
성균관대학교
SUNG KYUN KWAN UNIVERSITY

# CodingCAT: 온라인 코딩 테스트 플랫폼

## 소프트웨어 디자인 명세서

2022.11.13

소프트웨어공학개론
41분반 Team 2

Team Leader 박승호
Team Member 김동우
신주영
어영준
장병우
한태욱

CONTENTS

# LIST OF FIGURES

# LIST OF TABLES

# 1. Introduction

## 1.1. Objective

이 프로젝트는 성균관대학교 학생들을 위한  코딩 테스트 플랫폼, CodingCAT 서비스를 제공하기 위한 목적을 가지고 있다. 이 서비스는 성균관대학교 이은석 교수님의 2022년 2학기 소프트웨어공학개론 강의 41분반의 2조에 의해 고안되고 개발된다.

본 문서는 서비스 개발팀 6명과 강의 조교님, 강의 교수님이 본 서비스의 주요 독자이며, 예외적으로 동일 강의 수강자 또는 학습 및 교육의 용도로 열람할 수 있다. 본 문서를 재배포 및 수정하는 것에 제약은 없으나, 상업적 용도로 활용 시 반드시 개발팀의 허가를 얻어야 한다.

이 시스템은 웹을 통하여 제공될 예정이며, 전공자 및 강의 수강자가 본 문서의 주요 독자이다. 코딩은 소프트웨어 엔지니어로서 가장 기본이 되고 가장 근본이 되는 요구 역량이다. 이에 따라 전공자, 학생들은 많은 웹 또는 앱에서 소프트웨어 역량을 늘리고 있다. 현 실태에 맞춰 성균관대학교 학생들의 역량을 기르기 위해 자체적인 코딩테스트 플랫폼을 제공하고자 한다. 제공되는 서비스를 통해 전 보다 나아진 학습환경을 가지고 보다 역량을 발전시키는 것을 목표로 한다.

본 문서는 상기 프로젝트 시행을 위해 사용되었거나 사용될 디자인에 대한 설명을 담고 있다. 디자인은 이전에 쓰여진 요구사항 명세서의 내용을 토대로 만들어 졌다.

## 1.2. Applied Diagrams

이 장에서는 디자인 단계에서 개발팀이 사용한 다양한 기능들과 다이어그램(Diagram)에 대한 내용을 설명 할 것이다.

### 1.2.1. UML

UML은 Unified Modeling Language의 약자이다. UML은 개발자들이 실제 개발단계에 들어가기전, 다이어그램을 통해 프로그램의 전체적인 설계, 필요한 변수와 함수를 정하고, 함수들이 어떠한 과정을 거쳐 어떻게 작동하는지 등 전반적으로 코딩하기 전 계획을 디자인 혹은 시각화 한 것이다.

UML은 개발자가 프로젝트에 참여하는 다른 개발자 혹은 이외의 자들에게 더 쉽게 설명 및 이해가 가능하게 하며, 전체적인 프로젝트의 구조와 체계를 보다 쉽게 알 수 있게 한다. 이러한 장점을 바탕으로 하여 모든 주요 객체, 컴포넌트 소프트웨어 개발 방법 및 다양한 구현 플랫폼에 사용 할 수 있다. 시각적 표현으로 소프트웨어 개발 혹은 다른 단계에서 발생할 수 있는 문제나 오류사항들을 더 잘 이해할 수 있다.

### 1.2.2. Deployment Diagram

이 다이어그램(Diagram)은 시스템을 구성하는 HW자원 간의 연결 관계를 표현하고, HW자원에 대한 SW컴포넌트의 배치 상태를 표현한 다이어 그램이다. 대개 시스템의 설계 단계의 마지막에 작성하며, SW시스템이 배치 실행될 HW자원들을 정의한다. 또한, SW 컴포넌트가 어떤 HW 자원에 탑재되어 실행될지 정의하며, HW자원의 물리적인 구성을 정의한다. 구성은 Node와 Component 그리고 Connection과 Dependency를 통해 관계를 표현한다.

### 1.2.3. Class Diagram

이 다이어그램 (Diagram)은 클래스와 다른 클래스 사이의 관계를 모델링한 다이어 그램이다. 시스템의 클래스, 속성, 함수 및 각 클래스와 시스템 간의 관계를 다이어 그램의 형태로 보여줌으로써 시스템의 구조를 설명하는 정적 구조 다이어그램의 일종이다. 클래스가 객체의 구성 요소 인 것처럼 Class diagram은 UML의 구성요소 이다. 이 다이어 그램은 시스템의 구조를 나타내며, 클래스 끼리 연결 시킬 수 있고, 그

안에 필요한 변수 및 데이터 타입 함수를 정의할 수 있다. 클래스의 모양은 세개의 공간으로 나누어진 직사각형 모양으로 만들어 지며, 가장 위의 공간은 클래스의 이름을, 가운데 공간은 클래스의 속성을, 가장 아래의 공간은 클래스에서 사용할 수 있는 함수 또는 연산을 나타낸다.

### 1.2.4. Sequence Diagram

이 다이어그램 (Diagram)은 오브젝트, 클래스 끼리의 상호작용을 보여주는 다이어그램이다. 어떤 기능이 어떤 오브젝트 혹은 클래스와 연결되어 작동되어야 하는지, 그에 관한 결과는 어떤 값이 와야하는지, 전체적인 상호작용의 순서를 차례로 보여주는 다이어그램이다. 이 다이어그램을 통해 각 함수에 대응되는 파라미터를 알 수 있고, 그에 따라 도출되는 결과값을 확인할 수 있다. 모든 의사소통은 연대순으로 표현되며 Actor 또는 객체는 다른 객체가 상호작용을 할 때 등의 필요한 경우에만 활성화가 가능하다.

### 1.2.5. Use Case Diagram

이 다이어그램(Diagram)은 사용자(Actor)의 관점에서 시스템의 기능, 상호작용과 그들 간의 관계를 표현하는 다이어그램이다. 시스템에서 제공해야하는 기능이나 서비스가 명세되어 있으며, 사용자와 시스템 사이의 상호작용에 집중되어 있다. 외부에서 본 시스템의 기능을 표현하기 때문에, 실제 내부의 비즈니스 로직이 아닌, 사용자가 수행하는 기능을 파악하고 싶을 때 작성한다. 주요 구성요소는 'Scope' – 네모난 상자로 표현되며, 시스템이 제공하는 기능의 범위를 나타낼때 쓰인다, 'Use Case' – 시스템이 제공해주는 서비스와 기능을 나타내며, 사용자의 요구사항을 구조화한 것이다. 'Actor' – Actor은 구현 대상이 아닌 시스템 외부에서 시스템과 상호작용 하는 존재다. 사람 또는 외부 시스템을 액터로 표현한다. 각 액터끼리는 서로 상속 또는 일반화가 가능하다.

#### 1.2.6. Context Diagram

이 다이어그램(Diagram)은 시스템의 경계를 정의하고 구성요소와 외부와의 연동 관계를 나타내는 다이어그램이다. 하나의 목표 시스템과 주변의 Actor, 시스템과 Actors 간의 인터페이스를 표현하는 요소들로 이루어져 있다. Context diagram은 일반적으로 requirement specification에 포함된다. 시스템과 외부 엔티티, interface를 정의하며, 각 인터페이스와 관련한 기능적 요구사항과 품질 요구사항을 정의한다. 이 다이어그램은 주로 프로젝트 초기에 조사 대상의 범위를 결정하기 위해 종종 사용되며, 다이어그램 내에서 시스템은 중앙에 위치되며, 외부 엔티티와 interface에 의해 둘러싸여 있다.

### 1.3. Applied Tools

#### 1.3.1. diagram.net

별도로 소프트웨어를 로컬 머신에 설치할 필요없이 웹브라우저를 통해 접속하면 다양한 다이어그램을 그릴 수 있는 사이트이다. 다이어그램을 그리기 위한 컴포넌트들이 종류별로 그룹화 되어있으며, Google Drive, OneDrive등 원하는 곳에 다이어그램 파일을 저장할 수 있다.

#### 1.3.2. dbdiagram.io

ERD 자동화 도구이다. DB테이블 import를 통해 반자동화된 그리기가 가능하며, SQL과 유사한 모델링 전용 코드를 통해 모델 필드와 관계의 시각적 표현을 구성할 수 있다.

#### 1.3.3. Figma

웹 사이트 제작시 디자인을 빠르고 쉽게 할 수 있도록 도와주는 웹 기반 모델링 도구이며, 기존 사용되는 앱인 XD보다 간단한 component로 이루어져 있다. Export또한 잘 이루어져 있어 Frontend와 UI/UX디자인과의 협업에 유용하게 쓰인다.

### 1.3.4. Postman

API개발을 보다 빠르고 쉽게 구현 할 수 있도록 도와주며, 개발된 API를 실제와 유사한 환경에서 테스트 가능하여 서버-클라이언트 구조 시스템 개발시 많이 이용되는 플랫폼이다.

### 1.3.5. Swagger, ReDoc

Swagger은 Open Api Specification(OAS)를 위한 프레임워크이다. API들이 가지고 있는 명세와 스펙을 관리할 수 있는 프로젝트이다.

ReDoc은 Open Api Specification(OAS)파일을 읽어서 Construct해주는 도구로 로컬 및 개발 환경에서 작성한 API가 제대로 문서화되어있는지 빠르게 확인하고 테스트할 때 유용하다.

## 1.4. References


- Git Community, “Git”. https://git-scm.com/
- Facebook, “React”. https://reactjs.org/
- Django Software Foundation, “Django Project”. https://www.djangoproject.com/
- SQLite Team, “SQLite”. https://www.sqlite.org/
- Docker Incorporation, “Docker”. https://www.docker.com/
- MDN contributors, “HTTP”. https://developer.mozilla.org/
- Amazon aws, “RESTful API란 무엇입니까?”. https://aws.amazon.com/
- OpenJS Foundation, “About Node.js” https://nodejs.org/

# 2. Overall System Architecture

## 2.1. Purpose

본 장의 목적은 시스템의 전반적인 구조를 효과적으로 이해하기 위함이다. 시스템을 이루는 구성 요소와 물리적인 구조를 다이어그램을 통해 시각화하여 표현한다.

## 2.2. System Organization

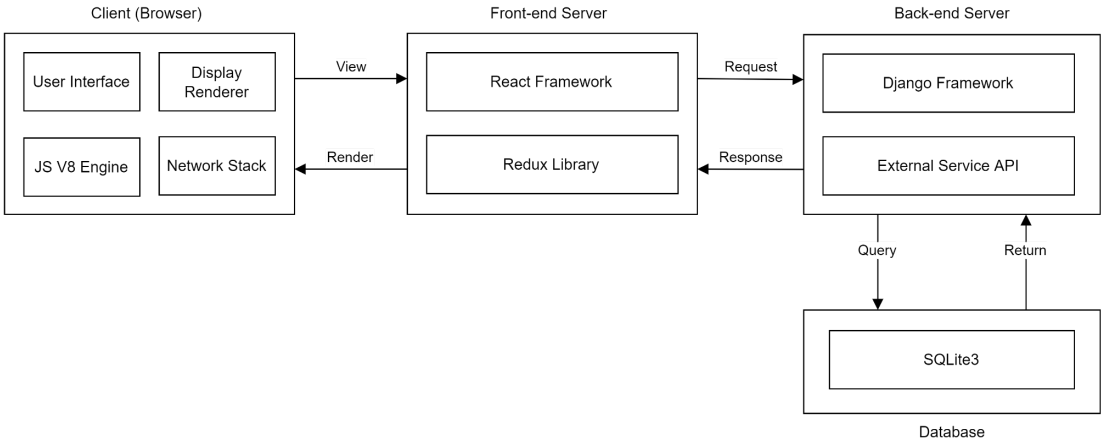


*Figure 1. Overall System Organization*

본 시스템은 크게 사용자, 프론트엔드, 백엔드 세 요소로 이루어져있다. 사용자는 주로 웹 브라우저를 통해 시스템에 접근할 수 있으며, 웹 브라우저에서 제공하는 Renderer와 JS 엔진을 이용해 서비스를 요청하거나 요청에 대한 응답 페이지가 렌더링된다. 본 시스템은 사용자가 Chrome 웹 브라우저를 이용한다고 가정한다.

프론트엔드 서버는 사용자와의 모든 상호작용을 처리한다. 사용자가 보낸 서비스 요청을 파싱하여 백엔드 서버에 전달하고, 요청에 대한 응답을 토대로 페이지를 렌더링하여 사용자에게 반환한다. 프론트엔드 서버는 React 프레임워크를 활용하여 구현되며, 필요에 따라 Redux 등 다른 라이브러리 또한 활용한다.

백엔드 서버는 요청에 따른 실질적인 동작을 처리한다. 프론트엔드 서버로 부터 전달된 요청을 파싱하여 비즈니스 로직 수행 후, 그 결과를 응답한다. 그 과정에 있어서 특정 데이터를 읽어오거나, 데이터를 추가하는 등 데이터베이스와의 상호작용이 발생할 수 있다. 백엔드 서버는 Django 프레임워크를 활용하여 구현되며, 이외에도 목표 서비스 제공을 위해 다양한 라이브러리들도 활용한다. 데이터베이스로는 SQLite3를 활용한다.

## 2.3. System Context

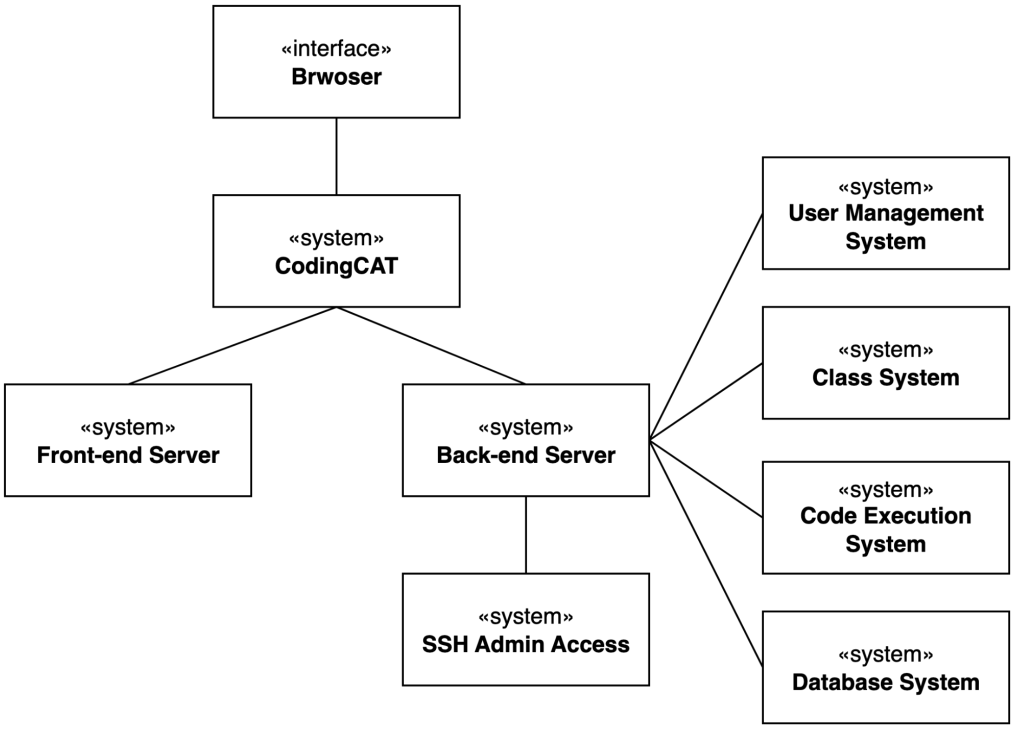


*Figure 2. System Context Model*

## 2.4. Sequence Diagram

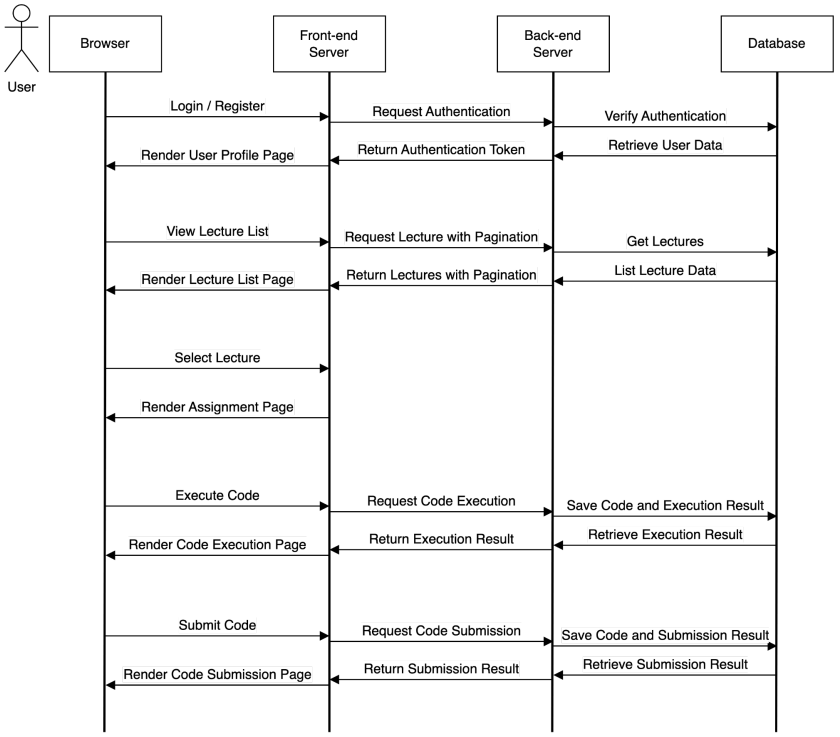


*Figure 3. Sequence Diagram*

## 2.5. Deployment Diagram

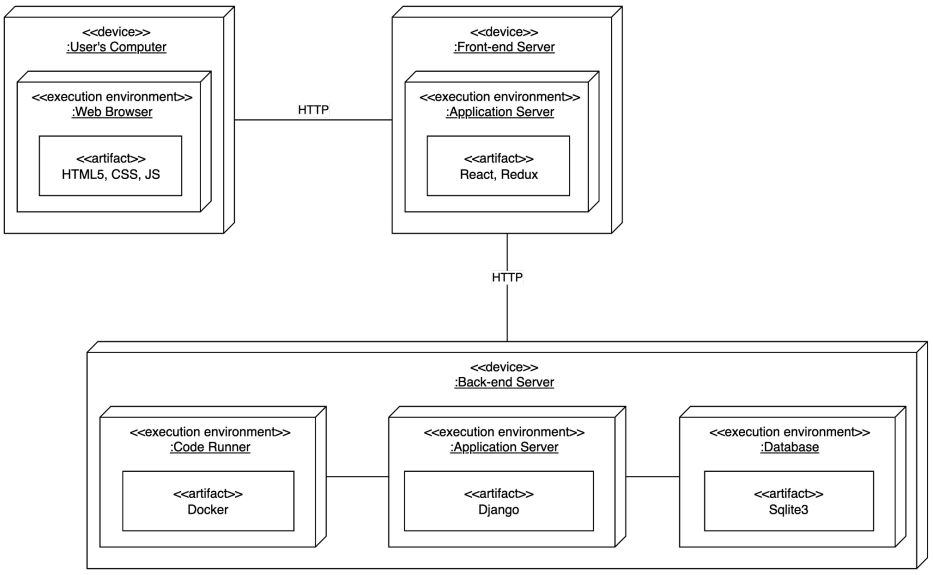


*Figure 4. Deployment Diagram*

# 3. Frontend System Architecture

## 3.1. Purpose

본 장에서는 Coding Cat 시스템 중, 유저와의 상호작용을 담당하는 프론트 엔드 시스템의 구조와 속성, 그리고 이 시스템이 포함하는 기능을 기술한다. 또한 프론트엔드 시스템을 구성하는 각 Sub Component 들간의 관계를 보여준다.

## 3.2. Overall Architecture

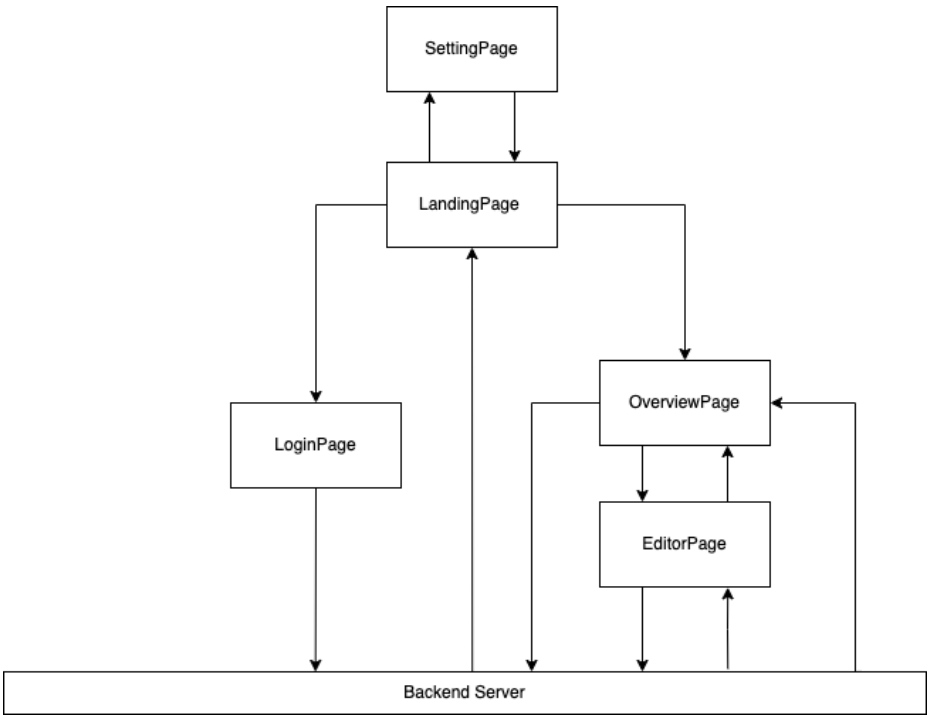


*Figure 5. Frontend System Architecture*

본 서비스의 시스템은 크게 프론트엔드와 백엔드, Docker로 구현된 서버측 코드 가상 실행 환경, 데이터베이스로 구성되어 있다. Frontend는 서버에 정보를 입출력하는 역할을 수행한다. 또한, Frontend 단에서 private route를 구현하여 로그인이 이루어지지 않은 경우, 본 시스템을 사용할 수 없고 로그인 페이지로 리다이렉션된다. 로그인이 이루어진 후, Landing page로 돌아와 사용자 설정, 과제 목록 조회 및 과제 수행 등 본

시스템의 모든 기능을 이용할 수 있다. Backend 및 Database는 유저의 로그인, 과제 정보, 강의 정보 등 시스템의 주요 기능 수행 시 Frontend로부터 호출되어 사용되어진다.

## 3.3. Sub Components

### 3.3.1. Profile Component

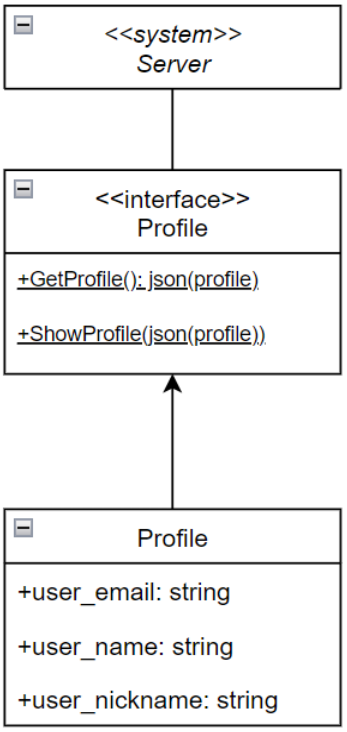


*Figure 6. Profile Component Class Diagram*

Profile 클래스는 유저의 개인 프로필 정보를 관리하는 객체이다. 개인 정보에는 사용자 별명(Nickname)과 아이디, 이메일 계정 정보가 있다. 사용자 이메일 계정 및 아이디, 별명은 모두 깃허브(Github)의 설정을 가져온 것이며, 변경 불가능하다.

- user_email: 사용자 계정 정보인 이메일 주소이다.
- user_nickname: 사용자 별명이다.
- user_name: 사용자 계정 정보에 있는 이름이다.
- GetProfile(): 서버로부터 사용자 프로필 정보를 가져오는 함수이다.

- ShowProfile(): 가져온 프로필 정보를 보여주는 함수이다.

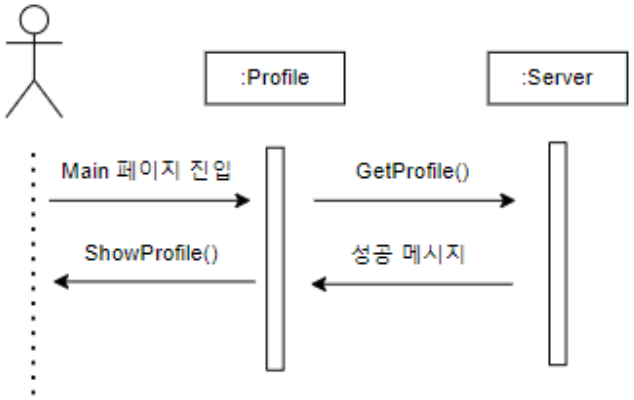


*Figure 7. Profile Component Class Sequence Diagram*

### 3.3.2. AuthManager

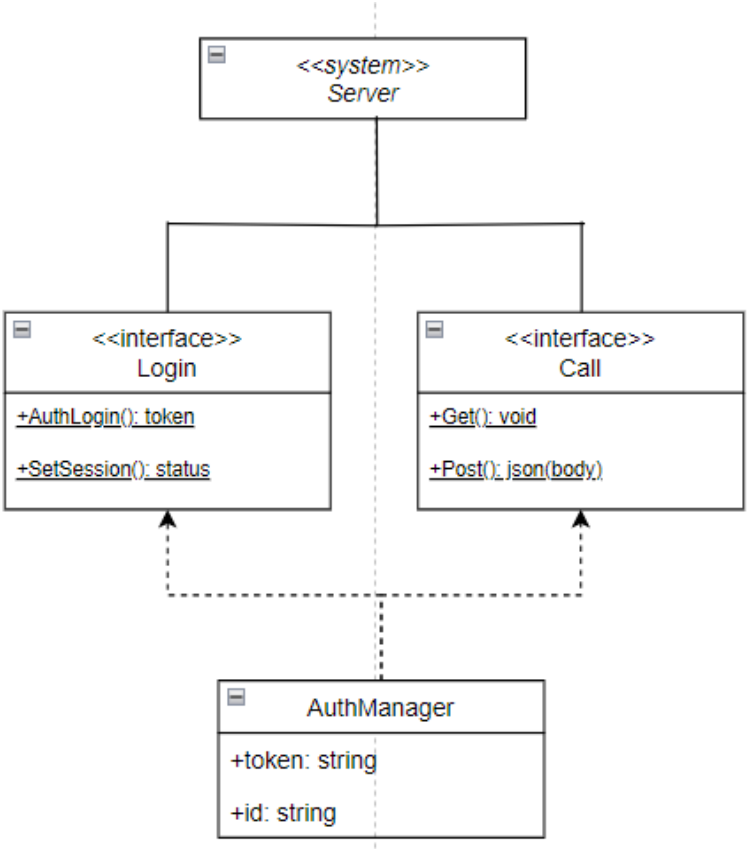


*Figure 8. Auth Manager Component Class Diagram*

이 클래스는 로그인과 세션 토큰을 관리하는 객체이다. Github Oauth 2.0을 사용한다.

- token: 사용자의 access token이다.
- id: 사용자의 식별 ID 값이다.
- SetSession(): 서버 응답을 바탕으로 유저의 세션을 관리하는 함수이다.
- Get(): 서버에 HTTP Get 요청을 보내는 함수이다.
- Post(): 서버에 HTTP Post 요청을 보내는 함수이다.

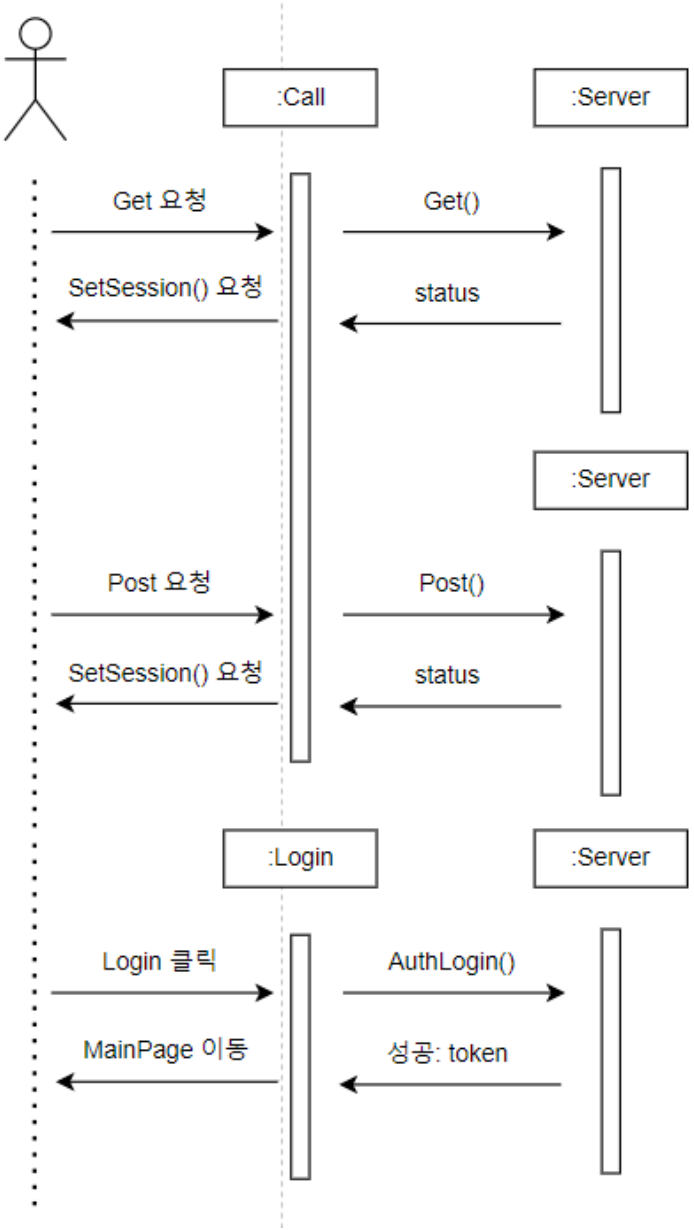


*Figure 9. Auth Manager Component Sequence Diagram*

### 3.3.3. Lecture Total Component

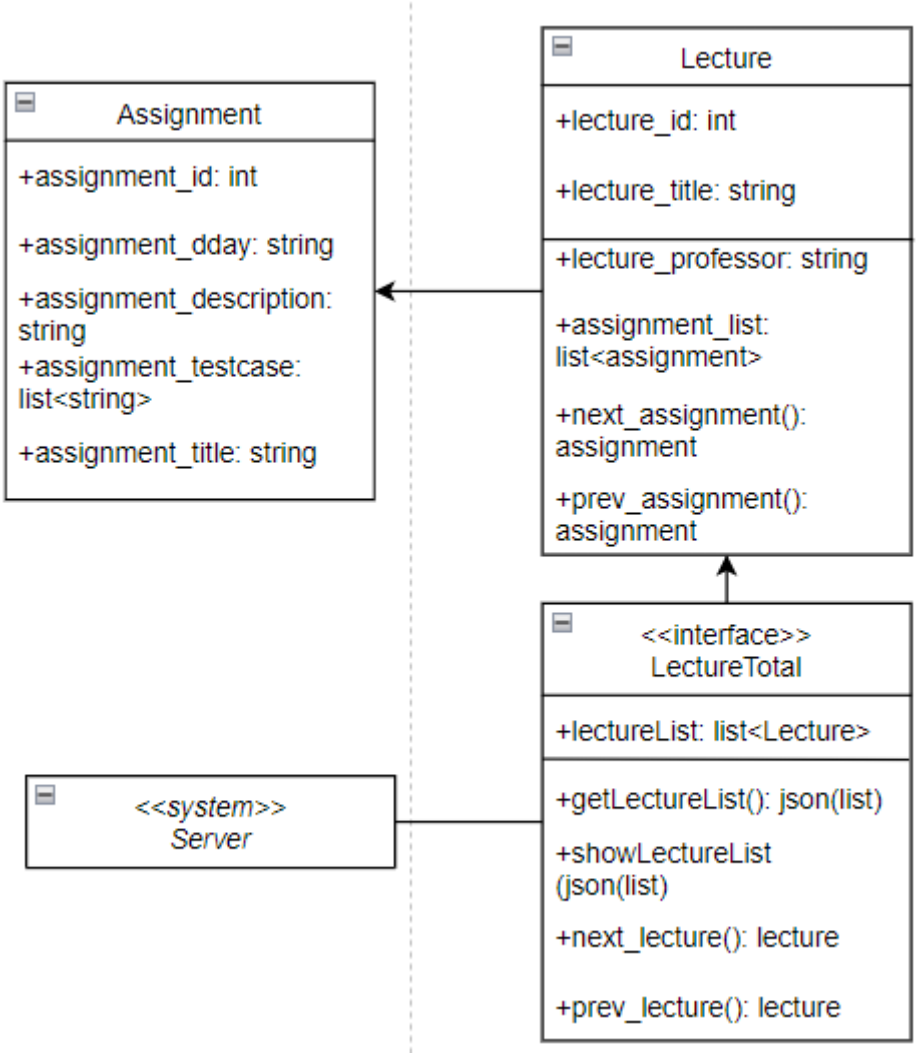


*Figure 10. Lecture Total Component Class Diagram*

강의 페이지를 관리하는 클래스이다. 현재 수강 중인 강의와 각 강의의 예정된 과제 및 진행한 과제에 대한 정보를 속성으로 가진다.

- lecture_list: 수강 중인 전체 강의를 관리하는 리스트이다.
- lecture_id: 강의를 구분하는 유일한 id 값이다.
- lecture_title: 강의 제목이다.
- lecture_professor: 강의를 담당하는 교수의 이름이다.
- assignment_list: 각 강의에 속하는 과제들을 관리하는 객체이다.

- assignment_id: 과제를 구분하는 유일한 id 값이다.
- assignement_deadline: 과제 제출 기한이다.
- assignment_description: 과제에 대한 설명이다.
- assignment_testcase: 과제의 테스트케이스이다.
- assignment_title: 과제의 제목이다.

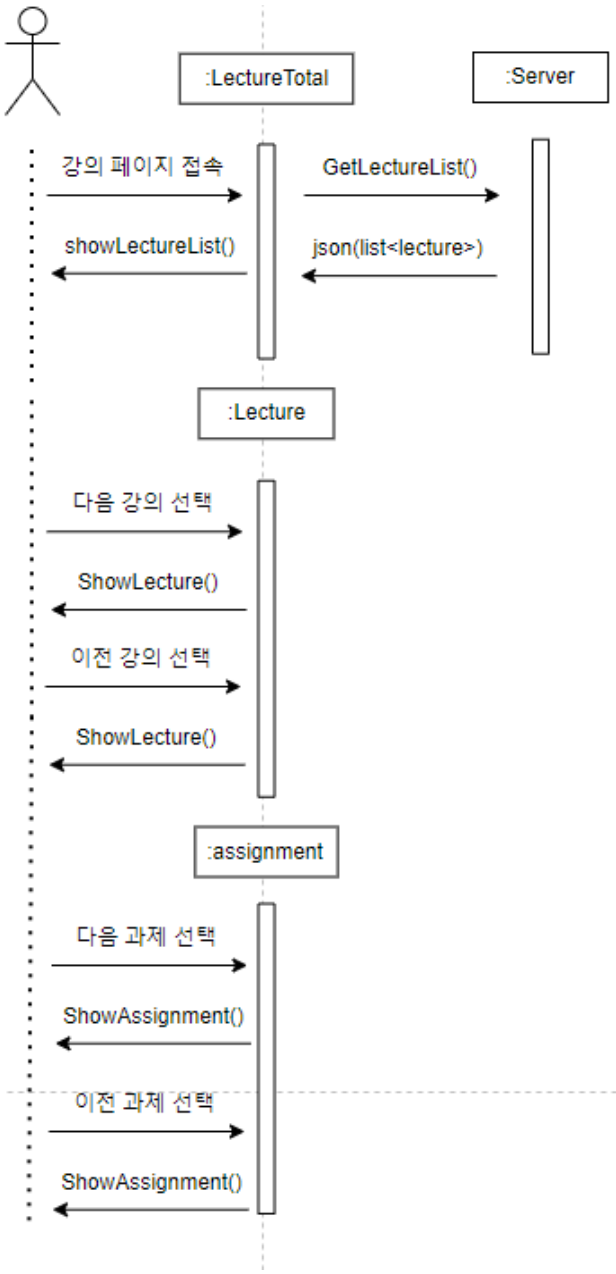


*Figure 11. Lecture Total Component Sequence Diagram*

### 3.3.4. Repo Component

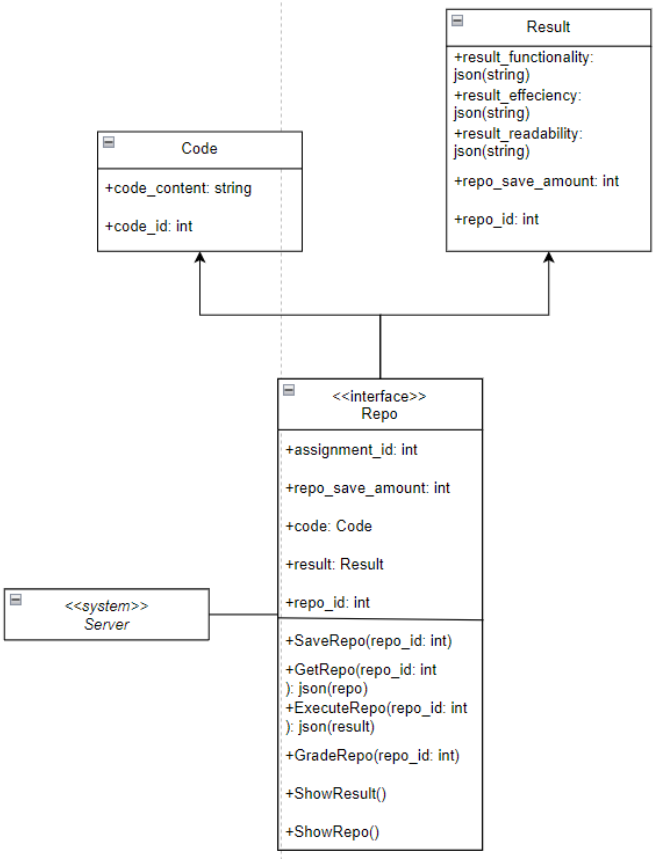


*Figure 12. Repo Component Class Diagram*

사용자가 과제 수행을 위해 코드 입력 및 디버깅하고 정답과 비교하는 프로세스를 담당하는 클래스이다.

- assignment_id: 과제의 id 값이다.
- repo_id: Repo 클래스의 고유한 id이다.
- repo_save_amount: 저장 횟수 값을 저장하는 int이다.
- result: 해당 Repo 클래스의 결과 값을 저장하는 클래스이다.
- code: 해당 Repo 클래스의 코드 입력 값을 저장하는 클래스이다.
- repo_id: result 값의 주인 repo id 값이다.
- result_functionality: 해당 저장소 코드의 기능성 채점 결과이다.
- result_efficiency: 해당 저장소 코드의 효율성 채점 결과이다.

- result_readability: 해당 저장소 코드의 가독성 채점 결과이다.
- code_content: 사용자가 입력한 코드 내용이다.
- code_id: 코드 식별 id이다.
- SaveRepo(): 현재까지 입력된 사용자의 코드 입력 값을 서버로 전송하여 저장한다.
- GetRepo(): 서버에 저장되어 있는 코드 입력 값을 요청하여 받아온다.
- ExecuteRepo(): 현재까지 입력된 사용자의 코드 입력 값을 서버로 보낸 후 실행하여 결과 값을 받는다.
- GradeRepo(): 서버에 저장되어 있는 코드를 채점하여 결과 값을 받아온다.
- ShowRepo(): Repo 클래스를 렌더링하여 시각화한다.

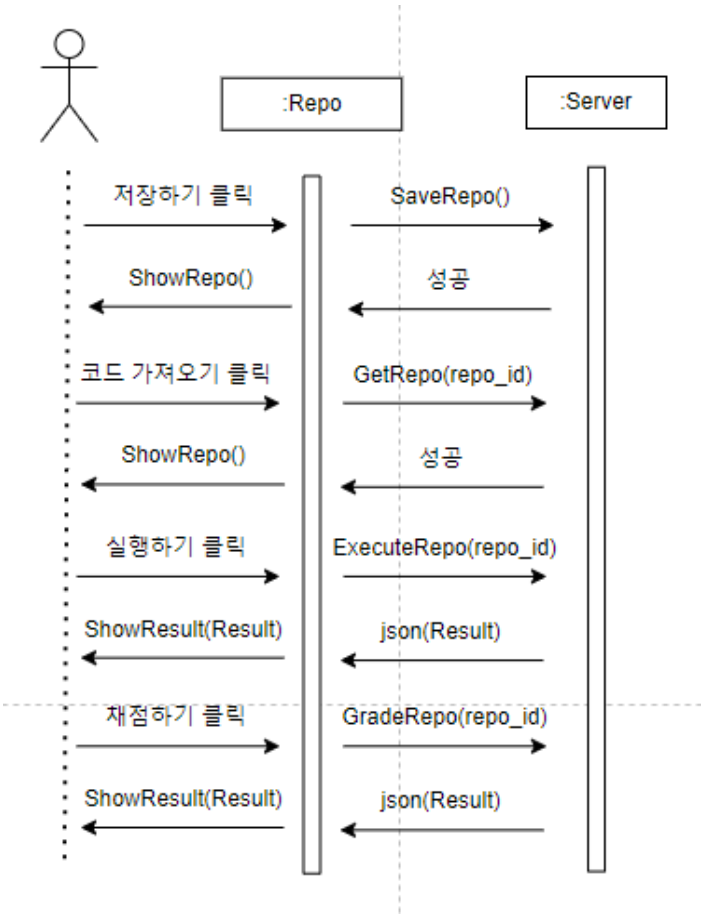


*Figure 13. Repo Component Sequence Diagram*

### 3.3.5. Setting Component

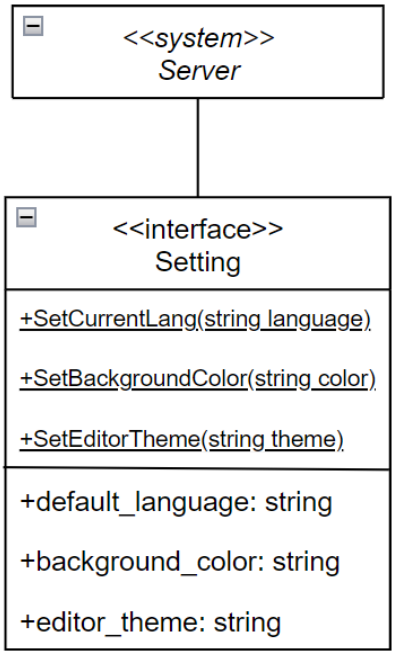


*Figure 14. Setting Component Class Diagram*

사용자의 개인 설정을 전역적으로 관리하는 클래스이다. 코드 에디터의 Theme, 사용 Language, 배경 색을 관리하는 객체를 속성으로 가진다.

- default_language: 사용자가 default로 이용할 프로그래밍 언어 값을 저장하는 변수이다.
- background_color: 해당 서비스의 전체 배경 색 값을 저장하는 변수이다.
- editor_theme: 코드 에디터의 theme 값을 저장하는 변수이다.
- SetCurrentLang(): 사용자가 선택한 프로그래밍 언어로 default_language 값을 설정한다.
- SetBackgroundColor(): 사용자가 선택한 배경 색 값으로 background_color 값을 설정한다.
- SetEditorTheme(): 사용자가 선택한 theme 값으로 editor_theme 값을 변경한다.

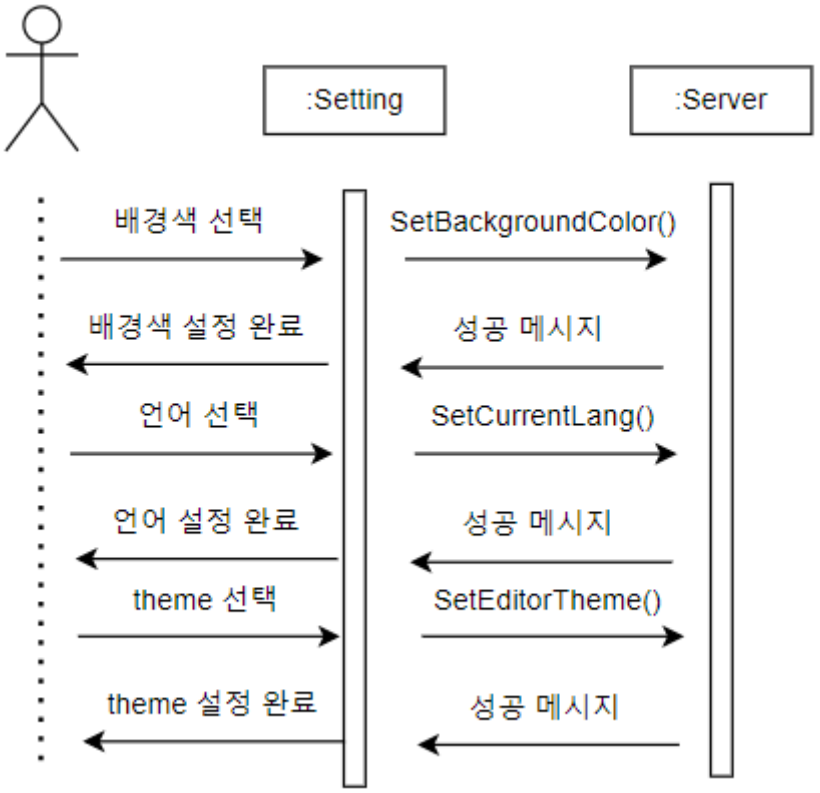


*Figure 15. Setting Component Sequence Diagram*

## 4. Backend System Architecture

### 4.1. Purpose

본 장의 목적은 백엔드 서버의 세부 구조를 명확히 이해하기 위함이다. 먼저 전반적인 구조 파악을 위해 백엔드를 이루는 컴포넌트들과 데이터베이스 간의 관계를 나타낸다. 그리고 세부 컴포넌트에 대한 이해를 위해 각 컴포넌트를 구성하는 클래스와 동작 흐름을 나타낸다.

## 4.2. Overall Architecture

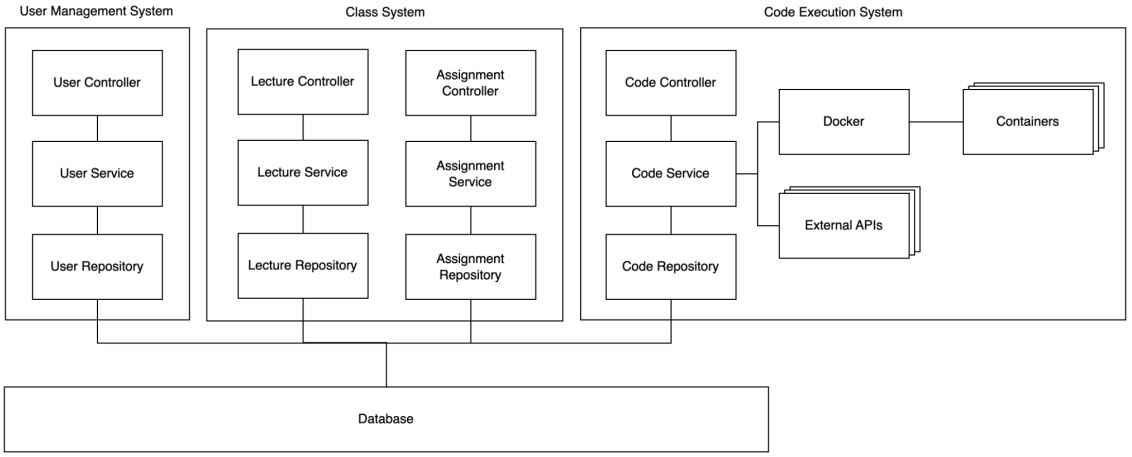


*Figure 16. Backend Architecture*

## 4.3. Sub Systems

### 4.3.1. User Management System

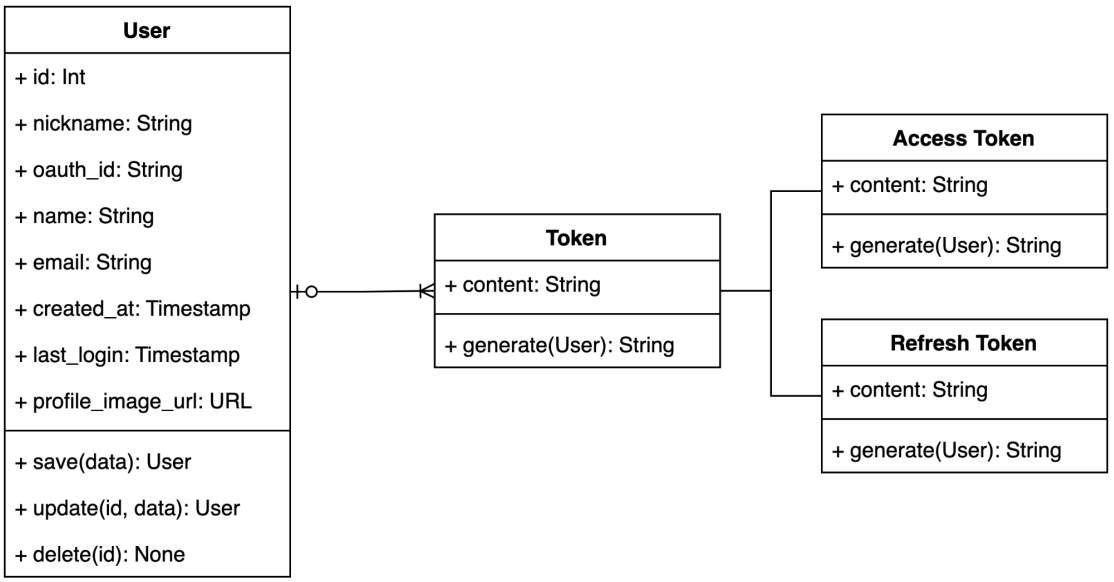


*Figure 17. User Management System Class Diagram*

본 시스템에 로그인하는 객체는 User로 관리되며, 소셜 로그인을 제공하므로 OAuth 구분을 위한 필드가 필요하다. OAuth 로그인 시 Provider로 부터 전달받은

값들을 가공하여 알맞은 필드와 맵핑한다. 또한 회원가입 혹은 로그인 시 변경된 값 반영을 위해 각 메서드를 제공하며, 회원탈퇴를 위한 메서드도 제공한다.

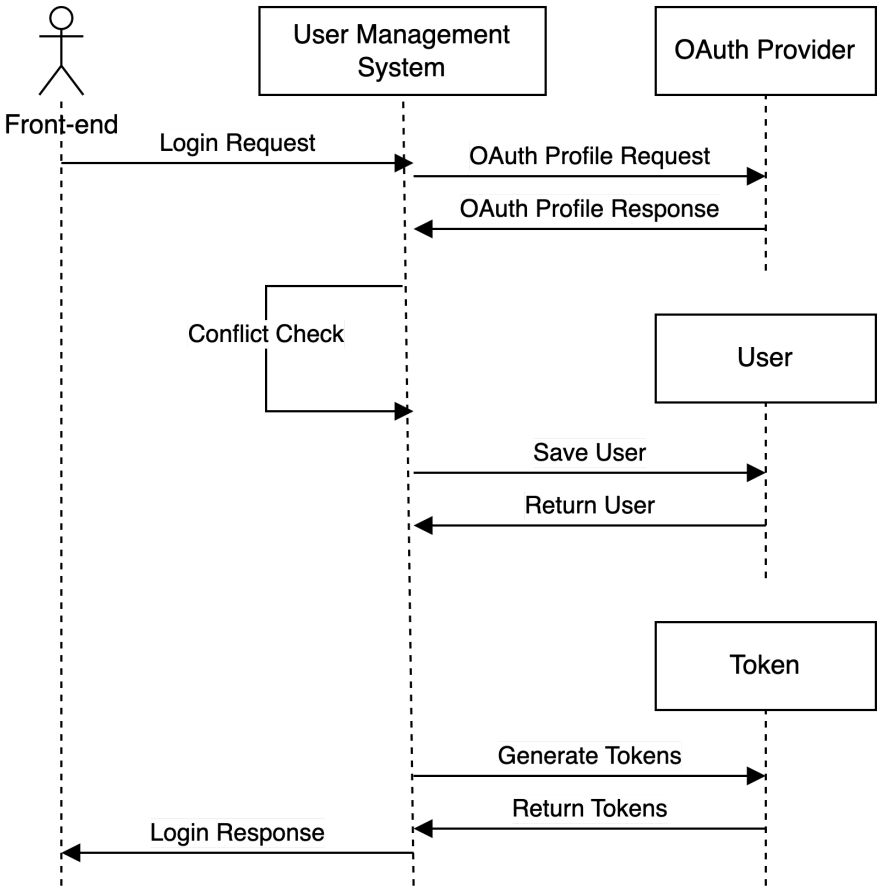


*Figure 18. User Management System Sequence Diagram*

성공한 사용자의 인증은 Token으로 관리된다. 이는 Stateless한 시스템 동작을 위한 것이며, 별도의 세션과 쿠키를 필요로 하지 않는다. Token은 Access Token과 Refresh Token으로 구분된다. Access Token은 비교적 짧은 만료 기간을 가진 인증 Token으로, 사용자가 본인 인증을 위해 주로 사용되는 수단이다. Refresh Token은 비교적 긴 만료 기간을 가진 인증 Token으로, 사용자의 Access Token이 만료될 경우 재발급 받을 때 사용되는 수단이다. 만약 Refresh Token도 만료된 경우, 사용자는 재로그인을 해서 각 Token들을 새로 발급받아야 한다.

### 4.3.2. Class System

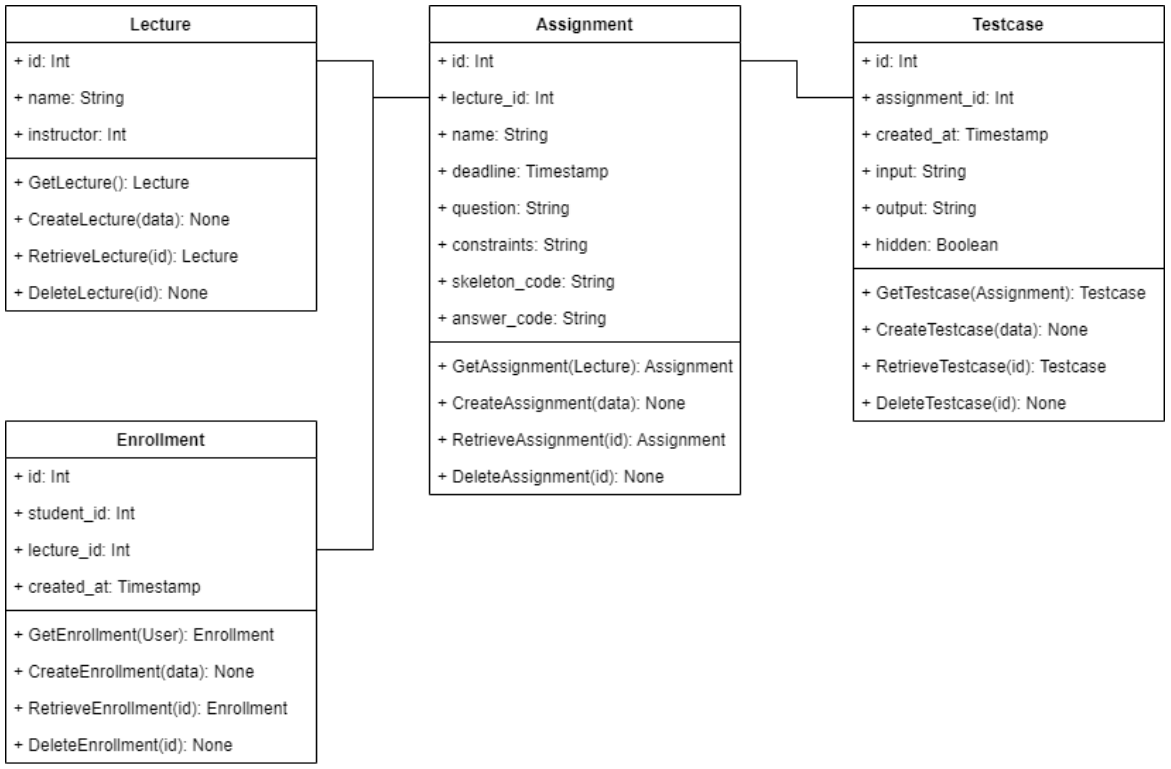


*Figure 19. Class System Class Diagram*

본 시스템에서는 문제 정보를 제공하기 위해, Lecture, 각 Lecture에 포함되는 Assignment, 각 Assignment에 포함되는 Testcase, 각 Lecture를 수강하는 학생을 등록한 Enrollment를 다룬다. 각 객체는 등록, 검색, 삭제를 위한 메서드를 제공한다.

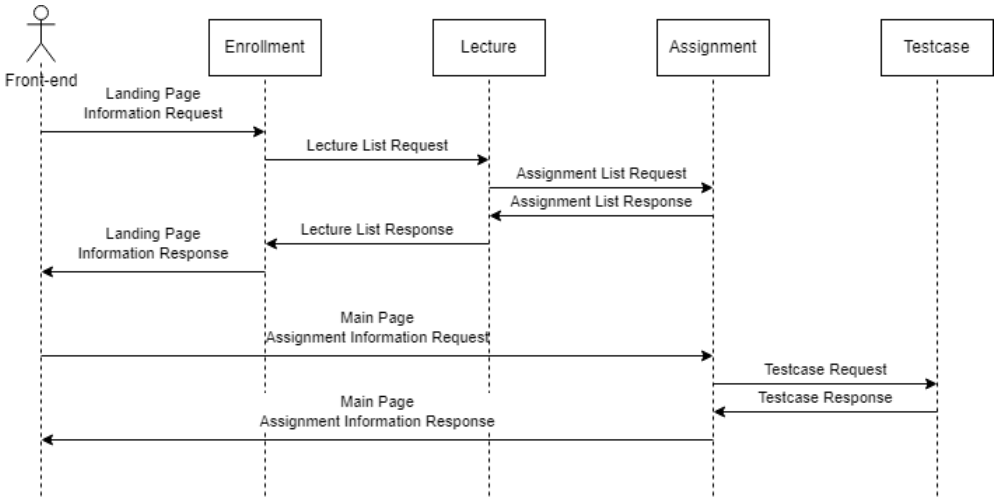


*Figure 20. Class System Sequence Diagram*

Lecture를 등록한 User는 Instructor로 등록되며, 객체의 수정은 Instructor만이 권한을 가진다. 마찬가지로 모든 객체에서의 삭제는 Instructor만이 권한을 가진다. 검색은 해당 리스트를 가져오는 Get과 id를 통해 검색하는 Retrieve가 있다.

### 4.3.3. Code Execution System

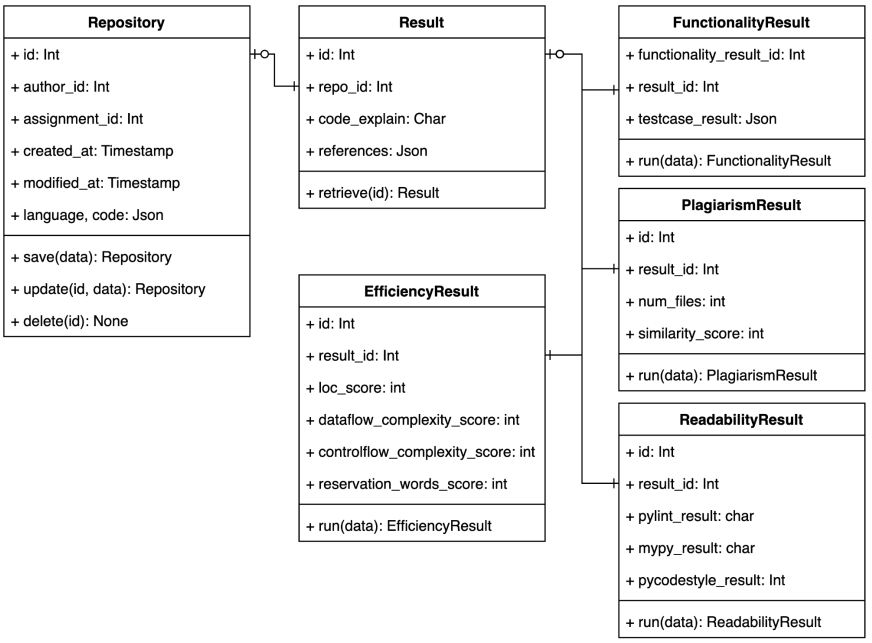


*Figure 21. Code Execution System Class Diagram*

본 시스템에 코드 실행 관련 기능을 요청하면 Repository 객체를 토대로 관리된다. 사용자는 시스템에서 지원하는 여러 프로그래밍 언어 중 한 가지에 대해 작성한 코드를 제출하여 코드 실행을 요청할 수 있다. 본 시스템의 첫 출시 시점 기준으로 Python, Javascript, C, C++ 언어를 지원한다. 코드 실행 시 다양한 언어 지원과 악의적인 공격을 방지 및 안정적인 구동 환경 제공을 위해 별도의 컨테이너 환경에서 수행된다.

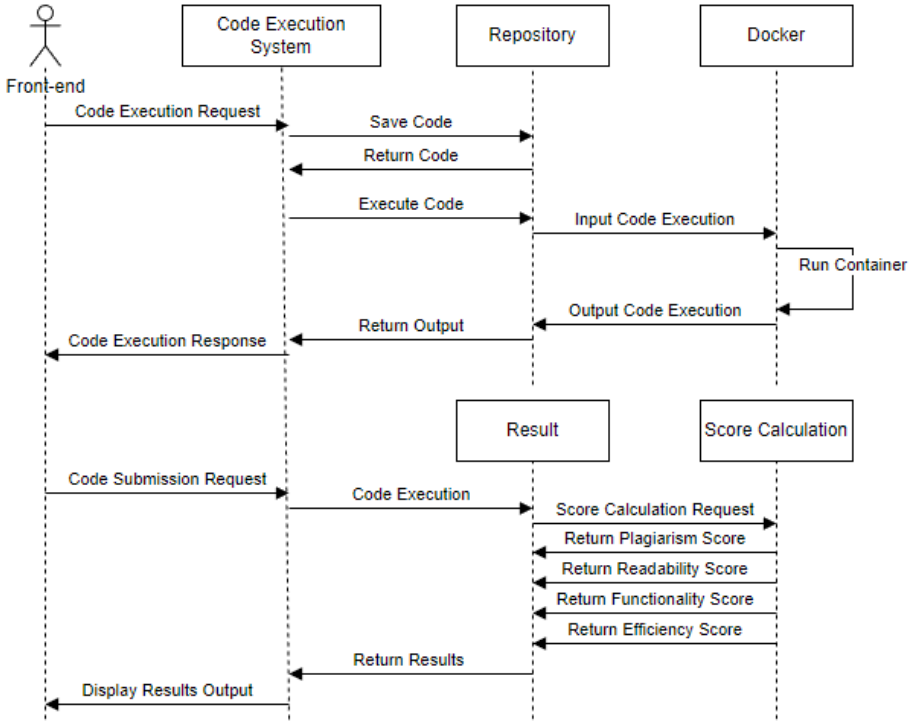


*Figure 22. Code Execution System Sequence Diagram*

시스템에 코드 제출 관련 기능을 요청하면 Result 객체가 추가로 생성되어 관리된다. 4가지 객체가 생성되며 각 객체에서 표절 점수, 효율 점수, 기능성 점수, 그리고 가독성 점수를 계산하여 저장된다. 표절 점수는 “PlagiarismResult” 객체에서 계산되며 비교한 파일 개수랑 표절 점수가 저장된다. 효율 점수는 “EfficiencyResult” 객체에서 계산되며 Source Line of Code 점수, Data Flow 복잡도 점수, Control Flow 복잡도 점수, Reservation Words 점수가 저장된다. 가독성 점수는 “ReadabilityResult” 객체에서 계산되면 PyLint 점수, MyPy 점수, PyCodeStyle 점수가 저장된다. 마지막으로 기능성

점수는 “FunctionalityScore” 객체에서 계산되며 테스트케이스 실행 결과로 전반적인 기능성 점수를 계산한다.

## 5. Protocol Design

본 장에서는 유저와의 상호작용을 담당하는 프론트엔드 시스템과 Coding Cat 서비스의 제공을 위해 필요한 연산을 수행하는 백엔드 시스템이 각각 어떤 방식으로 작동할 지에 대한 내용을 기술한다. 또한 이 두 가지의 시스템이 어떤 방식으로 소통하는지에 대한 내용을 기술한다.

### 5.1. HTTP

Hypertext Transfer Protocol (HTTP)는 OSI model 에서 application layer에 속하는 프로토콜로서, HyperText Markup Language (HTML)와 같은 hypermedia document를 전달하는 데 사용되는 프로토콜이다. 기본적으로는 웹 브라우저와 웹 서버간의 통신을 위해 설계되었지만, 이와는 다른 용도로도 사용될 수 있다. HTTP는 클라이언트가 request 를 위해 웹 서버로의 connection을 열고, 이에 대해 웹 서버가 response를 보낼 때까지 기다리는 전통적인 클라이언트-서버 모델을 따른다. 또한 HTTP는 stateless protocol로서, 웹 서버는 클라이언트로부터 받은 request에 대한 데이터 (state)를 저장하지 않는다. 본 문서에서 기술하고 있는 Coding Cat 시스템은 HTML과 같은 hypermedia document를 JSX를 통해 구현하여 HTTP 프로토콜을 사용한다.

### 5.2. REST

시스템 컴포넌트 간 통신을 위해 따라야 하는 규칙을 Application Programming Interface (API)라고 한다. 또한, Representational State Transfer (REST)는 이러한 API의 작동 방식에 대한 조건을 부과하는 소프트웨어 아키텍처이다. REST 기반의 아키텍처를 사용할 경우, 대규모의 고성능 통신을 안정적으로 지원할 수 있고, 구현과

유지보수에 용이하다는 장점이 있다. 따라서 RESTful API는 두개의 물리적으로 분리된 컴퓨터 시스템이 인터넷을 통해 정보를 안전하게 교환하기 위해 사용하는 인터페이스이다. Coding Cat 시스템은 프론트엔드 시스템과 백엔드 시스템의 두 서브 시스템으로 나누어져 작동하고, 상위 시스템 목적의 달성을 위해서는 두개의 서브 시스템 간 통신이 이루어져야 한다. 본 Coding Cat 시스템은 안전하고 신뢰할 수 있는, 그리고 효율적인 소프트웨어 통신 표준을 따르는 RESTful API를 사용해 프론트엔드 시스템과 백엔드 시스템 간의 통신 및 사용자 인증을 지원한다.

### 5.3. Node.js

Node.js는 V8과 같은 JavaScript engine에서 실행되는 오픈 소스, 크로스 플랫폼 JavaScript runtime 환경이다. Node.js는 Scalable 네트워크 애플리케이션의 개발을 위해 만들어졌으며, 웹 브라우저 밖에서 JavaScript 코드를 실행할 수 있다는 특징이 있다. 또한 주로 Non-Blocking I/O와 Single Threaded Event Loop를 사용함으로써 높은 컴퓨팅 성능을 특징으로 가져, 웹 서버를 구동하기에 적합하다. 하지만, 본 Coding Cat 시스템에서는 Node.js를 사용한 웹 서버를 대신하여 Django 프레임워크를 사용해 웹 서버를 구현한다. 대신 Node.js는 이에 포함된 패키지 관리자인 Node Package Manager (npm)을 구동함으로써 React 라이브러리를 사용한 프론트엔드 시스템에 사용된다.

### 5.4. JSX

JavaScript Syntax Extension (JSX)은 vanilla JavaScript language syntax에 React extension을 추가한 언어 형식이다. 웹 컴포넌트의 작성을 HTML과 비슷한 방법으로 작성할 수 있어 개발하기에 용이하고, 웹 컴포넌트들의 렌더링을 손쉽게 조작할 수 있어 시스템 state에 따라 동적인 웹 애플리케이션을 쉽게 만들 수 있다는 장점이 있다.

### 5.5. JSON

JavaScript Object Notation (JSON)은 open standard 파일 형식으로, 사람이 읽을 수 있는 형태를 띠는, 데이터 교환 즉, 데이터 저장 및 전송을 위한 파일 형식이다. JSON 파일은 Attribute-Value의 쌍과 Array의 집합으로 구성되고, 이 때 담을 수 있는 자료의 타입에 큰 제한이 없다. 이러한 특징을 바탕으로 Electronic Data Interchange (EDI)에 광범위 하게 사용되고, 특히 웹 애플리케이션 (클라이언트)와 웹 서버간의 통신을 위해 주로 사용된다. Coding Cat 시스템에서는 유저 인증을 위한 JWT, 그리고 프론트엔드 서버와 백엔드 서버 간의 상호작용을 위한 REST API에 사용된다.

### 5.6. OAuth

OAuth는 Open Standard for Authorization의 약자로, 2006년 정의된 개방형 인가 관리 표준이며 API를 제공하여 상용화될 목적으로 JSON 형식으로 정의되었다. 사용자는 제 3자의 웹 서비스를 인가 절차를 통해 이용할 때, 단일성 ID와 PW와 같은 로그인 자격 증명을 제 3자의 서버에 개인정보를 전송하지 않고도 사용자의 접근 또는 기타 권한을 부여할 수 있도록 하는 것을 목적으로 한다. 현재 상용화된 버전은 OAuth 2.0이다. 사용자는 OAuth Provider에게 로그인 자격 증명을 제출하고, OAuth Provider는 사용자 접근 권한이 담긴 인가 코드를 사전에 합의된 제 3자의 서버에 전달한다. 제 3자의 서버가 인가 코드를 다시 OAuth Provider에게 제출하면, OAuth Provider는 해당 사용자 권한을 토대로 엑세스 토큰을 발급하여 전달한다.

### 5.7. JWT

JWT는 Json Web Token의 약자로, 사용자에 대한 속성 값을 담은 Claim 기반 인증 방식을 사용한다. JWT 토큰 자체로 사용자의 상태를 포함하여 의미있는 토큰으로 구성되어 있기 때문에, 별도의 중앙화된 서버를 통해 세션 및 쿠키 검증할 필요 없이 하고자하는 동작을 수행할 수 있다. 이를 통해 서버에서 발생할 수 있는 중복 검증 동작

생략 및 Stateless 아키텍처를 구성할 수 있다. 다만, 토큰에 사용자 정보를 식별할 수 있는 값이 담겨있기 때문에 토큰을 탈취당할 경우 만료되기 전까지 부가적인 제어가 불가능하다. 따라서 보안 수준과 사용성 사이의 적절한 균형을 토대로 만료 시간을 짧게 설정해야 한다.

### 5.8. API

본 장에서는 프론트엔드 서버와 백엔드 서버 사이의 상호작용을 위해 필요한 API 디자인을 설명한다.

*Table 1. Retrieve User Data API*

| 이름 | 사용자 세부 정보 조회 | | |
|---|---|---|---|
| Component | Authentication | Content Type | application/json |
| Method | GET | URL | auth/{user_id}/ |
| 요청 | Authentication Token (Required) | | |
| 응답 | User ID / Created At / Last Login / Name / Email / Nickname / Profile Image URL | | |

*Table 2. OAuth 2.0 Github Callback API*

| 이름 | Github OAuth 2.0 로그인 요청 콜백 | | |
|---|---|---|---|
| Component | Authentication | Content Type | application/json |
| Method | GET, POST | URL | auth/github/callback/ |
| 요청 | Authorization Code | | |
| 응답 | User ID / Access Token / Refresh Token | | |

*Table 3. Re-Issue Access Token API*

| 이름 | Authentication Access Token 재발급 | | |
|---|---|---|---|
| Component | Authentication | Content Type | application/json |

| 이름 | Authentication Access Token 재발급 | | |
|---|---|---|---|
| Method | POST | URL | auth/refresh/ |
| 요청 | Refresh Token | | |
| 응답 | Access Token | | |

*Table 4. List Lectures API*

| 이름 | 모든 강의 목록 조회 | | |
|---|---|---|---|
| Component | Lecture | Content Type | application/json |
| Method | GET | URL | lectures/ |
| 요청 | Authentication Token (Optional) | | |
| 응답 | [ Lecture ID / Lecture Name ] | | |

*Table 5. Create New Lecture API*

| 이름 | 신규 강의 등록 | | |
|---|---|---|---|
| Component | Lecture | Content Type | application/json |
| Method | POST | URL | lectures/ |
| 요청 | Authentication Token (Required) / Lecture Name | | |
| 응답 | Lecture ID | | |

*Table 6. Retrieve Lecture Detail API*

| 이름 | 강의 세부 정보 조회 | | |
|---|---|---|---|
| Component | Lecture | Content Type | application/json |
| Method | GET | URL | lectures/{lecture_id}/ |
| 요청 | Authentication Token (Optional) | | |
| 응답 | Lecture ID / Lecture Name | | |

*Table 7. Delete Lecture API*

| 이름 | 강의 삭제 | | |
|---|---|---|---|
| Component | Lecture | Content Type | application/json |
| Method | DELETE | URL | lectures/{lecture_id} |
| 요청 | Authentication Token (Required) | | |
| 응답 | None | | |

*Table 8. List Enrollments API*

| 이름 | 사용자의 모든 수강 정보 조회 | | |
|---|---|---|---|
| Component | Enrollment | Content Type | application/json |
| Method | GET | URL | enrollments/ |
| 요청 | Authentication Token (Required) | | |
| 응답 | [ Enrollment ID / Created At / User ID / Lecture ID ] | | |

*Table 9. Create New Enrollment API*

| 이름 | 사용자의 신규 수강 정보 등록 | | |
|---|---|---|---|
| Component | Enrollment | Content Type | application/json |
| Method | POST | URL | enrollments/ |
| 요청 | Authentication Token (Required) / Lecture ID | | |
| 응답 | Enrollment ID | | |

*Table 10. Retrieve Enrollment Detail API*

| 이름 | 사용자의 수강 정보 세부 조회 | | |
|---|---|---|---|
| Component | Enrollment | Content Type | application/json |
| Method | GET | URL | enrollments/{enrollment_id}/ |
| 요청 | Authentication Token (Required) | | |

| 이름 | 사용자의 수강 정보 세부 조회 |
|---|---|
| 응답 | Enrollment ID / Created At / User ID / Lecture ID |

*Table 11. Delete Enrollment API*

| 이름 | 사용자의 수강 정보 삭제 | | |
|---|---|---|---|
| Component | Enrollment | Content Type | application/json |
| Method | DELETE | URL | enrollments/{enrollment_id}/ |
| 요청 | Authentication Token (Required) | | |
| 응답 | Enrollment ID | | |

*Table 12. List Assignments API*

| 이름 | 특정 강의 모든 과제 정보 조회 | | |
|---|---|---|---|
| Component | Assignment | Content Type | application/json |
| Method | GET | URL | assignments/ |
| 요청 | Authentication Token (Optional) / Lecture ID | | |
| 응답 | [ Assignment ID / Assignment Name / Deadline / Question / Constraints / Skeleton Code / Answer Code ] | | |

*Table 13. Create New Assignment API*

| 이름 | 특정 강의 신규 과제 등록 | | |
|---|---|---|---|
| Component | Assignment | Content Type | application/json |
| Method | POST | URL | assignments/ |
| 요청 | Authentication Token (Required) / Lecture ID / Assignment Name / Deadline / Question / Constraints / Skeleton Code / Answer Code | | |
| 응답 | Assignment ID | | |

*Table 14. Retrieve Assignment Detail API*

| 이름 | 과제 세부 정보 조회 | | |
|---|---|---|---|
| Component | Assignment | Content Type | application/json |
| Method | GET | URL | assignments/{assignment_id}/ |
| 요청 | Authentication Token (Optional) | | |
| 응답 | Assignment ID / Assignment Name / Deadline / Question / Constraints / Skeleton Code / Answer Code | | |

*Table 15. Delete Assignment API*

| 이름 | 과제 삭제 | | |
|---|---|---|---|
| Component | Assignment | Content Type | application/json |
| Method | DELETE | URL | assignments/{assignment_id}/ |
| 요청 | Authentication Token (Required) | | |
| 응답 | None | | |

*Table 16. List Test Cases API*

| 이름 | 특정 과제의 모든 공개 테스트 케이스 조회 | | |
|---|---|---|---|
| Component | Testcase | Content Type | application/json |
| Method | GET | URL | testcases/ |
| 요청 | Authentication Token (Required) / Assignment ID | | |
| 응답 | [ Test Case ID / Created At / Input / Output ] | | |

*Table 17. Create New Test Case API*

| 이름 | 특정 과제의 신규 테스트 케이스 등록 | | |
|---|---|---|---|
| Component | Testcase | Content Type | application/json |
| Method | POST | URL | testcases/ |

| 이름 | 특정 과제의 신규 테스트 케이스 등록 |
|---|---|
| 요청 | Authentication Token (Required) / Assignment ID / Input / Output / Hidden |
| 응답 | Test Case ID |

*Table 18. Retrieve Test Case Detail API*

| 이름 | 공개 테스트 케이스 세부 정보 조회 | | |
|---|---|---|---|
| Component | Testcase | Content Type | application/json |
| Method | GET | URL | testcases/{testcase_id}/ |
| 요청 | Authentication Token (Required) | | |
| 응답 | Test Case ID / Created At / Input / Output | | |

*Table 19. Delete Test Case API*

| 이름 | 테스트 케이스 제거 | | |
|---|---|---|---|
| Component | Testcase | Content Type | application/json |
| Method | DELETE | URL | testcases/{testcase_id}/ |
| 요청 | Authentication Token (Required) | | |
| 응답 | None | | |

*Table 20. List Repositories API*

| 이름 | 특정 과제의 모든 과거 풀이 코드 조회 | | |
|---|---|---|---|
| Component | Repository | Content Type | application/json |
| Method | GET | URL | repos/ |
| 요청 | Authentication Token (Required) / Assignment ID | | |
| 응답 | [ Repo ID / Created ID / Modified At / Content ] | | |

*Table 21. Create New Repository API*

| 이름 | 특정 과제의 신규 풀이 코드 저장 | | |
|---|---|---|---|
| Component | Repository | Content Type | application/json |
| Method | POST | URL | repos/ |
| 요청 | Authentication Token (Required) / Assignment ID / Content | | |
| 응답 | Repo ID | | |

*Table 22. Retrieve Repository API*

| 이름 | 과거 풀이 코드 세부 정보 조회 | | |
|---|---|---|---|
| Component | Repository | Content Type | application/json |
| Method | GET | URL | repos/{repo_id}/ |
| 요청 | Authentication Token (Required) / Assignment ID | | |
| 응답 | Repo ID / Created ID / Modified At / Content | | |

*Table 23. Update Repository API*

| 이름 | 풀이 코드 저장 | | |
|---|---|---|---|
| Component | Repository | Content Type | application/json |
| Method | PUT | URL | repos/{repo_id}/ |
| 요청 | Authentication Token (Required) / Content | | |
| 응답 | Repo ID | | |

*Table 24. Execution Exercises Result API*

| 이름 | 풀이 코드 실행 및 결과 전달 | | |
|---|---|---|---|
| Component | Output | Content Type | application/json |
| Method | POST | URL | outputs/exercises/ |
| 요청 | Authentication Token (Required) / Repo ID / Content / Input | | |

| 이름 | 풀이 코드 실행 및 결과 전달 |
|---|---|
| 응답 | Output |

*Table 25. Execution a Test Case Result API*

| 이름 | 특정 테스트 케이스 결과 전달 | | |
|---|---|---|---|
| Component | Output | Content Type | application/json |
| Method | POST | URL | outputs/testcases/{testcase_id}/ |
| 요청 | Authentication Token (Required) / Repo ID / Content | | |
| 응답 | Hidden / Expected Output / Actual Output | | |

*Table 26. Execution Submission Result API*

| 이름 | 과제 제출 및 결과 전달 | | |
|---|---|---|---|
| Component | Output | Content Type | application/json |
| Method | POST | URL | outputs/assignments/{assignment_id}/ |
| 요청 | Authentication Token (Required) / Repo ID / Content | | |
| 응답 | Answer Code / Code Explain / References / Functionality Result / Plagiarism Result / Efficiency Result / Readability Result | | |

## 6. Database Design

### 6.1. Entity Relationship Diagram

*Table 27. Table 'Users' 구조*

| Field | Key | Type | Description |
|---|---|---|---|
| id | PK | Int | 사용자 아이디 |
| nickname | | String | 사용자 닉네임 |
| oauth_id | | String | 사용자 oauth |

| Field | Key | Type | Description |
|---|---|---|---|
| name | | String | 사용자 이름 |
| email | | String | 사용자 이메일 |
| created_at | | Timestamp | 사용자 생성 시간 |
| last_login | | Timestamp | 마지막 로그인 시간 |
| profile_image_url | | URL | 프로필 이미지 url |
| github_api_url | | URL | github api url |
| github_profile_url | | URL | github profile url |

*Table 28. Table 'Lectures' 구조*

| Field | Key | Type | Description |
|---|---|---|---|
| id | PK | Int | 강의 아이디 |
| instructor_id | FK | Int | 강사 아이디 |
| name | | String | 강의 이름 |

*Table 29. Table 'Enrollments' 구조*

| Field | Key | Type | Description |
|---|---|---|---|
| id | PK | Int | 등록 아이디 |
| student_id | FK | String | 학생 아이디 |
| lecture_id | FK | String | 강의 아이디 |
| created_at | | Timestamp | 등록 생성 시간 |

*Table 30. Table 'Assignments' 구조*

| Field | Key | Type | Description |
|---|---|---|---|
| id | PK | Int | 과제 아이디 |
| lecture_id | FK | Int | 강의 아이디 |

| Field | Key | Type | Description |
|---|---|---|---|
| name | | String | 과제 이름 |
| deadline | | Timestamp | 과제 마감일 |
| question | | String | 과제 질문 사항 |
| constraints | | String | 과제 제약 사항 |
| skeleton_code | | Json | 초기 코드 |
| answer_code | | Json | 정답 코드 |

*Table 31. Table 'Testcases' 구조*

| Field | Key | Type | Description |
|---|---|---|---|
| id | PK | Int | 테스트케이스 아이디 |
| assignment_id | FK | Int | 과제 아이디 |
| created_at | | Timestamp | 테스트케이스 생성 시간 |
| input | | String | 테스트케이스 입력 |
| output | | String | 테스트케이스 출력 |
| hidden | | Boolean | 테스트케이스 공개 여부 |

*Table 32. Table 'Repo' 구조*

| Field | Key | Type | Description |
|---|---|---|---|
| id | PK | Int | 저장소 아이디 |
| author_id | FK | Int | 저장한 학생 아이디 |
| assignment_id | FK | Int | 과제 아이디 |
| created_at | | Timestamp | 저장소 생성 시간 |
| modified_at | | Timestamp | 저장소 수정 시간 |
| content | | Json | 저장 코드 |

*Table 33. Table 'Result' 구조*

| Field | Key | Type | Description |
|---|---|---|---|
| id | PK | Int | 결과 아이디 |
| repo_id | FK | Int | 저장소 아이디 |
| code_explain | | String | 코드 설명 |
| references | | Json | 코드 참조 사항 |

*Table 34. Table 'Functionality_result' 구조*

| Field | Key | Type | Description |
|---|---|---|---|
| id | PK | Int | 기능 채점 결과 아이디 |
| result_id | FK | Int | 결과 아이디 |
| testcase_result | | Json | 테스트케이스 실행 결과 및 정보 |

*Table 35. Table 'Plagiarism_result' 구조*

| Field | Key | Type | Description |
|---|---|---|---|
| id | PK | Int | 표절 채점 결과 아이디 |
| result_id | FK | Int | 결과 아이디 |
| num_files | | Int | 비교한 파일 개수 |
| similarity_score | | Int | 표절 점수 |

*Table 36. Table 'Efficiency_result' 구조*

| Field | Key | Type | Description |
|---|---|---|---|
| id | PK | Int | 효율 채점 결과 아이디 |
| result_id | FK | Int | 결과 아이디 |
| loc_score | | Int | Line of Score 점수 |
| dataflow_complexit | | Int | Data Flow 복잡도 점수 |

| Field | Key | Type | Description |
|---|---|---|---|
| y_score | | | |
| controlflow_complexity_score | | Int | Control Flow 복잡도 점수 |
| reservation_words_score | | Int | Unique operand 점수 |

*Table 37. Table 'Readability_result' 구조*

| Field | Key | Type | Description |
|---|---|---|---|
| id | PK | Int | 가독성 채점 결과 아이디 |
| result_id | FK | Int | 결과 아이디 |
| pylint_result | | Int | pylint 채점 결과 점수 |
| mypy_result | | Int | mypy 채점 결과 점수 |
| pycodestyle_resurlt | | Int | pycodestyle 채점 결과 점수 |

## 6.2. Relational Schema

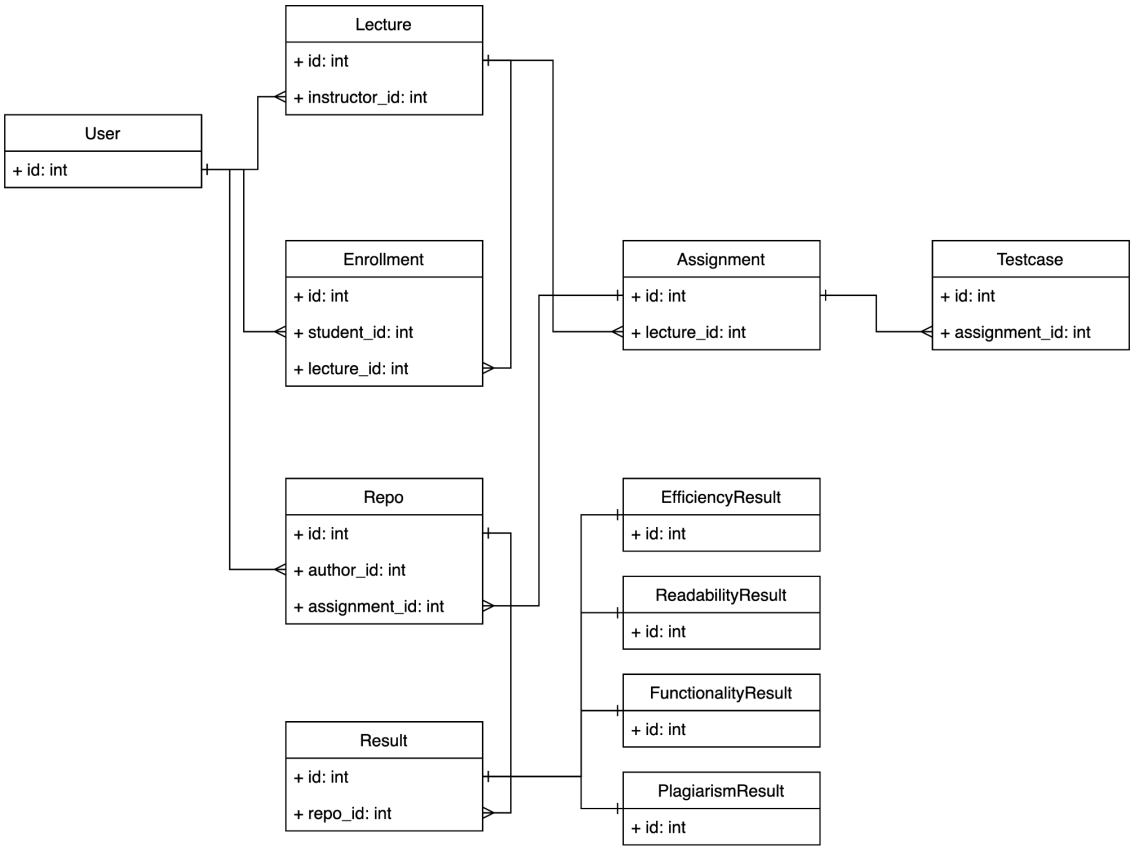


*Figure 23. Overall Relational Schema*

# 7. Testing Plan

## 7.1. Objective

본 장에서는 요구사항 명세서에서 기술한 시나리오를 바탕으로 통합 시스템이 동작하는 지 확인한다. 유닛 테스트, 통합 테스트 과정을 거쳐 시스템 출시 전에 발생할 수 있는 잠재적인 오류 혹은 결함을 찾아내고자 한다. 이를 위해 세가지 관점으로 나누어 테스트 과정을 설계하며, 세부 테스트 케이스를 설명한다.

## 7.2. Testing Policy

### 7.2.1. Development Testing

본 시스템 개발 과정 중에 발생할 수 있는 리스크와 비용을 줄이는 것을 목표로 하며, 먼저 시스템을 구성하는 서브 시스템 혹은 컴포넌트 단위의 독립적인 유닛 테스트를 진행한다. 유닛 테스트는 서비스하고자 하는 기능이 예상대로 동작하는 지를 중점적으로 확인한다. 더불어 구현 컴포넌트의 유지보수성을 높이고 잠재적인 결함을 발견하기 위해 상호 간의 코드 리뷰를 진행한다. 독립 테스트 진행 후, 각 컴포넌트들을 순차적으로 통합하여 통합 테스트를 진행한다. 통합 테스트는 기능적인 항목 뿐만 아니라 비기능적인 성능, 보안, 안정성 등을 검증해야 한다. 이를 위해 정적 코드 분석, 데이터 흐름 분석 등을 수행할 수 있다.

### 7.2.2. Release Testing

본 시스템의 새로운 버전 출시에 따른 배포 과정이 정상적으로 수행될 수 있는 지를 검증한다. 시스템이 최신 버전의 온전한 상태로 배포되어 작동에 어떠한 하자가 없어야 한다. 일반적으로 목표로 하는 기능을 포함한 알파 테스트를 거쳐, 그 과정에서 발견한 피드백 및 결함을 반영하여 베타 테스트를 진행한다. 알파 테스트의 경우 개발 진행 조직 내부에서 진행하며, 베타 테스트의 경우 제한된 사용자에게 공개하여 진행한다.

### 7.2.3. User Testing

본 시스템의 예상 사용자가 사용하는 과정을 시험함으로써, 본 시스템이 모든 요구사항 명세를 충족하는 지 검증한다. 이를 위해 동시에 접속하는 유저의 수를 포함하는 시스템 사용 시나리오를 작성하고, 이 시나리오에서 Coding Cat 시스템이 모든 Use Case를 충족할 수 있는지를 검증한다. 검증 결과를 바탕으로 Coding Cat 시스템의

베타 버전의 유용성을 평가하고, 최종 결과물로서 실제 환경에 Deployment 될 것인지를 결정한다.

## 7.3. Test Case

### 7.3.1. 소셜 로그인

*Table 38. 로그인 테스트*

| Element | Description |
|---|---|
| Summary | Coding Cat 시스템에 접근하기 위해서는 OAuth 로그인을 수행해야 한다. |
| Pre-Conditions | ● 사용자는 Github 계정을 소유하고 있다. |
| Test Steps | 1. 로그인 페이지 상의 Github 버튼을 클릭하여 로그인 요청<br>2. 사용자는 본인 계정으로 Github 로그인을 진행<br>3. Github이 서버로 사용자에 대한 인가 코드 발급<br>4. 서버는 인가 코드를 다시 Github에 제출하여 사용자의 프로필 정보를 요청<br>5. 서버는 응답 받은 프로필 정보를 토대로 DB에 기록<br>6. 서버는 사용자의 정보를 토대로 인증 토큰을 발급하여 반환 |
| Expected Results | ● 랜딩 페이지 상의 프로필 정보가 보여지게 된다. |

### 7.3.2. 시스템 접근 차단

*Table 39. 접근 차단 테스트*

| Element | Description |
|---|---|
| Summary | Coding Cat 시스템에 속한 모든 서비스에 접근하기 위해서는 OAuth로그인을 마친 상태여야 한다. |
| Pre-Conditions | ● 사용자는 로그인을 하지 않은 anonymous 상태이다.<br>● 사용자의 인터넷 환경이 원활해야 한다.<br>● anonymous 상태에서 유저는 랜딩 페이지의 어떤 버튼을 클릭하더라도, 로그인 페이지로 이동해야 한다. |

| Element | Description |
|---|---|
| Test Steps | 1. 랜딩 페이지에서 “회원가입 | 로그인” 버튼을 마우스로 클릭해 로그인 페이지로 이동<br>2. 나타나는 로그인 페이지에서 “로그인” 버튼을 마우스로 클릭해 GitHub OAuth2.0 로그인 수행<br>3. 로그인을 마치면 랜딩 페이지로 다시 이동 |
| Expected Results | ● 랜딩 페이지 상의 “회원가입 | 로그인” 버튼이 GitHub 계정을 표시하는 아이콘으로 바뀜<br>● 설정 버튼과 문제 버튼을 포함한 Coding Cat 시스템에 접근할 수 있게 됨 |

### 7.3.3. 시스템 설정

*Table 40. 설정 테스트*

| Element | Description |
|---|---|
| Summary | 로그인을 마친 사용자는 본인의 시스템 사용환경을 조작할 수 있다. |
| Pre-Conditions | ● GitHub OAuth 2.0 로그인을 마친 상태이다.<br>● 사용자의 인터넷 연결이 원활하다. |
| Test Steps | 1. 랜딩 페이지에서 좌측 상단에 위치한 “톱니바퀴 아이콘”을 마우스로 클릭하여 설정 페이지로 이동<br>2. 설정 페이지에서 마우스로 “배경 색” 우측에 위치한 흰색 원과 검은색 원을 마우스로 클릭함으로써 각각 화이트 모드와 다크모드를 선택<br>3. “코드 에디터” 하단에 위치한 “Language”와 “Theme” 드롭다운 메뉴를 마우스로 클릭해 프로그래밍 언어와 코드 에디터의 테마를 설정<br>4. 설정 박스 최하단의 “설정 완료” 버튼을 마우스로 클릭해 이전 페이지로 돌아감 |
| Expected Results | ● Coding Cat 시스템의 테마와 코드 에디터의 설정이 반영된다. |

### 7.3.4. 강의 및 과제 목록 확인

*Table 41. 및 과제 목록 확인 테스트*

| Element | Description |
|---|---|
| Summary | 로그인을 마친 사용자는 랜딩 페이지에서 문제 버튼을 클릭해 본인에게 할당된 강의 및 과제 정보를 열람할 수 있다. |
| Pre-Conditions | • GitHub OAuth 2.0 로그인을 마친 상태이다.<br>• 사용자의 인터넷 연결이 원활하다.<br>• 강의 및 과제 정보가 등록되어 있어야 한다.<br>• 사용자에게 할당된 강의가 적어도 하나 존재한다.<br>• 강의에 할당된 과제가 적어도 하나 존재한다. |
| Test Steps | 1. 랜딩 페이지에서 배너에 위치한 “문제” 버튼을 클릭하여 과제 정보 페이지로 이동<br>2. 데이터베이스에서 사용자의 Enrollment 정보 및 그에 따른 Lecture, Assignment 정보 가져오기<br>3. 가져온 정보를 바탕으로 사용자가 강의 및 과제 목록 확인 |
| Expected Results | • 사용자는 자신에게 할당된 과제에 대해 강의명, 과제명, 남은 기한을 확인할 수 있다.<br>• 남은 기간에 한해, 하루 이상의 시간이 남은 경우에는 검은색 스톱워치 아이콘이, 하루 미만의 시간이 남은 경우 빨간 스톱워치 아이콘이 나타난다.<br>• 제출 완료된 과제는 음영처리되어 표시된다. |

### 7.3.5. 과제 선택

*Table 42. 선택 테스트*

| Element | Description |
|---|---|
| Summary | 사용자는 과제 정보를 열람 중, 과제 항목을 선택하여 과제를 수행할 수 있다. |
| Pre-Conditions | • GitHub OAuth 2.0 로그인을 마친 상태이다.<br>• 사용자의 인터넷 연결이 원활하다.<br>• 강의 및 과제 정보가 등록되어 있어야 한다. |

| Element | Description |
|---|---|
| | • 사용자에게 할당된 강의가 적어도 하나 존재한다.<br>• 강의에 할당된 과제가 적어도 하나 존재한다. |
| Test Steps | 1. 과제 정보 페이지에서 과제를 마우스로 클릭하여 코드 에디터 페이지로 이동 |
| Expected Results | • 사용자는 코드 에디터 페이지로 이동해 과제를 수행할 수 있다. |

### 7.3.6. 코드 에디터 화면 표시

*Table 43. 에디터 화면 표시 테스트*

| Element | Description |
|---|---|
| Summary | 사용자는 선택한 과제에 대한 상세 정보를 코드에디터 상에서 올바르게 확인할수 있다. |
| Pre-Conditions | • GitHub OAuth 2.0 로그인을 마친 상태이다.<br>• 사용자의 인터넷 연결이 원활하다.<br>• 강의 및 과제 정보가 등록되어 있어야 한다.<br>• 사용자에게 할당된 강의가 적어도 하나 존재한다.<br>• 강의에 할당된 과제가 적어도 하나 존재한다.<br>• 사용자는 과제 정보 페이지에서 어떠한 과제를 선택하였다. |
| Test Steps | 1. 데이터베이스에서 Assignment 정보 및 그에 해당하는 Testcase 정보 가져오기<br>2. 강의명이 올바르게 표시되는지 확인<br>3. 과제명이 올바르게 표시되는지 확인<br>4. 남은 기한이 올바르게 표시되는지 확인<br>5. 문제, 참조 및 제약사항이 올바르게 표시되는지 확인<br>6. 과제에 사용되는 숨겨진 테스트 케이스를 제외한 모든 테스트 케이스들이 올바르게 표시되는지 확인<br>7. 코드 입력창 및 스켈레톤 코드가 올바르게 표시되는지 확인 |
| Expected Results | • 사용자는 올바르게 표시된 과제정보를 이해하고 과제를 수행할 수 있다. |

### 7.3.7. 코드 중간 저장

*Table 44. 중간 저장 테스트*

| Element | Description |
|---|---|
| Summary | 사용자는 선택한 과제에 대해 과제를 수행하는 중간에 코드를 저장할 수 있다. |
| Pre-Conditions | • GitHub OAuth 2.0 로그인을 마친 상태이다.<br>• 사용자의 인터넷 연결이 원활하다.<br>• 사용자는 코드 에디터 페이지에 위치한다. |
| Test Steps | 1. 코드에디터 창에 코드를 입력<br>2. 화면 우측 상단에 위치한 “폴더 버튼”을 클릭<br>3. 사용자, 과제, 코드, 저장 시간 정보 데이터베이스의 Repo에 저장<br>4. 폴더 버튼에 색이 채워진 경우 다시 클릭하여 코드를 불러오기<br>5. 데이터베이스에서 Repo 정보 가져오기<br>6. 해당 Repo 정보 데이터베이스에서 삭제 |
| Expected Results | • 코드를 불러온 후, 폴더 버튼은 색이 지워지고, 중간 저장되었던 코드는 코드 입력 창에 다시 나타난다. |

### 7.3.8. 과제 정보 페이지로 이동

*Table 45. 정보 페이지로 이동 테스트*

| Element | Description |
|---|---|
| Summary | 사용자는 선택한 과제에 대해 과제를 수행하는 중간에 과제 정보 페이지로 돌아갈 수 있다. |
| Pre-Conditions | • GitHub OAuth 2.0 로그인을 마친 상태이다.<br>• 사용자의 인터넷 연결이 원활하다.<br>• 사용자는 코드 에디터 페이지에 위치한다. |
| Test Steps | 1. 코드 에디터 좌측 상단에 위치한 “되돌아가기” 버튼을 마우스로 클릭<br>2. 과제 정보 페이지로 이동 |

| Element | Description |
|---|---|
| Expected Results | • 사용자는 본인의 과제 정보 페이지로 되돌아간다. |

### 7.3.9. 코드 복사

*Table 46. 복사 테스트*

| Element | Description |
|---|---|
| Summary | 사용자는 선택한 과제에 대해 과제를 수행하는 중간에 코드를 시스템 클립보드로 복사할 수 있다. |
| Pre-Conditions | • GitHub OAuth 2.0 로그인을 마친 상태이다.<br>• 사용자의 인터넷 연결이 원활하다.<br>• 사용자는 코드 에디터 페이지에 위치한다. |
| Test Steps | 1. 코드 에디터 우측 상단에 위치한 “코드 복사” 버튼을 마우스로 클릭<br>2. 클립보드에 코드 복사 |
| Expected Results | • 클립보드에 코드가 복사되어 사용자의 로컬 시스템에 코드를 붙여넣을 수 있다. |

### 7.3.10. 코드 초기화

*Table 47. 초기화 테스트*

| Element | Description |
|---|---|
| Summary | 사용자는 선택한 과제에 대해 과제를 수행하는 중간에 과제에 기본적으로 포함된 스켈레톤 코드로 사용자가 작성한 코드를 초기화할 수 있다. |
| Pre-Conditions | • GitHub OAuth 2.0 로그인을 마친 상태이다.<br>• 사용자의 인터넷 연결이 원활하다.<br>• 사용자는 코드 에디터 페이지에 위치한다. |
| Test Steps | 1. 코드 에디터 우측 상단에 위치한 “코드 초기화” 버튼을 마우스로 클릭 |

| Element | Description |
|---|---|
| | 2. 코드 에디터에 작성한 코드 삭제<br>3. 코드 에디터에 스켈레톤 코드 표시 |
| Expected Results | • 코드 에디터 페이지가 초기화되고, 스켈레톤 코드가 에디터 창에 표시된다. |

### 7.3.11. 로컬 코드 다운로드

*Table 48. 코드 다운로드 테스트*

| Element | Description |
|---|---|
| Summary | 사용자는 선택한 과제에 대해 과제를 수행하는 중간에 코드를 사용자 시스템으로 다운로드 할 수 있다. |
| Pre-Conditions | • GitHub OAuth 2.0 로그인을 마친 상태이다.<br>• 사용자의 인터넷 연결이 원활하다.<br>• 사용자는 코드 에디터 페이지에 위치한다. |
| Test Steps | 1. 코드 에디터 우측 상단에 위치한 “코드 다운로드” 버튼을 마우스로 클릭<br>2. 웹 브라우저 상의 다운로드 창에서 원하는 경로를 선택한 뒤, 코드를 다운로드 |
| Expected Results | • 다운로드 된 코드는 사용자의 로컬 시스템에서 접근할 수 있다. |

### 7.3.12. 로컬 코드 업로드

*Table 49. 코드 업로드 테스트*

| Element | Description |
|---|---|
| Summary | 사용자는 선택한 과제에 대해 과제를 수행하는 중간에 사용자 시스템에 위치한 코드를 사용자 Coding Cat 시스템으로 업로드 할 수 있다. |
| Pre-Conditions | • GitHub OAuth 2.0 로그인을 마친 상태이다.<br>• 사용자의 인터넷 연결이 원활하다. |

| Element | Description |
|---|---|
| | • 사용자는 코드 에디터 페이지에 위치한다. |
| Test Steps | 1. 코드 에디터 좌측 상단에 위치한 “코드 업로드” 버튼을 마우스로 클릭<br>2. 웹 브라우저 상의 업로드 창에서 원하는 파일을 선택한 뒤, 코드를 업로드 |
| Expected Results | • 업로드 된 코드는 사용자의 코드 에디터 페이지에 나타나고, 사용자는 과제를 계속해서 수행할 수 있다. |

### 7.3.13. 코드 실행

*Table 50. 실행 테스트*

| Element | Description |
|---|---|
| Summary | 사용자는 선택한 과제에 대해 과제를 수행하는 과정에서 코드의 실행 결과를 “실행” 눌렀을 때 “실행 결과” 창에 확인할 수 있다. |
| Pre-Conditions | • GitHub OAuth 2.0 로그인을 마친 상태이다.<br>• 사용자의 인터넷 연결이 원활하다.<br>• 사용자는 코드 에디터 페이지에 위치한다.<br>• 사용자는 코드 에디터에 코드를 작성하고 “실행”을 시킨다. |
| Test Steps | 1. 코드 에디터에 코드를 입력<br>2. 코드 에디터 우측에 위치한 “실행” 버튼을 마우스로 클릭 |
| Expected Results | • 실행시킨 코드 결과를 터미널 표준 출력처럼 우측 “실행결과” 화면에서 볼 수 있다. |

### 7.3.14. 코드 채점

*Table 51. 채점 테스트*

| Element | Description |
|---|---|
| Summary | 사용자는 선택한 과제에 대해 과제를 수행하는 과정에서 코드 에디터에 입력한 코드를 제출 전에 채점 해볼 수 있다. |

| Element | Description |
|---|---|
| Pre-Conditions | ● GitHub OAuth 2.0 로그인을 마친 상태이다.<br>● 사용자의 인터넷 연결이 원활하다.<br>● 사용자는 코드 에디터 페이지에 위치한다.<br>● 사용자는 코드 에디터에 코드가 존재한다. |
| Test Steps | 1. 코드 에디터에 코드를 입력<br>2. 코드 에디터 우측에 위치한 “채점” 버튼을 마우스로 클릭 |
| Expected Results | ● 성공적인 코드 실행시 우측에 위치한 “채점결과” 화면에서 총점, 각 테스트케이스 간 통과 또는 실패를 확인 할 수 있다.<br>● 코드가 실행 안될 경우 traceback을 같은 화면에서 보여준다. |

#### 7.3.15. 과제 제출

*Table 52. 과제 제출 테스트*

| Element | Description |
|---|---|
| Summary | 사용자는 선택한 과제에 대해 과제를 수행하는 과정에서 코드의 실행 결과를 “실행” 눌렀을 때 “실행 결과” 창에 확인할 수 있다. |
| Pre-Conditions | ● GitHub OAuth 2.0 로그인을 마친 상태이다.<br>● 사용자의 인터넷 연결이 원활하다.<br>● 사용자는 코드 에디터 페이지에 위치한다.<br>● 사용자는 코드 에디터에 코드 작성 완료 상태 |
| Test Steps | 1. 코드 에디터에 코드를 입력<br>2. 코드 에디터 우측에 위치한 “제출” 버튼을 마우스로 클릭 |
| Expected Results | ● 코드가 성공적으로 실행된 경우 우측에 위치한 “제출결과” 화면에서 제출한 코드의 총점, 기능 점수, 효율 점수, 가독성 점수를 확인 할 수 있다.<br>● 코드 실행이 실패한 경우, 동일한 우측 화면에서 Traceback을 알려준다. |

# 8. Development Design

## 8.1. Objective

본 장에서는 시스템 구현을 위한 기술 항목, 개발 환경과 개발 과정에 있어서 지켜져야 하는 제약 사항을 기술한다.

## 8.2. Development Environment

### 8.2.1. Git


*Figure 24. Logo of Git*

Git은 분산된 환경에서의 개발 환경을 구축할 수 있고, 버전 컨트롤이 가능한 오픈소스 서비스이다. 본 시스템을 개발함에 있어서 컴포넌트 별로 개발 사항들을 브랜치로 구분하기 용이하고, 코드 리뷰 진행 및 시스템 통합하는 과정에 유용하기에 Git을 활용한다.

### 8.2.2. React

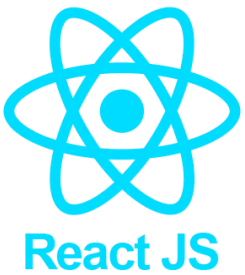


*Figure 25. Logo of React*

React (React.js 또는 React JS)는 무료 오픈 소스 프론트엔드 자바스크립트 라이브러리로서, 시스템의 UI/UX를 구축하는데 사용된다. 본 시스템은 Functional Programming으로 React를 사용하여 웹페이지의 유지 보수를 용이하게 하며, React Hook 및 Redux를 기반으로 상태를 관리한다.

### 8.2.3. Redux


*Figure 26. Logo of Redux*

Redux는 오픈소스 JavaScript 라이브러리로, JavaScript 애플리케이션의 State를 관리한다. 순수함수를 사용하기 때문에 State를 예측 가능케 하여 동적 웹 애플리케이션의 유지 보수에 용이하다. 따라서 본 시스템은 Redux를 시스템의 상태 관리 라이브러리로서 사용한다.

### 8.2.4. styled-components

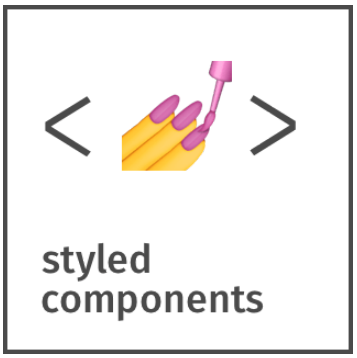


*Figure 27. Logo of styled-components*

Styled-Components는 React 개발자를 위해 만들어진 라이브러리로서, React Component 수준에서 CSS 스타일링을 가능케 한다. Styled-Components 라이브러리는 CSS-in-JS 기술을 활용하여 JavaScript 파일안에서 CSS 파일 형식을 담는 React

Component를 손쉽게 생성하고, 이를 렌더링할 수 있게 한다. 본 시스템은 컴포넌트 별 디자인에 독립성을 부여하여 유지 보수를 용이하게 하기 위해, 이 라이브러리를 사용하여 개발되었다.

### 8.2.5. Django


*Figure 28. Logo of Django*

Django는 Python 기반의 오픈소스 웹 애플리케이션을 위한 프레임워크이다. 웹 서버를 개발함에 있어서 상당 부분을 추상화하여 비교적 수월하게 개발할 수 있고 이를 통해 짧은 시간 주기의 개발이 가능하면서도 준수한 성능을 보여준다. 더불어 확장성을 고려하여 설계되어 다양한 추가 기능 및 미들웨어 적용이 가능하다. 본 시스템 제약 사항으로 Python 기반으로 개발되어야 한다는 점과 추후 확장 가능하다는 점, 그리고 범용적으로 이용될 수 있다는 점을 고려하여 Django를 활용한다.

### 8.2.6. SQLite


*Figure 29. Logo of SQLite*

SQLite는 C언어 기반의 SQL 데이터베이스 엔진이다. 서버 없이 시스템에 내장되어, 적은 용량을 차지하고 빠르게 동작 가능하다는 특징이 있다. 또한 다양한 플랫폼에서 동작 가능하고, 하위 호환성을 지원한다. 이러한 점 때문에 모바일 기기나

비교적 데이터 보관 소요가 적은 시스템에서 많이 활용된다. 본 시스템의 예상 규모가 크지 않고 빠른 개발이 필요로 하므로, SQLite 데이터베이스를 활용한다.

#### 8.2.7. Docker

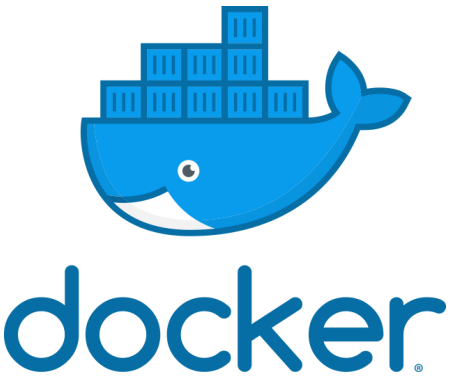


*Figure 30. Logo of Docker*

Docker는 매우 다양한 애플리케이션들을 독립적인 격리 공간, 하나의 컨테이너에서 구동하여 관리 가능한 서비스이다. 하나의 시스템에서도 크고 작은 규모의 수 많은 서버가 동작 해야하는 최근 기술 소요에 적합한 서비스를 제공한다. 본 시스템에서는 다양한 언어를 컴파일 및 구동시키면서도 백엔드 서버의 안정성을 유지해야하기 때문에 Docker를 활용한다.

### 8.3. Development Constraints

본 시스템은 소프트웨어 요구사항 명세서와 소프트웨어 디자인 명세서에 기술된 내용을 기반으로 설계 및 구현된다. 이외의 세부 사항은 개발진의 기술 능력을 반영하여, 다음 사항을 준수해야 한다.

- 범용적으로 사용되며 검증된 기술 및 서비스를 사용한다.
- 상업용 라이선스가 필요하거나 추가 비용을 지불 해야하는 서비스는 사용을 지양한다.
- 오픈소스 소프트웨어 사용을 지향한다.

- 시스템의 가용성과 확장성을 고려하여 설계 및 구현한다.
- 시스템의 개발 및 유지보수 비용을 고려하여 설계 및 구현한다.
- Git Flow 브랜치 전략을 채택한다. 단, 개발 수준에 따라 일부 수정될 수 있다.
- 프론트엔드 시스템 개발에 atomic design pattern을 적용함으로써 코드의 재사용성을 높이고, 시스템의 유지 보수를 쉽게 만든다.

# 9. Supporting Information

## 9.1. Software Design Specification

본 소프트웨어 디자인 명세서는 IEEE 권장 사항에 맞추어 작성되었다 (IEEE Standard for Information Technology Systems Design Software Design Descriptions, IEEE-Std-1016). 다만 본 시스템의 설계 사항을 용이하게 파악할 수 있도록 본래의 양식에서 일부 추가되거나 제외된 항목이 있다.

## 9.2. Document History

*Table 53. Document History*

| Date | Version | Description | Write |
|---|---|---|---|
| 2022.11.11 | 0.1 | 초안 작성 완료 | 박승호 외 5인 |
| 2022.11.13 | 1.0 | 최초 버전 발행 | 박승호 외 5인 |