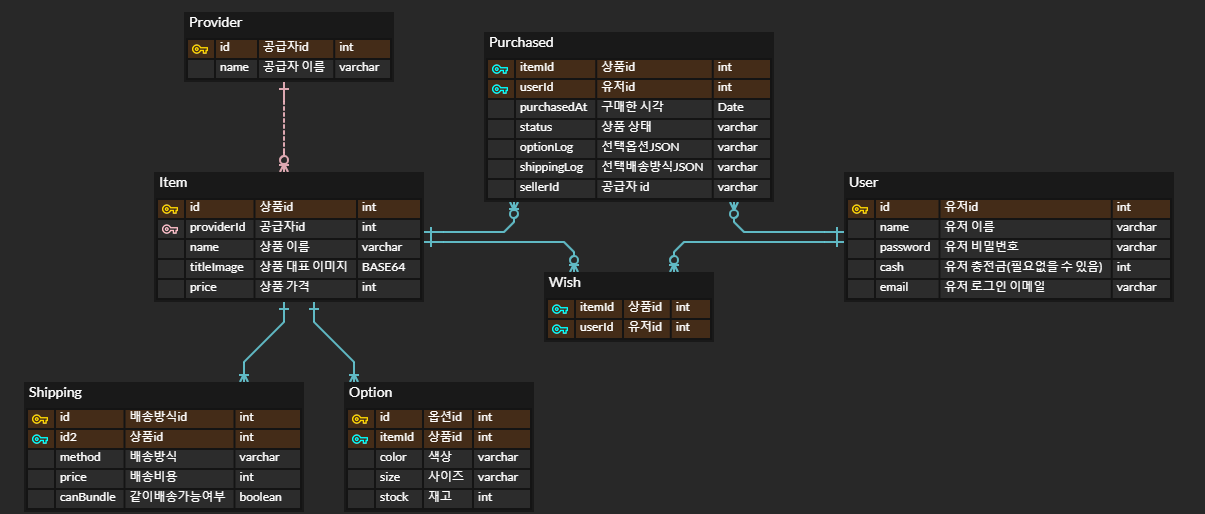
1. **User**

**상품을 위시리스트(장바구니)에 등록하고, 구매할 수 있는 서비스의 사용자를 객체로 표현한다.**

**상품을 장바구니에 등록하면, Wish에 추가되고, 상품을 구매하면 Purchased에 추가된다. Wish와 Purchased 모두 User id와 Item id를 id로 가지는 associative entity로서 Item과 User의 M : N 관계를 구현한다.**

**특히 Purchased의 경우, 판매자가 삭제되거나, item이 삭제되거나 option 또는 shipping 정보가 변경된다고 해도, 구매 정보를 온전히 보존하기 위해서 sellerId, optionLog, shippingLog에 판매자id와 실제 구매 당시 옵션을 저장한다.**

**Purchased의 purchased\_at은 구매 시각을 나타내고, status는 배송 대기, 배송 완료 등 상품 상황을 저장한다.**

**ER Diagram**

**ER Diagram을 Relational Model로 변환한 결과다.**

**유저 충전금은 필요 없을 수 있지만 이후 마일리지나 포인트 개념을 구현할 수   
있어서 cash라는 이름으로 구현해두었다.**