REVIEW REVOLUTION

design specification



|  |  |
| --- | --- |
| Student Number | Name |
| 2017314380 | 윤성경 |
| 2014311058 | 김진태 |
| 2014314650 | 유종현 |
| 2013310608 | 정창호 |

Contents

[1. Preface](#_Toc24553745)

[1.1. Objectives](#_Toc24553746)

[1.2. Readership](#_Toc24553747)

[1.3. Document Structure](#_Toc24553748)

[A. Introduction](#_Toc24553749)

[B. System Architecture](#_Toc24553750)

[C. Protocol Design](#_Toc24553751)

[D. Database Design](#_Toc24553752)

[E. Testing Plan](#_Toc24553753)

[F. Development Plan](#_Toc24553754)

[G. Index](#_Toc24553755)

[2. Introduction](#_Toc24553756)

[2.1. Objectives](#_Toc24553757)

[2.2. Applied Diagram](#_Toc24553758)

[A. UML](#_Toc24553759)

[B. Class Diagram](#_Toc24553760)

[C. Sequence Diagram](#_Toc24553761)

[D. State Diagram](#_Toc24553762)

[E. Package Diagram](#_Toc24553763)

[F. ER Diagram](#_Toc24553764)

[2.3. Applied Tool](#_Toc24553765)

[A. Draw.io](#_Toc24553766)

[B. PowerPoint](#_Toc24553767)

[C. ERDPlus](#_Toc24553768)

[2.4. Project Scope](#_Toc24553769)

[3. System Architecture – Overall](#_Toc24553770)

[3.1. Objectives](#_Toc24553771)

[3.2. System Organization](#_Toc24553772)

[A. Overall System Architecture](#_Toc24553773)

[B. Frontend Architecutre](#_Toc24553774)

[C. Backend Architecture](#_Toc24553775)

[3.3. Overall System State Diagram](#_Toc24553776)

[4. System Architecture – Frontend](#_Toc24553777)

[4.1. Objective](#_Toc24553778)

[4.2. Overall Architecture](#_Toc24553779)

[4.3. Subcomponents](#_Toc24553780)

[A. Customized Search Result](#_Toc24553781)

[B. Item Page](#_Toc24553782)

[C. Keyword Visualizer](#_Toc24553783)

[D. Review Page](#_Toc24553784)

[E. Review Post](#_Toc24553785)

[F. User Page](#_Toc24553786)

[5. System Architecture – Backend](#_Toc24553787)

[5.1. Objectives](#_Toc24553788)

[5.2. Overall Architecture](#_Toc24553789)

[5.3. Subcomponents](#_Toc24553790)

[A. Login](#_Toc24553791)

[B. Review Analysis](#_Toc24553792)

[C. Similar Item Finder](#_Toc24553793)

[D. 상품 정보 제공](#_Toc24553794)

[E. 개인정보 변경](#_Toc24553795)

[6. Protocol Design](#_Toc24553796)

[6.1. Objectives](#_Toc24553797)

[6.2. REST API](#_Toc24553798)

[A. Item](#_Toc24553799)

[B. Review](#_Toc24553800)

[C. User](#_Toc24553801)

[7. Database Design](#_Toc24553802)

[7.1. Objectives](#_Toc24553804)

[7.2. ER Diagram](#_Toc24553805)

[A. Overall Entities](#_Toc24553806)

[B. User](#_Toc24553807)

[C. Item](#_Toc24553808)

[D. Review](#_Toc24553809)

[E. Keyword](#_Toc24553810)

[7.3. JSON](#_Toc24553811)

[A. User](#_Toc24553812)

[B. Item](#_Toc24553813)

[C. Review](#_Toc24553814)

[D. Keyword **오류! 책갈피가 정의되어 있지 않습니다.**](#_Toc24553815)

[8. Testing Plan](#_Toc24553816)

[8.1. Objectives](#_Toc24553818)

[8.2. Testing Policy](#_Toc24553819)

[A. Development Test](#_Toc24553820)

[B. Release Test](#_Toc24553821)

[9. Development Plan](#_Toc24553822)

[9.1. Objectives](#_Toc24553824)

[9.2. Overall Framework Architecture](#_Toc24553825)

[9.3. Frontend Environment](#_Toc24553826)

[A. Vue.js](#_Toc24553827)

[9.4. Backend Environment](#_Toc24553828)

[A. Node.js](#_Toc24553829)

[B. Express](#_Toc24553830)

[C. Firebase](#_Toc24553831)

[9.5. Schedule](#_Toc24553832)

[10. Index](#_Toc24553833)

[10.1. Tables](#_Toc24553834)

[10.2. Figures](#_Toc24553835)

[10.3. Diagrams](#_Toc24553836)

[11. References](#_Toc24553837)

# Preface

## Objectives

이 장에서는 예상 독자를 정의하고, 각 장의 내용을 요약한다. 또한 현재 version과 이전 version의 차이점에 대해 설명한다. 하지만 본 문서는 초기 버전이기 때문에 이 부분을 생략한다.

## Readership

본 문서는 다양한 독자에게 읽힐 것이라고 상정하고 있다. 따라서 각 부분을 서술하는 데 있어 어떠한 독자층을 상정하고 있는지를 설명한다.

## Document Structure

### Introduction

본 문서를 서술하는 데 사용된 다양한 다이어그램과 표현 도구들에 대해 설명하고, 본 소프트웨어 프로젝트가 다루는 시스템의 범위에 대해 서술한다.

### System Architecture

시스템과 각 서브시스템의 구조를 개괄적으로 기술하고, 시스템의 전체 기능이 각 서브시스템과 하드웨어에 어떻게 할당되었는지 설명한다.

### Protocol Design

기본적으로 시스템의 각 컴포넌트, 특히 프론트엔드 시스템과 백엔드 시스템간의 상호작용을 규정하는 인터페이스와 프로토콜을 어떻게 구성하는지에 대해 기술하고, 해당 인터페이스가 어떤 기술에 기반해 있는지 설명한다.

### Database Design

Requirements 문서에서 규정된 데이터베이스 요구 사항을 기반으로, 각 데이터 엔티티의 속성과 관계를 ER diagram을 통해 표현하고 최종적으로 Relational Schema, SQL DDL를 작성한다.

### Testing Plan

미리 작성된 Test를 이용해, verification 과 validation을 시행한다. 이 Test 작성에 대한 계획을 설명한다.

### Development Plan

시스템을 구현하는 데 필요한 개발 도구와 프로그래밍 언어, 라이브러리 등의 개발 환경에 대해 설명하고, 시스템 개발 일정을 기술한다.

### Index

본 문서에서 사용된 그림, 표, 다이어그램 등의 색인을 기술한다.

# Introduction

## Objectives

이 장에서는 본 시스템이 등장하게 된 배경과 필요성에 대해 설명하고, 시스템의 대략적인 기능에 대해 설명한다. 또한 해당 시스템을 개발함으로써 기대되는 효과에 대해 설명한다

## Applied Diagram

### UML

UML 이란 범용적으로 쓰이는 모델링 언어로, 서로 다른 시스템의 특성을 나타내기 위해 사용하는 기법들을 담고 있다. 이 기법들은 각각 연관관계, 과정 또는 결과 등을 담고 있으며, 각 기법들에 대한 설명은 다음 섹션에서 찾을 수 있다. 우리는 서비스를 설명하고 가시화하기 위해 UML을 이 문서 전반적으로 사용하였음을 밝힌다.

UML은 단순히 한가지 모델링 기법만을 설명하는 것이 아닌 여러가지 기법과 다이어그램들로 이루어져 있으므로, 각 기법을 설명하는 것은 이 문서를 읽게 될 독자와 개발자 그리고 사용자가 원활한 소통을 하기 위해 중요하다. 이 문서에서 설명할 UML 다이어그램은 Class Diagram, Sequence Diagram, State Diagram, Package Diagram, ER Diagram이다. Use Case Diagram은 requirement specification에서 설명했기 때문에 제외했다.

### Class Diagram

![스크린샷이(가) 표시된 사진  자동 생성된 설명](data:application/octet-stream;base64,)

Figure 1 : Example of Class Diagram

Class Diagram은 시스템을 객체와 객체 간의 연관성을 통해 표현하는 diagram이다. Class Diagram은 각 객체의 attribute와 method를 통해 객체를 설명하며, 객체 간의 의존성 또는 계층적 구조를 화살표 등의 기호를 통해 객체 간의 연관성을 표현한다.

위 다이어그램에서 보는 것과 같이 객체는 큰 사각형의 형태로 표현하며, 그 안의 구조는 객체의 이름, 각 객체가 지니는 값인 attribute, 각 객체가 수행하는 행동인 method로 형성되어 있다.

### Sequence Diagram

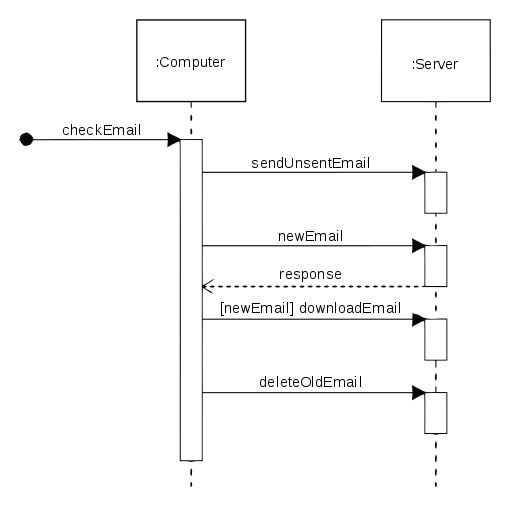


Figure 2 : Example of Sequence Diagram

Sequence diagram은 actor의 행동에 대해 actor와 object간의 상호작용이 어떻게 이루어지는지를 표기한 다이어그램이다. 이 다이어그램을 통해 상호작용이 어떤 object간에 일어나는지, 어떤 순서로 일어나는지를 볼 수 있게 된다. Sequence Diagram에서 순서는 화살표의 형태로 알기 쉽게 표현된다.

### State Diagram

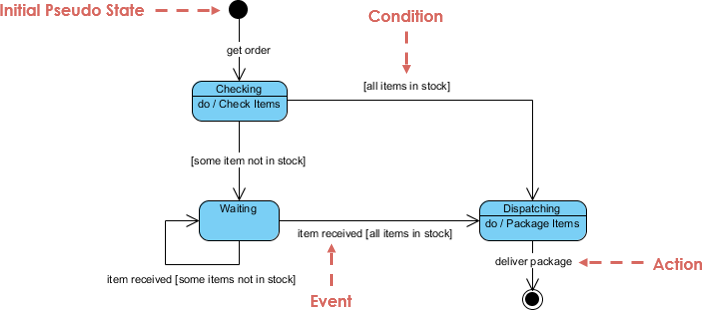


Figure 3 : Example of State Diagram

State Diagram은 시스템의 상태(State)의 변화를 통해 시스템을 설명한 다이어그램이다. 각각의 event에 대해 시스템의 상태는 한 상태에서 다른 상태로 이동하게 되어 독자가 시스템의 동작 과정을 이해할 수 있게 도와준다.

이 State Diagram을 만들기 위해 한 상태에서 일어날 수 있는 모든 시나리오를 분석하여야 하므로 Class Diagram의 객체를 디자인 또는 정의하는 데에 도움을 주며, 시스템의 구체적인 상태를 예측할 수 있다는 장점도 있다.

### Package Diagram

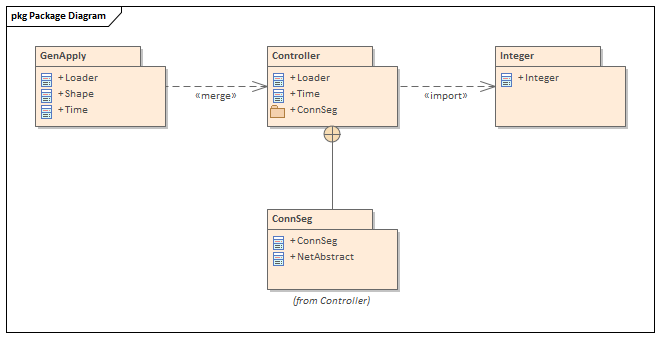


Figure 4 : Example of Package Diagram

Package diagram은 모델의 element들의 배치와 구성에 대한 내용을 담고 있는 구조적 다이어그램이다. Package diagram은 시스템 구조만을 표현하는 것이 아닌 sub-system간의 의존성을 추상적으로 표현한다. 이를 통해 복잡한 시스템을 여러 element의 그룹인 package로 표현하여 시스템을 단순하게 표현할 수 있다.

### ER Diagram

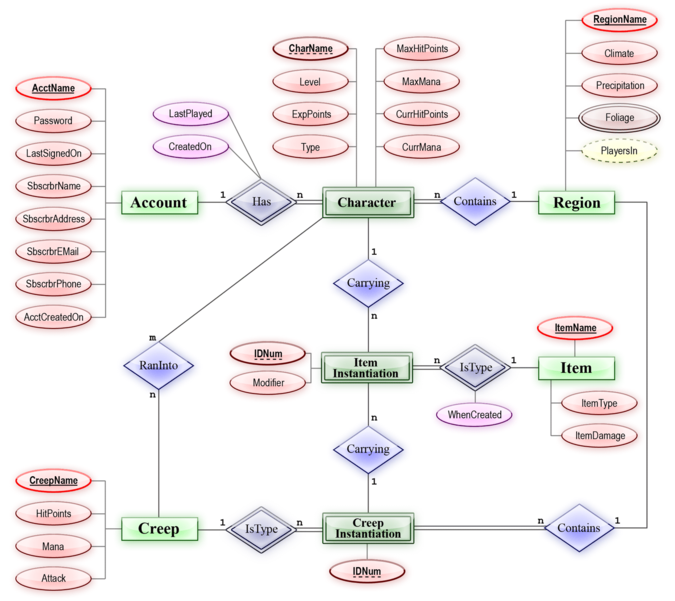


Figure 5 : Example of ER Diagram

ER Diagram은 Entity라는 element로 이루어진 다이어그램으로, Entity의 attribute와 Entity 간의 관계(relationship)을 통해 나타내어진 다이어그램이다. Class Diagram과 달리 ER Diagram은 데이터베이스 설계를 위해 주로 사용된다. 이 다이어그램을 바탕으로 SQL DDL 등을 만들어내며 이를 바탕으로 데이터베이스를 구현하게 된다.

## Applied Tool

### Draw.io



Figure 6 : Draw.io Logo

Draw.io는 온라인 모델링 툴로서 많은 기본 템플릿과 도형을 제공하기 때문에 사용자가 직접 다이어그램에 사용하기 위해 도형을 만들 필요가 없다. 또한 도형 간 연결선을 간단하게 만들 수 있고, 격자에 위치를 맞출 수 있기 때문에 도형을 정렬하기 편리하다. 이 문서에서 사용된 대부분의 다이어그램은 본 도구로 작성되었다.

### PowerPoint



Figure 7 : Powerpoint logo

Powerpoint는 그래픽 프레젠테이션 툴이다. 주로 발표용으로 사용되지만 내장된 도형 작성 기능이 매우 강력하기 때문에 draw.io에서 만들기 힘든 복잡한 다이어그램을 작성하기 위해 사용하였다.

### ERDPlus



Figure 8 : ERDPlus Logo

ERDPlus 는 ER Diagram 을 간단한 버튼 클릭으로 생성할 수 있게 해 주는 온라인 툴이다. 적은 노력으로 ER DIagram을 draw.io에 비해 간단하게 만들 수 있기 때문에 ER Diagram을 작성하는 데 사용하였다.

## Project Scope

본 시스템은 상품의 리뷰를 찾아보고 상품들을 비교하는 번거로운 과정에서 소비자들에게 편리를 제공하기 위해서 설계되었다. 후보 상품들을 비교할 때, 한눈에 대표 리뷰와 키워드를 같이 비교할 수 있는 기능을 제공해서 사용자들이 여러 사이트를 돌아다니며 정보를 찾는 불편을 줄이고, 자연어 처리 기반으로 고객들에게 리뷰를 키워드 형태로 이용할 수 있게 사용성을 높였다. 인터넷 쇼핑몰 입장에서는 고객이 결정하는 시간을 줄여서 상품 구매율을 높이고, 고객에게 보다 질 높은 정보를 제공해서 고객이 다른 곳으로 이탈하는 경우를 방지할 수 있게 도와준다.

본 시스템은 리뷰를 분석하는 리뷰 분석 시스템과, 연관 상품을 파악하는 연관 상품 탐색 시스템을 주 시스템으로 하며, 질 높은 상품 리뷰 비교를 목표로 한다.

리뷰 분석과 비교 기능에 집중하였고, 기존 인터넷 커머스 시스템 안에서 사용되는 Subsystem으로 동작할 수 있게 설계하였다. 그렇기 때문에 system boundary 안에 이미 기존 시스템에 구현되어 있는 component를 최소로 넣어 overhead를 줄였고, 본 시스템의 기능을 사용 시에 기존 시스템과 밀접하게 interaction하는 부분만 넣어서 실제로 본 시스템이 사용될 때 어떻게 기능하는지 보여줄 수 있도록 만들었다.

먼저 Frontend System은 사용자와 직접 상호작용을 하며, 상품 정보와 함께 리뷰를 효과적으로 보여주고, 다양한 사용자의 입력에 따라 반응하며 사용자의 결정을 돕는 역할을 한다.

다음으로 Backend System은 사용자가 요청하는 정보를 Database와 통신해서 다루는 역할을 하고 효과적으로 정보를 정리하는 역할을 한다.

특히 리뷰 분석 시스템과 연관 상품 탐색 시스템은 리뷰 정보를 잘 정리하고 유사 상품을 잘 파악하는 역할을 수행해서 사용자에게 편리하고 좋은 정보를 제공하는 목표에 필수적인 시스템이라고 할 수 있다. 따라서 Backend System 안에서도 따로 분리해서 중요하게 다룰 것이다.

# System Architecture – Overall

## Objectives

이번 장 에서는 본 시스템의 구조에 대해서 설명한다.

본 시스템의 전체적인 구조를 다이어그램으로 그리고 다이어그램에 포함된 각 서브시스템과 서브시스템 간의 상호작용에 대해서 설명한다.

## System Organization

### Overall System Architecture

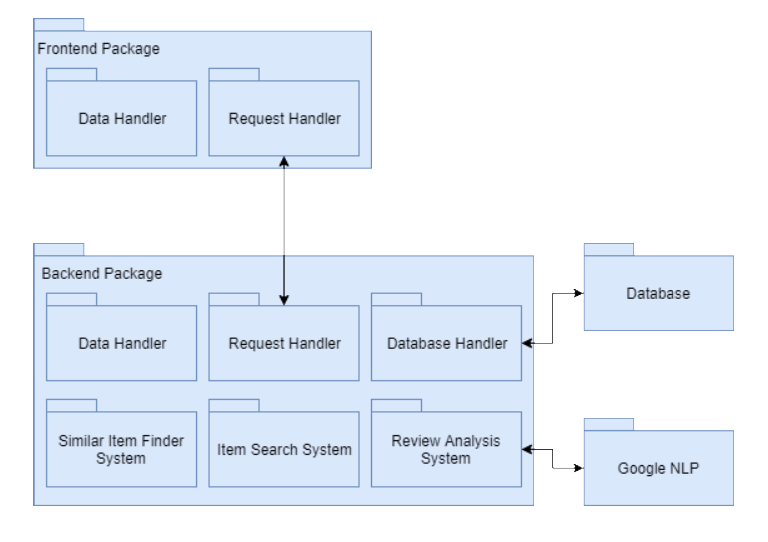


Diagram 1 : Overall System Architecture – Package Diagram

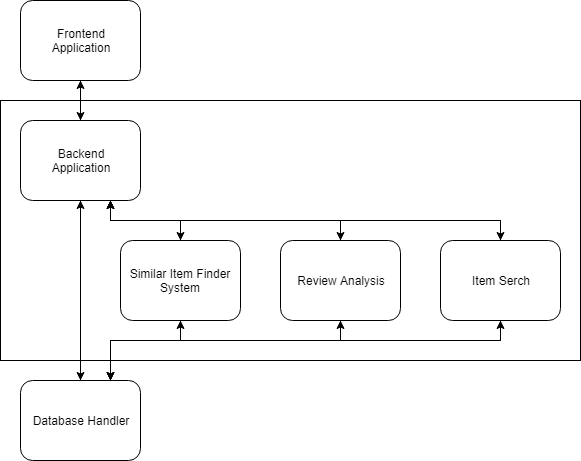


Diagram 2 : Overall System Architecture

시스템은 크게 Frontend Application, Backend Application, Database 로 나누어져 있다.

Frontend Application은 사용자로부터 입력을 받아 Backend Application에 필요한 정보 혹은 정보 수정을 요청한다. 두 Application간의 통신은 HTTP 프로토콜 위에서 JSON 혹은 XML의 형식으로 이루어진다. Backend Application으로부터 필요한 정보를 받으면, Frontend Application은 이 정보를 이용해 사용자에게 보여지는 페이지를 구성하고 사용자에게 전달한다.

Backend Application은 Frontend Application에서 요청한 정보를 제공하거나 정보를 수정한다. 이런 요청들은 Backend Application 에서 실행되거나, 복잡한 요청의 경우 하위 시스템에 의해 실행된다. 각 시스템들은 데이터베이스와 통신하기 위해 Database Handler를 이용한다.

Database는 사용자에게 보여지는 페이지를 구성하는데 필요한 정보들을 저장하고 있다. Backend Application의 요청에 따라 정보의 추가, 수정, 삭제, 조회가 이루어진다.

### Frontend Architecutre

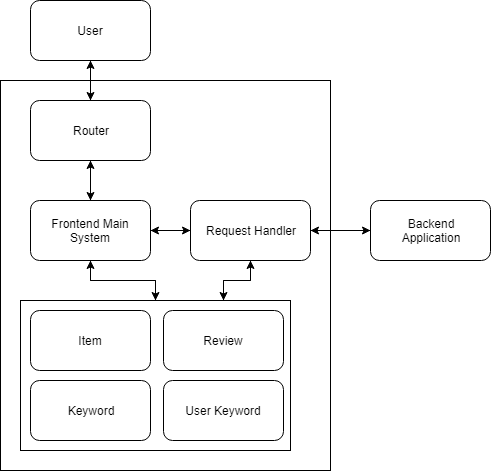


Diagram 3 : Frontend Architecture

Frontend는 사용자와 직접 상호작용을 하는 시스템을 말한다. Frontend Main System은 사용자의 입력을 받고 적절한 페이지를 만들어 사용자에게 전송한다. Item, Review, Keyword, User Keyword 등 페이지를 구성하는데 필요한 정보는 Backend Application을 통해 제공받으며, Backend와의 통신은 Request Handler을 통한다. Request Handler는 REST API를 이용해 Backend Application과 정보를 주고 받는다.

### Backend Architecture

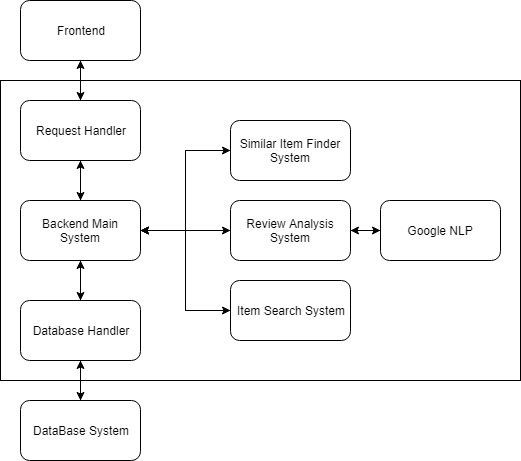


Diagram 4 : Backend Architecture

Backend Application은 사용자와 직접 상호작용을 하지 않지만, Frontend Application이 요청하는 정보를 제공하는 방식으로 페이지 구성을 돕는다. 필요한 정보를 제공하기 위해 데이터베이스에서 Item, Review 등을 조회하며, 정보를 가공하기도 한다. 유사한 상품 검색, 리뷰 분석 등 복잡한 작업은 하위 시스템이 전담하여 처리하며, 리뷰 분석의 경우 Google NLP API를 이용한다.

## Overall System State Diagram

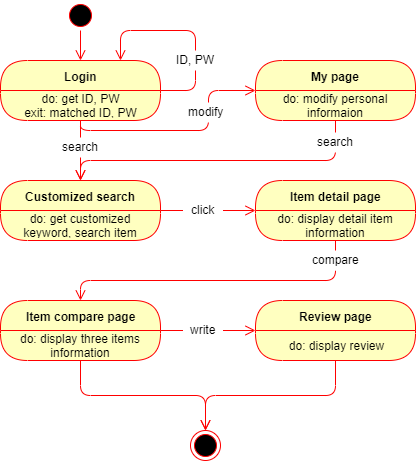


Diagram 5 : Overall System State Diagram

|  |  |
| --- | --- |
| **State** | **Description** |
| Login | 사용자가 로그인 하기를 기다리는 상태 |
| My page | 사용자 정보 페이지를 display |
| Customized search | 사용자가 사전에 설정해 둔 키워드와 검색어를 기준으로 상품 검색 |
| Item detail page | 상품 클릭 시, 상세 정보를 display |
| Item compare page | 3개의 상품을 동시에 display |
| Review page | 리뷰 작성 페이지를 display |

Table 1 : Overall System State Diagram – State Table

|  |  |
| --- | --- |
| **Stimulus** | **Description** |
| ID, PW | 유저가 ID, PW를 입력 |
| Modify | 유저가 정보 수정을 클릭 |
| Search | 유저가 상품을 검색 |
| Click | 검색된 상품을 클릭 |
| Compare | 상품 비교를 유저가 요청 |
| Write | 유저가 리뷰 작성 요청 |

Table 2 : Overall System State Diagram – Stimulus Diagram

# System Architecture – Frontend

## Objective

전체 시스템 아키텍처 중 사용자와의 상호작용을 담당하는 프론트엔드 시스템의 구조와 각 컴포넌트의 구성, 컴포넌트간의 관계를 서술한다.

## Overall Architecture

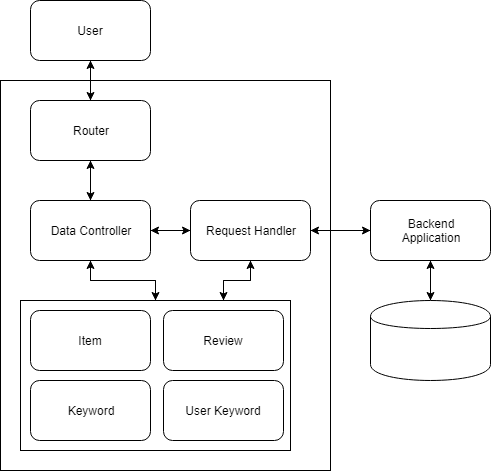


Diagram 6: System Architecture - Frontend – Overall

Frontend Application은 Router을 통해 사용자와 직접 상호작용을 하고, Request Handler을 통해 Backend Application과 통신한다. Backend Application를 통해 Item, Review, Keyword, User Keyword등 필요한 정보를 받으면 Data Controller는 이를 이용해 페이지를 만든다.

## Subcomponents

### Customized Search Result

1. Class Diagram

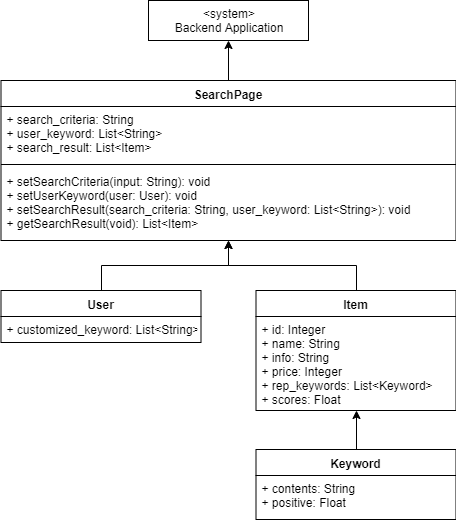


Diagram 7 : System Architecture - Frontend – Customized Search Result

* 1. SearchPage – 검색 페이지 객체

1. attributes

* search\_criteria: SearchPage 객체에서 아이템을 검색할 때 기준이 되는 단어
* user\_keyword: user가 사전에 설정한 선호 키워드 목록
* search\_result: 아이템 검색 후, 해당하는 아이템에 대한 목록

1. methods

* setSerachCriteria(input: String): 유저가 입력한 검색어로 search\_criteria를 설정한다.
* setUserKeyword(user: User): user 객체에서 유저가 사전에 설정한 선호 키워드를 가져와 user\_keyword에 저장한다.
* setSearchResult(search\_criteria: String, user\_keyword: List<String>): 검색 조건에 해당하는 아이템 목록을 가져오고, 유저 키워드에 따라 상품 키워드를 정렬한 후 search\_result에 저장한다.
* getSearchResult(void): search\_result에 저장된 값을 반환한다.
  1. User – 유저 객체

A. attributes

* customized\_keyword: 유저가 사전에 설정해둔 선호 키워드 목록

B. 기타사항

* 실제 User 객체는 그림보다 더 많은 속성 값을 가지고 있지만 사용되어지는 속성만을 표현했다.
  1. Item – 상품 객체

A. attributes

* id: 상품 ID
* name: 상품 이름
* info: 상품 정보
* price: 상품 가격
* rep\_keywords: 상품 대표 키워드 목록
* scores: 상품 평점

B. 기타사항

* 실제 Item 객체는 그림보다 더 많은 속성 값을 가지고 있지만 사용되는 속성만을 표현했다.
  1. Keyword – 키워드 객체

A. attributes

* contents: 키워드 내용
* positive: 리뷰 긍정도

1. Sequence Diagram

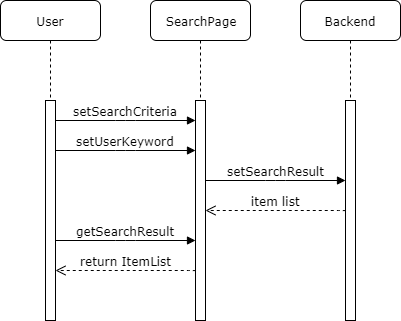


Diagram 8 : System Architecture – Frontend - Customized Search Result - sequence

### Item Page

1. Class Diagram

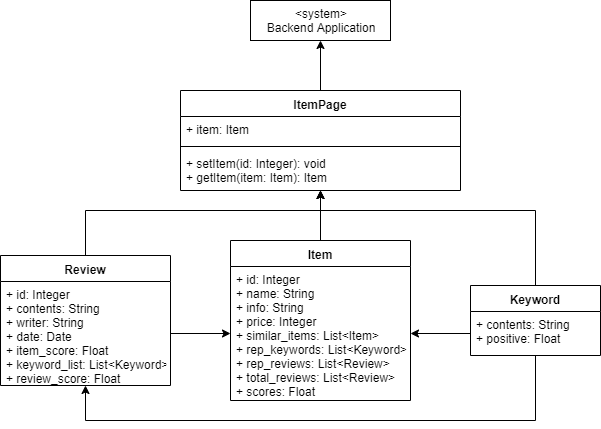


Diagram 9 : System Architecture – Frontend – Item Page

1. ItemPage – 아이템 상세 페이지 객체
   1. Attributes
      * item: 상세 페이지에서 보여줄 아이템 객체
   2. methods
      * setItem(id: Integer): 유저가 선택한 아이템을 id 조회를 통해 item에 설정
      * getItem\*(item: Item): getItem으로 시작하는 모든 함수는 item에 저장되어 있는 객체에서 정보를 추출
2. Review – 리뷰 객체
   1. Attributes
      * id: 리뷰의 id
      * contents: 리뷰 내용
      * writer: 리뷰 작성자
      * date: 리뷰 작성 날짜
      * item\_score: 리뷰 작성자가 매긴 아이템의 점수
      * keyword\_list: 리뷰의 키워드 목록
      * review\_score: 리뷰 자체 점수
3. Item – 상품 객체

A. attributes

* id: 상품 ID
* name: 상품 이름
* info: 상품 정보
* price: 상품 가격
* similar\_items: 해당 상품과 관련 있는 상품 목록
* rep\_keywords: 상품 대표 키워드 목록
* rep\_reviews: 상품 대표 리뷰 목록
* total\_reviews: 상품 전체 리뷰 목록
  + - scores: 상품 평점

1. Keyword – 키워드 객체
   1. Attributes
      * contents: 키워드 내용
      * positive: 리뷰 긍정도

2. Sequence Diagram

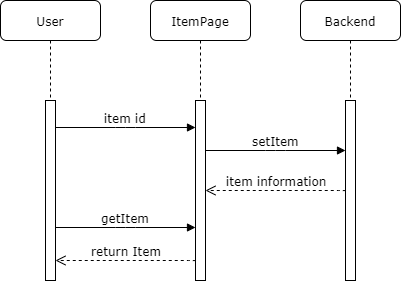


Diagram 10 : System Architecture – Frontend – Item Page - sequence

### Keyword Visualizer

1. Class Diagram

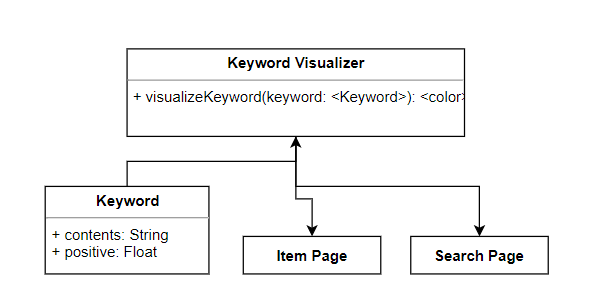


Diagram 11 : System Architecture - Frontend – Keyword Visualizer

1. Keyword Visualizer – 키워드 시각화 객체
2. attributes
3. methods

* visualizeKeyword(keyword: <Keyword>): 키워드의 positive value를 이용해서 키워드의 긍정도를 색으로 시각화해준다. Item Page와 Search Page에서 키워드를 표시할 때에 사용된다.

1. Keyword – 키워드 객체
   * + - 1. attributes

* contents: 키워드 내용
* positive: 리뷰 긍정도

1. Item Page – 상품 페이지 객체

A. 기타사항

* 실제 상품 페이지 객체를 추상화해서 해당 페이지에서 키워드 시각화 기능이 사용됨만을 나타냈다.

1. Search Page – 검색 페이지 객체

A. 기타사항

* 실제 검색 페이지 객체를 추상화해서 해당 페이지에서 키워드 시각화 기능이 사용됨만을 나타냈다.

1. Sequence Diagram

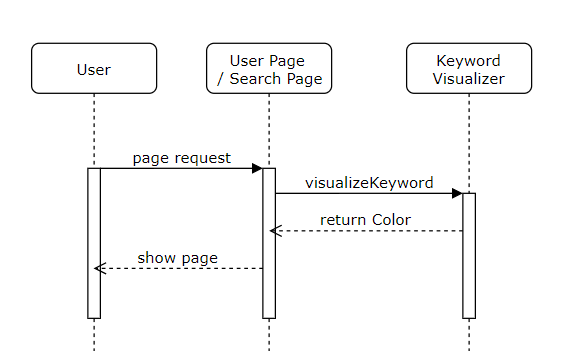


Diagram 12 : System Architecture – Frontend – keyword Visualizer - sequence

### Review Page

1. Class Diagram

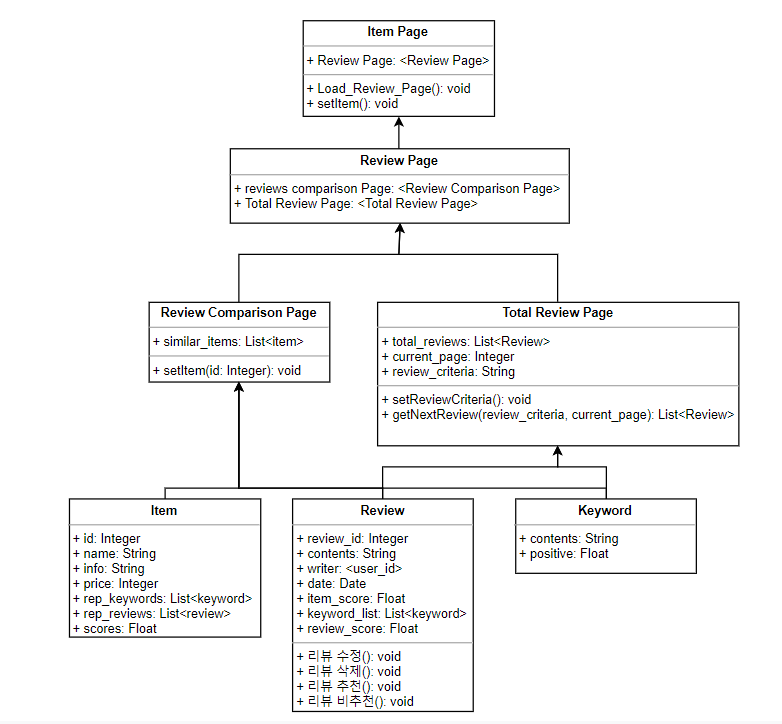


Diagram 13 : System Architecture - Frontend – Review Page

1. Item Page – 상품 페이지
2. attributes

* Review Page: 상품에 해당하는 리뷰 정보 페이지

1. methods

* Load\_Review\_Page(): 상품의 리뷰 페이지를 로드한다.
* setItem(): 상품 페이지에서 정보를 표시할 대상 상품을 설정한다.

1. 기타사항

실제 Item Page 객체를 Review Page입장에서 필요한 요소로만 나눠서 표현하였다.

1. Review Page – 리뷰 페이지 객체

A. attributes

* Reviews comparison Page: 유사상품 리뷰 비교 페이지
* Total Reviews Page: 전체 리뷰 페이지

B. 기타사항

* Review Page가 수행하는 두 가지 기능을 두 가지 객체로 나눠서 표현한다.

1. Review Comparison Page – 리뷰 비교 페이지

A. attributes

* Similar\_items: 유사 상품 목록

B. methods

* setItem(id: integer): 사용자가 관심 유사상품을 클릭하면 해당 상품의 상품 페이지로 넘어간다.

B. 기타사항

* 상품 페이지에서 원래 갖고 있는 속성, method들을 리뷰 페이지 관점으로 구체화해서 정의했다.

1. Total Review Page – 전체 리뷰 페이지
   1. attributes

* Total\_reviews: 전체 리뷰 목록
* Current\_page: 현재 리뷰 페이지 위치 (한번에 모든 리뷰를 요청하지 않고 사용자가 원하는 리뷰만 요청해서 자원 활용을 효율적으로 한다.)
* Review\_criteria: 리뷰 요청 시의 기준을 말한다. 특정 키워드 혹은 평점 순 등의 기준이 있다.
  1. methods
* setReviewCriteria(): 리뷰 요청 시의 기준을 설정한다.
* getNextReview(review\_criteria, current\_page): 설정한 리뷰 요청 기준으로 더 볼 리뷰를 불러온다. 키워드 별 리뷰 모아보기, 평점 높은 리뷰 순으로 보기, 리뷰 더 보기 기능을 구현할 수 있다.
  1. 기타사항
* 상품 페이지에서 원래 갖고 있는 속성, method들을 리뷰 페이지 관점으로 구체화해서 정의했다.

1. Item – 상품 객체

A. attributes

* id: 상품 ID
* name: 상품 이름
* info: 상품 정보
* price: 상품 가격
* rep\_keywords: 상품 대표 키워드 목록
* scores: 상품 평점

B. 기타사항

* 실제 Item 객체는 그림보다 더 많은 속성 값을 가지고 있지만 사용되는 속성만을 표현했다.

1. Review – 리뷰 객체
   * + - 1. attributes

* review\_id: 리뷰 아이디
* contents: 리뷰 내용
* writer: 작성자
* date: 마지막 수정 시각
* item\_score: 상품 평점
* keyword\_list: 키워드 목록
* review\_score: 리뷰 추천수
  + - * 1. methods
* 리뷰 수정( ): 작성자가 리뷰를 수정한다.
* 리뷰 삭제( ): 작성자가 리뷰를 삭제한다.
* 리뷰 추천( ): 자신이 구매한 상품의 리뷰를 최대 한번 추천한다. 추천수를 1 늘린다.
* 리뷰 비추천( ): 자신이 구매한 상품의 리뷰를 최대 한번 비추천한다. 추천수를 1 줄인다.

1. Keyword – 키워드 객체

A. attributes

* contents: 키워드 내용
* positive: 리뷰 긍정도

1. Sequence Diagram

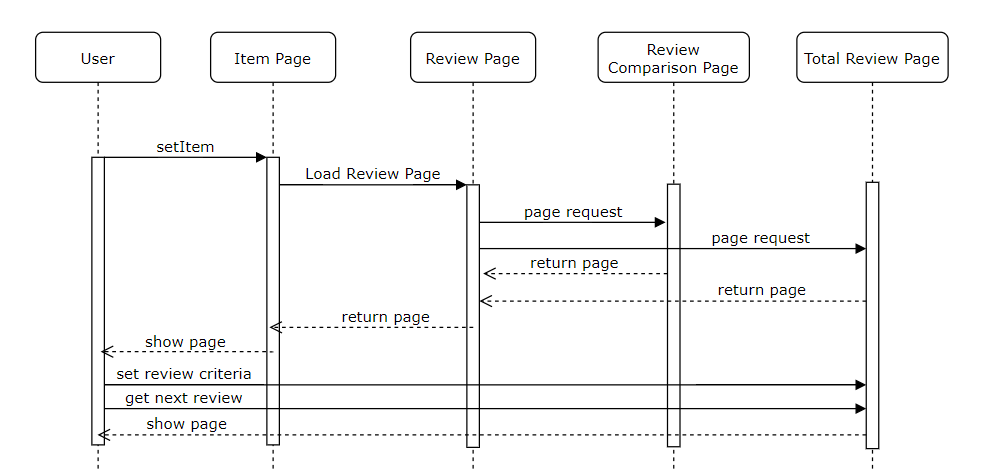


Diagram 14 : System Architecture – Frontend - Review Page - sequence

### Review Post

1. Class Diagram

![스크린샷이(가) 표시된 사진  자동 생성된 설명](data:application/octet-stream;base64,)

Diagram 15 : System Architecture – Frontend - Review Post

* 1. ReviewPoster : 리뷰 등록 객체
     1. Methods
        + PostReview(review: Review, item Item)

입력으로 들어온 리뷰를 상품의 리뷰로 등록 시킨다.

* 1. Review : Review 객체
     1. Attributes
        + Contents:사용자가 직접 입력한 리뷰 내용
        + User: 해당 리뷰를 등록한 유저
        + 기타사항: Review에는 keyword 등의 attribute가 더 있으나, 등록 시점에는 설정되지 않으므로 클래스 다이어그램에서 제외하였음
  2. User : 리뷰를 등록한 유저 객체
     1. Attributes
        + userId: 사용자의 ID
        + 기타사항: Review에 사용자의 다른 정보가 필요하지 않으므로 다른 attribute를 적지 아니하였음
  3. Item – 리뷰의 대상이 되는 상품
     1. Attributes
        + itemId: 상품의 ID
        + name: 상품의 이름
        + reviews: 해당 상품에 작성된 리뷰들의 리스트

1. Sequence Diagram

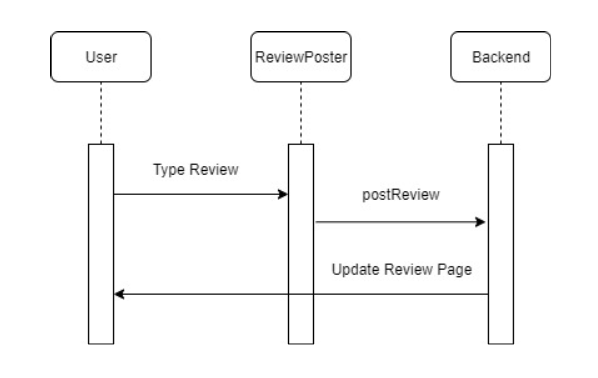


Diagram 16 : System Architecture – Frontend - Review Post – sequence

1. State Diagram

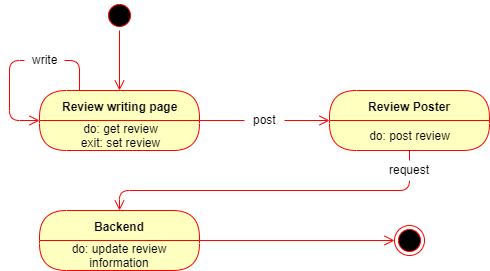


Diagram 17 : System Architecture – Frontend - Review Post – State

|  |  |
| --- | --- |
| **State** | **Description** |
| Review writing page | 유저가 리뷰를 작성하는 페이지 |
| Review Poster | 리뷰를 등록하기 위한 상태 |
| Backend | 리뷰를 Database에 업데이트 |

Table 3 : Review Post – State Diagram – State Table

|  |  |
| --- | --- |
| **Stimulus** | **Description** |
| Write | 유저가 리뷰를 작성 |
| Post | 리뷰 작성 후, 완료 클릭 |
| Request | 백엔드에 리뷰 포스팅 요청 |

Table 4 : Review Post – State Diagram – State Table

### User Page

1. Class Diagram

![스크린샷이(가) 표시된 사진  자동 생성된 설명](data:application/octet-stream;base64,)

Diagram 18 : System Architecture – Frontend - User Page

* 1. User Page
     1. Attributes
        + User: 해당 사용자 페이지를 방문하는 사용자 객체
     2. Methods
        + enterUserPage(password: String):해당 유저의 유저페이지에 접근하기 위해 비밀번호를 다시 입력하여 사용자를 인증. 인증 실패 시 접근 불가.
        + changePassword(password String): 사용자 계정의 비밀번호를 바꿈
        + changeKeyword(keywords: Lisng<Keyword>):사용자의 선호 키워드의 순서, 내용등을 바꿈
  2. User
     1. Attributes
        + userId: 사용자의 ID
        + name: 표시 될 사용자의 이름
        + password: 사용자의 비밀번호. 이 값은 해시 된 형태로 저장된다.
        + customized\_keyword: 사용자가 사전에 설정한 키워드들의 목록. 사용자가 중요하게 생각하는 순서대로 정렬되어 있다.
  3. Keyword
     1. Attributes
        + Contents: keyword의 이름. 리뷰 분석을 통해 얻어지며, 사용자가 중요하게 생각하는 내용과 관련이 있다.
        + Score: Keyword의 점수. 높을수록 리뷰에서 좋은 뜻으로 쓰였음을 의미하지만 사용자 페이지 기능에서는 쓰이지 않는다.

1. Sequence Diagram

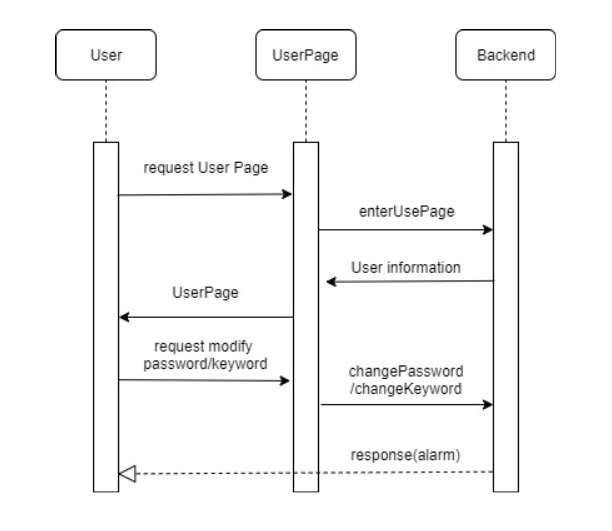


Diagram 19 : System Architecture – Frontend – User page – sequence

# System Architecture – Backend

## Objectives

전체 시스템 아키텍처 중 요청 처리와 리뷰 분석 역할을 하는 백엔드 시스템의 구조와 각 컴포넌트의 구성, 컴포넌트간의 관계를 서술한다

## Overall Architecture

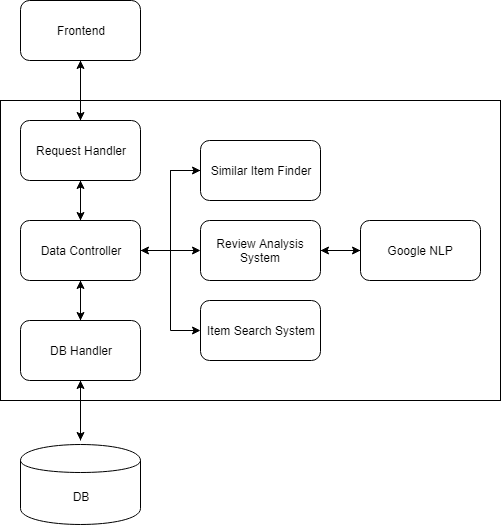


Diagram 20 : System Architecture – Backend – Overall

Backend Application은 Request Handler을 통해 Frontend와 통신하며, DB Handler을 통해 DB에 접근한다. Frontend에서 정보를 요청하면 DB에 접근해 데이터를 가져오고, 필요한 경우 하위 시스템을 호출해 정보를 가공한다. 하위 System인 Review Analysis System은 Google NLP를 이용한다.

## Subcomponents

### Login

1. Class Diagram

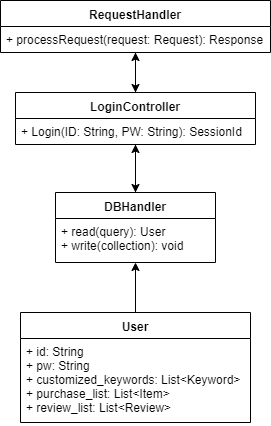


Diagram 21 : System Architecture – Backend – Login

* + - 1. RequestHandler – request 처리 객체
         1. methods

processRequest(request: Request): request 요청이 오면 요청받은 내용에 따라 알맞은 controller에 넘겨주는 메서드. 요청을 처리한 후 적절한 값을 반환

* + - 1. LoginController – 로그인 처리 객체
         1. methods

Login(ID: String, PW: String): ID와 PW를 이용해 database에 해당 유저 조회를 요청하는 메서드

* + - 1. DBHandler – database 제어 객체
         1. Methods

Read(query): 요청받은 query문을 사용해 databse에서 조회, User 객체를 반환

Write(collection): colleciton에 있는 데이터를 database에 저장

* + - 1. User – 유저 객체

1. Sequence Diagram

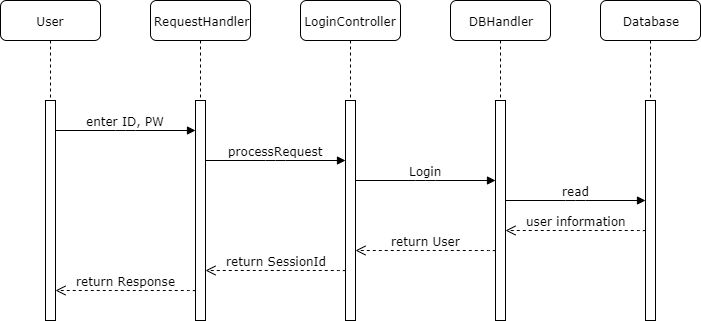


Diagram 22 : System Architecture – Backend – Login - sequence

### B. Review Analysis

1. Class Diagram

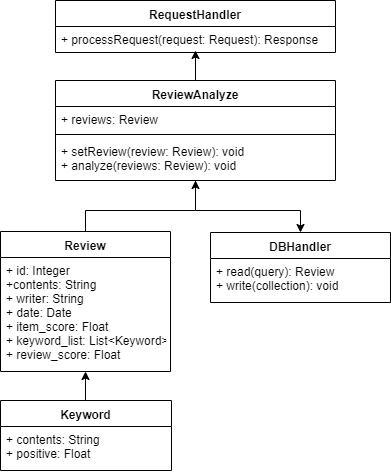


Diagram 23 : System Architecture – Backend – Review Analysis

* + - 1. RequestHandler – request 처리 객체
         1. methods

processRequest(request: Request): request 요청이 오면 요청받은 내용에 따라 알맞은 controller에 넘겨주는 메서드. 요청을 처리한 후 적절한 값을 반환

* + - 1. ReviewAnalyze – Review 분석 객체
         1. Attributes

reviews: 분석할 리뷰

* + - * 1. methods

setReview(review: Review): 포스팅 된 리뷰를 불러오는 메소드

analyze(reviews: Review): Google NLP API를 사용해 리뷰를 분석 후 변경된 정보를 database에 저장하는 메소드

* + - 1. DBHandler – database 제어 객체
         1. Methods

Read(query): 요청받은 query문을 사용해 databse에서 조회

Write(collection): colleciton에 있는 데이터를 database에 저장

1. Sequence Diagram

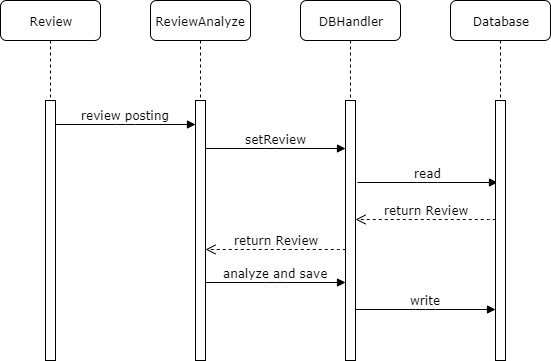


Diagram 24 : System Architecture – Backend – Review Analysis - sequence

1. State Diagram

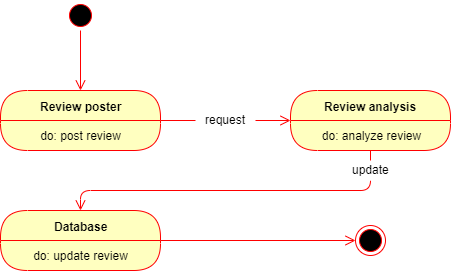


Diagram 25 : System Architecture – Backend – Review Analysis – State

|  |  |
| --- | --- |
| **State** | **Description** |
| Review poster | 리뷰 등록을 요청하는 상태 |
| Review analysis | 자연어 처리를 통한 리뷰 분석 |
| Database | Database에 분석된 리뷰를 업데이트 |

Table 5 : Review Analysis – State Diagram – State Table

|  |  |
| --- | --- |
| **Stimulus** | **Description** |
| Request | 자동으로 리뷰 분석을 요청 |
| Update | Databse에 리뷰 정보 업데이트 요청 |

Table 6 : Review Analysis – State Diagram – Stimulus Table

### C. Similar Item Finder

1. Class Diagram

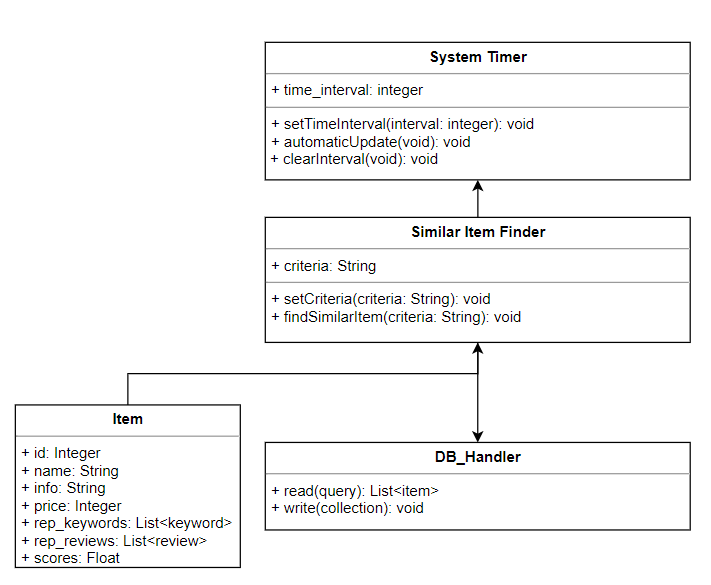


Diagram 26 : System Architecture – Backend – Similar Item Finder

* + - 1. System Timer: 시스템 자동 실행 시스템
         1. Attributes

time\_interval: 자동 실행 주기

* + - * 1. methods

setTimeInterval(interval: Integer): 자동 실행 주기 설정

automaticUpdate(void): 일정 주기마다 update 진행, Similar Item Finder 실행

clearTimeInterval(void): 자동 실행 취소

* + - 1. Similar Item Finder: 유사 상품 탐색 시스템
         1. Attributes

criteria: 유사 상품 탐색의 기준

* + - * 1. Methods

setCriteria(criteria:String): 유사 상품 탐색의 기준을 설정

findSimilarItem(criteria:String): criteria를 기준으로 유사 상품을 탐색하고 상품 DB에 업데이트한다.

* + - 1. DBHandler – database 제어 객체
         1. Methods

Read(query): 요청받은 query문을 사용해 databse에서 조회

Write(collection): colleciton에 있는 데이터를 database에 저장

* + - 1. Item – 상품 객체

A. attributes

* id: 상품 ID
* name: 상품 이름
* info: 상품 정보
* price: 상품 가격
* rep\_keywords: 상품 대표 키워드 목록
* scores: 상품 평점

B. 기타사항

* 실제 Item 객체는 그림보다 더 많은 속성 값을 가지고 있지만 사용되는 속성만을 표현했다.

1. Sequence Diagram

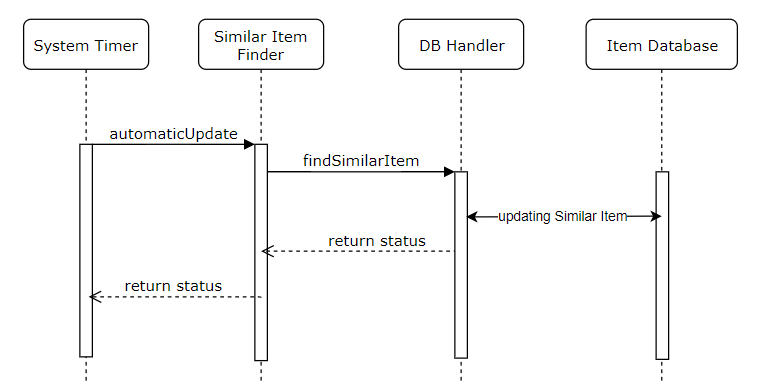


Diagram 27 : System Architecture – Backend – Similar Item Finder - sequence

### D. 상품 정보 제공

1. Class Diagram

![스크린샷이(가) 표시된 사진  자동 생성된 설명](data:application/octet-stream;base64,)

Diagram 28 : System Architecture – Backend – 상품 정보 제공

* 1. RequestHandler
     1. Methods
        + processRequest(request: Request): 사용자 Frontend에서 들어오는 요청을 처리하고 그에 따른 응답을 사용자 Frontend로 보내는 역할. 상품 정보 제공에서는 상품에 대한 요청이 들어왔을 때 DB로부터 상품정보를 받아 해당 정보를 응답으로 보낸다.
  2. DBHandler
     1. Methods
        + getItem(query): 상품을 DB에서 검색하여 Item 객체를 반환한다.
        + putItem(query, input: List<Item>): 상품 객체를 DB에 등록하는 기능. 그러나 이 기능에서는 쓰이지 않는다.
        + 기타사항: 그 외에 다른 method들이 많으나 해당 기능과 관련이 적어 적지 아니하였다.
  3. Item
     1. Attributes
        + Item\_id: 상품의 ID
        + name: 상품의 이름
        + price: 상품의 가격
        + keywords: 상품이 가지고 있는 keyword들의 목록
        + img\_src: 상품의 사진 주소
  4. Keywords
     1. Attributes
        + Contents: keyword의 이름. 리뷰 분석을 통해 얻어지며, 사용자가 중요하게 생각하는 내용과 관련이 있다.
        + Score: Keyword의 점수. 높을수록 리뷰에서 좋은 뜻으로 쓰였음을 의미한다.

1. Sequence Diagram

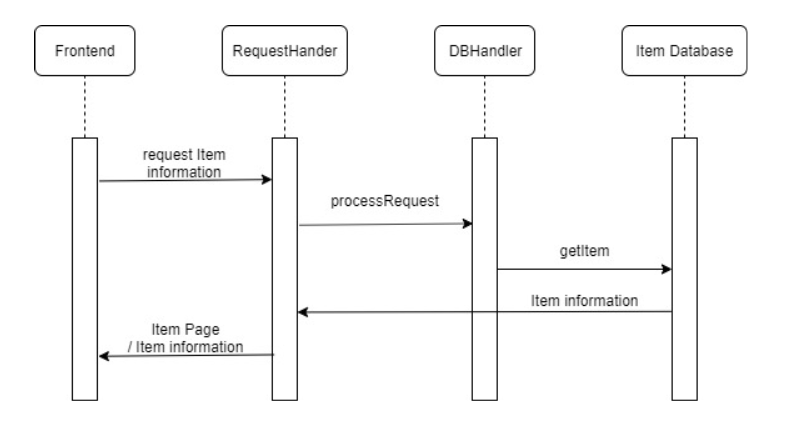


Diagram 29 : System Architecture – Backend – 상품 정보 제공 – sequence

### E. 개인정보 변경

1. Class Diagram

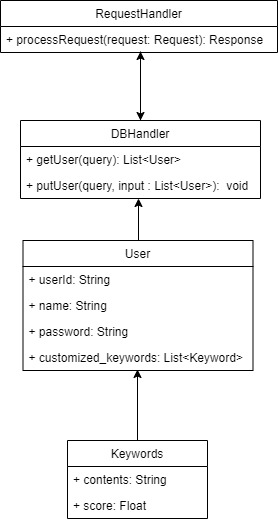


Diagram 30 : System Architecture – Backend – 개인 정보 변경

* 1. RequestHandler
     1. Methods
        + processRequest(request: Request): 사용자 Frontend에서 들어오는 요청을 처리하고 그에 따른 응답을 사용자 Frontend로 보내는 역할. 개인정보 변경 기능에서는 비밀번호 또는 사용자 설정 키워드를 변경하고자 하는 응답을 받아 처리하게 된다.
  2. DBHandler
     1. Methods
        + getUser(query): 상품을 DB에서 검색하여 User 객체를 반환한다.
        + putUser(query, input: List<Item>): User 객체를 DB에 등록하는 기능. 이 기능에서는 단순 등록이 아닌 기존에 있는 User 객체를 Update하는 것을 의미한다.
        + 그 외에 다른 method들이 많으나 해당 기능과 관련이 적어 적지 아니하였다.
  3. User
     1. Attributes
        + userId: 사용자의 ID
        + name: 표시 될 사용자의 이름
        + password: 사용자의 비밀번호. 이 값은 해시 된 형태로 저장된다.
        + customized\_keyword: 사용자가 사전에 설정한 키워드. 사용자가 중요하게 생각하는 순서대로 정렬되어 있다.
  4. Keywords
     1. Attributes
        + Contents: keyword의 이름. 리뷰 분석을 통해 얻어지며, 사용자가 중요하게 생각하는 내용과 관련이 있다.
        + Score: Keyword의 점수. 높을수록 리뷰에서 좋은 뜻으로 쓰였음을 의미하지만 개인정보 변경 기능에서는 쓰이지 않는다.

1. Sequence Diagram

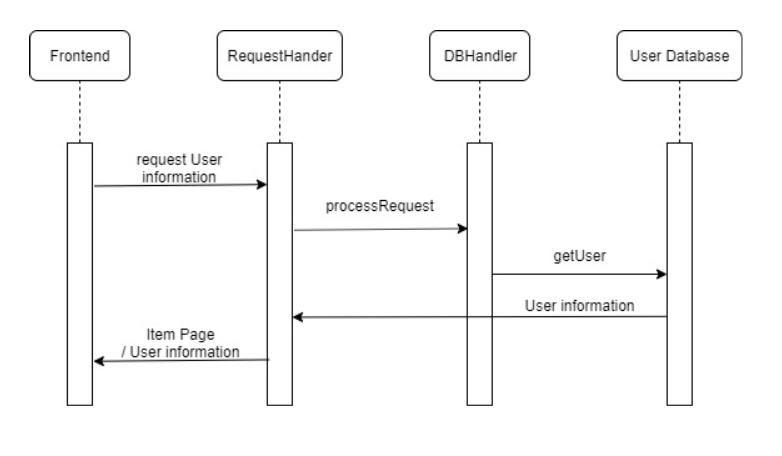


Diagram 31 : System Architecture – Backend – 개인 정보 변경 – sequence

# Protocol Design

## Objectives

이 문서에서는 backend에 정의된 rest api의 호출 방법과 반환 값에 대해 기술한다.

## REST API

### Item

1. Get (Item Search)

* Request

|  |  |  |
| --- | --- | --- |
| Method | GET | |
| URI | /items/?search=query | |
| Query | String | 검색 문자열 |

Table 7 : item - get – request

* Response

|  |  |  |
| --- | --- | --- |
| Success Code | 200 OK | |
| Failure Code | 404 Not Found | |
| Success  Response Body | item | 검색 문자열을 포함한 Item list |
| Failure  Response Body | message | Fail reasons   * 해당 item이 존재하지 않음 |

Table 8 : item - get - response

1. Get

* Request

|  |  |  |
| --- | --- | --- |
| Method | GET | |
| URI | /items/:item\_id | |
| Parameters | Item\_id | Item의 id |

Table 9 : item - get – request

* Response

|  |  |  |
| --- | --- | --- |
| Success Code | 200 OK | |
| Failure Code | 404 Not Found | |
| Success  Response Body | item | Item 객체 |
| Failure  Response Body | message | Fail reasons   * 해당 item이 존재하지 않음 |

Table 10 : item - get - response

1. Post
   * Request

|  |  |  |
| --- | --- | --- |
| Method | POST | |
| URI | /items | |
| Body | Item | Item 객체 |

Table 11 : item - post - request

* + Response

|  |  |  |
| --- | --- | --- |
| Success Code | 201 Created | |
| Failure Code | 400 Bad Request | |
| Success  Response Body | message | Item 추가 성공 메시지 |
| Failure  Response Body | message | Fail reasons   * 해당 item이 이미 존재 |

Table 12 : item - post - response

1. Put
   * Request

|  |  |  |
| --- | --- | --- |
| Method | PUT | |
| URI | /items/:id | |
| Body | item | Item 정보 |

Table 13 : item - put - request

* + Response

|  |  |  |
| --- | --- | --- |
| Success Code | 200 OK(기존에 존재하는 item의 경우) | |
| 201 Created(기존에 존재하지 않는 item의 경우) | |
| Failure Code | 400 Bad Request | |
| Success  Response Body | message | Item 수정 성공 메시지 |
| Failure  Response Body | message | Fail reasons   * Firebase 오류 |

Table 14 : item - put - response

1. Delete
   * Request

|  |  |  |
| --- | --- | --- |
| Method | DELETE | |
| URI | /items/:id | |
| Parameters | Item\_id | Item의 id |
| Header |  |  |

Table 15 : item - delete - request

* + Response

|  |  |  |
| --- | --- | --- |
| Success Code | 200 OK | |
| Failure Code | 404 Not Found(해당 id를 가진 item이 존재하지 않음) | |
| Success  Response Body | message | Item 삭제 성공 메시지 |
| Failure  Response Body | message | Fail reasons   * 해당 item이 존재하지 않음 |

Table 16 : item - delete - response

### Review

1. Get (all review)

* Request

|  |  |  |
| --- | --- | --- |
| Method | GET | |
| URI | /reviews/:item\_id/:offsetValue/?criteria | |
| Parameters | Item\_id, offsetValue | item id와 page number |
| Query | criteria | 리뷰 정렬 기준 |

Table 17 : review - get - request

* Response

|  |  |  |
| --- | --- | --- |
| Success Code | 200 OK | |
| Failure Code | 404 Not Found | |
| Success  Response Body | review | Criteria에 따라 정렬된 리뷰 중 지정된 페이지의 Review data list |
| Failure  Response Body | message | Fail reasons   * 해당 item이 존재하지 않음 |

Table 18 : review - get - response

1. Get (rep. review)

* Request

|  |  |  |
| --- | --- | --- |
| Method | GET | |
| URI | /reviews/:item\_id | |
| Parameters | Item\_id | Item의 id |

Table 19 : review - get - request

* Response

|  |  |  |
| --- | --- | --- |
| Success Code | 200 OK | |
| Failure Code | 404 Not Found | |
| Success  Response Body | review | 높게 평가된 review data list (3개) |
| Failure  Response Body | message | Fail reasons   * 해당 item이 존재하지 않음 |

Table 20 : review - get - response

1. Get (one review)

* Request

|  |  |  |
| --- | --- | --- |
| Method | GET | |
| URI | /reviews/:item\_id/:review\_id/1 | |
| Parameters | Item\_id, review\_id | item id와 review id |

Table 21 : review - get - request

* Response

|  |  |  |
| --- | --- | --- |
| Success Code | 200 OK | |
| Failure Code | 404 Not Found | |
| Success  Response Body | review | Review 객체 |
| Failure  Response Body | message | Fail reasons   * 해당 item 혹은review id가 존재하지 않음 |

Table 22 : review - get - response

1. Post
   * Request

|  |  |  |
| --- | --- | --- |
| Method | POST | |
| URI | /reviews/:item\_id/:user\_id | |
| Parameters | Item\_id, review\_id | 리뷰한 item의 id와 user의 id |

Table 23 : review - post - response

* + Response

|  |  |  |
| --- | --- | --- |
| Success Code | 201 Created | |
| Failure Code | 400 Bad Request | |
| Success  Response Body | message | 리뷰 작성 완료 메시지 |
| Failure  Response Body | message | Fail reasons   * User가 해당 item을 사지 않았음 * User가 이미 해당 item에 대한 review 작성 |

Table 24 : review - post – response

1. Put (review edit)
   * Request

|  |  |  |
| --- | --- | --- |
| Method | PUT | |
| URI | /reviews/:id | |
| Parameters | Item\_id, review\_id | 리뷰가 있는 item의 id와 리뷰의 id |

Table 25 : review - put – request

* + Response

|  |  |  |
| --- | --- | --- |
| Success Code | 200 OK(기존에 존재하는 item의 경우) | |
| Failure Code | 404 Not Found(해당 id를 가진 review가 존재하지 않음) | |
| Success  Response Body | message | review update success |
| Failure  Response Body | message | Fail reasons   * 해당 id를 가진 review가 존재하지 않음 |

Table 26 : review - put – response

1. Put (review recommend)
   * Request

|  |  |  |
| --- | --- | --- |
| Method | PUT | |
| URI | /reviews/:id | |
| Parameters | Item\_id, review\_id, user\_id | 리뷰가 있는 item의 id와 리뷰의 id, 리뷰를 추천한 user의 id |

Table 27 : review - put – request

* + Response

|  |  |  |
| --- | --- | --- |
| Success Code | 200 OK(기존에 존재하는 item의 경우) | |
| Failure Code | 404 Not Found(해당 id를 가진 review가 존재하지 않음) | |
| Success  Response Body | message | review update success |
| message | 생성된 review id를 경로로 반환 |
| Failure  Response Body | message | Fail reasons   * 해당 id를 가진 review가 존재하지 않음 |

Table 28 : review - put – response

1. Delete
   * Request

|  |  |  |
| --- | --- | --- |
| Method | DELETE | |
| URI | /reviews/:item\_id/:review\_id/:user\_id/?mode | |
| Parameters |  |  |
| Header |  |  |

Table 29 : review – delete – request

* + Response

|  |  |  |
| --- | --- | --- |
| Success Code | 200 OK | |
| Failure Code | 404 Not Found(해당 id를 가진 review가 존재하지 않음) | |
| Success  Response Body | message | review 삭제 성공 메시지 |
| Failure  Response Body | message | Fail reasons   * 해당 review가 존재하지 않음 |

Table 30 : review - delete – response

### User

1. Get
   * Request

|  |  |  |
| --- | --- | --- |
| Method | GET | |
| URI | /users/:user\_id/?mode | |
| Parameters | User\_id | User의 id |
| Query | Mode | Purchased or recommend or *undefined* |

Table 31 : user – get – request

* + Response

|  |  |  |
| --- | --- | --- |
| Success Code | 200 OK | |
| Failure Code | 404 Not Found (해당 사용자가 존재하지 않음) | |
| Success Response body |  | 사용자 객체(*undefined* mode),  item\_id list(purchased mode),  review\_id list(recommend mode) |
| Failure Response body | - | Fail reason   * 해당 user가 존재하지 않음 |

Table 32 : user – get – response

1. Post
   * Request

|  |  |  |
| --- | --- | --- |
| Method | POST | |
| URI | /users | |
| Parameters | User | 새 사용자 객체 정보 |

Table 33 : user - post – request

* + Response

|  |  |  |
| --- | --- | --- |
| Success Code | 200 OK | |
| Failure Code | 400 Bad Request (이미 등록된 사용자) | |
| Success Response body | - | DB에 user 등록 성공 메시지 |
| Failure Response body | - | Fail reason   * 이미 등록된 사용자 |

Table 34 : user - post – response

1. Put

* Request

|  |  |  |
| --- | --- | --- |
| Method | PUT | |
| URI | /users/:user\_id | |
| Parameters | User\_id | User의 id |
| body | keyword, PW | 유저 키워드 혹은 패스워드 변경 |

Table 35 : user – put – request

* Response

|  |  |  |
| --- | --- | --- |
| Success Code | 200 OK | |
| Failure Code | 404 Not Found (해당 user가 존재하지 않음) | |
| Success Response body | - | 유저 정보 변경 성공 메시지 |
| Failure Response body | - | Fail reason   * 해당 user가 존재하지 않음 |

Table 36 : user – put – response

1. Put (item 구매시 user data에 저장)

* Request

|  |  |  |
| --- | --- | --- |
| Method | PUT | |
| URI | /users/:user\_id/:item\_id | |
| Parameters | User\_id, item\_id | User의 id와 item의 id |

Table 37 : user – put – request

* Response

|  |  |  |
| --- | --- | --- |
| Success Code | 200 OK | |
| Failure Code | 404 Not Found (해당 user가 존재하지 않음) | |
| Success Response body | - | 유저 객체에 purchased\_item 추가 성공 메시지 |
| Failure Response body | - | Fail reason   * 해당 user가 존재하지 않음 |

1. Delete

* Request

|  |  |  |
| --- | --- | --- |
| Method | DELETE | |
| URI | /users/:user\_id | |
| Parameters | User\_id | 제거하고자 하는 사용자의 ID |

Table 38 : user – delete – request

* Response

|  |  |  |
| --- | --- | --- |
| Success Code | 200 OK | |
| Failure Code | 400 Bad Request (해당 사용자가 존재하지 않음) | |
| Success Response body | - | 유저 정보 삭제 성공 메시지 |
| Failure Response body | - | Fail reason   * 해당 user가 존재하지 않음 |

Table 39 : user – delete – response

1. Delete (구매 취소 시 user data에서 삭제)

* Request

|  |  |  |
| --- | --- | --- |
| Method | DELETE | |
| URI | /users/:user\_id/:item\_id | |
| Parameters | User\_id, item\_id | User의 id와 item의 id |

Table 40 : user – delete – request

* Response

|  |  |  |
| --- | --- | --- |
| Success Code | 200 OK | |
| Failure Code | 400 Bad Request (해당 사용자가 존재하지 않음) | |
| Success Response body | - | 구매 정보 삭제 성공 메시지 |
| Failure Response body | - | Fail reason   * 해당 user가 존재하지 않음 |

Table 41 : user – delete – response

### login

1. Post
   * Request

|  |  |  |
| --- | --- | --- |
| Method | Post | |
| URI | /login | |
| Body | Id, pw | User의 login id와 pw |

Table 42 : user – get – request

* + Response

|  |  |  |
| --- | --- | --- |
| Success Code | 200 OK | |
| Failure Code | 404 Not Found (해당 사용자가 존재하지 않음) | |
| Success Response body | - | 로그인 성공 메시지   * Session 생성 후 반환 |
| Fail  Response body | - | Fail reason   * 로그인 정보 불일치 |

1. Post
   * Request

|  |  |  |
| --- | --- | --- |
| Method | Get | |
| URI | /login/info | |
| Body | - | - |

Table 43 : user – get – request

* + Response

|  |  |  |
| --- | --- | --- |
| Success Code | 200 OK | |
| Failure Code | 404 Not Found (해당 사용자가 존재하지 않음) | |
| Success Response body | - | 현재 세션 반환 |
| Fail  Response body | - | 현재 세션 없음 |

### Logout

1. Post
   * Request

|  |  |  |
| --- | --- | --- |
| Method | Delete | |
| URI | /logout | |
| Body | - | - |

Table 44 : user – get – request

* + Response

|  |  |  |
| --- | --- | --- |
| Success Code | 200 OK | |
| Failure Code | 404 Not Found | |
| Success Response body | - | 로그아웃 성공 메시지 |
| Fail  Response body | - | Fail reason   * 로그인 상태 아님 |

# Database Design



## Objectives

Requirement specification과 위에 서술한 object model을 통해서 구체적 Database의 design을 기술한다. ER Diagram을 통해서 Entity와 Entity 관계를 기술하고, NoSQL 방식을 사용하는 Firebase Database에서 실제로 자료구조가 구현되는 방식인 JSON 형식으로 Database Design Specification을 진행한다.

## ER Diagram

본 시스템에는 User, Item, Review, Keyword 로 총 4 개의 Entity 가 존재한다. 각각의 Entity 는 네모 박스의 형태로 표현되고, Entity 간의 관계는 마름모꼴로 표현된다. 각 Entity는 사각형 안에 표시되며, attributes는 동그라미 안에 표시된다. 이 때 한 속성이 여러 개 값을 갖는 경우엔 동그라미 테두리가 2개이며, id같이 유일한 값을 갖는 속성은 밑줄을 그어 나타낸다. 관계에서는 상대 객체에 자신 객체 여러 개가 관련되어 있는 경우, 줄을 세 개 연결해서 표시한다.

### Overall Entities

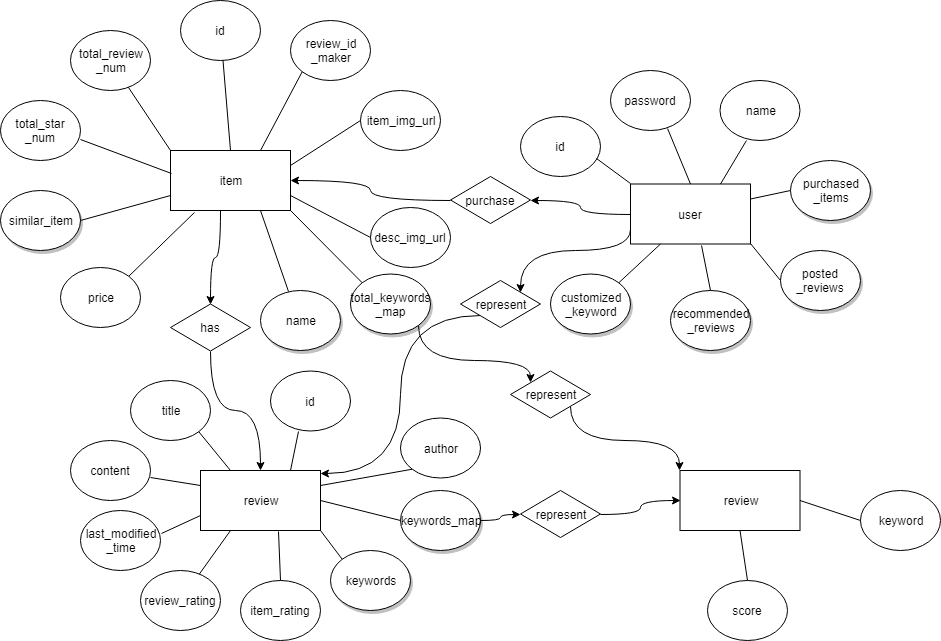


Diagram 32 : ER Diagram

### User

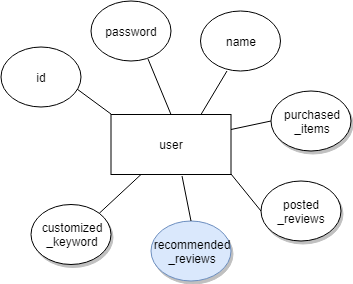


Diagram 33 : ER Diagram - User

User Entity는 각 사용자의 정보를 표시한다. Primary key는 id이며, 속성으로는 id, name, password, purchased items, posted\_reviews, customized keyword로 구성되어 있다.

추가 recommended\_reviews: 마이페이지에서 추천한 리뷰를 확인하기 위해 추가.

### Item

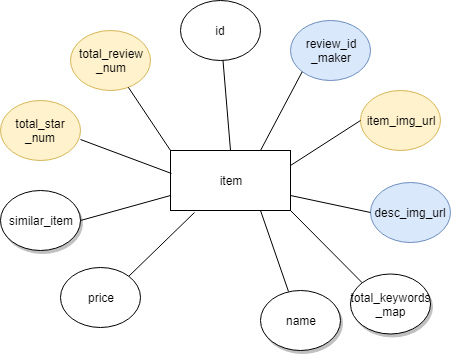


Diagram 34 : ER Diagram - Item

Item Entity는 각 상품의 정보를 표시하며, primary key는 id이다. 속성으로는 id, name, info, price, rep\_keywords, rep\_reviews, similar items, scores, img\_src로 구성되어 있다.

추가 review\_id\_make: 다음에 생성될 리뷰의 id를 저장하기 위해 추가

desc\_img\_url: item의 상세 설명 이미지를 저장하기 위해 추가

수정 item\_img\_url: item 객체에는 url을 저장하고, 다른 경로에 이미지 저장

total\_review\_num, total\_star\_num: 리뷰가 추가될 때 평점을 계산하기 위해 수정

### Review

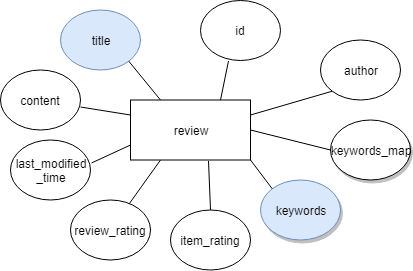


Diagram 35 : ER Diagram - Review

Review Entity는 각 리뷰의 정보를 표시하며, primary key는 id이다. 속성으로는 id, contents, writer, date, item score, keyword list, review score로 구성되어 있다.

추가 title: 리뷰의 제목 추가

keywords: 검색을 쉽게 하기 위해 포함된 키워드만 포함한 list 추가

### Keyword

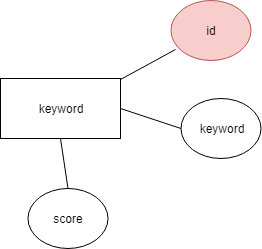


Diagram 36 : ER Diagram - Keyword

Keyword Entity는 각 키워드의 정보를 표시하며, primary key는 id이다. 속성으로는 id, contents, score로 구성되어 있다.

삭제 keyword가 item혹은 review에 포함됨에 따라 id를 삭제

## JSON

Firebase를 사용한 데이터베이스에 맞는 자료 구조 형식은 JSON형식이다. JSON 형식을 사용하게 되면 기존 RDBMS와 다르게 적은 제약으로 유연한 데이터베이스 구조를 만들 수 있다. 이를 통해 데이터의 체계성은 줄어들었지만, 빅데이터로의 확장 가능성이 높아지고, 간단한 구현을 보장한다. 이런 NoSQL 구조가 짧은 시간 안에 우리가 원하는 시스템을 테스트하는 데 최적의 데이터베이스 구조였기 때문에 이와 같은 구조를 채택했다.

다음은 JSON상으로 구현된 모델이다.

### User

{

id: 아이디,

name: 이름,

password: 비밀번호(hashed),

purchased items: [구매 상품 id 목록],

posted review: [등록한 리뷰 목록]

recommended review: [추천한 리뷰 목록],

customized keyword: [선호 키워드 목록]

}

### Item

{

id: 상품 아이디,

name: [상품 이름 단어 목록],

item\_img\_url: 상품 이미지 url,

desc\_img\_url: 상품 상세 정보 url,

price: 상품가격,

similar\_item: [유사 상품1, 유사 상품 2],

total\_star\_num: 평가 점수의 합,

total\_review\_num: 리뷰 개수,

total\_keywords\_map: { 키워드: 키워드 점수, … },

review\_id\_make: 마지막 리뷰의 id,

review: <<review collection>>

}

### Review

{

id: 리뷰 아이디,

author: 리뷰 작성자,

title: 리뷰 제목,

content: 리뷰 내용,

last\_modified\_time: 마지막 수정 시간,

keywords\_map: [{ name: 키워드, score: 키워드 점수 } , … ]

keywords: [키워드 목록]

item\_rating: 리뷰 평점

}

### Keyword

{

Content: 키워드,

Score: 키워드 점수,

Id: 키워드 id

}

# Testing Plan



## Objectives

본 장에서는 Testing Plan의 목적과 프로세스에 대해 설명한다.

Software Test는 사용자에게 특정한 품질 이상의 Software를 제공하기 위해, Software을 Test하는 과정이다. 이 과정을 통해 다음과 같은 조건을 만족해야 한다.

* Development/ Design Requirement를 만족해야 한다.
* 모든 종류의 입력에 대해서 올바르게 반응한다.
* 허용되는 시간 내에 기능을 수행한다.
* 사용자의 개인정보를 보호한다.
* 결과가 사용자를 만족시킬 수 있어야 한다.

Test는 개발 과정 중에 진행하는 Development Test와 개발이 완료되고 Release하기 전 진행하는 Release Test로 나뉜다.

## Testing Policy

### Development Test

Development Testing은 개발 과정 중에서 시스템의 오류를 찾아내는 프로세스이다. 개발 과정 중에 즉각적인 수정을 하기 때문에 개발 비용, 시간을 줄일 수 있다.

이 단계에서는 코드 분석, 데이터 흐름 분석, Unit Test 등을 진행하며, Reliability Test, Security Test, Performance Test에 초점을 맞춘다.

1) Reliability Test

Reliability Test는 Software의 함수들이 주어진 시간 내에 제대로 작동하는지 확인하기 위한 Test이다. 이 Test를 통해 함수의 설계의 문제를 발견하거나 기능적 오류를 발견할 수 있다.

Similar Item Finder, Review Analysis 등의 함수들에 대해 Test Case를 설정하고 만족스러운 결과가 나오도록 반복적인 수정을 한다. 또한 이런 함수들이 모든 유효한 입력에 대해서 처리할 수 있도록 꼼꼼히 예외 처리를 하도록 한다.

2) Security Test

Security Test는 System이 데이터를 보호하는 기능이 제대로 작동하는지 확인하기 위한 Test이다. 하지만 몇몇 Test를 통과하더라도 예상하지 못한 곳에서 결함이 발견될 수 있으며, 문제가 발생한 경우 많은 사용자에게 피해를 줄 수 있기 때문에, 주의를 기울여 Test해야 한다.

고객이 입력한 비밀번호는 DB에 저장된 비밀번호와 값이 동일한지만 확인하면 되기 때문에 해시 알고리즘을 이용할 수 있다. 복호화가 힘들다는 해시 알고리즘 의 특성 때문에, 누군가 DB에 직접 접근하더라도 고객의 비밀번호를 알 수 없다.

DB는 사용자의 개인정보가 포함되어 있기 때문에, Backend Application을 제외한 외부 침입을 차단해야 한다. 이를 위해 Backend과의 통신에서 서로의 신원을 확인하기 위한 검증을 한다.

개발 시 System간 통신시 검증을 하도록 하고, 코드 구조에 결함은 없는지 검토하며, 이후 해킹 상황을 가정해 테스트 하도록 한다.

3) Performance Test

Performance Test는 특정 워크로드에서 시스템의 속도와 안정성을 측정하기 위한 Test이다. 시스템이 예상되는 동시 사용자 수에서 주어진 응답시간을 만족하는지 확인한다. 동시 사용자 수가 많은 상황을 가정해 부하를 증가시키며 시스템의 동작을 관찰한다. Frontend, Backend Application과 Database를 관찰하며 처리량과 응답시간 등을 확인한다.

### Release Test

Release Test는 하위 시스템을 결합한 전체 시스템이 문제가 없이 의도한대로 작동하는지 확인하는 프로세스이다. 최종 발표 이전까지 완료해야 하며, 알파 버전, 베타 버전 등 Proto Type을 만들어 확인한다. 이 과정에서 실제 사용자로부터 피드백을 받을 수 없기 때문에, Test Case를 만들고 직접 확인하도록 한다.

Test에 필요한 데이터 중, Item과 Review는 Amazon.com에서 가져오며, 회원정보 같은 경우 직접 입력하여 진행한다. Test 과정에서 다음과 같은 항목을 확인한다.

* Requirement에서 작성한 Scenario Flow에서 문제가 없는지 확인
* Page 구성 등 시각적인 부분에 문제가 없는지 확인
* 시스템의 반응 속도를 확인
* 페이지 이동간에 로그인 인증이 잘 유지되는지 확인
* Search, Profile Edit, Similar Item Finder, Review Analysis등 중요한 기능이 잘 작동하는지 확인

### Automized Test

Mocha는 Node.js에서 사용하는 테스트 프레임워크이다. 본 시스템에서는 mocha를 이용해 backend server test를 자동화한다. Backend server에 rest api를 전송하고 예상되는 응답을 한는지 확인한다. 테스트 내용은 다음과 같다.

1. Item test

* 모든 item 데이터 가져오기
* Item등록하기
* Item 데이터 가져오기
* Item 데이터 수정하기
* Item 데이터 삭제하기
* 등록되지 않는 item 데이터 가져오기

1. User test

* User 등록하기
* 선호 키워드 수정하기
* 패스워드 변경하기
* 아이템 구매하기, 구매 취소하기
* User 데이터 삭제하기
* 등록되지 않은 User 데이터 가져오기

1. Review test

* 아이템의 리뷰 가져오기
* 리뷰 등록하기
* 특정 리뷰 가져오기
* 유저가 등록한 리뷰 가져오기
* 유저가 추천한 리뷰 가져오기
* 리뷰 삭제하기
* 리뷰를 작성하지 않은 유저가 리뷰 삭제하기

1. Login, logout

* 회원등록하기
* 정상 로그인하기
* 틀린 패스워드로 로그인하기
* 등록되지 않은 아이디로 로그인하기
* 로그아웃 하기
* 회원 탈퇴하기

# Development Plan



## Objectives

본 챕터에서는 개발 시 실제로 사용할 플랫폼과 환경에 대해 설명할 것이다. 각 플랫폼의 특징에 대해 설명하고 최종적으로 어떤 구조를 가지게 되는지 기술한다. 또한 개발 일정을 간트 차트 형식을 통해 기술할 것이다.

## Overall Framework Architecture

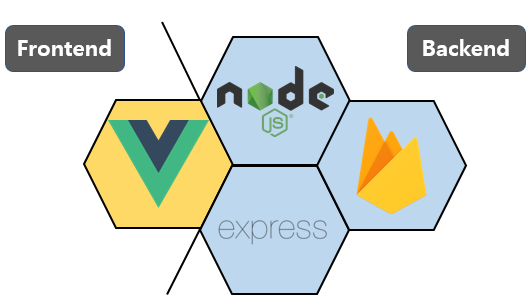


Figure 9 : Overall Framework

우리 서비스에서 최종적으로 사용할 프레임워크 구조이다. 프론트엔드로는 vue.js를 사용했으며 백엔드로는 Node.js, Express, Firebase를 사용할 것이다. 각각의 프레임워크에 대한 설명을 아래 파트에 상세히 기술했다.

## Frontend Environment

### Vue.js



Figure 10 : Vue.js logo

Vue.js는 최근 계속해서 상승세를 타고 있는 프론트엔드 자바스크립트 프레임워크이다. Angular, React에 비해 매우 작고 가벼우며 복잡도도 낮기 때문에 사용하기에 매우 간편하고 처음 배우는 사람이 시작하기에 다른 프레임워크보다 쉽다. 또한 성능이 다른 프론트엔드 프렘워크에 비해 우수하고 빠른편이며, Angular의 장점과 React의 장점을 모두 가지고 있다.

Vue.js의 특징은 다음과 같다.

Vue.js는 MVVM(Model-View-ViewModel) 패턴을 사용한다. MVVM 패턴이란 화면을 모델(Model) – 뷰(View) – 뷰 모델(ViewModel)로 구조화하여 개발하는 방식이다. 여기서 모델은 어플리케이션에서 사용되는 데이터와 그 데이터를 처리하는 부분이며, 뷰는 사용자에게 보여지는 UI 부분, 마지막으로 뷰 모델은 뷰를 표현하기 위해 만든 뷰를 위한 모델이다. 뷰를 나타내 주기 위한 모델이자 뷰를 나타내기 위한 데이터를 처리하는 부분인 것이다. 이 MVVM패턴을 사용함으로써 뷰와 모델 사이의 의존성을 없애 각각을 모듈화하여 개발할 수 있게 된다.

Vue.js는 컴포넌트 기반의 프레임워크이다. 각각의 컴포넌트는 HTML, CSS, 자바스크립트를 모두 가지고 있어 코드 집적도와 유지보수성을 높일 수 있으며, 코드의 재사용성을 높일 수 있다.

이외에 가상돔(Virtual DOM) 렌더링 방식, 양방향 데이터 바인딩 등 많은 좋은 특징을 가지고 있다.

## Backend Environment

### Node.js

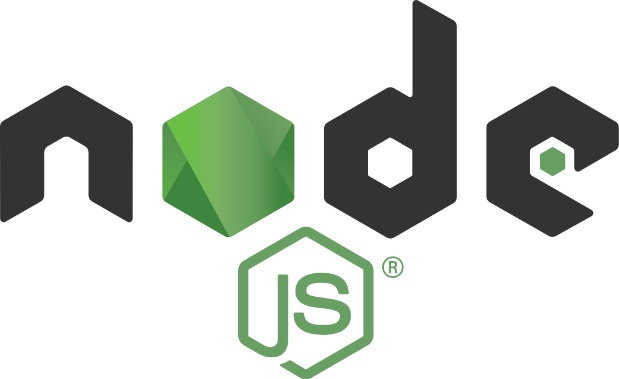


Figure 11 : Node.js logo

Node.js는 구글 크롬의 자바 크롬의 자바스크립트 엔진(V8 Engine)으로 빌드 된 자바스크립트 런타임이다. 이 이야기를 쉽게 하면 다음과 같다. 자바스크립트는 일반적으로 크롬과 같은 브라우저에 내장되어 있다. 그래서 자바스크립트는 브라우저에 종속되어 있었지만 이를 크롬 같은 브라우저에서만 쓰는 것이 아닌 브라우저 밖, 즉 내 컴퓨터에서 다양한 용도로 확장하기 위해 만들어진 것이 바로 Node.js이다. Node.js를 이용하면 python과 같이 내 컴퓨터에서 File System을 이용할 수도 있고, 서버를 만들 수도 있고, 크롤링도 할 수 있게 된다.

Node.js의 특징은 다음과 같다.

비동기식 I/O 처리를 지원한다. Node.js의 모든 라이브러리는 비동기식(async), 즉 Non-blocking 방식으로 작업을 처리한다. 하나의 API가 실행되고 데이터를 반환할 때까지 기다리는 것이 아니라 바로 다음 API를 계속해서 실행하는 것이다. 만약 이전에 실행했던 API가 결과값을 반환한다면 Node.js의 이벤트 알림 메커니즘을 통해 결과값을 받아오게 된다.

이 외 구글 크롬의 자바스크립트 엔진으로 인한 빠른 속도, 단일 쓰레드 사용, 뛰어난 확장성 등의 특징을 가지고 있다.

### Express



Figure 12 : Express logo

Express는 Node.js의 개발을 더 쉽게 하기위해 만들어진 간편한 웹 프레임워크이다. Node.js의 핵심 모듈인 http와 Connect 컴포넌트를 기반으로 한다. 쉽게 말하면 Node.js개발 시 개발을 빠르고 손쉽게 할 수 있도록 도와주는 역할을 한다. Python의 Django, java의 spring과 같은 역할을 한다고 생각하면 된다.

### Firebase



Figure 13 : Firebase logo

Firebase는 웹과 모바일 개발에 필요한 기능을 제공하는 BaaS(Backend as a Service)이다. 쉽게 말해 백엔드 개발을 통해 서버를 따로 설계, 구현하지 않고도 클라우드와 연동해 웹, 모바일 응용 프로그램의 프론트엔드 개발에 집중할 수 있도록 도와주는 서비스이다. 즉 하나의 서버를 개발하기 위한 인증, 데이터베이스, 푸시 알람, 스토리지, API 등을 따로 개발하지 않아도 Friebse가 이 모든 기능을 프로젝트 구축 시 자동적으로 만들어 주는 것이다. 또한 서버를 구축하기 위해 리눅스 명령어를 알 필요도 없으며 도메인을 구입할 필요도 없고 개발하는 동안에는 서버를 구입할 필요도 없게 해준다. Firebase가 가지고 있는 대표적인 기능은 다음과 같다.

* 실시간 데이터베이스
* 간편한 사용자 인증
* 클라우드 저장소
* 호스팅
* 앱 테스트
* 성능 모니터링
* 오류 보고

이 외에도 많은 기능을 가지고 있으며, 이는 Firebase 프로젝트를 생성하기만 하면 모두 사용할 수 있다.

## Schedule

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 항목 | | 11월 | | | 12월 | |
| 3주차 | 4주차 | 5주차 | 1주차 | 2주차 |
| 프론트엔드 | 쇼핑몰 첫 페이지 |  |  |  |  |  |
|  | 검색 결과 페이지 |  |  |  |  |  |
|  | 마이 페이지 |  |  |  |  |  |
|  | 상품비교 페이지 |  |  |  |  |  |
| 백엔드 | 로그인 |  |  |  |  |  |
|  | 개인정보 수정 |  |  |  |  |  |
|  | 상품 검색 |  |  |  |  |  |
|  | 연관상품 찾기 |  |  |  |  |  |
|  | 리뷰 분석 |  |  |  |  |  |
| 시스템 통합 & 테스트 | |  |  |  |  |  |
| 최종 발표 준비 | |  |  |  |  |  |

Table 45 : schedule table

앞으로의 일정은 위의 표와 같다. 프론트엔드와 백엔드 개발을 병렬적으로 진행해 개발 속도를 높일 것이다.

# 10. Index

## 10.1. Tables

[Table 1 : Overall System State Diagram – State Table](#_Toc24553582)

[Table 2 : Overall System State Diagram – Stimulus Diagram](#_Toc24553583)

[Table 3 : Review Post – State Diagram – State Table](#_Toc24553584)

[Table 4 : Review Post – State Diagram – State Table](#_Toc24553585)

[Table 5 : Review Analysis – State Diagram – State Table](#_Toc24553586)

[Table 6 : Review Analysis – State Diagram – Stimulus Table](#_Toc24553587)

[Table 7 : item - get - request](#_Toc24553588)

[Table 8 : item - get - response](#_Toc24553589)

[Table 9 : item - post - request](#_Toc24553590)

[Table 10 : item - post - response](#_Toc24553591)

[Table 11 : item - put - request](#_Toc24553592)

[Table 12 : item - put - response](#_Toc24553593)

[Table 13 : item - delete - request](#_Toc24553594)

[Table 14 : item - delete - response](#_Toc24553595)

[Table 15 : item - search - request **오류! 책갈피가 정의되어 있지 않습니다.**](#_Toc24553596)

[Table 16 : item - search - response **오류! 책갈피가 정의되어 있지 않습니다.**](#_Toc24553597)

[Table 17 : review - get - request](#_Toc24553598)

[Table 18 : review - get - response](#_Toc24553599)

[Table 19 : review - post - response](#_Toc24553600)

[Table 20 : review - post – response](#_Toc24553601)

[Table 21 : review - put – request](#_Toc24553602)

[Table 22 : review - put – response](#_Toc24553603)

[Table 23 : review – delete – request](#_Toc24553604)

[Table 24 : review - delete – response](#_Toc24553605)

[Table 25 : user – get – request](#_Toc24553606)

[Table 26 : user – get – response](#_Toc24553607)

[Table 27 : user - post – request](#_Toc24553608)

[Table 28 : user - post – response](#_Toc24553609)

[Table 29 : user – put – request](#_Toc24553610)

[Table 30 : user – put – response](#_Toc24553611)

[Table 31 : user – delete – request](#_Toc24553612)

[Table 32 : user – delete – response](#_Toc24553613)

[Table 33 : schedule table](#_Toc24553614)

## 10.2. Figures

[Figure 1 : Example of Class Diagram](#_Toc24553429)

[Figure 2 : Example of Sequence Diagram](#_Toc24553430)

[Figure 3 : Example of State Diagram](#_Toc24553431)

[Figure 4 : Example of Package Diagram](#_Toc24553432)

[Figure 5 : Example of ER Diagram](#_Toc24553433)

[Figure 6 : Draw.io Logo](#_Toc24553434)

[Figure 7 : Powerpoint logo](#_Toc24553435)

[Figure 8 : ERDPlus Logo](#_Toc24553436)

[Figure 9 : Overall Framework](#_Toc24553437)

[Figure 10 : Vue.js logo](#_Toc24553438)

[Figure 11 : Node.js logo](#_Toc24553439)

[Figure 12 : Express logo](#_Toc24553440)

[Figure 13 : Firebase logo](#_Toc24553441)

## 10.3. Diagrams

[Diagram 1 : Overall System Architecture – Package Diagram](#_Toc24556239)

[Diagram 2 : Overall System Architecture](#_Toc24556240)

[Diagram 3 : Frontend Architecture](#_Toc24556241)

[Diagram 4 : Backend Architecture](#_Toc24556242)

[Diagram 5 : Overall System State Diagram](#_Toc24556243)

[Diagram 6: System Architecture - Frontend – Overall](#_Toc24556244)

[Diagram 7 : System Architecture - Frontend – Customized Search Result](#_Toc24556245)

[Diagram 8 : System Architecture – Frontend - Customized Search Result - sequence](#_Toc24556246)

[Diagram 9 : System Architecture – Frontend – Item Page](#_Toc24556247)

[Diagram 10 : System Architecture – Frontend – Item Page - sequence](#_Toc24556248)

[Diagram 11 : System Architecture - Frontend – Keyword Visualizer](#_Toc24556249)

[Diagram 12 : System Architecture – Frontend – keyword Visualizer - sequence](#_Toc24556250)

[Diagram 13 : System Architecture - Frontend – Review Page](#_Toc24556251)

[Diagram 14 : System Architecture – Frontend - Review Page - sequence](#_Toc24556252)

[Diagram 15 : System Architecture – Frontend - Review Post](#_Toc24556253)

[Diagram 16 : System Architecture – Frontend - Review Post – sequence](#_Toc24556254)

[Diagram 17 : System Architecture – Frontend - Review Post – State](#_Toc24556255)

[Diagram 18 : System Architecture – Frontend - User Page](#_Toc24556256)

[Diagram 19 : System Architecture – Frontend – User page – sequence](#_Toc24556257)

[Diagram 20 : System Architecture – Backend – Overall](#_Toc24556258)

[Diagram 21 : System Architecture – Backend – Login](#_Toc24556259)

[Diagram 22 : System Architecture – Backend – Login - sequence](#_Toc24556260)

[Diagram 23 : System Architecture – Backend – Review Analysis](#_Toc24556261)

[Diagram 24 : System Architecture – Backend – Review Analysis - sequence](#_Toc24556262)

[Diagram 25 : System Architecture – Backend – Review Analysis – State](#_Toc24556263)

[Diagram 26 : System Architecture – Backend – Similar Item Finder](#_Toc24556264)

[Diagram 27 : System Architecture – Backend – Similar Item Finder - sequence](#_Toc24556265)

[Diagram 28 : System Architecture – Backend – 상품 정보 제공](#_Toc24556266)

[Diagram 29 : System Architecture – Backend – 상품 정보 제공 – sequence](#_Toc24556267)

[Diagram 30 : System Architecture – Backend – 개인 정보 변경](#_Toc24556268)

[Diagram 31 : System Architecture – Backend – 개인 정보 변경 – sequence](#_Toc24556269)

[Diagram 32 : ER Diagram](#_Toc24556270)

[Diagram 33 : ER Diagram - User](#_Toc24556271)

[Diagram 34 : ER Diagram - Item](#_Toc24556272)

[Diagram 35 : ER Diagram - Review](#_Toc24556273)

[Diagram 36 : ER Diagram - Keyword](#_Toc24556274)

11. References

김준현, 유도영, 최지혜, 허준범, 모하메드, "requirement.pdf", Github, 2019. 6. https://github.com/skkuse/2019spring\_41class\_team5/blob/master/docs/requirement.pdf

김준현, 유도영, 최지혜, 허준범, 모하메드, "system-design.pdf", Github, 2019. 6. https://github.com/skkuse/2019spring\_41class\_team5/blob/master/docs/system-design.pdf