## 5. 기타 사항 [본문에서 표현되지 못한 작품의 가치(Value)] 및 제작 노력

<목차>


- ㅇ 쇼핑몰을 직접 구축한 이유
- ㅇ 어떤 목적으로 Kafka를 선택했는가?
- ㅇ 어떤 목적으로 ElasticSearch를 선택했는가?
- ㅇ 어떤 목적으로 추천 알고리즘: UserCF & ItemCF를 선택했는가?
- ㅇ 어떤 과정으로 Web Socket를 선택했는가?
- ㅇ 가상 사용자 시뮬레이션


### ㅇ 쇼핑몰을 직접 구축한 이유

우리 프로젝트는 쇼핑몰과 쇼핑몰을 기반으로 Kafka를 통해 데이터를 수집하고 저장, 분석하는 시스템 2가지 요소로 구성되어 있다. 사용자의 쇼핑몰 활동의 어느 시점에서 데이터를 수집할 것인지 정교하게 제어하기 위해 쇼핑몰이라는 다소 규모있는 시스템을 직접 구현하기로 결정하였다.

이외에도 쇼핑몰을 직접 구현하여 Kafka 동작 원리를 직접 구현하여 효과적으로 학습 및 적용할 수 있었고, 3초 이내에 의미있는 분석 결과를 전달하는 위한 쇼핑몰의 화면과 서버의 웹 소켓 연동도 수월하였다. 또한 고안한 패턴 및 알고리즘을 직접 적용하여 실제로 우리의 로직을 검증할 수 있었다.

현업에서 직접 쓰이는 아키텍처로 우리 넷의 생각을 구현하고 검증했다는 점에 큰 의미를 가진다.

### ㅇ 어떤 목적으로 Kafka를 선택했는가?

"빅데이터를 통한 실시간 사용자 추적을 위해서 적용"

이 프로젝트의 초기 기획은 '실시간으로 사용자 행동을 수집하고 분석한 후에 의미 있는 결과를 제공하고 싶다!' 라는 아주 막연한 아이디어에서 시작되었다. 이러한 추상적인 기획을 어떻게 구현할 수 있을지 고민하며 사례조사를 진행을 시작하였다.

사례조사 과정에서 많은 기업의 기술 블로그에서 Kafka가 있는 빅데이터 아키텍처를 접하게 되어, Kafka가 무엇이고 어떤 역할을 할 수 있을지에 대한 호기심을 가지게 되었고, 우리가 생각하는 기능을 Kafka로 구현할 수 있겠다는 흐름으로 프로젝트가 진행되었다.

Kafka를 프로젝트에 적용하기까지 많은 시행착오가 필요했다. 우선 쇼핑몰을 구현해야 하는 Spring Boot에 대해서도 모르는 부분이 너무나 많았다. 초기 세팅부터 배포까지, 데이터베이스를 관리하는 JPA 기술마저도 학습이 필요하였다.

Kafka 관련 국내 서적은 많지 않았고, 한빛 미디어에서 Kafka에 대한 책이

나온다는 소식을 듣고 베타 리더로 지원하여 이론을 습득하기도 했다. 하지만 결국 영어로 작성된 해외 자료를 보고 공부해야 했다.

찾아본 많은 예제들은 순수 Java로 작성이 되어 있었고, Spring에서 Kafka를 위한 프로젝트가 있는 상황이었다. 따라서 찾은 예제들을 통해 Java로 Kafka를 동작시키는 방법을 이해한 뒤 Spring 공식문서를 정독하여, Spring이 지원하는 Kafka 관련 추상화 모듈을 능숙하게 이용할 수 있게끔 추가 학습 과정을 거쳤다.

이와 같은 과정을 통해 Spring Boot에 Kafka Producer와 Consumer 로직을 구현할 수 있었다.


** 우리 프로젝트의 Kafka 관련 Log

### ㅇ 어떤 목적으로 ElasticSearch를 선택했는가?

"Kafka Topic에 대한 데이터를 저장하여 쉽게 질의하기 위해 적용"

ElasticSearch는 엘라스틱의 분산형 RESTful 검색 및 분석 엔진이고, ElasticSearch에 저장된 데이터 확인과 분석은 Kibana를 이용하여 편리하게 모니터링할 수 있다.

이 프로젝트에서 데이터 저장 단계에 ElasticSearch를 채택한 이유는 이를 통한 이 쇼핑몰의 검색 기능 구현과 Nosql 형태로의 데이터 저장 및 수월한 쿼리 작성 가능한 인터페이스 때문이다.

하지만 처음부터 Kafka 와 ElasticSearch 를 사용해야지, 하며 시작된 것은 아니었다. Spring Boot와 Kafka 를 이용한 데이터 수집의 일부를 구성한 후, 뒤늦게 Kafka Topic이 Database처럼 쿼리를 작성하여 데이터를 가져올 수 있는 형태가 아니라는 치명적인 사실을 발견하였다.

기획하고 있는 기능을 구현하기 위해 dnflsms Kafka SQL과 Elastic Search 등 언뜻 빠르게 조사해보기에 대안처럼 보이는 기술들을 적용을 검토해보았다. 이 중 Kakfa SQL은 우리가 원하는 성능으로 동작시키기 어렵다고 멘토님께 조언을 구하여 수월하게 시간을 단축시킬 수 있었다.

다행히 ElasticSearch는 검색엔진에 관심이 가졌던 시절에 잠시 살펴보았던

검색엔진 기술이었다. Kafka와 함께 사용할 수도 있다는 사실은 프로젝트를 진행하며 새로 알았지만, 평소 관심을 가져 알고있던 지식을 활용하여 빠르게 프로젝트에 적용시킬  수 있었다.

이러한 과정을 겪으며 왜 ElasticSearch를 사용하는지 몸소 깨달을 수 있었다.

### ㅇ 어떤 목적으로 추천 알고리즘: UserCF & ItemCF를 선택했는가?

"다양한 다른 사용자들의 유저보이스를 간접적으로 제공하기 위해 적용"

팀원 모두가 프로젝트를 구체화하면서 쇼핑몰 사용자들의 결정을 돕는 추천 시스템의 중요성을 느꼈다. 쇼핑몰이라는 플랫폼에서 고객은 수많은 물품들 속에서 자신이 원하는 무엇인가를 선택하고 결정하게 된다. 개별 사용자를 위한 추천 알고리즘으로 고객에게 추천할 만한 아이템을 쉽게 제공한다면 사용자에게 훨씬 편리한 쇼핑을 제공할 수 있고, 나아가 판매자 입장에게는 매출 증가를 제공해 줄 수 있다. 추천 시스템은 이처럼 서비스에 큰 역할을 하게 되는데 많은 알고리즘들이 연구되고 있다.

이 프로젝트에서는 추천 시스템의 주요 기술 중 UserCF와 ItemCF를 선택하였으며 우리는 UserCF와 ItemCF가 가진 장점들을 본 프로젝트에 활용한다면 큰 효과를 기대해 볼 수 있다고 여겼다. UserCF(User-based Collaborative Filtering) 방식은 나와 비슷한 성향을 가진 사용자를 기반으로 그 사람이 구매했거나 선호하는 상품을 추천하는 방식이다. 예를 들어 한 고객이 바게트, 통밀빵과 같은 베이커리류 상품을 주로 구매하였다면 그와 비슷한 성향을 지닌 사용자도 빵을 좋아할 것이라 예측하고 추천하는 방식이다. 비슷한 성향이라는 것을 계산할 때는 유사도 값을 사용할 수 있는데 피어슨 상관계수 등을 활용해서 구할 수 있으며 이 값으로 가장 유사하다고 생각하는 사람이나 물품을 선정할 수 있다. 이러한 사용자기반 협업 필터링 알고리즘은 아이템 자체에 많은 기준을 부여할 필요가 없다는 장점이 있다. 하지만 유사한 사람이라는 성향에는 명확한 기준을 부여하기 힘든 경우가 나타나기 때문에 본 프로젝트에서는 UserCF 추천시스템과 함께 ItemCF도 활용하여 전반적으로 더 높은 수준의 서비스를 제공하고자 하였다. ItemCF(Item-based Collaborative Filtering) 방식은 아이템 자체의 유사도를 기반으로 즉, 사용자가 구매했었던 물품을 기반으로 그 상품과 유사한 성향을 가진 물품을 사용자에게 추천을 제공하는 방식이다. 예를 들어 한 사용자가 분유나 이유식을 주로 구매하였다면 그 사용자에게 유사한 유아식을 추천해 주는 것이다. 이러한 아이템 기반 협업 필터링에서는 코사인유사도 등을 활용하여 유사도를 계산하게 되는데 비교적 메모리에 부담도 적게 주고 정확도도 높일 수 있다는 장점이 있다. 이러한 두 가지 협업 필터링을 사용하여 본 프로젝트에서는 목표로 하는 추천 시스템을 제작해 나가고자 한다.

### ㅇ 어떤 과정으로 Web Socket를 선택했는가?

"3초 이내의 빠른 피드백을 제공하기 위해 적용"

이 프로젝트에서 수집한 수많은 데이터들은 UserCF와 ItemCF 등의 추천 알고리즘을 거쳐 수많은 요소가 복잡한 관계를 이루는 분석 결과로 가공된다. 우리는 이러한 결과를 쇼핑몰이라는 플랫폼에서 사용자의 행동을 구매로 유도하기 위해 다양한 상황에서 3초 이내에 행동에 피드백을 제공하고자 하였다.

이 과정에서 우리는 분석 결과를 제공하기 위한 수단으로 브라우저에서 요청 후 서버 응답을 받는 Ajax 기술과 언제든 서버에서도 브라우저에 결과를 전달할 수 있는 Web Socket 기술 중에서 고민한 끝에, 많은 피드백을 전달하기 위해 Web Socket을 적용하기로 결정하였다.

적용 초기에는 Web Socket이라는 기술이 동작하는 흐름에 대한 명확한 이해가 없던 상황에서 Spring Boot로 구현하기 위해 참고할 자료들이 많지 않다는 상황에 맞닥뜨리게 되었다.

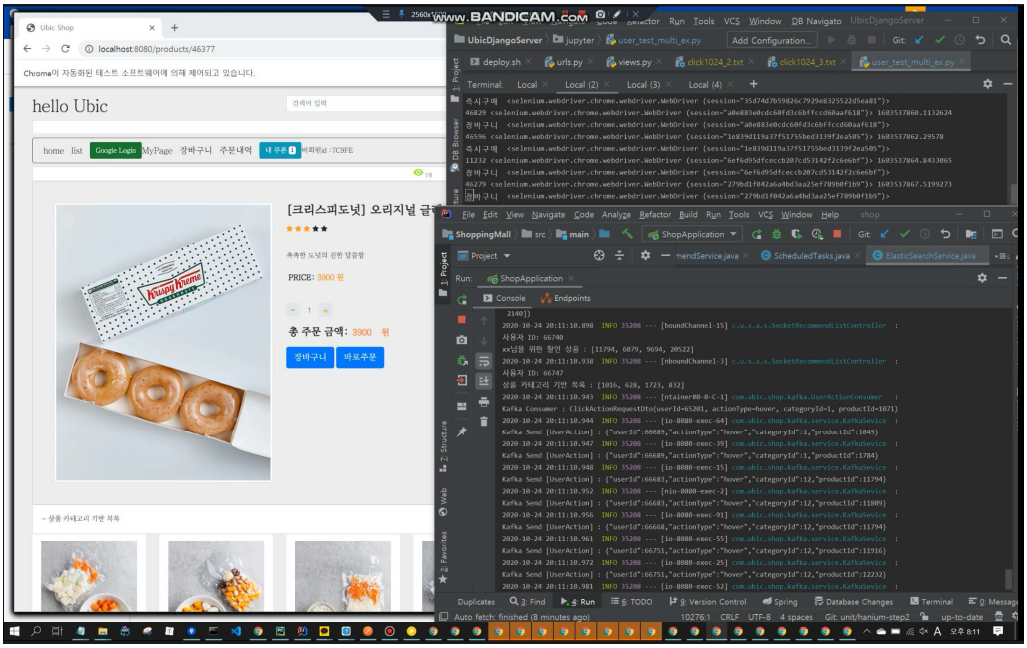
고민 끝에 웹 소켓의 활용 방법을 정확하게 이해 후 적용하여가 언제든 기획대로 구현할 수 있을 것이라 판단했고, 백엔드 프레임워크의 공통되는 지식에 익숙한 팀원이 socket.io를 이용하여 웹 소켓을 구현한 node.js 프로젝트를 공식문서를 참여하고 분석하였다. 그리고 WebSocket을 지원하는 Spring 프로젝트의 공식문서를 학습하여, 우리의 알고리즘을 적용한 Spring Boot의 기능을 구현하였던 어마어마한 시행착오가 있었다.

이런 과정으로 적용한 소켓 기술을 통해 우리 프로젝트는 쿠폰과 추천목록을 3초 이내로 갱신할 수 있게 되었다.

### ㅇ 가상 사용자 시뮬레이션

본 프로젝트는  로컬 환경에서 python selenium 과 asyncio를 이용하여 비지니스 환경에서 실제로 하는것처럼 시뮬레이션을 수행하였다. 이 상황은 30명이 실제 동작하는것과 동일하게 환경을 구성하였다.

총 30개의 브라우저를 통해 상품 상세 화면을 랜덤 방문하고, 랜덤으로 장바구니에 담거나 바로 주문하는 기능을 테스트하였고, 상품 상세 화면 하단의 추천목록이 우리가 작성한 로직대로 잘 동작함을 확인했다.




** 화면 구성 : 좌측은 작성한 프로그램으로 제어하는 가상 브라우저,
우측 상단은 가상 사용자 프로그램 동작 로그, 우측 하단은 쇼핑몰 동작 로그