# Design Specification


Team 1
2016311264 김준우
2017310463 최재익
2017312780 유찬우
2017314819 장윤진
2019314623 장채윤
2019314352 김이지

# 목차

# 그림 목차

# 표 목차

# 1. Introduction

## 1.1. Objectives

본 문서는 파이썬 교육 시스템을 설계 및 구현하기 위한 요구사항 명세서이며 SWE3002-41분반 Team 1에 의해 운용된다. 본 문서에서 파이썬 교육 시스템을 위한 요구사항을 정리, 분석, 기재한 내용을 바탕으로 시스템을 설계 및 구현한다.

본 문서는 전반적인 요구 사항을 서술하여 완성도 높은 웹 서비스를 개발하고 발전된 서비스를 제공하는 것을 최우선 목적으로 한다.

본 코딩 교육 프로그램의 목적은 사용자들에게 양질의 코딩 문제들을 전달함과 동시에 이를 웹상에서 풀어 구체적으로 평가받을 수 있는 인터페이스를 제공함으로써 비전공자들이 코딩에 대한 보다 발전된 역량을 기르도록 돕는 것이다.

본 문서는 이전에 쓰여진 Requirements specification의 내용을 기반으로 쓰여졌다.

## 1.2. Applied Diagrams

### 1.2.1. UML

UML은(Unified Modeling Language)는 소프트웨어 공학에서 사용되는 표준화된 범용 모델링 언어이다. 이 표준은 UML을 고안한 객체 관리 그룹에서 관리 하고 있다. UML은 소프트웨어 집약 시스템의 시각적 모델을 만들기 위한 도안 표기법을 포함한다.

UML은 원래 소프트웨어 설계에 대한 상이한 표기 체계와 접근 방식을 표준화하기 위해 만들어졌다. 1994년부터 1995년까지 Rational Software에서 개발되었으며 1996년까지 개발을 주도하였다.

1997년, UML은 객체 관리 그룹(OMG)에 의해 표준으로 채택되었고, 그 이후로 이 조직에서 관리되고 있다. 2005년에는 국제 표준화 기구(ISO)가 승인한 ISO 표준으로 UML을 발표하였다.

### 1.2.2. Use Case Diagram

Use case diagram은 시스템과 사용자의 가능한 상호 작용을 그래픽으로 묘사한 것이다. Use case diagram은 시스템이 가지고 있는 다양한 사용 사례와 다양한 유형의 user를 보여주며 종종 다른 유형의 다이어그램과 함께 제공된다. use case는 원 또는 타원으로 표시된다. actor들은 막대기로 표시된다.

use case 자체는 모든 가능성에 대해 많은 세부 정보를 제공할 수 있지만, use case diagram은 시스템에 대한 더 높은 수준의 보기를 제공하는 데 도움이 될 수 있다.

use case diagram은 단순하기 때문에 stakeholders에게 좋은 커뮤니케이션 도구가 될 수 있다. diagram은 stakeholders이 시스템이 어떻게 설계되는지에 대해 이해하기 쉽게 해준다.

### 1.2.3. Sequence Diagram

Sequence diagram은 소프트웨어 엔지니어링 분야에서 시간 순서대로 정렬된 프로세스 상호 작용을 보여준다. 관련 프로세스와 기능 수행에 필요한 프로세스 간에 교환되는 메시지의 순서를 나타낸다. sequence diagram은 일반적으로 개발 중인 시스템의 4+1 아키텍처 뷰 모델에서 use case realizations과 관련이 있다. sequence diagram은 event digram 또는 event scenario 라고도 한다.

use case의 특정 시나리오의 경우, diagram은 외부 행위자가 생성하는 이벤트, 순서 및 가능한 시스템 간 이벤트를 보여준다. 모든 시스템은 블랙박스로 취급되며, 다이어그램은 행위자에서 시스템으로 시스템 경계를 가로지르는 이벤트에 중점을 둡니다. 사용 사례의 주요 성공 시나리오와 빈번하거나 복잡한 대안 시나리오에 대해 시스템 시퀀스 다이어그램을 수행해야 한다.

### 1.2.4. Class Diagram

통합 모델링 언어의 클래스 다이어그램(class diagram)은 소프트웨어 공학에서 시스템의 클래스, 속성, 작업(또는 방법) 및 객체 간의 관계를 보여줌으로써 시스템의 구조를 설명하는 static structure diagram의 한 유형이다.

class diagram은 object-oriented modeling의 주요 구성 요소이다. 응용 프로그램 구조의 일반적인 개념 및 세부 모델링에 사용되며, 모델을 프로그래밍 코드로 변환한다. class diagram은 데이터 모델링에도 사용할 수 있다. class diagram의 클래스는 주요 요소, 응용프로그램의 상호 작용 및 프로그래밍된 클래스를 모두 나타낸다.

diagram에서 클래스는 세 개의 칸으로 구성되어 있다. 맨 위 칸에는 클래스 이름이 들어 있고 가운데 칸에는 클래스의 속성이 포함되어 있다. 마지막으로 아래 칸에는 클래스가 실행할 수 있는 작업이 포함되어 있다.

시스템 설계과정에서 많은 클래스들이 식별되고 클래스들 사이의 static relations를 결정하는 데 도움이 되는 클래스 다이어그램으로

함께 그룹화된다. 상세 모델링에서 개념 설계의 클래스는 종종 하위 클래스로 분할된다.

시스템의 동작을 더 자세히 설명하기 위해, 이러한 class diagram을 보완하기 위해 state diagram 또는 UML state machine을 이용할 수 있다.

### 1.2.5. Context Diagram

Context diagram은 시스템 또는 시스템의 일부와 그 환경 사이의 경계를 정의하는 다이어그램으로, 시스템과 상호 작용하는 entity를 보여준다.

### 1.2.6. Entity Relationship Diagram

enhanced entity–relationship model(EER)은 컴퓨터 과학에서 원래 entity–relationship model의 확장을 포함하는 높은 수준의 개념적 데이터 모델이며, 데이터베이스 설계에 사용된다.

엔지니어링 설계 및 제조(CAD/CAM), 통신, 복잡한 소프트웨어 시스템 및 지리 정보 시스템과 같이 보다 복잡한 데이터베이스에서 발견되는 속성과 제약 조건을 보다 정확하게 반영하기 위해 개발되었다.

## 1.3. Applied tools

### 1.3.1. Visual studio code

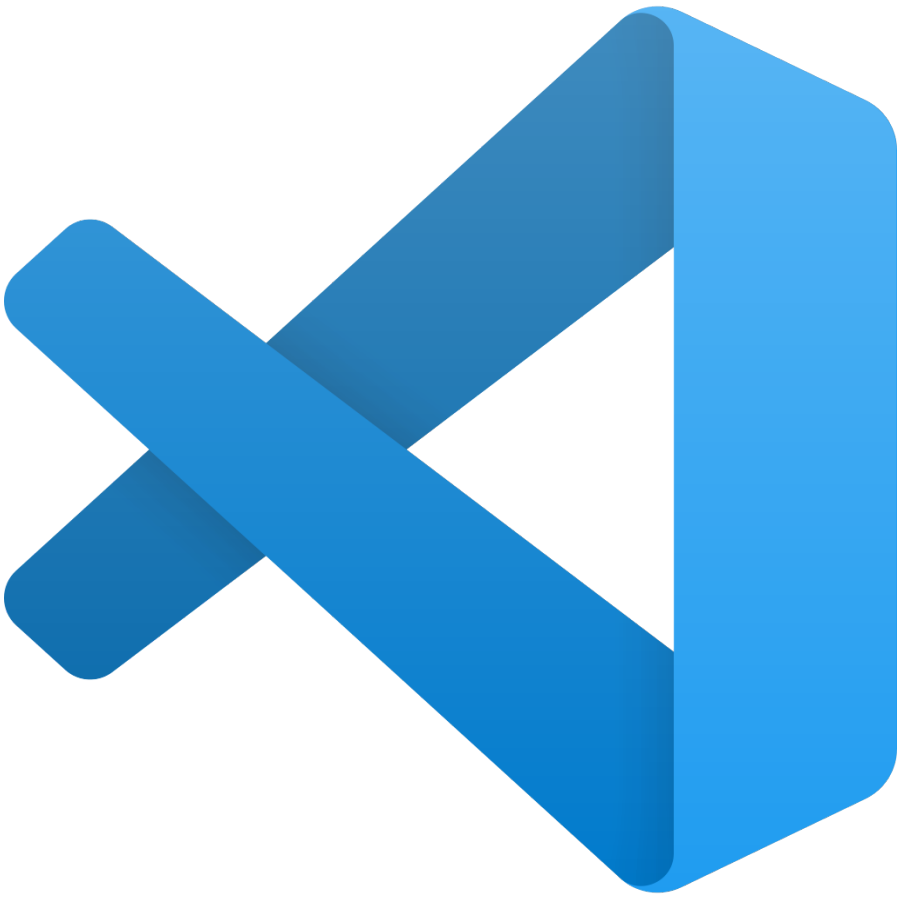

그림 1.1 - Visual studio code logo

마이크로소프트에서 개발한 텍스트 에디터이다. Electron 프레임워크를 기반으로 만들었다. Visual Studio Team Services(현 Azure DevOps)에 있던 웹 에디터를 발전시켜 Electron 프레임워크를 통해 로컬에서 쓸 수 있게 만든 것에서 출발했다. MS의 개발 툴 중 최초로 크로스 플랫폼을 지원하는 에디터이며 윈도우, macOS, 리눅스를 모두 지원한다.

### 1.3.2. Draw.io

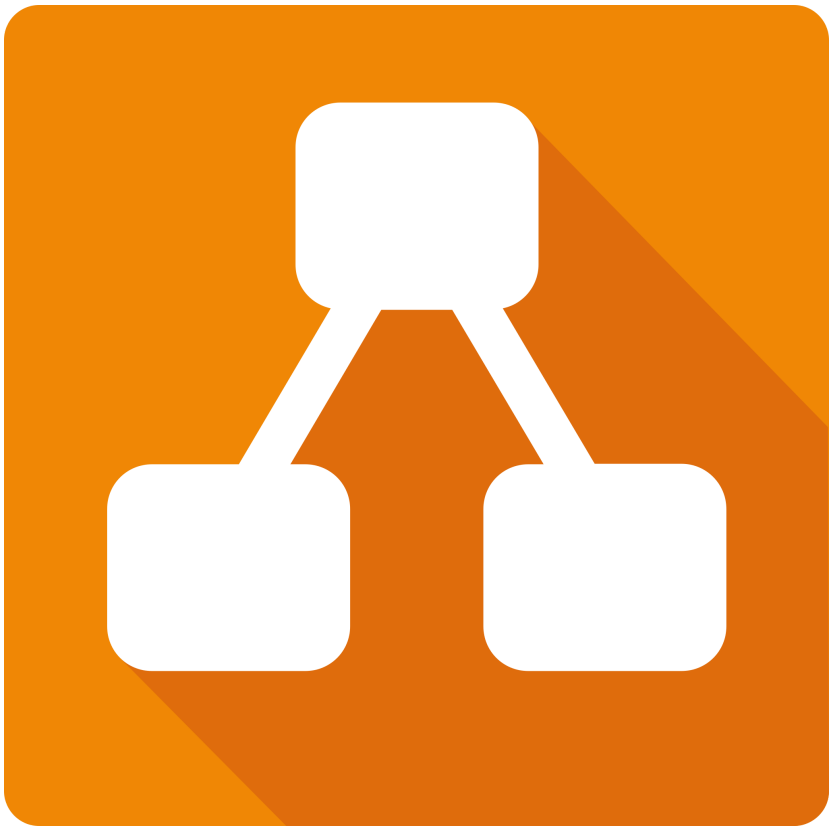

그림 1.2 - Draw.io logo

HTML5와 자바스크립트로 개발된 무료, 오픈 소스, 크로스 플랫폼을 지원하는 그래프 그리기 소프트웨어이다. draw.io는 플로우 차트, 와이어 프레임, UML 다이어그램, 조직 차트 및 네트워크 다이어그램과 같은 다이어그램을 만드는 데 사용할 수 있다.

draw.io는 크로스오버 웹 앱으로 온라인으로도 이용할 수 있으며, 리눅스, macOS, 윈도용 오프라인 데스크톱 애플리케이션으로도 이용할 수 있다. draw.io의 오프라인 애플리케이션은 전자 프레임워크를 사용하여 구축된다. 웹 앱은 온라인 로그인 또는 등록이 필요하지 않으며 로컬 하드 드라이브에서 열고 저장할 수 있다.

다운로드할 수 있는 저장소 및 내보내기 형식에는 PNG, JPEG, SVG 및 PDF가 있다.

### 1.3.3. Git

그림 1.3 - Git logo

Git은 컴퓨터 파일의 변경사항을 추적하고 여러 명의 사용자들 간에 해당 파일들의 작업을 조율하기 위한 스냅샷 스트림 기반의 분산 버전 관리 시스템이다. 또는 이러한 명령어를 가리킨다. 소프트웨어 개발에서 소스 코드 관리에 주로 사용되지만 어떠한 파일 집합의 변경사항을 지속적으로 추적하기 위해 사용될 수 있다. 기하학적 불변 이론을 바탕으로 설계됐고, 분산 버전 관리 시스템으로서 빠른 수행 속도에 중점을 두고 있는 것이 특징이며 데이터 무결성, 분산, 비선형 워크플로를 지원한다

Git은 2005년에 리눅스 커널 개발을 위해 초기 개발에 기여한 다른 커널 개발자들과 함께 2005년에 리누스 토르발스가 처음 개발한 것이다. 2005년부터 지금까지 Junio Hamano가 소프트웨어의 유지보수를 맡고 있다.

다른 대부분의 분산 버전 관리 시스템처럼, 또 대부분의 클라이언트-서버 시스템과 달리, 모든 노드의 모든 깃 디렉터리는 네트워크 접속이나 중앙 서버와는 독립적으로 동작하는 완전한 이력 및 완전한 버전 추적 등, 모든 기능을 갖춘 저장소이다.

### 1.3.4. Figma

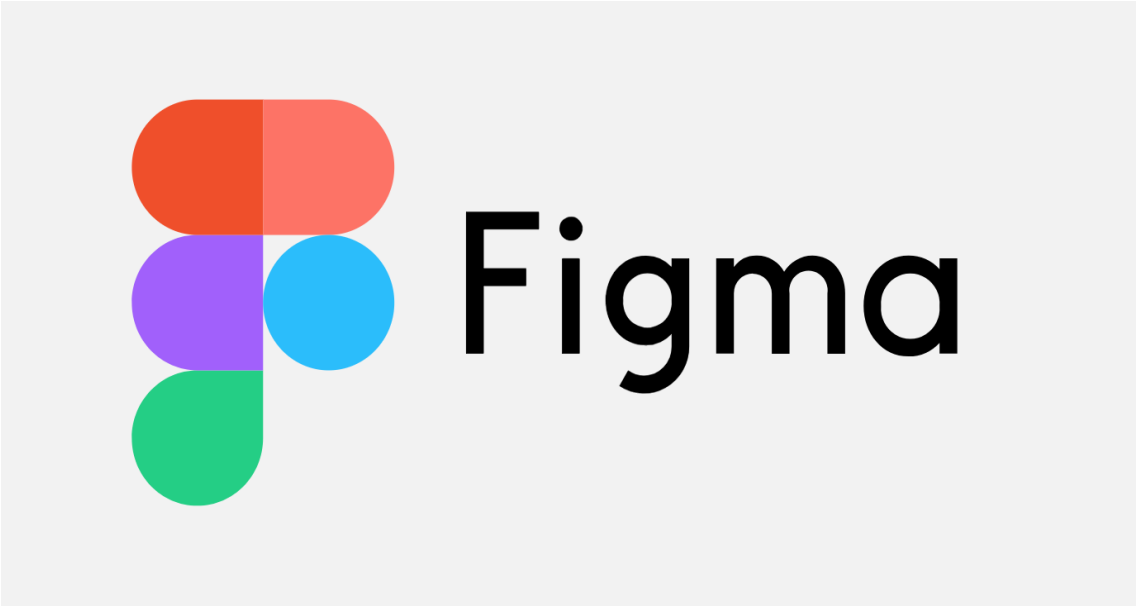


그림 1.4 - Figma logo

피그마는 인터페이스 설계를 위한 협업 웹 애플리케이션으로, 맥 OS와 윈도우용 데스크톱 애플리케이션에 의해 추가적인 오프라인 작업이 가능하다. Figma의 기능들은 사용자 인터페이스와 사용자 경험 설계에 중점을 두고 실시간 협업을 강조하며, 다양한 벡터 그래픽 편집기 및 프로토타이핑 도구를 활용한다. 안드로이드 및 iOS용 피그마 모바일 앱은 모바일 및 태블릿 기기에서 실시간으로 피그마 프로토타입을 볼 수 있다.

### 1.4. References

이 디자인 명세서는 다음과 같은 자료를 참고하였다.

- Team2_Design Description, 2022 Spring, SKKU
- Design Specification Team #3, 2022 Spring, SKKU
- Team5_Design_Specification, 2022 Spring, SKKU

# 2. System Architecture - Overall

## 2.1. Objectives

시스템의 구조를 프로젝트의 frontend와 backend를 기준으로 전체적으로 설명한다.

## 2.2. System Organization

이 서비스는 client-server architecture 모델을 적용하여 설계되었다. Frontend 애플리케이션은 사용자와의 상호작용을 위해 웹페이지를 설계하고 페이지 간, 사용자와의 Input&Output등의 작용을 설계한다. Frontend 애플리케이션과 backend 애플리케이션은 JSON 기반의 HTTP 통신을 통해 데이터를 주고 받는다. Frontend에서 Input 정보를 Backend로 넘겨주면, 이를 Backend에서 받고, 데이터베이스에서 필요한 객체 정보를 가져와서 함수를 작동시켜 그 output을 Frontend에 다시 JSON 형식으로 전달한다. Frontend가 이를 가져와서 다시 디스플레이를 통해 사용자에게 보여준다.

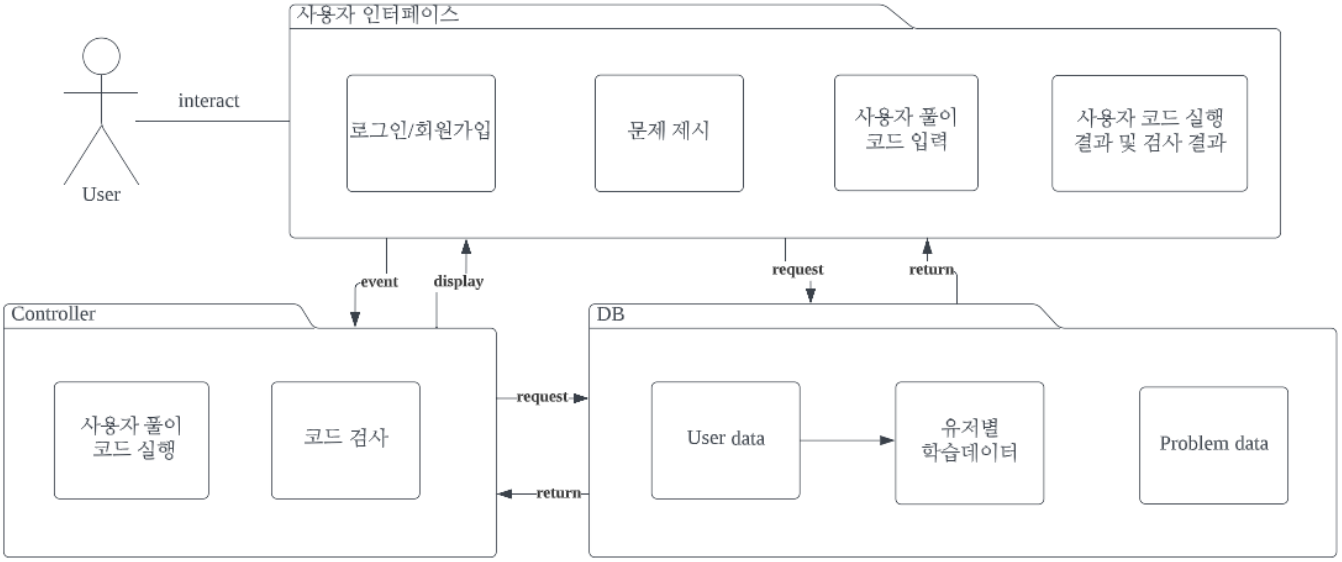


그림 2.1 - System architecture diagram

## 2.3. Context Diagram

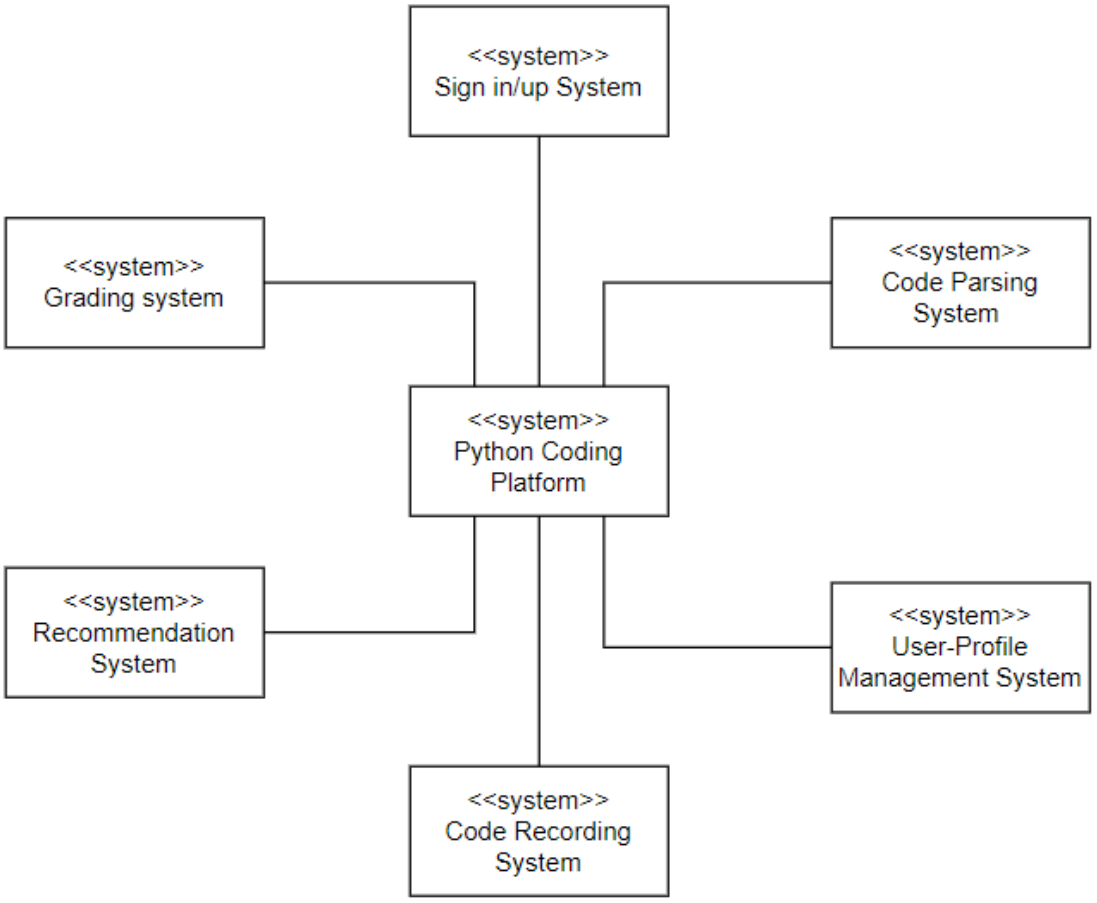


그림 2.2 - System context architecture diagram

## 2.4. Sequence Diagram

### 2.4.1. For Lecturer

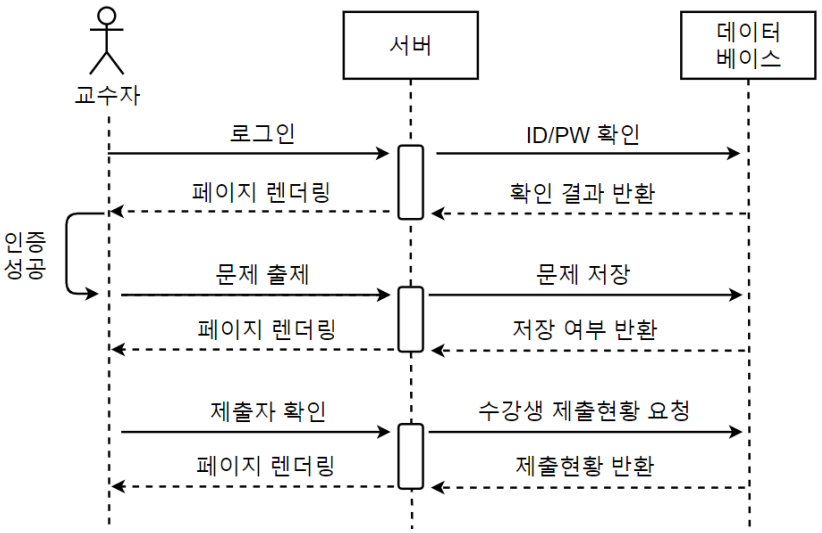


그림 2.3 - System sequence architecture diagram for lecturer

### 2.4.2. For Learner

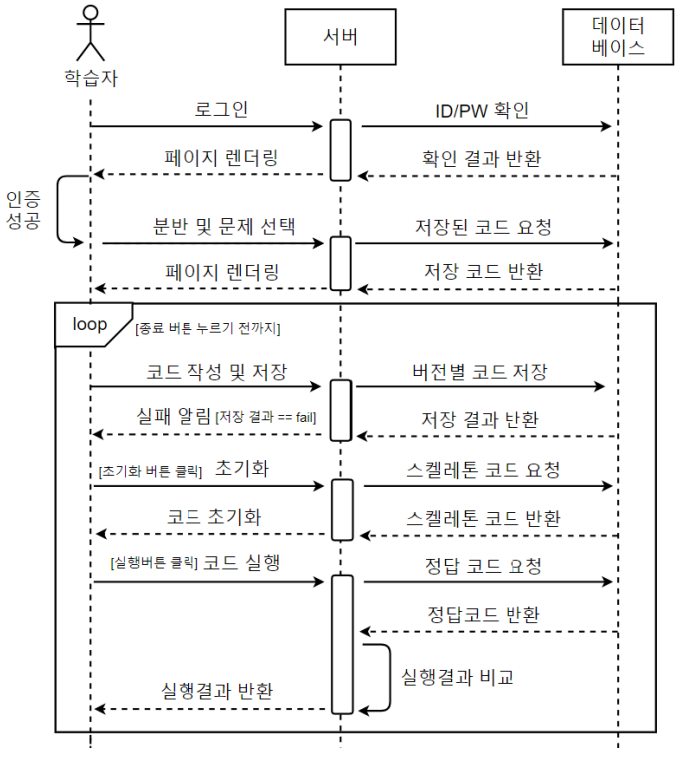


그림 2.4 - System sequence architecture diagram for learner

## 2.5. Use Case Diagram

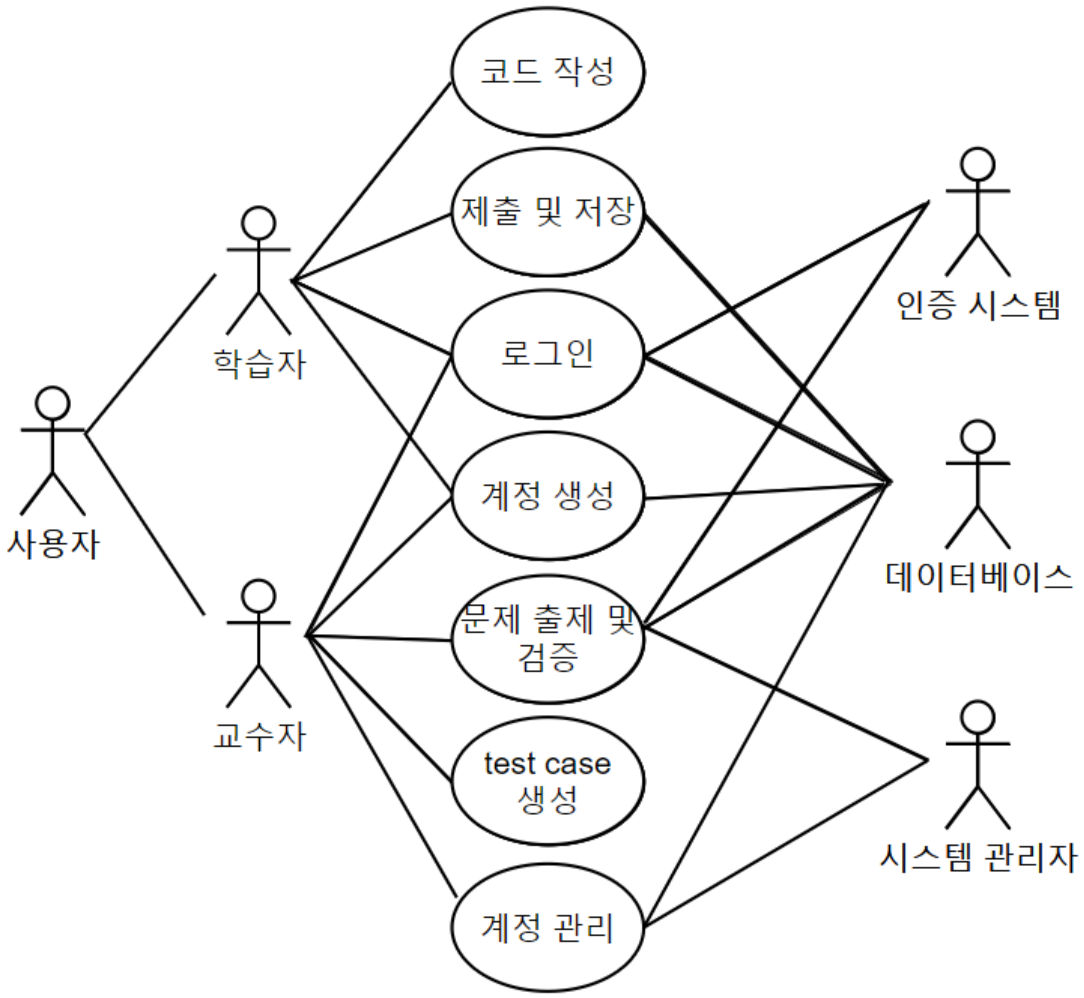


그림 2.5 - System use-case architecture diagram

# 3. System architecture - Frontend

## 3.1. Objective

이 장에서는 프론트엔드 시스템의 구조, 속성 및 기능에 관한 설명을 객체 기준으로 기술하고, 각 객체를 구성하는 각 요소들의 관계를 설명한다.

## 3.2. Overall structure

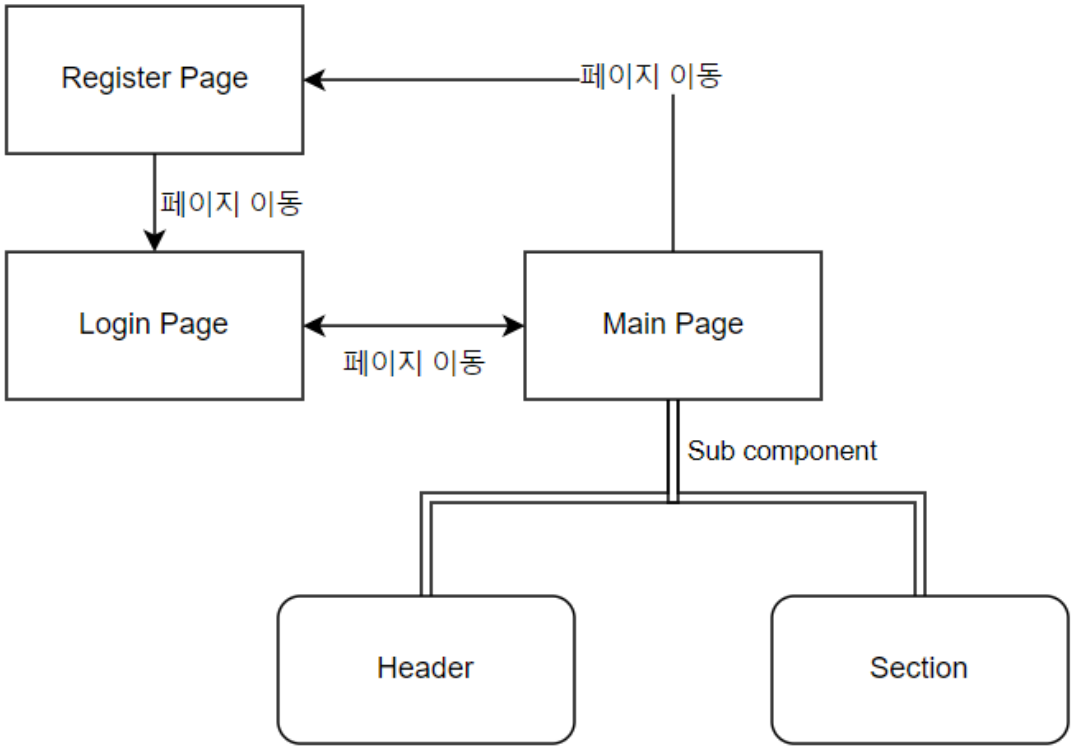


그림 3.1 - Overall structure diagram

## 3.3. Sub components

### 3.3.1. Login

#### A. LoginPage

- **Field**

Email:
사용자 계정인 이메일 주소이다.

Password:
사용자 계정인 비밀번호이다.

LoginMsg:
로그인이 실패하였을 때 뜨는 오류 메시지이다.

- **Methods**

setEmail(), setPassword():
e.currentTarget.value로 value를 바꿔준다.

onEmailHandler(), onPasswordHandler():
onChange event를 발생시켜 state를 변경한다.

onSubmitHandler():
로그인 button을 클릭하였을 때 onSubmit event를 발생시킨다.

loginUser():
axios를 통해 request를 진행한다.

### B. Register Page

- **Field**

Email: 사용자 계정인 이메일 주소이다.

Password: 사용자 계정인 비밀번호이다.

PasswordConfirmed: 비밀번호 설정 확인 창이다.

Disable, PmMsg:
회원가입 오류 메시지이다.

- **Methods**

setEmail(), setPassword(), setPasswordConfirmed():
e.currentTarget.value로 value를 바꿔준다.

setDisable():
이메일 입력값의 '@' 포함 여부에 따라 설정된다.

setPmMsg():
비밀번호 입력값의 길이 제한에 따라 설정된다.

onEmailDisableHandler(), onPasswordHandler(),
onPasswordConfirmHandler():
onChange event를 발생시켜 state를 변경한다

onSubmitHandler():
회원가입 button을 클릭하였을 때 onSubmit event를 발생시킨다.
useDispatch를 통해 action인 user_action의 registerUser로 data를 전달한다.

registerUser():
axios를 통해 request를 진행한다.

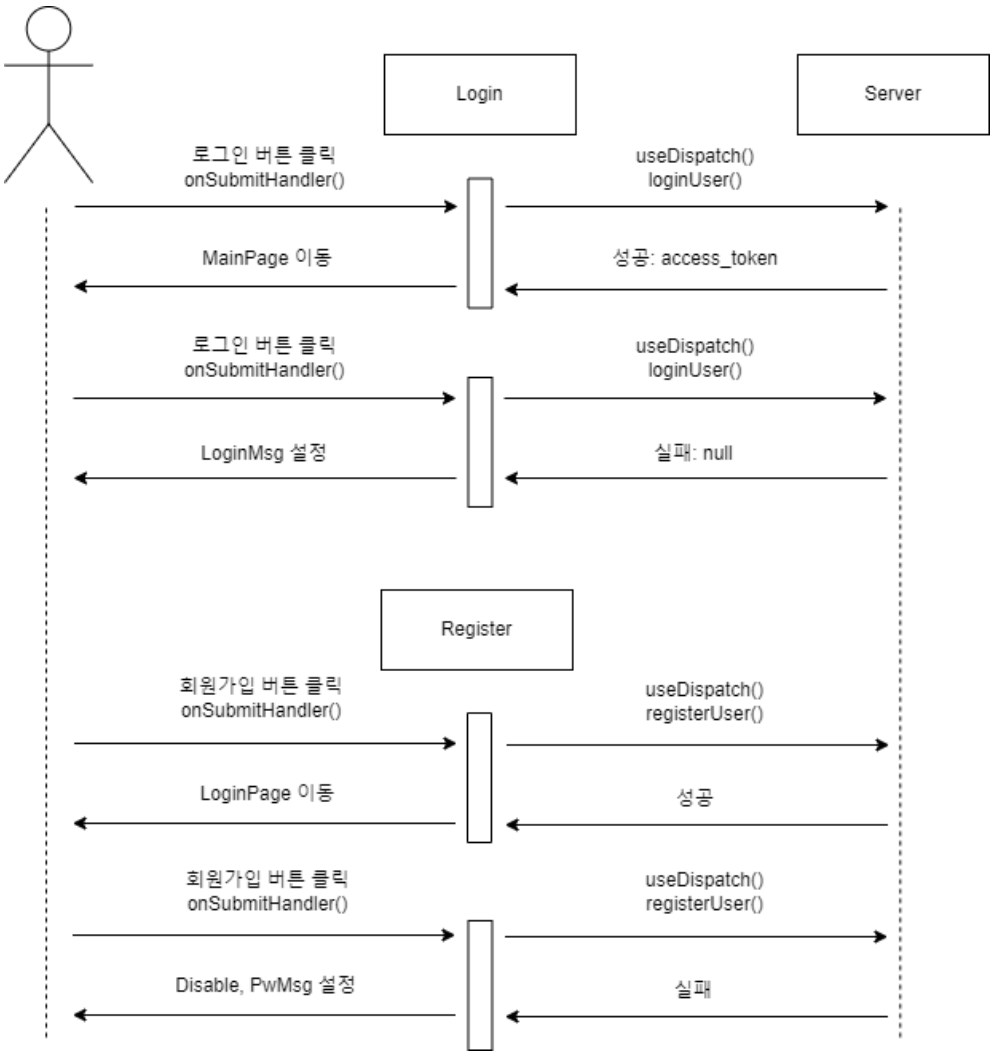


그림 3.2 - Login Page Sequence Diagram

### 3.3.2. MainPage

**A. Header &** 공통

- Question 정보 안내 및 기타 페이지 이동, Question number 변경
- **Field**
  userEmail: 사용자 정보. 이메일을 기준으로 사용자를 구분한다.
  questionNo: Backend로부터 가져올 question index. 사용자는 해당 번호의 question을 보고 문제를 풀게된다.

**B. Question Display**

- **Field**

  questionText: Backend로부터 가져온 question의 텍스트 정보
  testCases: question과 함께 가져온 test cases들의 정보

- **Methods**

  getQustion(): Backend로부터 question을 받아와서 questionText 및 testCases에 정보들을 저장하는 함수.
  displayQuestion(): Question항목에 questionText 정보를 유저에게 보여주는 함수.
  displayTestCases(): testCase 항목에 testCases 정보를 유저에게 보여주는 함수.

**C. Python Editor**

- **Fields**

  monaco: 파이썬 에디터, 파싱하여 파이썬의 형태로 디스플레이 해주는 함수. npm에서 제공해주는 것을 사용한다.

- **Methods**

  monacoMountHandler(): 모나코 에디터를 마운트하여 처리하는 작업을 하는 함수. 에디터의 텍스트를 가져온다.
  sendValue(): 에디터의 현재 텍스트를 Backend로 전송한다.

**D. Result Display**

- **Fields**

  resultText: Backend로 부터 받아올 결과 텍스트 정보

- **Methods**

  getResult(): Backend로 부터 editor를 실행하고 난 결과를 받아오는 함수.
  displayResult(): resultText에 저장된 result 정보를 유저에게 보여주는 함수.

**E. Evaluation Display**

- **Fields**

  correctAnswer: Backend로 부터 받아올 문제의 정확한 답 코드 텍스트 정보.
  evaluation: Backend로 부터 받아올 사용자의 코드 평가 정보.

- **Methods**

  getEvaluation(): Backend로 부터 evaluation과 correctAnswer및 기타 정보를 가져오는 함수.

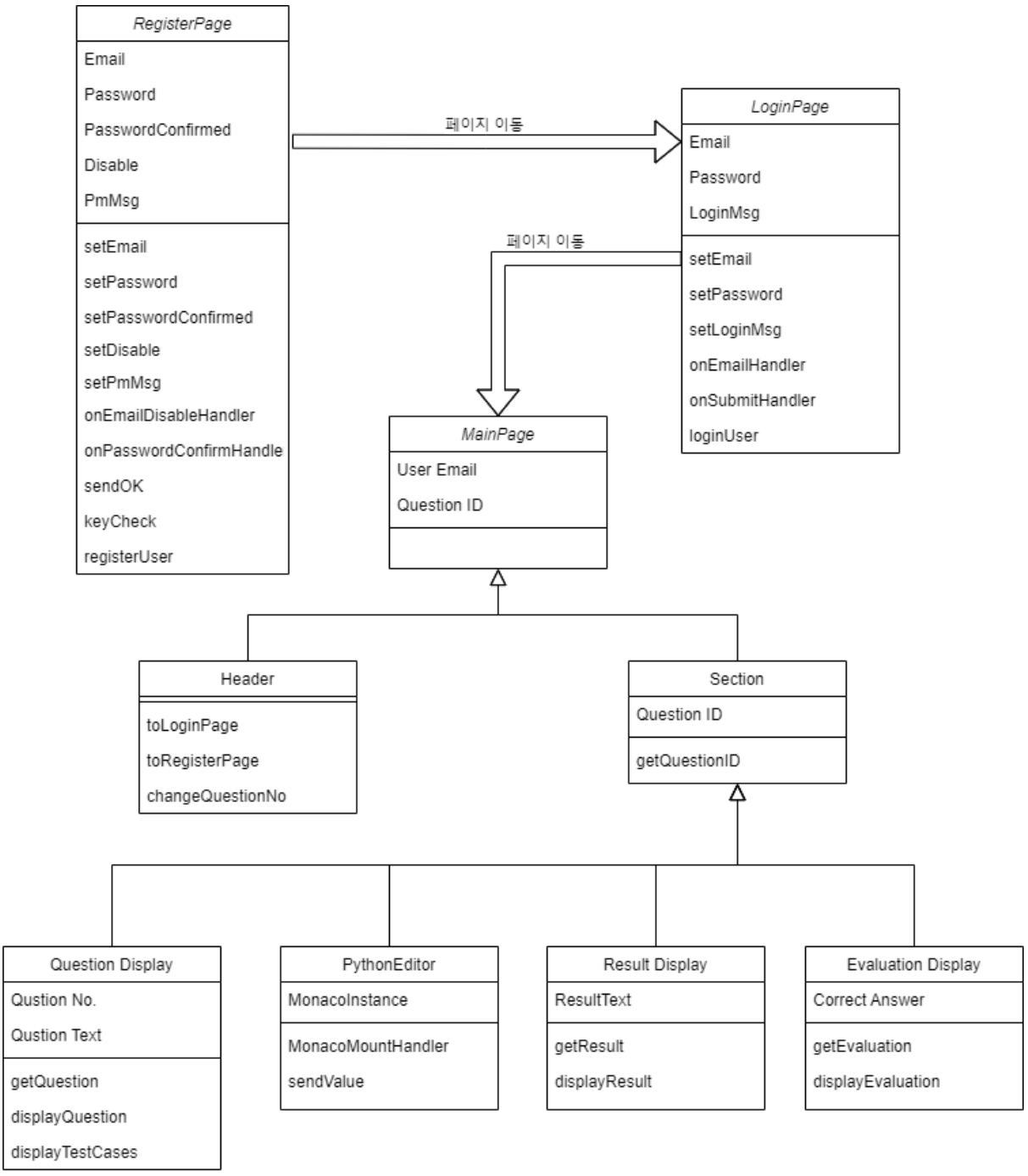


그림 3.3 - Main Page Class Diagram

.

# 4. System architecture - Backend

## 4.1. Objective

이번 장에서는 DB와 서버의 구조를 설명한다.

## 4.2. Overall Architecture

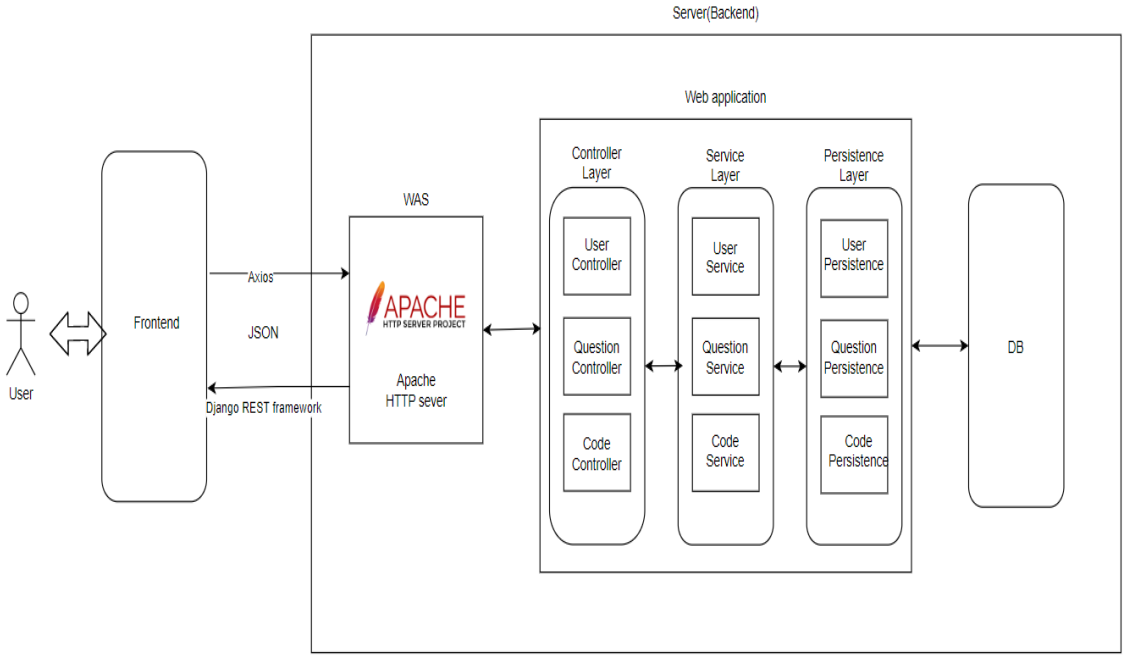


그림 4.1 - Overall architecture diagram

서버의 전체 구조는 다음과 같다. 먼저 Frontend 에서 axios를 통해 API를 호출하면 Apache HTTP Server를 통해 web application 에서 기능이 호출이 된다. Request 와 Response 를 처리하는 Controller layer, 비지니스 로직을 처리하는 Service Layer, 마지막으로 DB 와의 통신을 담당하는 Persistence Layer 로 구성되어있다.

## 4.3. Subcomponents

### 4.3.1. User System

System User에 관련된 작업들이 묶여져 있는 System 이다.

#### A. User System Class Diagram

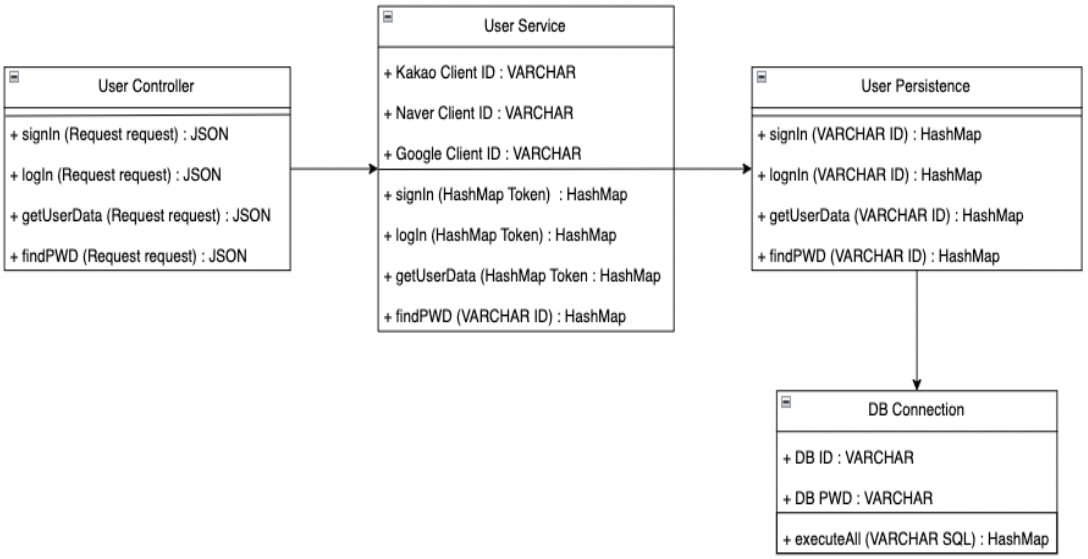


그림 4.2 - Class diagram - user system

- **User Controller Class**

HTTP Server 로 부터 받은 Request 에서 Parameter 를 받아서 이를 HashMap 으로 Parsing 한 다음 Service Layer를 호출하는 역할을 가지고 있다. 해당 Class는 4가지 Method가 존재한다. 첫번째는 회원가입 Method인 singIn, 두번째로는 로그인 Method인 logIn 그리고 세번째는 사용자의 정보를 가져오는 getUserData, 네번째는 사용자의 비밀번호를 찾을 수 있는 findPWD가 있다. 해당 Method들은 모두 JSON 파일을 반환하게 되어 있다.

- **User Service Class**

Controller Layer 로 부터 받은 Parameter 를 Parsing 하고 Persistence Layer 를 Request 하고 해당 Layer 에서 반환된 값을 Controller 에 넘겨주는 역할을 한다. User Service Class 에는 마찬가지로 4가지 Method가 있다. 회원가입과 관련된 signIn, 로그인과 관련된 logIn. 사용자의 정보를 가져오는 getUserData 이다. 마지막으로 사용자의 비밀번호를 찾을 수 있는 finePWD가 있다. 각 Method 가 받는 Parameter 는 Token 이다. 해당 Token 은 OAuth 서버로 부터 받아온 것이다. 이들은 아래 레이어에서 받은 Hashmap 을 controller 에게 넘겨준다.

- **User Persistence Class**

Service Layer 에서 받은 Parameter 를 바탕으로 SQL 문을 만들어서 DB Connection 에서 넘겨준다. 또한 DB Connection 에서 DB 로 부터

받은 값을 HashMap 으로 Service Layer 에 넘겨주는 역할을 한다. 해당 서비스에는 총 4 가지 메소드가 존재한다. 첫번째는 회원가입과 관련된 signIn 이 존재한다. 두번째는 로그인과 관련된 loginIn 이다. 세번째는 사용자의 정보를 가져오는 getUserData 이다. 마지막은 비밀번호를 찾을 수 있는 findPWD 이다.

- **Database Connection**

DB 랑 연결을 담당하는 부분으로 Persistence Layer 로 부터 받은 SQL 을 DB 에 넘겨 결과값을 받는 역할을 한다. 이후 결과를 받으면 Persistence Layer 로 넘겨준다.

## B. User Sequence Diagram

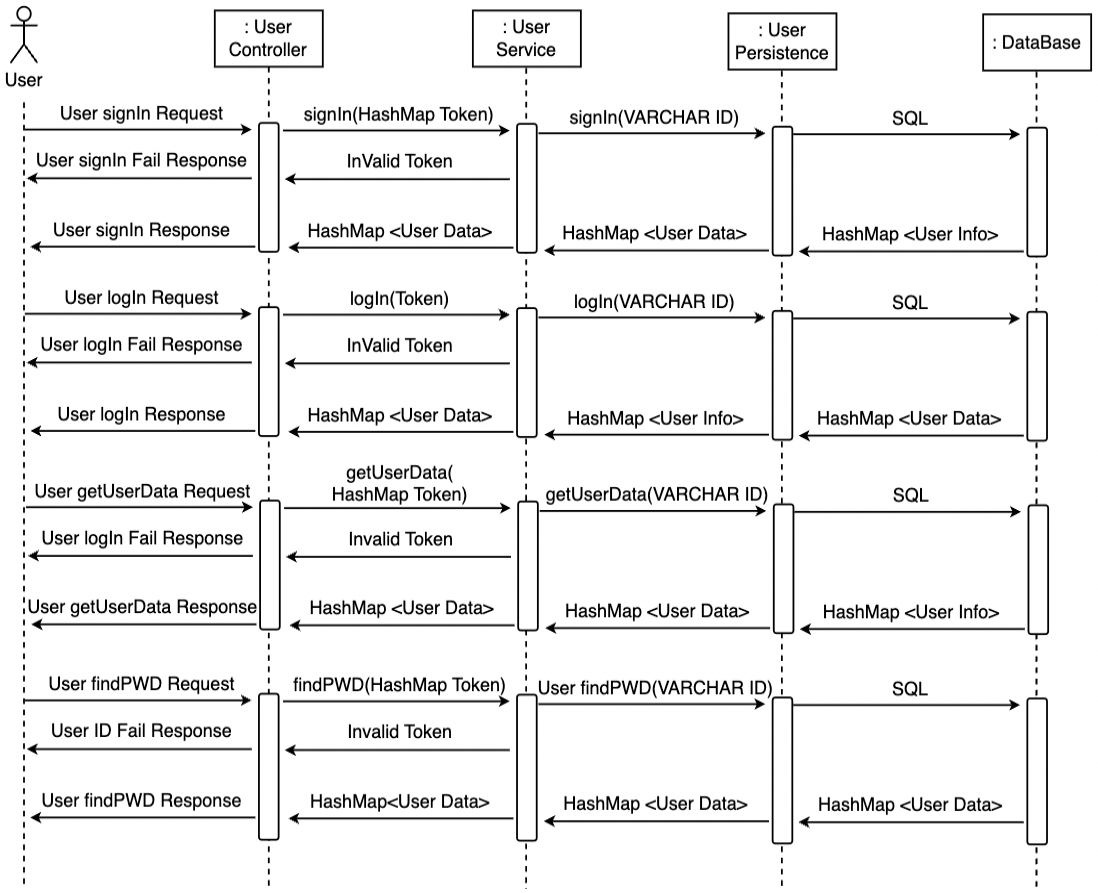


그림 4.3 - Sequence diagram - user system

## 4.3.2. Question System

System에서 Question과 관련된 작업들이 묶여져 있는 System 이다. Question을 생성하고, 조회하며, 관련된 Data를 불러오는 작업들이 포함되어 있다.

## A. Question System Class Diagram

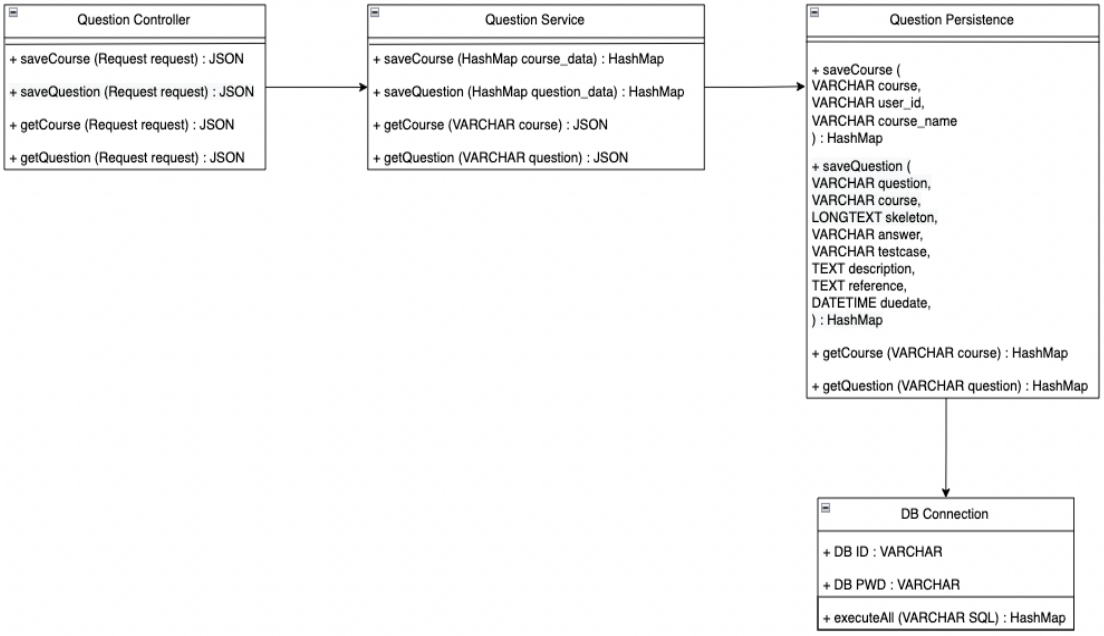


그림 4.4 - Class diagram - question system

### - Question Controller Class

HTTP Server 로 부터 받은 Request 에서 Parameter 를 받아서 이를 HashMap 으로 Parsing 한 다음 Service Layer를 호출하는 역할을 가지고 있다. 해당 Class는 4가지 Method가 존재한다. 첫번째는 새로운 강의를 등록하는 Method인 saveCourse, 두번째로는 특정 강의에 새로운 Question을 등록하는 Method인 saveQuestion 그리고 세번째는 Course 정보를 가져오는 getCourse, 네번째는 Question의 정보를 가져오는 getQuestion이 있다. 해당 Method들은 모두 JSON 파일을 반환하게 되어 있다.

### - Question Service Class

Controller Layer 로 부터 받은 Parameter 를 Parsing 하고 Persistence Layer 를 Request 하고 해당 Layer 에서 반환된 값을 Controller 에 넘겨주는 역할을 한다. Question Service Class 에는 마찬가지로 4가지 Method가 있다. 강의 등록과 관련된 saveCourse, 문제 등록과 관련된 saveQuestion. 강의를 가져오는 getCourse 이다.

마지막으로 문제를 가져오는 getQuestion. 각 Method 가 받는 Parameter 는 차례대로 course_data, question_data, course(course_id), question(question_id) 이다.

- **Question Persistence Class**

Service Layer 에서 받은 Parameter 를 바탕으로 SQL 문을 만들어서 DB Connection 에서 넘겨준다. 또한 DB Connection 에서 DB 로 부터 받은 값을 HashMap 으로 Service Layer 에 넘겨주는 역할을 한다. 해당 서비스에는 총 4 가지 메소드가 존재한다. 강의 등록과 관련된 saveCourse, 문제 등록과 관련된 saveQuestion. 강의를 가져오는 getCourse 이다. 마지막으로 문제를 가져오는 getQuestion.

- **Database Connection**

DB 랑 연결을 담당하는 부분으로 Persistence Layer 로 부터 받은 SQL 을 DB 에 넘겨 결과값을 받는 역할을 한다. 이후 결과를 받으면 Persistence Layer 로 넘겨준다.

## B. Question Sequence Diagram

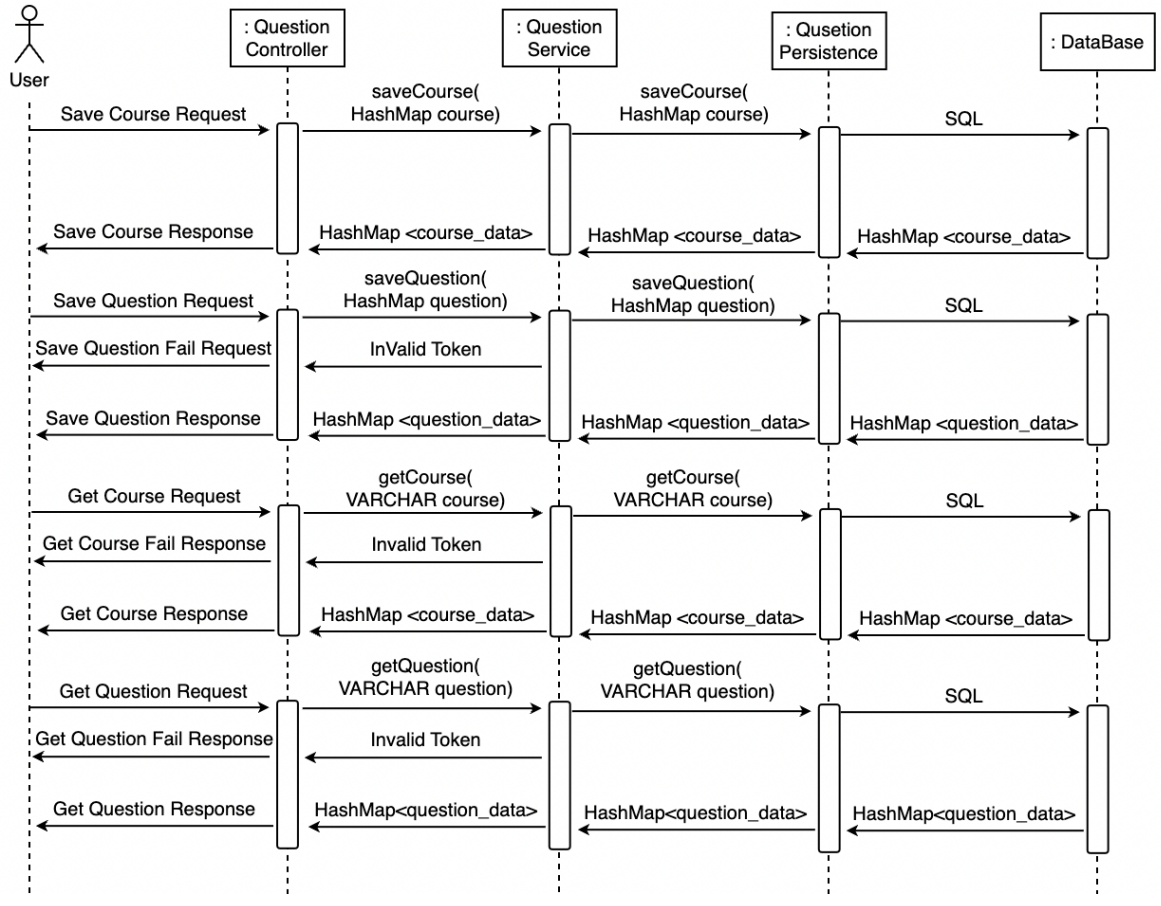


그림 4.5 - Sequence diagram - question system

### 4.3.3. Code System

System에서 User가 작성한 Code data와 관련된 작업들이 묶여져 있는 System 이다.

## A. Code System Class Diagram

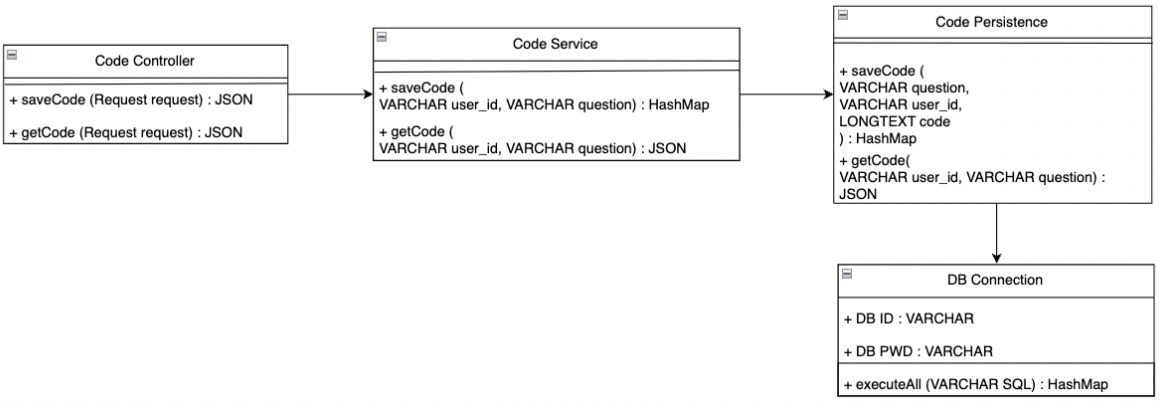


그림 4.6 - Class diagram - code system

### - Code Controller Class

HTTP Server 로 부터 받은 Request 에서 Parameter 를 받아서 이를 HashMap 으로 Parsing 한 다음 Service Layer를 호출하는 역할을 가지고 있다. 해당 Class는 2가지 Method가 존재한다. 첫번째는 작성한 코드를 저장하는 Method인 saveCode, 두번째로는 저장한 코드를 불러오는 getCode Method가 있다. 해당 Method들은 모두 JSON 파일을 반환하게 되어 있다.

### - Code Service Class

Controller Layer 로 부터 받은 Parameter 를 Parsing 하고 Persistence Layer 를 Request 하고 해당 Layer 에서 반환된 값을 Controller 에 넘겨주는 역할을 한다. Question Service Class 에는 마찬가지로 2가지 Method가 있다. 작성한 코드를 저장하는 Method인 saveCode, 저장한 코드를 불러오는 getCode Method가 있다.

### - Code Persistence Class

Service Layer 에서 받은 Parameter 를 바탕으로 SQL 문을 만들어서 DB Connection 에서 넘겨준다. 또한 DB Connection 에서 DB 로 부터 받은 값을 HashMap 으로 Service Layer 에 넘겨주는 역할을 한다. 해당 서비스에는 총 2 가지 메소드가 존재한다. 작성한 코드를 저장하는 Method인 saveCode, 저장한 코드를 불러오는 getCode Method.

- **Database Connection**

DB 랑 연결을 담당하는 부분으로 Persistence Layer 로 부터 받은 SQL 을 DB 에 넘겨 결과값을 받는 역할을 한다. 이후 결과를 받으면 Persistence Layer 로 넘겨준다.

## B. Code Sequence Diagram

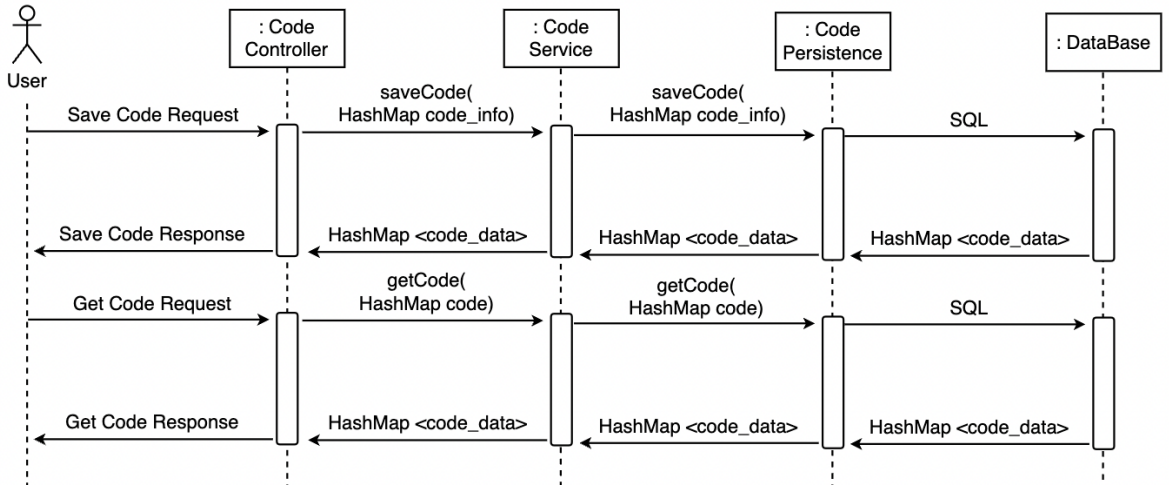


그림 4.7 - Sequence diagram - question system

# 5. Protocol

## 5.1. Overall

해당 장에서는 하위 시스템간 상호작용을 위한 필수적인 프로토콜을 기술하고 sub-system간에서 통신하는 메시지 형식과 용도에 대해서 기술하였다.

## 5.2. HTTP

HTTP(HyperText Transfer Protocol)은 W3상에서 정보를 주고받을 수 있는 프로토콜이다. HTTP는 클라이언트와 서버 사이에 이루어지는 요청/응답 프로토콜이다. 해당 시스템은 HTTP를 통하여서 클라이언트와 서버끼리 요청/응답을 진행하고 해당 요청과 응답의 형식은 아래에서 서술하는 JSON이라는 형태로 진행된다.

## 5.3. JSON

JSON(JavaScript Object Notation)은 속성-값 쌍으로 이루어진 데이터 오브젝트를 전달하기 위해 인간이 읽을 수 있는 텍스트를 사용하는 개방형 표준 포맷이다. 주로 인터넷에서 자료를 주고 받을 때 해당 자료를 표현하는 방법이다.

## 5.4. OAuth

OAuth(Open Authorization)는 인터넷 사용자들이 비밀번호를 제공하지 않고 다른 웹사이트 상의 자신들의 정보에 대해 웹사이트나 애플리케이션의 접근 권한을 부여할 수 있는 공통적인 수단으로서 사용되는, 접근 위임을 위한 개방형 표준이다. 이 매커니즘은 여러 기업들에 의해 사용되는데, 이를테면 아마존, 구글, 페이스북, 마이크로소프트, 트위터가 있으며 사용자들이 타사 애플리케이션이나 웹사이트의 계정에 관한 정보를 공유할 수 있게 허용한다.

## 5.5. Protocol Description

### 5.5.1. Overview

이 장에서는 프로토콜의 형식을 표로 나타내었다.

### 5.5.2. Authentification

해당 절에서는 client와 서버간 응답 및 요청 할때 주고 받는 구체적인 메시지를 프로토콜의 구현 기능별로 서술하였다.

- Signin/Signup Protocol(OAuth)

| GET | api/signin | | |
|---|---|---|---|
| 사용자가 다른 사이트 정보로 로그인/회원가입할 때 요청하는 주소 | | | |
| 분류 | 항목명 | 설명 및 제약사항 | 예시 |
| Parameter | user_email | 사용자의 이메일 | |
| | redirect url | 리다이렉트 uri | |
| | client_id | 사용자 정보 | |
| Response | access token | OAuth 접근 토큰 | |
| | message | “로그인 실패" | |
| | status code | 상태 코드 | |
| Status Code | 200 | 로그인 성공 | |
| | 400 | 로그인 실패 | |

표 5.1 - Signin/Signup protocol(OAuth)

- Signin Protocol(JSON)

| GET | api/signin | | |
|---|---|---|---|
| 사용자가 로그인할 때 요청하는 주소 | | | |
| 분류 | 항목명 | 설명 및 제약사항 | 예시 |
| Parameter | user_id | 사용자의 아이디 | ‘jcy9911’ |
| | user_password | 사용자의 비밀번호 | ‘123456a’ |
| Response | message | “로그인 실패" | |
| | status code | 상태 코드 | |
| Status Code | 200 | 로그인 성공 | |
| | 400 | 로그인 실패 | |

표 5.2 - Signin protocol(JSON)

- Signup Protocol

| POST | api/signup | | |
|---|---|---|---|
| 사용자가 회원가입할 때 요청하는 주소 | | | |
| 분류 | 항목명 | 설명 및 제약사항 | 예시 |
| Parameter | user_id | 사용자의 아이디 | ‘jcy9911’ |
| | user_pwd | 사용자의 비밀번호 | ‘123456a’ |
| | user_type | 사용자 타입 | ‘student’ |
| | user_org | 사용자 소속 | 성균관대학교 |
| | uesr_name | 사용자의 이름 | |
| Response | message | “회원가입 실패" | |
| | status code | 상태 코드 | |
| Status Code | 201 | 계정 생성 성공 | |
| | 400 | 회원가입 실패 | |

표 5.3 - Signup protocol

### 5.5.3. Problem

- Problem Load Protocol

| GET | api/problem | | |
|---|---|---|---|
| 사용자가 문제를 불러올때 요청하는 주소 | | | |
| 분류 | 항목명 | 설명 및 제약사항 | 예시 |
| Parameter | course | 수업 제목 | ‘소프트웨어공학개론’ |
| | question | 문제 제목 | ‘히노이의 탑' |
| Response | description | 문제 설명 | |
| | reference | 제약 사항 | |
| | testcase | 테스트 케이스 | |
| | skeleton | 스켈레톤 코드 | |
| | hint | 문제 힌트 | |
| | duedate | 제한 시간 | |
| | status code | 상태 코드 | |

| | | | |
|---|---|---|---|
| Status Code | 200 | 문제 불러오기 성공 | |
| | 204 | 해당 문제 없음 | |

표 5.4 - Problem load protocol

### 5.5.4. Code

- Code Validation Protocol

| POST | api/problem/validation/{testcase_num} | | |
|---|---|---|---|
| 사용자가 코드를 검증할 때 요청하는 주소 | | | |
| 분류 | 항목명 | 설명 및 제약사항 | 예시 |
| Parameter | code | 코드 | |
| | testcase_num | 테스테 케이스 번호 | |
| Response | output_data | 사용자의 출력값 | |
| | status code | 상태 코드 | |
| Status Code | 200 | 테스트 검증 성공 | |

표 5.5 - Code validation protocol

- Code Execute Protocol

| POST | api/problem/test | | |
|---|---|---|---|
| 사용자가 코드를 실행할 때 요청하는 주소 | | | |
| 분류 | 항목명 | 설명 및 제약사항 | 예시 |
| Parameter | code | 코드 | |
| Response | compile_result | 실행 결과 | |
| | error_line | 에러 위치 | |
| | error_result | 에러 내용 | |
| | status code | 상태 코드 | |
| Status Code | 200 | 코드 실행 성공 | |

표 5.6 - Code execute protocol

- Code Grading Protocol

| POST | api/problem/grade | | |
|---|---|---|---|
| 사용자가 코드를 채점할 때 요청하는 주소 | | | |
| 분류 | 항목명 | 설명 및 제약사항 | 예시 |
| Parameter | code | 코드 | |
| Response | testcase_num | 테스트 케이스 순서 | |
| | test_result | 테스트 통과 여부 | |
| | hidden | 히든 테스트 여부 | |
| | user_answer | 사용자 코드 결과 | |
| | correct_answer | 정답 값 | |
| | status code | 상태 코드 | |
| Status Code | 200 | 코드 채점 성공 | |

표 5.7 - Code grading protocol

- Code Submission Protocol

| POST | api/problem/submit | | |
|---|---|---|---|
| 사용자가 코드를 채점할 때 요청하는 주소 | | | |
| 분류 | 항목명 | 설명 및 제약사항 | 예시 |
| Parameter | code | 코드 | |
| Response | exe_result | 채점 결과 | |
| | grade_result | 기능 점수 | |
| | cheat_result | 표절율 | |
| | efficiency | 효율 점수 | |
| | readability | 가독성 점수 | |
| | recommend | 관련 자료 | |
| | status code | 상태 코드 | |
| Status Code | 200 | 코드 제출 성공 | |

표 5.8 - Code submission protocol

# 6. Database Design

## 6.1. Objective

해당 절에서는 시스템 데이터 구조와 데이터 구조를 데이터베이스에 표시하는 방법을 설명한다. 우선 ER-diagram(Entity Relationship diagram)을 통해 엔티티와 그 관계를 식별한다. 이후 관계형 스키마 및 SQL DDL(Data Description Language) 명세서를 생성한다.

## 6.2. EER Diagram

EER-diagram은 모델에서 테이블간의 관계를 시각적으로 표현한다. 테이블이 다른 테이블과 여러 관계가 있는 경우 이를 나타내기 위해 닻 모양이 사용된다. 테이블이 다른 테이블과 하나의 관계만 있는 경우, 십자가 모양을 사용하여 이를 나타낸다. 테이블의 속성은 테이블 안에 표현된다. 테이블을 고유하게 식별하는 키 속성은 속성의 왼쪽에 키 모양으로 표시되어 있다.

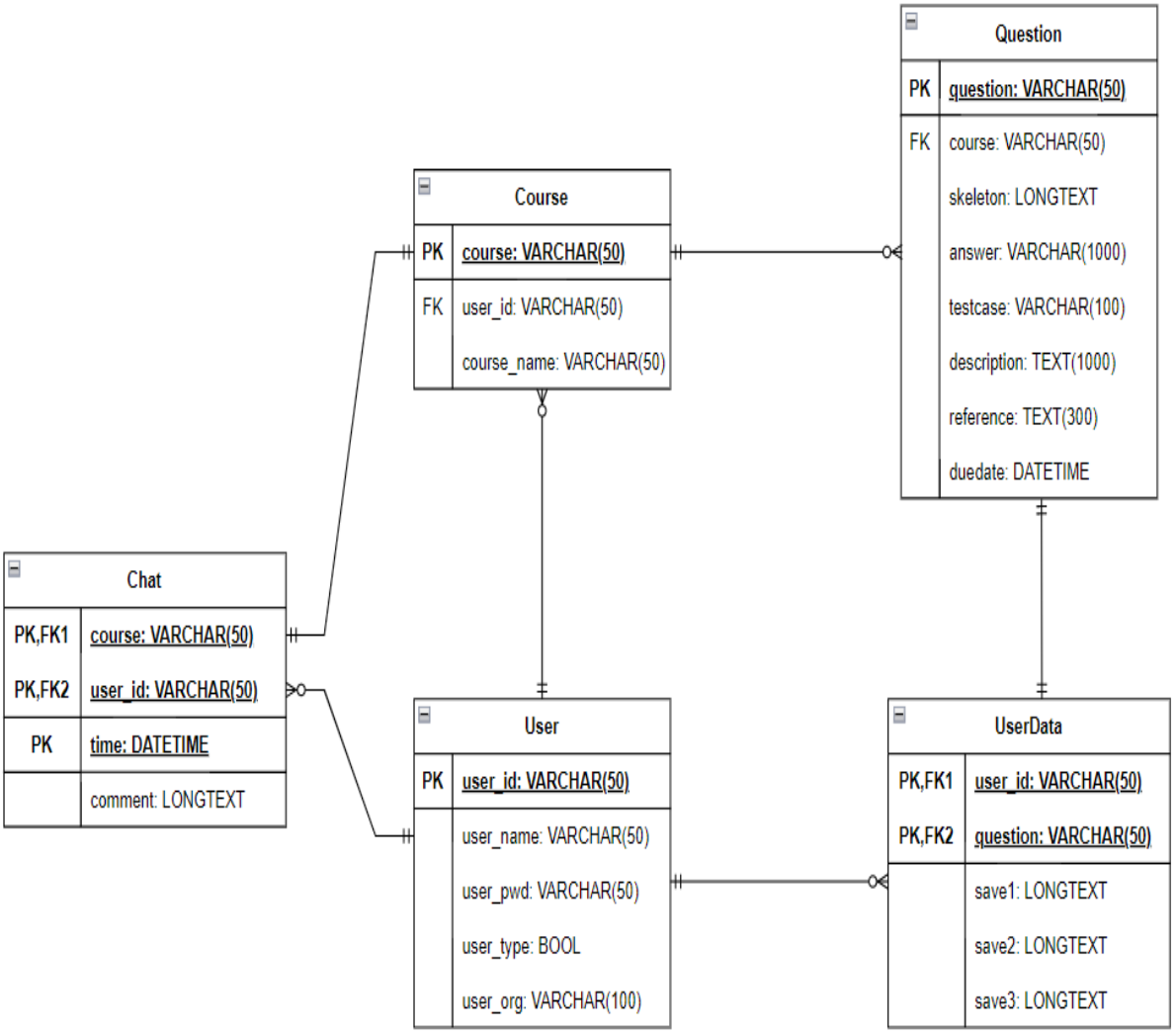


그림 6.1 - EER-Diagram

### 6.2.1. Table

## A. User

User 테이블은 본 시스템의 사용자를 나타낸다. user_id(사용자의 ID), user_name(사용자의 이름), user_pwd(사용자의 비밀번호), user_type(교수자/학습자), user_org(소속단체)로 구성되어 있다. user_id가 프라이머리 키이고, 이 필드로 Chat 테이블, Course 테이블, UserData 테이블과 1 대 다의 관계를 맺는다.

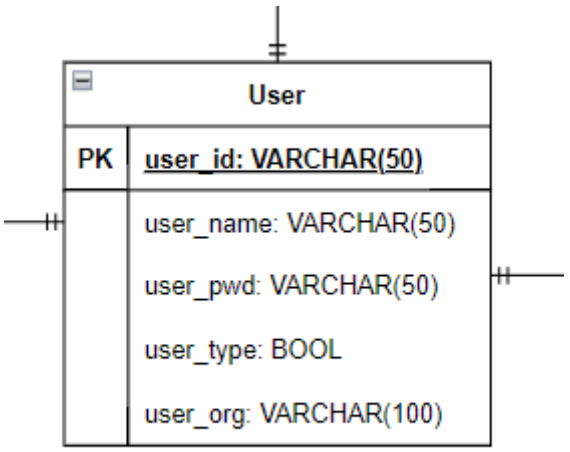


그림 6.2 - User 테이블

## B. Course

Course 테이블은 본 시스템에서 교수자 강의하는 수업을 나타내며 user_id (교수자의 아이디)와 course(과목 아이디), course_name(과목명)으로 구성되어 있다. course가 프라이머리 키이고 User 테이블에서 user_id를 외래키로 가져온다. Course 테이블은 course 필드를 이용하여 Chat 테이블과 1 대 1의 관계를, Question 테이블과 1 대 다의 관계를 맺는다.

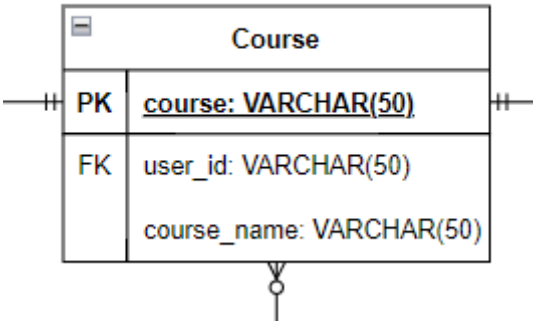


그림 6.3 - Course 테이블

## C. Chat

Chat 테이블은 본 시스템에서 과목별 채팅방 내용을 관리하며, course(과목 아이디)와 user_id(교수자의 아이디), time(입력 시간), comment(입력 내용)으로 구성되어 있다. course와 user_id는 각각 Course 테이블, User 테이블의 프라이머리 키를 외래키를 가져오며 time 필드와 함께 Chat 테이블의 프라이머리 키로 사용된다.

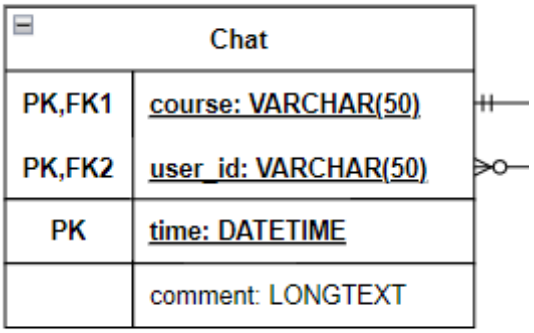


그림 6.4 - Chat 테이블

## D. Question

Question 테이블은 과목별 문제에 대한 정보를 관리하는 테이블이며, question(문제별 고유식별자), course(문제가 속한 과목 식별자), skeleton(스켈레톤 코드), answer(해당 문제의 정답), testcase(해당 문제에서 제공하는 테스트 케이스), description(문제 설명), reference(문제의 참조사항 내용), duedate(해당 문제의 제출기한) 필드로 구성되어 있다. question 필드가 프라이머리 키이며 이 필드는 과목명과 숫자의 조합으로 표현된다. 또한 Course 테이블의 course 필드를 외래 키로 가져온다. Question 테이블은 프라이머리 키를 이용하여 UserData 테이블과 1 대 1의 관계를 맺는다.

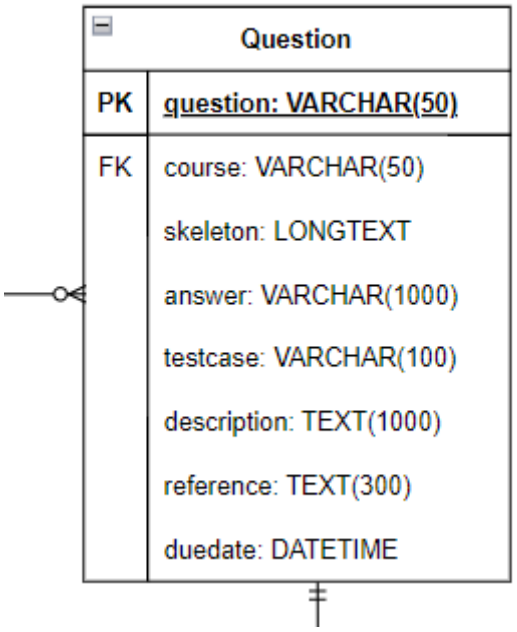


그림 6.5 - Question 테이블

## E. UserData

UserData 테이블은 학습자가 문제를 풀 때 3번에 걸쳐서 사용자가 제출한 코드를 관리한다. user_id (사용자 고유 식별자), question (문제 고유 식별자)가 외래 키이자 프라이머리 키로, 각각 User 테이블과 Question 테이블에서 가져오며, 이외에도 save1 (첫 번째 제출 코드), save2 (두 번째 제출 코드), save3 (세 번째 제출 코드) 필드가 있다.

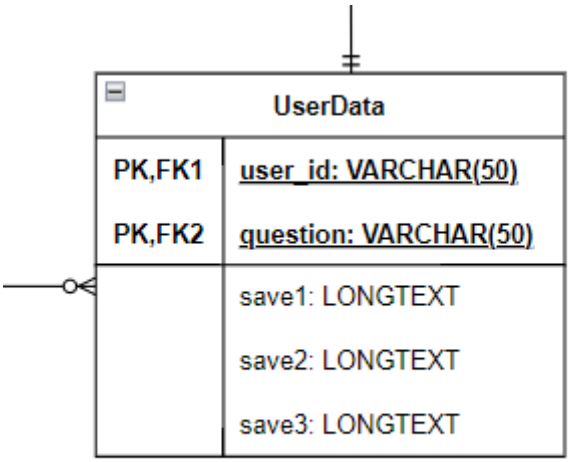


그림 6.6 - UserData 테이블

# 7. Testing plan

이 챕터에서는 testing 과정을 Development testing, Release testing, User testing의 세 단계로 분류해서 설명할 예정이다. 이렇게 테스트의 목적, 방법에 따라 테스트의 방식을 다르게 해서 테스트를 함으로써 오류가 발생할 확률을 낮출 수 있다.

## 7.1. Developing Testing

시스템을 개발하는 개발하면서 진행하는 테스트이다. developing testing은 소프트웨어 개발 위험, 시간 및 비용을 줄이기 위해 광범위한 결함 방지 및 탐지 전략의 동기화된 적용을 포함하는 소프트웨어 개발 프로세스이다.

소프트웨어 개발에 대한 조직의 기대치에 따라 개발 테스트에는 static code abnalysis, data flow analysis, metrics analysis, peer code analysis, unit testing, code coverage analysis, traceability와 other software verification practice이 포함될 수 있다.

## 7.2. Release Testing

Release Testing은 소프트웨어를 릴리스하기 전에 사용자가 소프트웨어를 사용하기에 충분하다는 확신을 개발자에게 주기 위해 시행되는 코딩 테스트이다. Release Testing은 새로운 버전의 소프트웨어를 릴리스할 때 새로운 버전에 결함이 존재하는지 테스트하는 것이다.

## 7.3. User Testing

이 단계에서는 실제 사용자들의 환경에서 테스트를 진행하게 된다. 실제 사용자들이 시스템을 사용해보면서 시스템의 출시 여부를 최종적으로 결정하게 된다.

## 7.4. Test Case Example

**A.** 회원가입

| | |
|---|---|
| 테스트 이름 | 회원가입 테스트 |
| 사전 조건 | 1. 아이디가 기존 데이터베이스에 저장되어 있는 아이디와 중복되어서는 안된다.<br>2. 아이디, 비밀번호,이름,소속은 문자열 형식이다. |
| 실행 | 1. 아이디 중복 체크<br>2. 비밀번호 형식 체크<br>3. 이름 형식 체크<br>4. 소속 형식 체크<br>5. 사용자 타입 선택 여부 체크 |
| 예상 결과 | 회원 가입 후 데이터베이스에 회원 정보 저장<br>로그인 화면으로 전환 |

표 7.1 - 회원가입 확인

**B.** 로그인

| | |
|---|---|
| 테스트 이름 | 로그인 테스트 |
| 사전 조건 | 1. 이메일 형식 (@, . 등)을 지켜야 한다.<br>2. 비밀번호는 문자열 형식이다. |

| | |
|---|---|
| | 3. 이메일과 비밀번호가 데이터베이스에 저장되어 있어야한다. |
| 실행 | 1. 이메일 형식 체크<br>2. 비밀번호 형식 체크<br>3. 이메일 DB에 존재여부 체크<br>4. 이메일과 비밀번호 일치 여부 체크 |
| 예상 결과 | 사용자 로그인 상태로 기록, 사용자 정보를 저장한 후 메인 화면으로 전환 |

표 7.2 - 로그인 확인

**C.** 주차 과제 확인

| | |
|---|---|
| 테스트 이름 | 주차 과제 확인 |
| 사전 조건 | 사용자는 로그인한 상태여야 한다. |
| 실행 | 1. 로그인 한 후 사용자는 헤더의 배너를 통해 과목을 선택<br>2. 해당 과목의 주차 과제 목록을 확인<br>3. 과제 선택 시 출제한 문제가 좌측 화면에 표시<br>4. 파이썬 에디터에 스켈레톤 코드 생성 |

| | |
|---|---|
| 예상 결과 | 사용자가 선택한 문제에 대한 번호를 화면에 표시<br>이전에 저장한 코드가 존재할 경우 코드를 불러옴<br>이전에 저장한 코드가 존재하지 않은 경우 skeleton code를 불러옴 |

표 7.3 - 주차과제 확인

**D.** 코드 실행 결과 및 평가 확인

| | |
|---|---|
| 테스트 이름 | 코드 실행 및 평가 |
| 사전 조건 | 1. 사용자는 로그인한 상태여야 한다.<br>2. 사용자가 코드를 실행하기 전 에디터에 코드를 입력해야 한다. |
| 실행 | 1. 사용자는 문제 칸에 뜨는 문제를 확인한다<br>2. 사용자가 해당 문제를 풀 수 있는 코드를 작성한다.<br>3. 제출을 누를 경우 에디터 창에 있는 사용자의 코드를 저장하며 해당 코드에 대해 채점을 진행한다. |
| 예상 결과 | 코드실행이 되어 결과를 표시<br>해당 평가를 사용자가 볼 수 있도록 화면에 표시 |

표 7.4 - 코드 실행 결과 및 평가 확인

# 8. Development plan

## 8.1. Development Environment

### 8.1.1. React


그림 8.1 – React logo

메타에서 개발한 오픈 소스 자바스크립트 라이브러리이다. SPA을 전제로 하고 있으며, Dirty checking과 Virtual DOM을 활용하여 업데이트 해야하는 DOM 요소를 찾아서 해당 부분만 업데이트하기 때문에, 리렌더링이 잦은 동적인 모던 웹에서 엄청나게 빠른 퍼포먼스를 내는게 가능하다. 기본적으로 모듈형 개발이기 때문에 생산성 또한 상당히 높은 라이브러리이다. 거기에 기본적으로 프레임워크가 아니라 라이브러리인지라 다른 프레임워크에 간편하게 붙여서 사용하는 것도 가능하며, React Hooks라는 강력한 메소드들을 지원하면서 사실상 웹 프론트엔드 개발의 표준으로 자리잡았다. 이와 더불어 타입스크립트나 Sass 같은 언어도 지원한다. 또한 Next.js 등의 등장으로 인해 SSG, SSR등을 할 수 있게 되면서 사용 범위 또한 기하급수적으로 늘어났다.

### 8.1.2. Django


그림 8.2 – Django logo

Django는 파이썬으로 작성된 오픈 소스 웹 프레임워크로, model-template-view(MTV) 패턴을 따르고 있다. 현재는 장고 소프트웨어 재단에 의해 관리되고 있다. 고도의 데이터베이스 기반 웹사이트를 작성하는 데 있어서 수고를 더는 것이 장고의 주된 목표이다. 장고는 콤포넌트의 재사용성(reusability)과 플러그인화 가능성(pluggability), 빠른 개발 등을 강조하고 있다. 또한, "DRY(Don't repeat yourself: 중복배제)" 원리를 따랐다. 설정 파일부터 데이터 모델에까지 파이썬 언어가 구석구석에 쓰였다.

### 8.1.3. SQLite


그림 8.3 – SQLite logo

SQLite는 MySQL나 PostgreSQL와 같은 데이터베이스 관리 시스템이지만, 서버가 아니라 응용 프로그램에 넣어 사용하는 비교적 가벼운 데이터베이스이다.

일반적인 RDBMS에 비해 대규모 작업에는 적합하지 않지만, 중소 규모라면 속도에 손색이 없다. 또 API는 단순히 라이브러리를 호출하는 것만 있으며, 데이터를 저장하는 데 하나의 파일만을 사용하는 것이 특징이다. 버전 3.3.8에서는 풀텍스트 검색 기능을 가진 FTS1 모듈이 지원된다. 컬럼을 삭제하거나 변경하는 것 등이 제한된다.

## 8.2. Constraints

본 시스템은 문서에 적힌 제약사항을 준수하여 개발된다.

- 검증된 기술을 사용한다.

- Typescript 언어를 사용하지 않는다.

- 웹 프레임워크로써 React, Django, SQLite만을 사용한다.

- 비용을 지불해야 하는 소프트웨어는 사용을 지양한다.

- 브라우저는 Chrome 107.0.5304.110 버전을 기준으로 개발한다.

- 사용자가 사용하기 편하고 직관적인 UI/UX를 사용한다.

## 8.3. Assumptions & Dependencies

이 문서에 기술된 모든 시스템은 네트워크 통신이 원활하고 브라우저가 react를 지원한다는 가정 하에 작성되었다. 때문에 react를 지원하는 브라우저 중 하나인 chrome 107.0.5304.110 버전을 기준으로 작성되었다.