|  |  |
| --- | --- |
| 미반납 기자재 없음 확인란  (공기실 조교 작성) | 확인자 성명:  확인자 서명:  확인 일자: |

|  |
| --- |
| **2021학년도 1학기 졸업작품 결과 보고서** |
| **속닥속닥 책 추천 애플리케이션** |
| **주제: 빅데이터 분산 처리를 위한 파이프라인 구축과 추천 서비스** |
|  |

|  |  |
| --- | --- |
| **팀명 :** | 머글머글 |
| **팀원 :** | 이름 / 학번 |
| **제출자 :** | 이름 / 학번 (서명) |
| **제출일 :** |  |
| **지도교수 :** | (서명) |

**1. 서론**

**1.1. 개발 동기 및 목표**

코로나 시대에 들어서면서, 집안에서 여가시간을 보내는 시간이 증가함에 따라, 독서의 수요가 증가하게 되었다. 넷플릭스, 유투브 등의 영상 스트리밍 서비스에 대한 수요가 가장 크게 증가한 측면이 있지만, 집안에서 책에 자연스레 손이 가고 조금 더 의미 있게 시간을 보내고자 한 사람들을 독서를 통해 여가시간을 보내고자 하였다. 또한, 이 기회를 통해 독서 습관을 기르고자 한 사람들도 있다는 사실을 알게 되었다.

하지만, 기존에 독서습관 혹은 관심 도서 분야가 없었던 사람들에게 자신에게 알맞은 책을 찾아보는 것은 하나의 과제로 주어졌다. 너무나도 방대한 양의 책들이 존재하고, 카테고리, 시대, 분야 별 도서는 무수히 많기에 책 하나를 선정해 독서를 시작하는 것 조차 어려움을 겪는 사람들도 많은 것으로 파악되었다. 위 사실을 파악하면서, 사람들이 어떤 책이 좋은 책인지, 어떤 방식으로 책을 골라하야 하는지에 대한 어려움을 겪는 사람들도 많다는 것을 알게 되었다.

이러한 상황을 파악하면서, 어떻게 하면 독서를 처음 시작하는 이들이 쉽게 도서를 선택하여 독서를 시작할 수 있을지, 그리고 어떻게 하면 그들에게 유익한 독서 습관을 만들어 줄 수 있을지에 대한 고민을 하게 되었다. 그래서, 이들을 위한 책 추천 시스템, 그리고 올바른 독서 방법을 통해 독서의 효율을 높일 수 있는 기능들을 포함한 어플리케이션을 개발하고자 하였다.

**1.2. 프로젝트 개요**

독서를 시작하는 사람들에게 상황에 따른 선호도에 따른 맞춤형 도서를 추천하고, 독서 초보자들을 위한 독서법 7단계를 기반으로 이에 맞는 기능들을 포함한 어플리케이션을 개발하고자 하였다.

먼저, 도서라는 빅데이터 분석을 위해, 네이버 책의 도서 정보들을 Crawling하여 책 데이터를 수집하고자 하였다. 이후, 빅데이터 분석에 가장 많이 사용되는 Spark를 통해 대용량의 책 데이터를 처리하고, Milb를 이용하여 맞춤형 책 추천 알고리즘을 개발하고자 하였다.

어플리케이션 내에서 독서 초보자를 위한 독서법 7단계의 1,2 단계인 ‘책과 친해지기’, ‘독서 목표설정과 동기부여하기’를 실천할 수 있는 Book MBTI 기능을 구현하고자 하였다. 최근 가장 인기를 끌고 있는 하나의 컨텐츠인 MBYI를 기반으로 책과 관련된 성향을 파악함에 있어 책에 대한 흥미를 유발하고자 하였다.

다음 단계인 3단계, ‘속독보다는 정독하는 습관’을 위한 타이머 기능을 추가하고자 하였다. 속독을 하게 된다면 자신이 책을 읽고 있는 시간이 생각보다 빠르다는 것을 분명 스스로도 인식하고 있을 것이다. 타이머 기능을 통해, 정독을 할 수 있도록 지정된 적절한 시간 동안 그에 맞는 속도로 정독을 할 수 있도록 유도하고자 하였다.

다음, 4단계인 ‘자기 수준에 맞는 책을 시작하기’ 단계를 실현하기 위해, 어플리케이션의 메인 기능인 맞춤형 책 추천 기능을 구현하고자 하였다. 빅데이터 처리 시스템인 spark를 기반으로 spark milb를 통해 맞춤형 도서 추천 알고리즘을 구현하고, 이를 기반으로 유저들에게 맞춤형 책을 추천하고자 하였다.

또한, 5단계인 ‘중요한 부분은 필사하고 상상력은 메모하기’와 7단계 ‘책을 모두 읽은 후에도 책 내용을 상기하기’실현을 위해, 메모자 기능을 구현하고자 하였다. 필사하기, 메모하기, 그리고 책 내용 상기하기를 실현 할 수 있도록 자신이 읽은 도서의 내용을 간단한 메모장 기록을 통해 실현 할 수 있도록 도움을 주고자 하였다.

마지막으로, 6단계인 ‘한꺼번에 오랜시간 일기보단 조금씩 자주 읽기’ 단계 실현을 돕기 위해 최근 읽은 책 기능 저장하기 기능을 구현하고자 하였다. 최근 읽은 책을 마이페이지를 통해 확인하고, 읽은 페이지를 기록할 수 있음으로서, 추후 책을 다시 읽게 될 때 도움이 되고자 하였다.

**1.3. 1.1. 개발 언어 및 환경**

개발 언어와 환경은 크게 client, user sever, data server, 그리고 platform server로 나뉘게 된다. 유저에게 제공되는 어플리케이션 개발을 위해 자바 기반의 안드로이드 프로그램을 활용하였다. 유저 데이터는 전반적으로 firebase 서버를 통해 관리된다. 빅데이터를 처리하고, 알고리즘을 적용하여 맞춤형 정보를 추출해내기 위한 data server는 docker 환경 기반으로 구축하였다. docker 환경 안에서 spark, spark Mlib, 그리고 하둡 데이터 파일시스템을 활용하게 된다. docker환경과 user server 연동을 위해 google cloud platform을 활용하여 데이터를 주고 받을 수 있다.

**2. 본론**

**2.1. 사용기술 및 특징**

**2.1.1.** 빅데이터 분석을 이용한 책 줄거리 분석

: Crawling을 이용한 책 데이터를 수집하였다. 그 후 Spark 를 이용한 대용량의 책 데이터를 처리하였다. 처리한 책 데이터를 이용하여 Spark Mllib 을 이용한 맞춤형 책 추천 알고리즘을 구현하였다.

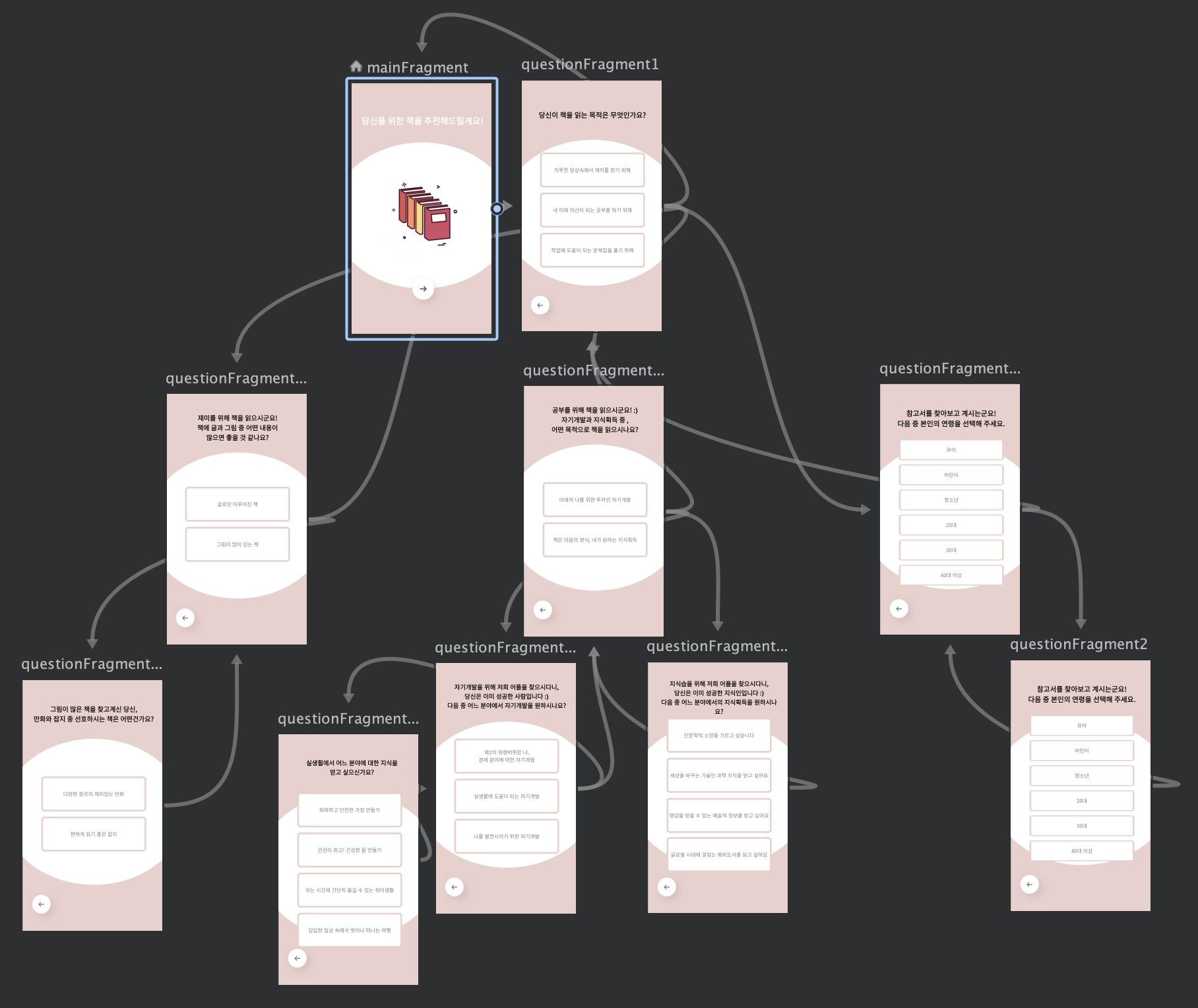
**2.1.2.** 사용자의 선호하는 책 내용을 기반으로 추천 알고리즘에 의한 맞춤 책 추천

먼저 데이터 수집을 위해 Selenium을 이용하여 ‘네이버 책’에서 책 정보 Crawling을 진행하였다. 또한, 앱 실행 초기에 사용자로부터 선호하는 책 카테고리에 대한 최소한의 정보를 입력받았다.

Chrome 자동제어를 이용하여 Naver 책 카테고리별 best top 100을 크롤링하였다.



또한, Android Studio 의 navigation 기능을 이용하여 앱 실행 초기에 사용자의 책에 대한 선호 정보를 입력받았다.



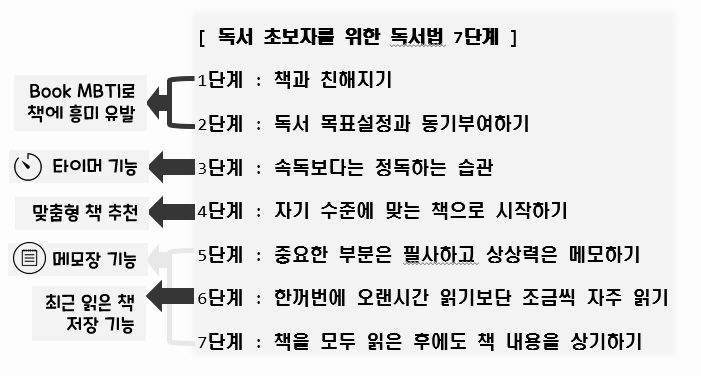
네이버 책 정보에 있는 책 소개 부분을 크롤링하여 책 줄거리 데이터를 얻었다. 이를 기준으로 책들 간의 유사도를 계산하여 신규 사용자에게 선호하는 책 정보를 받고, 해당 책과 줄거리의 유사도가 높은 책을 추천해준다면 정보가 부족한 사용자에게도 의미있는 추천을 해줄 수 있을 것이라 기대했다. 따라서 줄거리에 있는 단어들의 중요도를 수치화하기 위해 문서 내 등장하는 단어의 빈도수를 기준으로 중요도를 계산하는 tf-idf 알고리즘과, 문서들 간 비슷한 단어가 얼마나 공통적으로 사용되었는지를 기준으로 유사도를 측정하는 코사인 유사도 측정 방식으로 유사도가 높은 책들을 계산했다. 네이버 책 정보의 경우 책들마다 소개글의 길이가 저마다 매우 상이했기 때문에, 벡터의 방향, 즉 패턴에 초점을 두어 유사도를 구하는 코사인 유사도 알고리즘을 통해 해당 문제를 완화한 유사도를 계산했다.

단어 행렬을 사용해 문서에서 각 단어의 빈도수를 수치화하고, 그 중요도를 파악하는 방식이기 때문에, 불용어와 특수문자 등을 제거하는 전처리 단계가 필요하다. 이를 위해 각 책의 줄거리 문서를 토큰화 한 후 RanksNL 에서 제공하는 한국어 불용어 리스트를 사용하여 조사, 접속사 및 의미없는 단어들을 제거하고 명사와 형용사만을 뽑아냈다. 또한 tf-idf 알고리즘을 적용하기 전 줄거리가 없는 책들에 대해 모두 0으로 채워 null값을 제거했다.

이후 tf-idf 알고리즘을 사용해 전처리로 얻은 각 단어들의 중요도를 계산했다. tf-idf 알고리즘은 각 문서에서 각 단어의 빈도수를 계산한 행렬 값에 여러 문서에서 등장한 단어의 가중치를 낮추는 역할을 하는 idf값을 곱하는 방식이다. 따라서 특정 문서에서만 자주 등장하는 단어의 가중치를 높게 두어 중요한 단어라고 판단할 수 있다.

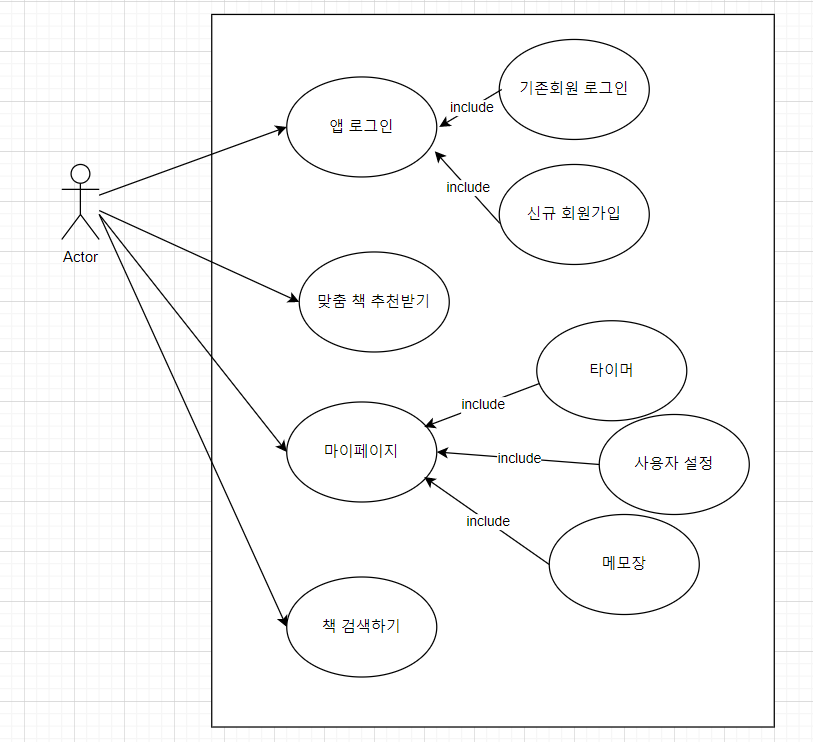
줄거리 문서 내의 단어들을 수치화하여 각 책을 숫자로 표현한 tf-idf 값을 기준으로 코사인 유사도를 이용해 문서 간의 유사도를 구했다. 코사인 유사도를 이용해 문서의 길이가 서로 매우 상이한 책 집합에 대해 비슷한 단어들이 얼마나 많이 등장하는지를 벡터화하여 벡터의 방향이 유사한 책을 비슷한 내용을 갖는 책이라고 판단하여 책끼리의 유사도를 구할 수 있었다. 이후 계산한 유사도와 초기에 navigator를 사용하여 사용자에게 받은 선호하는 책 정보를 기준으로 사용자가 기존에 흥미있게 읽었던 책과 유사한 책 목록을 추천해준다.

**2.1.3.** 올바른 독서를 위해, ‘독서법 7단계’를 도와줄 수 있는 기능 개발

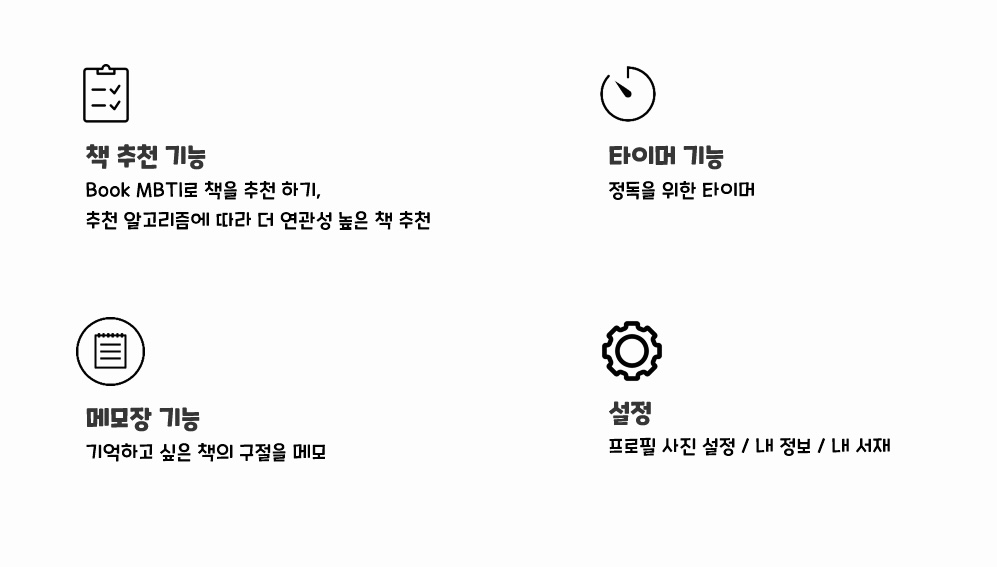


**2.2 요구사항 분석**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 요구 사항 ID | 요구사항명 | 기능ID | 기능명 | 세부사항 | 예외사항 | 유즈케이스 ID |
| A01 | 로그인 | A01\_B01 | 기존회원 로그인 | 회원가입이 되어있는 사용자는 로그인 후 앱을 이용한다. |  | UC\_A01\_B01 |
| 회원가입 | A01\_B02 | 신규회원 가입 | 신규회원은 하단의 신규회원가입 버튼을 누르면 신규회원가입 activity로 넘어간다. | 신규회원 | UC\_A01\_B02 |
| A02 | 책 MBTI | A02\_B01 | 맞춤 책 추천받기 | 책 MBTI검사를 통해 사용자의 선호 책 성향을 파악한 후 사용자에게 맞는 책을 추천해준다. | 책 선호 검사실행 | UC\_A02\_B01 |
| A03 | 마이페이지 | A03\_B01 | 타이머 | 정독을 위한 타이머 기능이 있다. 사용자가 타이머를 설정하면, 시간에 맞춰 알람이 울린다. |  | UC\_A03\_B01 |
| A03\_B02 | 메모장 | 기억하고 싶은 책의 구절을 메모할 수 있는 메모장 기능이 있다. 사용자가 메모를 한 후 저장하면, 메모가 저장되고 언제든지 다시 메모를 확인할 수 있다. |  | UC\_A03\_B02 |
| A03\_B03 | 사용자 설정 | 프로필 사진 설정 / 내 정보 / 내 서재 기능이 있다. 내 서재에는 내가 관심있어 하는 책이 들어가있다. |  | UC\_A03\_B03 |
| A04 | 책 검색 | A04\_B01 | 책  검색하기 | 사용자는 전체 책 목록 중 원하는 책을 검색을 통해 찾을 수 있다 |  | UC\_A04\_B01 |



**2.3. 기능 설명 및 구조도**



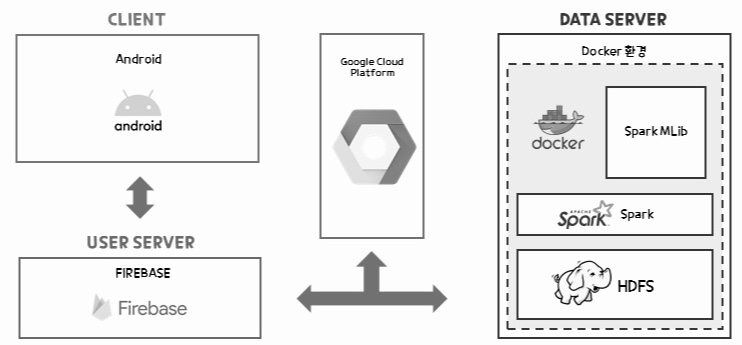
기능 1) 책 추천 기능 : Book MBTI로 책을 추천해준다. 추천 알고리즘에 따라 더 연관성 높은 책 추천을 진행한다.

기능 2) 타이머 기능 : 정독을 위한 타이머 기능이 있다. 사용자가 타이머를 설정하면, 시간에 맞춰 알람이 울린다.

기능 3) 메모장 기능 : 기억하고 싶은 책의 구절을 메모할 수 있는 메모장 기능이 있다. 사용자가 메모를 한 후 저장하면, 메모가 저장되고 언제든지 다시 메모를 확인할 수 있다.

기능 4) 설정 : 프로필 사진 설정 / 내 정보 / 내 서재 기능이 있다. 내 서재에는 내가 관심있어 하는 책이 들어가있다.

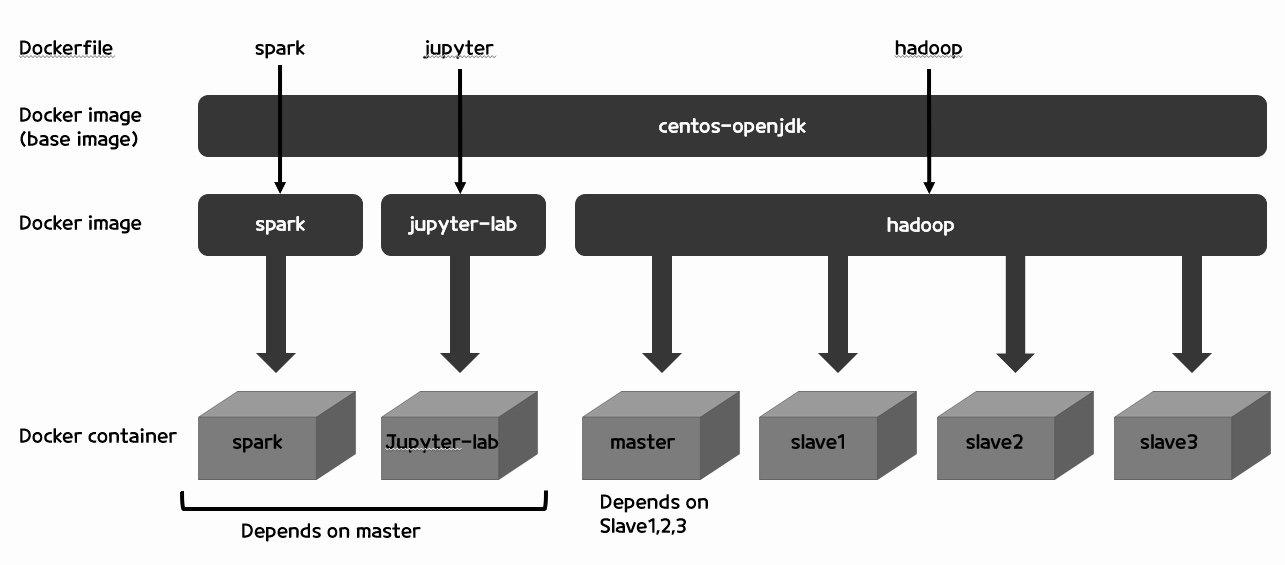
-구조도



Client는 안드로이드 스튜디오를 이용하여 앱 개발을 하였다. 사용자 서버는 firebase를 이용하여 구축하였고, 안드로이드 스튜디오에서는 firebase와의 통신으로 사용자의 정보를 받아오고, 입력받은 사용자의 정보를 firebase에 저장한다.

Data server에서는 크롤링한 데이터를 HDFS에 저장한 후 spark MLlib를 이용하여 데이터를 분석하였다. 도커 compose로 spark와 jupyter 컨테이너를 연결해서 빌드한 후 크롤링한 데이터를 가져와서 모델링하였다.

-Hadoop 시스템 구조



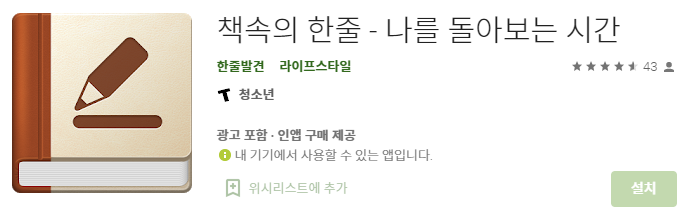
단순 파이썬으로 모델 연산을 수행하기보다 spark 엔진을 사용해 in-memory 기반으로 빠르게 연산을 수행하고, 더 나아가 연산을 scale-out하기 적합한 환경을 구축하는 것을 목표로 했다. 따라서 Hadoop의 hdfs와 spark, mysql을 사용해 네이버 책에서 스크래핑한 데이터를 수집 및 적재하고 모델을 적용해 데이터를 가공해 적재하는 백단 DB를 구축했다. 이때 hdfs, spark, mysql을 각각 docker 컨테이너로 패키징하여 각 애플리케이션을 일관되고 격리된 환경에서 운영하고, 컨테이너 클러스터를 docker-compose를 사용하여 오케스트레이션했다. 다음은 도커 가상화 기술을 사용하게 된 배경이다.

1. HDFS는 Linux os와 JDK 1.6 이상의 버전 시스템 요구 사항을 만족해야 구축할 수 있기 때문에 가상 머신이 필요했다. 기존 VM는 hypervisor를 통해 기동되어 host os의 리소스에 가상으로 엑세스할 수 있는 guest os를 실행시키는 방식이므로 오버 헤드가 크고 느리며 os 로딩을 위해 메모리를 부가적으로 더 사용한다. 따라서 빠르고 경량화된 가상화 기술이 필요했고, 도커는 도커 엔진을 통해 실행되어 host os 위에 image를 배포하기만 하면 되기 때문에 이러한 요구사항을 만족시킬 수 있다.

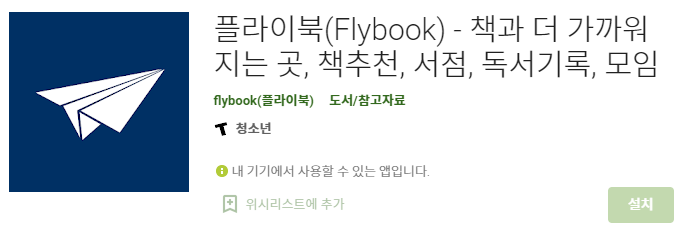
2. 도커를 사용하면 실행에 필요한 서비스를 모두 패키지화하고 분리하기 때문에, 각 애플리케이션을 다양한 환경에서 동일하게 구축, 배포, 복사할 수 있다. 또한 컨테이너를 재사용하고, 확장하기 용이하기 때문에 분산 환경 및 클라우드 환경에 매우 적합하다.

spark의 경우 데이터를 디스크가 아닌 메모리에 로드하여 메모리 기반으로 연산을 수행해 빠르고, 분산 병렬 처리가 가능한 프레임워크다. 따라서 pyspark의 DataFrame 데이터api 와 mllib 를 사용하여 데이터에 추천 모델을 적용한 후 mysql에 저장했다.

**2.4. 유사기능 프로그램들과의 비교**



책을 다른 사용자에 의해서만 추천 받을 수 있는 프로그램이다. 따라서 추천 요청 글에 댓글이 안 달리면 추천을 받지 못할 수도 있다는 단점이 존재한다.



책을 카테고리에 따른 분류로만 추천 받는다. 따라서 처음에 선택한 카테고리 외에 있는 카테고리의 책은 추천 받지 못하게 된다.

**3. 결론**

**3.1. 개발 중 발생한 문제점 및 해결책**

**3.1.1. 추천 모델링 기법 발전 방향**

크롤링한 책의 줄거리를 단어의 빈도수를 기준으로 벡터화하여 문서 간의 유사도를 구하는 방식으로 사용자에게 책을 추천했다. 이 경우 해당 단어가 갖는 의미를 반영할 수 없다는 점에서 사용자가 추천된 책을 선호하지 않을 수 있다.

책 정보를 크롤링할 때 책의 카테고리도 구체적으로 크롤링해서 저장해두었기 때문에, 이 카테고리를 적극적으로 반영하면서도 단순 단어가 나타난 빈도수만을 기준으로 문서를 벡터화하는 것이 아닌 단어의 의미도 반영해서 문서를 표현하고자 했다. 책의 카테고리와 단어의 의미를 결합하여 문서를 벡터화해야 신규 사용자에게 좀 더 유의미한 추천을 해줄 수 있기 때문이다.

따라서 기존 tf-idf 알고리즘으로 얻은 단어 행렬에 추가적으로 LSA(Latent Semantic Analysis) 모델링을 적용한다면 단어의 잠재된 의미까지 고려하여 문서 집합을 벡터화할 수 있다. LSA 알고리즘은 특이값 분해를 통해 행렬을 3개의 행렬 곱으로 분해한 후 차원을 축소하여 설명력이 높은 정보만 남기는 방식이다. 불필요한 토픽(문서 집합의 추상적인 주제)을 제거하고 문서의 잠재 의미, 즉 주제와 이에 상응하는 중요한 단어만 얻어낼 수 있다.

또한 초기 사용자에 대한 정보가 없을 때는 문서 간 유사도만을 기준으로 추천했으나 사용자의 책에 대한 리뷰나 평점 데이터를 남긴다면, 해당 데이터를 활용하여 matix factorization 모델을 사용하는 등 좀 더 개인화된 추천을 제공할 수 있을 것이다.

**3.1.2. 정기적인 크롤러 필요**

책은 하루에서 수십권씩 출판되고 인기를 끄는 책들도 계속 달라지기 때문에, 자체적으로 책 정보를 입력받는 시스템을 구축하지 않는 이상 주기적으로 크롤러를 실행시켜 책 데이터를 업데이트 해야한다.

현재는 프로젝트를 진행하는 기간동안 크롤링한 책 데이터로 모델링했기 때문에 데이터를 업데이트해서 모델에 반영하는 일이 없었다. 따라서 LSA의 새로운 데이터가 들어왔을 때 계산을 처음부터 다시 해야하는 단점이 두각되지 않았지만 데이터가 점점 더 커진다면 Word2Vec 등 단어의 의미를 벡터화화여 저장해둘 수 있는 모델을 사용하여 이를 극복할 수 있을 것이다.

**3-2. 기대효과 및 발전 방향**

**3.2.1. 기대효과**

프로젝트에서 사용한 하둡과 스파크는 분산 저장, 처리에 최적화된 오픈소스 도구다. 따라서 데이터가 기가, 테라 단위로 들어올 때에도 빠르고 효율적으로 핸들링할 수 있으며 운영 비용 측면에서도 매우 뛰어나다. 책 정보나 로그 등 영구적으로 저장해야 할 데이터, spark로 배치처리를 거친 대용량 데이터를 HDFS에 분산 저장할 수 있으며 spark로 메모리 위에서 데이터를 빠르게 가공 및 머신러닝 모델을 적용할 수 있다.

사용자가 속닥속닥 앱에 선호하는 책 정보를 저장해둘 수 있기 때문에, 이를 이용해 사용자가 앱을 이용하면서 선호하는 책, 관심있는 책 정보를 더 남길수록 개인화된 추천을 제공할 수 있다. 현재는 사용자에 대한 데이터가 부족한 문제를 보완하고자 아이템끼리의 유사도만을 기준으로 한 추천 알고리즘을 사용한다. 하지만 사용자마다 선호하는 책 데이터를 수집하여 적재해둔다면 추후 사용자와 책들 간의 상호작용과 다른 사용자들과의 유사성을 기반으로 Collaborative Filtering 추천 모델을 사용하여 개인 맞춤형으로 책을 추천해줄 수 있다.

**3.2.2. 발전방향**

하둡 에코시스템에 최적화된 관계형 데이터베이스를 수집, 적재할 수 있는 도구를 mysql 과 HDFS사이에 두어 대용량 데이터를 처리하는데에 더 적합한 파이프라인으로 발전시킬 수 있다. 예를 들어 Sqoop은 클러스터 모드로 분산 저장, 처리 엔진을 사용할 수 있다. 하둡의 MapReduce로 원격지 장비의 리소스를 활용하여 병렬 분산 처리가 가능하고 원격 디스크에 저장할 수 있으므로, 데이터 수집 도구로 더 적절하다.

현재 하둡, 스파크 모두 standalone 환경으로 구축하였다. 이는 각 요소가 모두 로컬 리소스를 사용하기 때문에 분산 처리 효과를 보지 못한다. 그러나 애플리케이션을 모두 컨테이너 환경으로 패키지화했기 때문에, 클라우드 환경에서 리소스를 추가적으로 발급받거나 실제로 노드를 추가했을 때 해당 파이프라인을 동일한 환경을 유지하며 배포하기 용이하다. 또한 해당 데이터 파이프라인을 Airflow 등 워크플로우 관리 도구를 사용하여 자동화하거나, 쿠버네티스를 사용하여 컨테이너 오케스트레이션을 자동화할 수 있는 등 확장 가능성이 매우 높다.