Software Design Specification of Web Coding Test Service

Prepared by:

강지수, 박경수, 양환석, 조용현, 한건희, 홍시현

2022.11.12

SUNG KYUN KWAN UNIVERSITY(SKKU)

CONTENTS

1. Introduction

1.1 Purpose of this document

이 문서는 2022-2 소프트웨어공학개론 수업에서 제작하는 코딩 테스트 소프트웨어의 고수준 디자인 요구사항을 포함한다. 사용되는 컴포넌트를 설명하고 컴포넌트의 관계를 다이어그램으로 표현하였다.

1.2 Development range

이 프로젝트는 성균관대학교 학생들을 위한 코딩 관련 편의 기능을 갖춘 프로그램을 만드는 목적을 가지고 있다. 이 프로그램은 웹을 통하여 제공될 예정이며 학생들이 편하게 코딩하고 과제를 제출할 수 있게 도와줄 것이다. 아울러 여러가지 코딩 채점 Metrics와 관련 학습 자료를 제공하여 학생들의 코딩 능력을 향상하는데 도움을 줄 것이다.

1.3 Definitions, acronyms

- **User**: 코딩 테스트 서비스를 이용하는 성균관대 학생
- **Frontend**: 소프트웨어에서 유저 인터페이스의 동작을 담당하는 부분
- **backend**: 소프트웨어에서 서비스의 로직과 데이터 관리를 담당하는 부분
- **API**: Application Programming Interface
- **DDL**: Database Definition Language
- **ER Diagram**: Entity Relationship Diagram
- **End-to-end**: Frontend와 backend사이의 관계

2. System Architecture

2.1 Architecture Overview

소프트웨어의 백엔드 구조, 프론트엔드 구조, 백엔드-프론트 연결 구조를 아래의 도식으로 표현하였다.

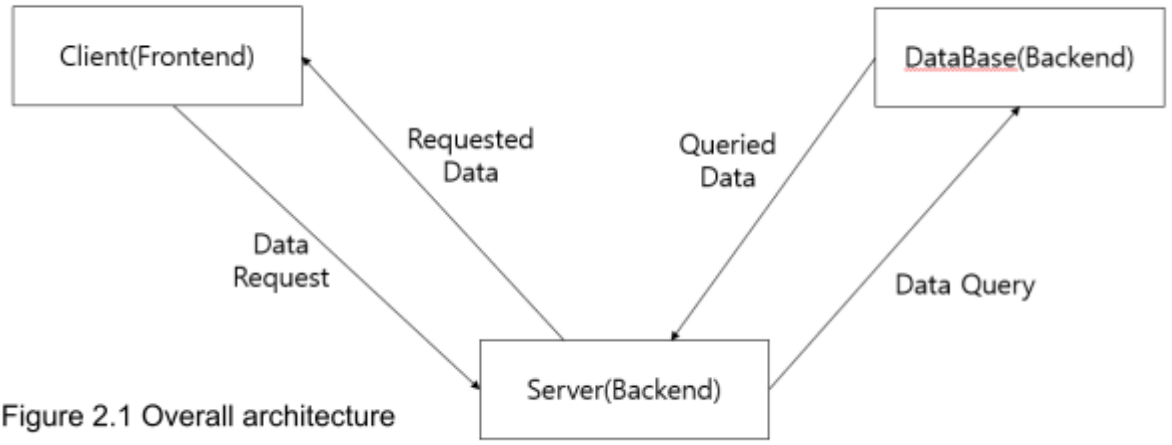

Figure 2.1 Overall architecture

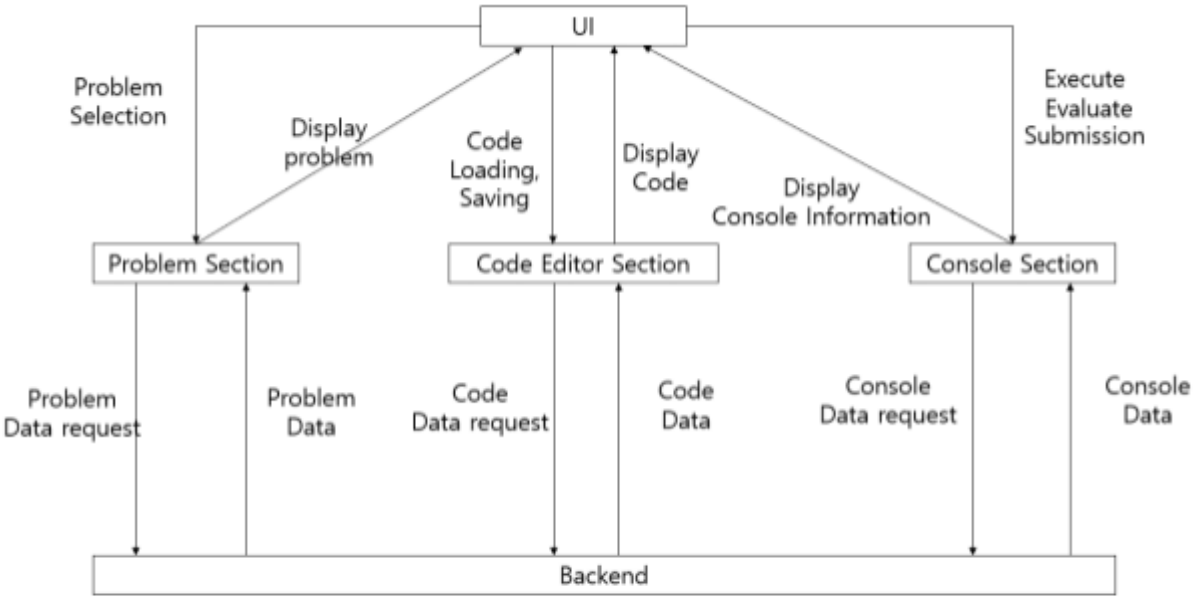

Figure 2.2 Frontend architecture

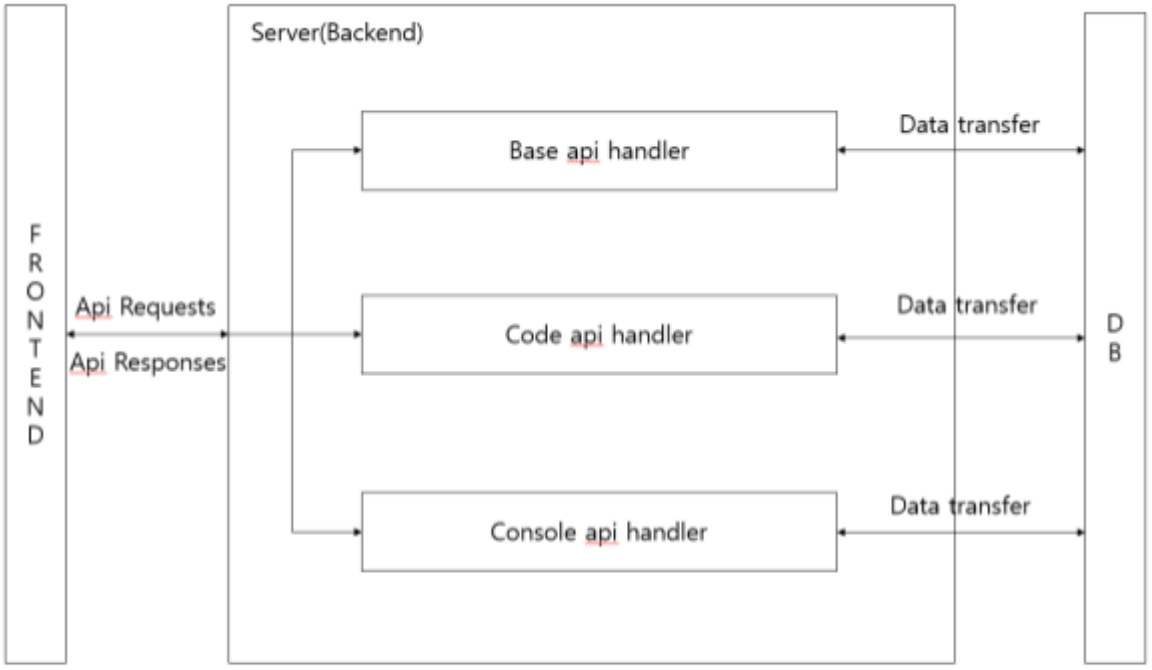

Figure 2.3 Backend architecture

2.2 Database Structure

해당 절에서는 시스템 데이터 구조와 데이터 구조를 데이터베이스에 표시하는 방법을 설명한다. ER-diagram을 통해 엔티티와 그 관계를 식별한다. 이후 SQL DDL를 생성한다.

2.2.1 ER Diagram

EER-diagram 은 모델의 테이블 간의 관계를 시각적으로 표현한다. table을 고유하게 식별하는 primary key는 끝이 빨간 열쇠 모양으로 속성 좌측에 표시한다. 다른 table에서 참조하는 foreign key는 일반 열쇠 모양으로 속성 좌측에 표시한다. 속성 우측에는 변수 Type과 빈칸 허용 여부인 Not Null이 위치한다. NN이 적혀 있는 경우 빈칸이 허용되지 않기 때문에 필수적으로 입력되어야한다. 다른 Table과의 관계는 선으로 나타낼 수 있는데 테이블이 다른 테이블과 여러 관계가 있는 경우 이를 나타내기 위해 닻 모양이 사용된다. 테이블가 다른 테이블과 하나의 관계만 있는 경우, 십자가 모양을 사용하여 이를 나타낸다.

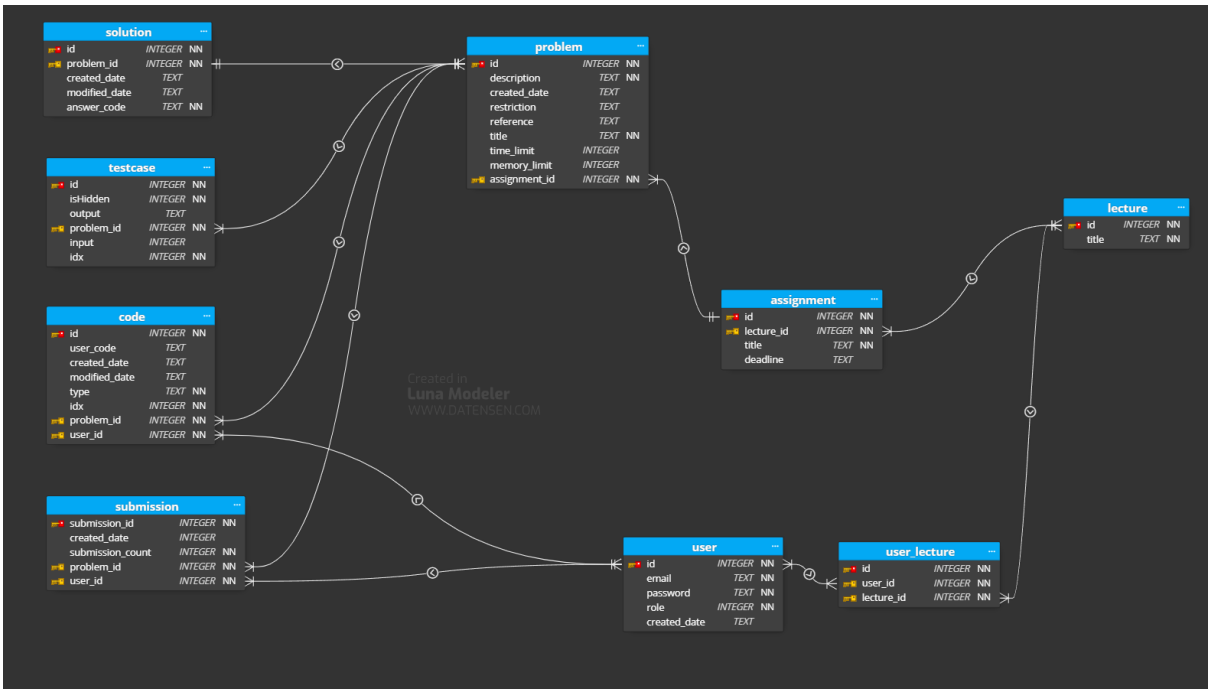

Figure 2.4 ER Diagram

2.2.2 Tables

1. Lecture

Lecture Table은 강의를 나타낸다. id와 title로 구성되어 있고 id를 primary key로 가진다. assignment와는 1:N, user와는 N:M의 관계를 가진다.

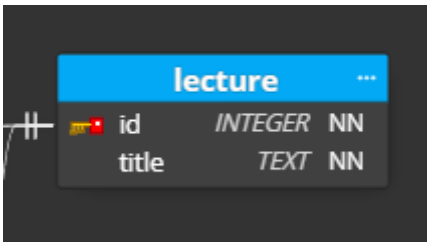

Figure 2.5 lecture table

2. Assignment

assignment Table은 강의의 과제에 해당하는 Table이다 id, lecture_id, title, deadline의 정보를 가지고 id를 primary key로 가진다. lecture_id는 foreign key로서 lecture table의 id값을 참조한다. lecture와 N:1, problem과 1:N 관계를 가진다.

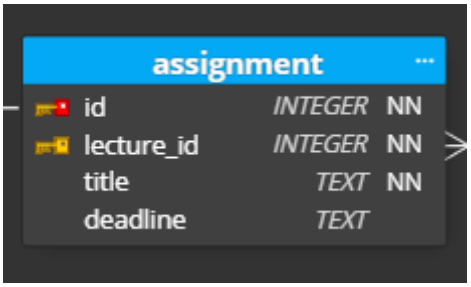

Figure 2.6 assignment table

3. Problem

problem table은 assignment에 배정된 문제 table이다. id를 primary key로 가지며 문제 설명과 제약사항에 해당하는 description, restriction을 정보로 가지고 created_data, referencem title, time_limit으로 구성되어있다. assignment_id는 assignment table의 id를 참조한다. assignment와 N:1 관계를 맺는다. solution과는 1:1, test case와는 1:N 관계를 가진다. code table과 submission table과는 N:M 관계를 가진다.

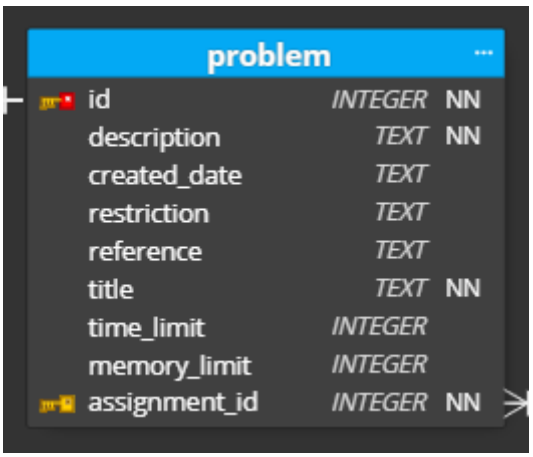

Figure 2.7 problem table

4. Solution, testcase

solution과 test case table은 problem의 정답 코드와 test case 정보를 가진 table이다. 두 table 모두 id에 의해 구분 되며 problem table의 id를 참조한다. problem table과는 N:1 관계를 가진다.

solution table은 created_date, modified_date, answer_code로 이루어져있다. test case는 input, output, idx(test case의 index)와 비공개 test case인지에 대한 isHidden으로 이루어져있다.

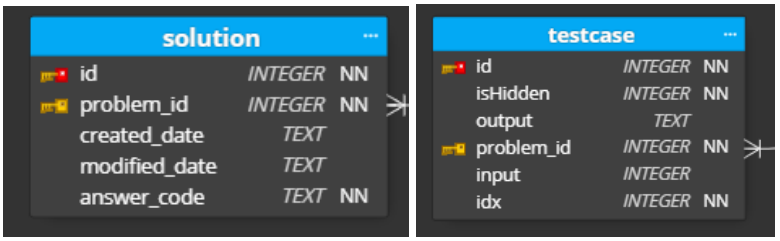

Figure 2.8 solution, test case table

5. Code

code table은 학생이 problem 풀이 과정에서 저장하거나, 풀이 후 제출한 코드 정보를 가지는 table이다. 따라서 problem, user table과 N:M 관계를 가진다. user_code, created_date, modified_date 정보를 가지고, 이 code가 단순 저장인지 제출용인지에 대한 정보인 type과 몇 번째 저장본인지에 대한 index 정보인 idx 또한 table의 속성으로 가진다.

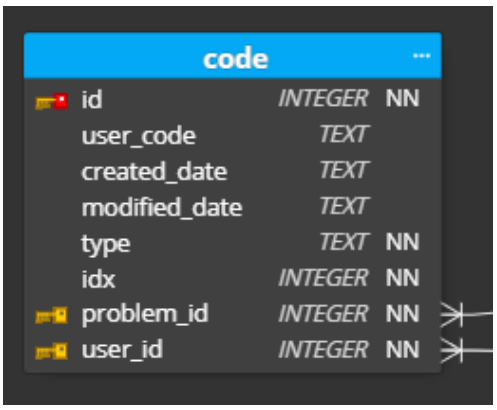

Figure 2.6 code table

6. Submission

submission table은 학생이 problem 풀이 후 제출에 관한 정보를 가진다. problem당 최대 제출 횟수가 3회이기 때문에 제출 횟수를 세기 위해 만들어진 table이다. problem, user table과는 N:M 관계를 가지며 학생이 코드를 제출할 경우 제출한 문제에 해당하는 database가 새로 생성된다.

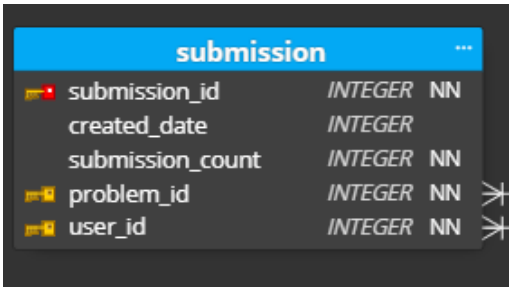

Figure 2.7 submission table

7. User

User table은 system을 사용하는 사용자 정보를 가진 table이다. id로 구분 되며, email, password, role, created_date 속성을 가진다. 이중 role은 교수자, 학생을 구분하는 속성이다. user가 수강하는 lecture에 대한 정보를 가진 user_lecture table과 N:M 관계를 가지고, submission, code table과도 N:M 관계를 가진다.

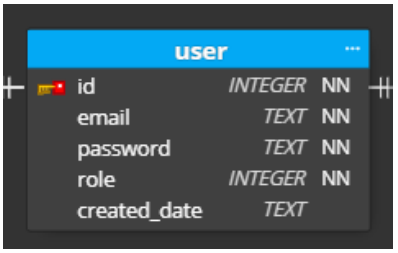

Figure 2.8 user table

8. User_lecture

user가 수강하는 lecture에 대한 정보를 가진 user_lecture table이다. user, lecture table과 N:M 관계를 가진다.

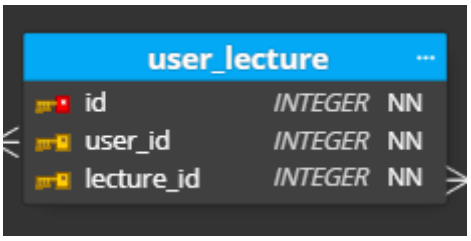

Figure 2.9 user_lecture table

2.2.3 SQL DDL

1. Lecture

```
CREATE TABLE lecture(id INTEGER NOT NULL, title TEXT, PRIMARY KEY(id));
```

2. Assignment

```
CREATE TABLE assignment(
  id INTEGER NOT NULL,
  lecture_id INTEGER NOT NULL,
  title TEXT,
  deadline TEXT,
  PRIMARY KEY(id),
  CONSTRAINT lecture_assignment FOREIGN KEY (lecture_id) REFERENCES lecture (id)
);
```

3. Problem

```
CREATE TABLE problem(
  id INTEGER NOT NULL,
  description TEXT,
  created_date TEXT,
  restriction TEXT,
  reference TEXT,
  title TEXT,
  time_limit INTEGER,
  memory_limit INTEGER,
  lecture_id INTEGER NOT NULL,
  assignment_id INTEGER NOT NULL,
  PRIMARY KEY(id),
  CONSTRAINT assignment_problem
    FOREIGN KEY (assignment_id) REFERENCES assignment (id),
  CONSTRAINT lecture_problem FOREIGN KEY (lecture_id) REFERENCES lecture (id)
);
```

4. Solution

```
CREATE TABLE solution(
  id INTEGER NOT NULL,
  problem_id INTEGER NOT NULL,
  created_date TEXT,
  modified_date TEXT,
  answer_code TEXT NOT NULL,
  PRIMARY KEY(id),
  CONSTRAINT problem_solution FOREIGN KEY (problem_id) REFERENCES problem (id)
);
```

5. Testcase

```
CREATE TABLE testcase(
  id INTEGER NOT NULL,
  isHidden INTEGER NOT NULL,
  output TEXT,
  problem_id INTEGER NOT NULL,
  input INTEGER,
  idx INTEGER NOT NULL,
  PRIMARY KEY(id),
  CONSTRAINT problem_testcase FOREIGN KEY (problem_id) REFERENCES problem (id)
);
```

6. Code

```
CREATE TABLE code(
  id INTEGER NOT NULL,
  user_code TEXT,
  created_date TEXT,
  modified_date TEXT,
  type TEXT NOT NULL,
  idx INTEGER NOT NULL,
  problem_id INTEGER NOT NULL,
  user_id INTEGER NOT NULL,
  PRIMARY KEY(id),
  CONSTRAINT problem_code FOREIGN KEY (problem_id) REFERENCES problem (id),
  CONSTRAINT user_code FOREIGN KEY (user_id) REFERENCES user (id)
);
```

7. Submission

```
CREATE TABLE submission(
  submission_id INTEGER NOT NULL,
  created_date INTEGER,
  submission_count INTEGER NOT NULL,
  problem_id INTEGER NOT NULL,
  user_id INTEGER NOT NULL,
  PRIMARY KEY(submission_id),
  CONSTRAINT problem_submission FOREIGN KEY (problem_id) REFERENCES problem (id),
  CONSTRAINT user_submission FOREIGN KEY (user_id) REFERENCES user (id)
);
```

8. User

```
CREATE TABLE user(
  id INTEGER NOT NULL,
  email TEXT,
  password TEXT,
  role INTEGER,
  created_date TEXT,
  PRIMARY KEY(id)
);
```

9. User_lecture

```
CREATE TABLE user_lecture(
  id INTEGER NOT NULL,
  user_id INTEGER NOT NULL,
  lecture_id INTEGER NOT NULL,
  PRIMARY KEY(id),
  CONSTRAINT user_user_lecture FOREIGN KEY (user_id) REFERENCES user (id),
  CONSTRAINT lecture_user_lecture FOREIGN KEY (lecture_id) REFERENCES lecture (id)
);
```

2.3 End-To-End API

해당 절에서는 프론트엔드와 백엔드에서 엔드투엔드(End-To-End) 통신 구조와 구체적인 API들을 설명한다. 먼저 Overall Structure를 통해 엔드투엔드 통신의 구조를 설명한다. 이후 각 API에 대하여 자세히 설명한다.

2.3.1 Overall Structure

프론트엔드에서는 React를 프레임워크로 사용하였고, 백엔드에서는 Django를 프레임워크로 사용하였다. 데이터베이스 프레임은 SQLite를 사용하였다. 프론트에서는 백엔드와의 통신 API로 axios API를 활용하였고, 백엔드에서는 프론트엔드와 데이터베이스와의 통신 API로 REST API를 활용하였다.

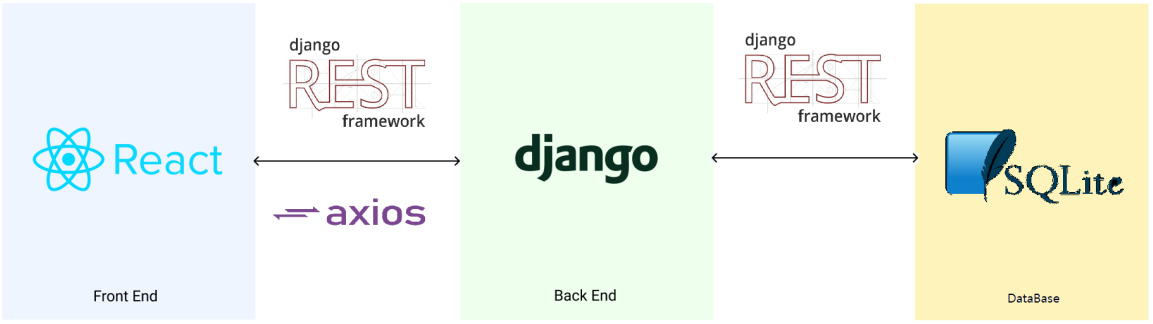

Figure 2.10 End-To-End Overall Structure

2.3.2 APIs

- **getLecture**

```
const getLecture = async () => {
  return await instance.post("/study/");
};
```

  - **method**: POST
  - **URL patterns**: /study/
  - **Parameter**: None
  - **Request Body**: None
  - **Response body**: {[lecture_id: number, title: string]}
  - 모든 강의를 가져오는 요청

- **getAssignment**

```
const getAssignment = async (lecture_id) => {
  return await instance.post("/study/lecture/", {
    lecture_id,
  });
};
```

  - **method**: POST
  - **URL patterns**: /study/lecture/
  - **Parameter**: lecture_id: number(강의 ID번호)
  - **Request** Body: {lecture_id}
  - **Response body**: {[assignment_id: number, title: string, deadline: date, lecture: number]}
  - 특정 강의 번호를 가지고 그 강의에 속한 과제를 모두 가져오는 요청

- **getProblems**

```
const getProblems = async (lecture_id, assignment_id) => {
  return await instance.post("/study/assignment/", {
    lecture_id,
    assignment_id,
  });
};
```

  - **method**: POST
  - **URL patterns**: /study/assignment/
  - **Parameter**: lecture_id: number(강의 ID번호), assignment_id(과제 ID번호)
  - **Request Body**: {lecture_id, assignment_id}
  - **Response body**: {[problem_id: number, title: string, description: string, restriction: string, memorylimit: number, timelimit: number, reference: string, lecture: number, assignment: number]}
  - 특정 강의 번호와 그 강의에 속한 과제 번호를 가지고 과제에 속한 모든 문제를 가져오는 요청

- **getProblem**

```
const getProblems = async (lecture_id, assignment_id) => {
  return await instance.post("/study/assignment/", {
    lecture_id,
    assignment_id,
  });
};
```

  - **method**: POST
  - **URL patterns**: /study/problem/
  - **Parameter**: lecture_id: number(강의 ID번호), assignment_id(과제 ID번호), **problem_id**: number(문제 ID번호)
  - **Request** Body: {lecture_id, assignment_id, problem_id}

- Response body: {problem_id: number, title: string, description: string, restriction: string, memorylimit: number, timelimit: number, reference: string, lecture: number, assignment: number}
- 특정 강의 번호와 그 강의에 속한 과제 번호, 그 과제에 속한 문제 번호를 가지고 특정 문제 하나의 정보만을 가져오는 요청

- **getProblemDetail**

```
const getProblemDetail = async (user_id, problem_id) => {
  return await instance.get(`/study/${user_id}/${problem_id}`);
};
```

- **method**: GET
- **URL patterns**: /study/${user_id}/${problem_id}
- **Parameter**: user_id: number(사용자 ID번호), problem_id(문제 ID번호)
- **Request Body**: None
- **Response body**: {

  [{problem_id: number, title: string, description: string, restriction: string, memorylimit: number, timelimit: number, reference: string, lecture: number, assignment: number}],

  [{testcase_id: number, idx: number, isHidden: boolean, input: string, output: string, problem: number}],

  [{code_id: number, created_date: string modified_date: string, user_code: string, problem: number, user: number}]

  }

- 바로 위의 함수와 기능은 동일. 배열 3개로 리턴하는데 첫 번째는 해당 문제에 대한 정보 하나를 가진 길이 1의 배열, 두 번째와 세 번째는 테스트케이스와 작성 코드 배열.

- **getRecentProblem**

```
const getRecentProblem = async () => {
  return await instance.get(`/study/recent`);
};
```

  - **method**: GET
  - **URL patterns**: /study/recent
  - **Parameter**: None
  - **Request Body**: None
  - **Response body**: { ***getProblemDetail***과 동일

    [{problem_id: number, title: string, description: string, restriction: string, memorylimit: number, timelimit: number, reference: string, lecture: number, assignment: number}],

    [{testcase_id: number, idx: number, isHidden: boolean, input: string, output: string, problem: number}],

    [{code_id: number, created_date: string modified_date: string, user_code: string, problem: number, user: number}]

    }

  - 사이트를 처음 열었을 때 이전에 풀던 문제 정보를 가져오는 요청

- **saveCodeInDB**

```
const saveCodeInDB = async (problem, user, user_code) => {
  return await instance.post("/study/save/", {
    problem,
    user,
    user_code,
  });
};
```

  - **method**: POST
  - **URL patterns**: /study/save/
  - **Parameter**:
    - problem: number(문제 ID번호), user: number(사용자 ID번호)
    - user_code: string(사용자가 작성한 코드를 스트링화)
    - code_idx: number(1,2,3코드 슬롯)
  - **Request Body**: {problem, user, user_code}
  - **Response body**: {code_id, created_date, modified_date, users_code, code_idx, problem, user}
  - 데이터베이스에 사용자가 푼 코드를 저장하는 요청

- **executeCode**

```
const executeCode = async (user_code) => {
  return await instance.post(`/study/run`, {
    user_code,
  });
};
```

  - **method**: POST
  - **URL patterns**: /study/run/
  - **Parameter**: None
  - **Request Body**: {user_code: string}
  - **Response body**: {line_number: number, message: string}

- 실행하고자 하는 코드를 문자열로 전달한다. 실행에 실패할 경우 line_number를 통해 사용자 코드에서 에러가 발생한 위치를 알 수 있으며, message를 통해 Traceback 정보를 받을 수 있다. 실행에 성공한 경우 line_number는 null값으로 반환된다.

2.4 User Interface

2.4.1 Main Page

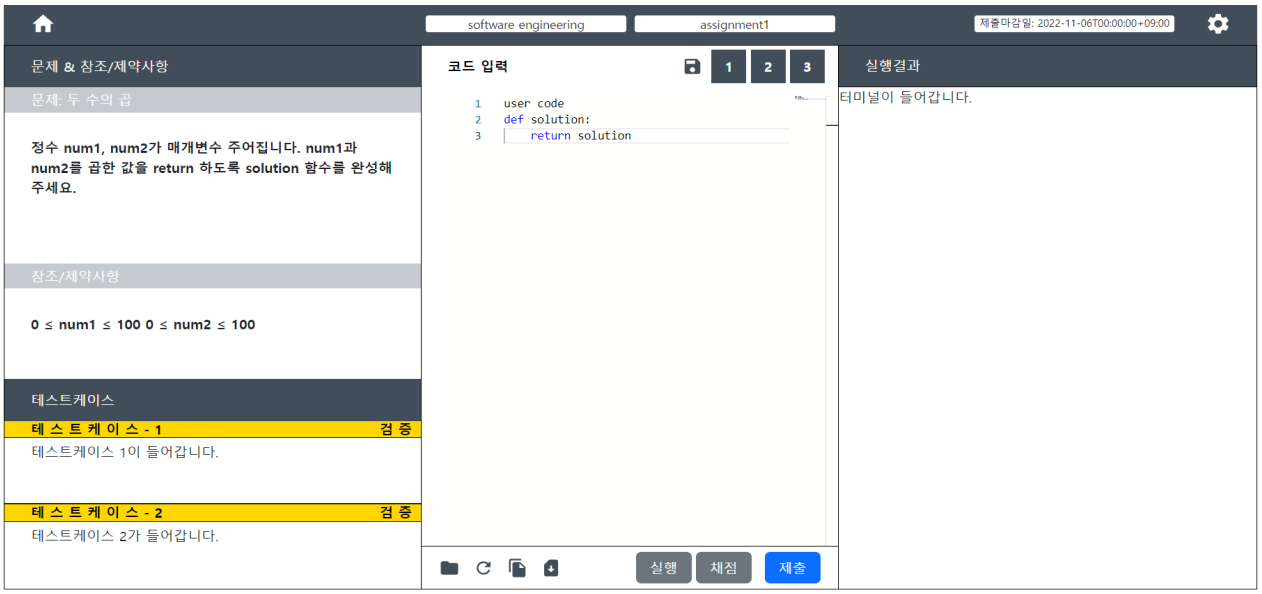

Figure 2.11 Main Page

사용자는 메인 페이지에 있는 1)헤드 섹션, 2)문제 설명 섹션, 3)테스트 케이스 섹션, 4)코드 에디터 섹션, 5)각종 기능 버튼 섹션과 6)결과 섹션 각각의 섹션을 클릭하여 원하는 작업을 할 수 있다.

2.4.2 Head Section

Figure 2.12 Head Section

헤드 섹션은 홈 버튼, 강의명, 과제명, 마감일, 설정으로 구성된다.

1) 홈 버튼을 클릭하면 맨 처음 화면을 돌아간다.

2) 강의명에는 선택한 강의의 제목이 출력된다.

3) 과제명에는 과제명이 출력된다. 이때 양쪽에 화살표 아이콘이 존재하여 이를 클릭하면 다른 과제로 이동한다.

4) 마감일에는 해당하는 과제의 마감일이 출력된다.

5) 헤드 섹션의 가장 우측에 설정 아이콘이 존재하며 이를 클릭하면 배경 색 변경, 코드 에디터 변경 등의 기능을 지원한다.

2.4.3 Problem Section

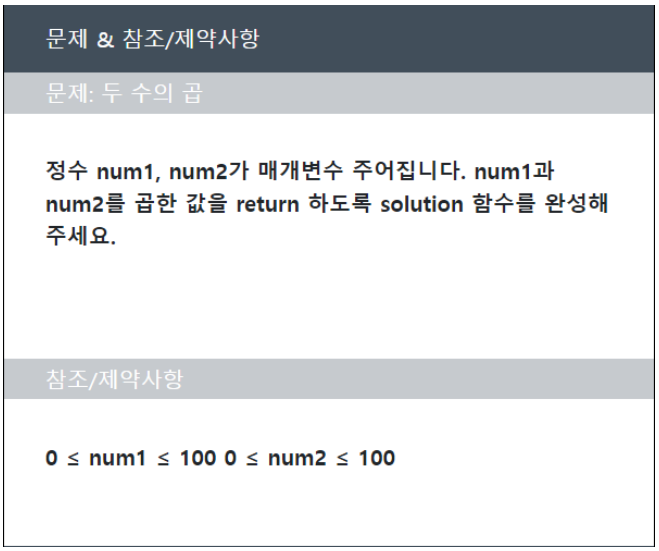

Figure 2.12 Problem Section

문제 설명 섹션에는 문제와, 참조/제약사항으로 구성된다.

1) 문제 설명에는 해당 문제의 내용이 출력된다.

2) 참조/제약사항에는 문제에서 참조해야 할 내용과 문제의 제약사항이 출력된다.

2.4.4 Testcase Section

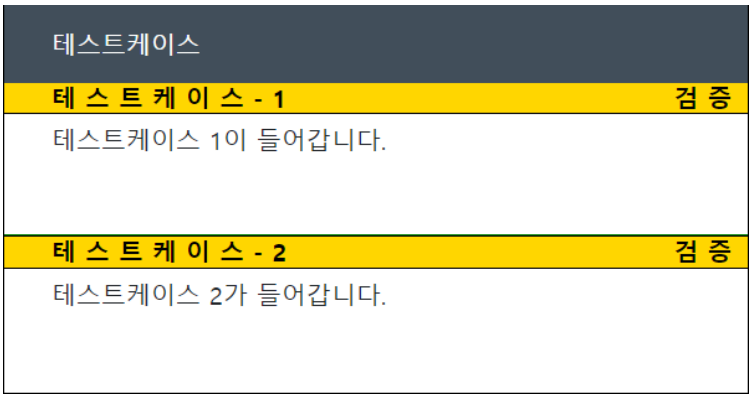

Figure 2.13 Testcase Section

테스트 케이스 섹션에는 기본적으로 2개의 (테스트케이스, 검증 영역)으로 구분된다.

1) 테스트 케이스 섹션에는 해당 문제에 대한 테스트 케이스가 출력된다. 이때 공개 테스트 케이스만 보이도록 구분하며 비공개 테스트 케이스는 출력되지 아니한다.

2) 검증 영역에는 검증 아이콘(혹은 링크 형태의)을 두고 이것을 클릭하면 사용자가 입력한 코드와 미리 정해져 있는 테스트 케이스 입력에 따라 테스트의 pass, fail을 결정하여 이를 출력, 만약 fail일 경우 사용자의 코드에서 나온 output결과 또한 함께 출력

2.4.5 Code Editor Section

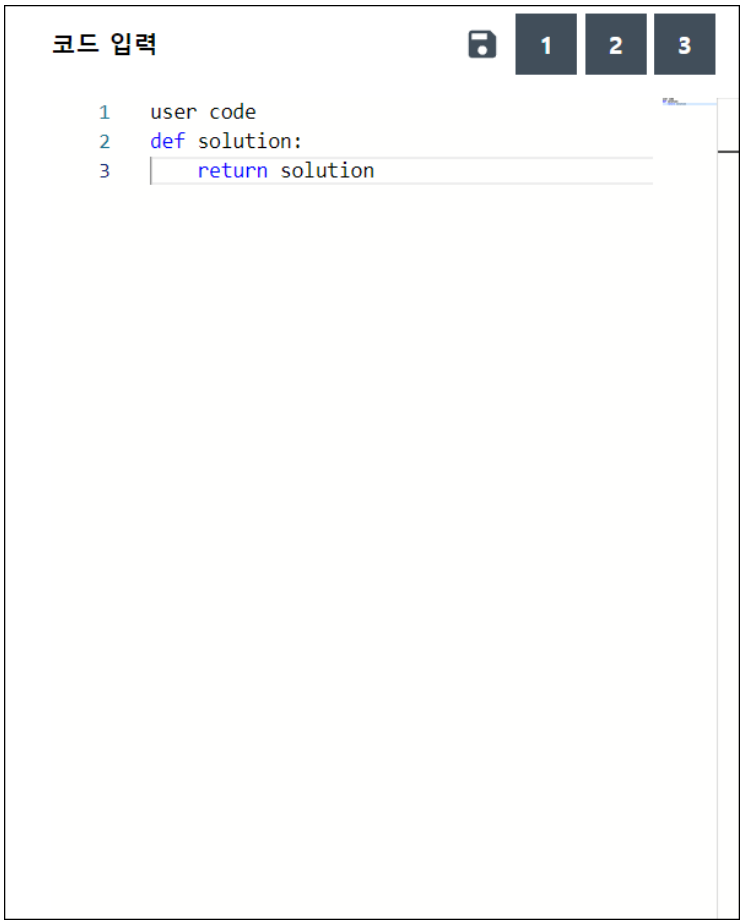

Figure 2.14 Code Editor Section

코드 에디터 섹션에는 코드 입력 에디터와, 코드 중간 저장 컴포넌트로 구성한다.

1) 코드 입력 에디터의 경우 모나코 라이브러리를 활용하며, 사용자가 코드를 작성할 수 있다.

2) 코드 중간 저장 기능의 경우 코드 입력 에디터 우측 상단에 저장 아이콘을 배치하여 이를 클릭하면 최대 세 번까지 작성한 코드를 중간에 저장한다.

2.3.6 Multi Function Section

Figure 2.15 Multi Function Section

각종 기능 버튼 섹션의 경우 파일 불러오기 버튼, 코드 초기화 버튼, 코드 복사 버튼, 코드 다운로드 버튼과 코드 실행, 코드 채점, 코드 제출 버튼 영역으로 나뉜다.

1) 가장 좌측에 파일 불러오기 버튼을 배치하여 이를 클릭하면 기존의 소스코드 파일 혹은 텍스트 파일을 불러와서 에디터에 보여준다.

2) 파일 불러오기 버튼 오른쪽에 코드 초기화 버튼을 배치하여 이를 클릭하면 문제 초기 스켈레톤 코드로 초기화한다. 사용자가 저장하지 않은 내용은 사라진다.

3) 코드 복사 버튼을 코드 초기화 버튼 오른쪽에 배치하여 이를 클릭하면 작성중인 코드를 텍스트 형태로 클립보드에 복사한다.

4) 코드 다운로드 버튼을 코드 복사 버튼 오른쪽에 배치하여 이를 클릭하면 코드를 소스코드 파일로 저장할 수 있다.

5) 코드 실행 버튼을 코드 채점 버튼 왼쪽에 배치하여 이를 클릭하면 코드를 실행하고 해당 코드에 대한 출력을 오른쪽 결과 섹션에서 보여주게 된다.

6) 코드 채점 버튼을 코드 제출 버튼 왼쪽에 배치하여 사용자의 코드로 테스트케이스를 돌렸을 때와 정답 코드로 테스트케이스를 돌렸을 때의 결과를 비교하여 이를 채점한다. 또한 사용자의 코드에서 에러가 발생할 경우 이를 우측의 결과 섹션에서 보여주게 된다. 코드 채점에 대한 패널티는 없기 때문에 사용자는 한 문제에 대해 여러 번 코드를 채점할 수 있도록 한다.

7) 코드 제출 버튼을 가장 오른쪽에 배치하여 이를 클릭하면 사용자가 작성한 코드를 제출하고 이를 평가한다. 코드 채점과는 달리 코드 제출은 한 문제 당 최대 세 번만

가능하며 과제의 경우 사용자가 작성한 해당 소스파일을 과제의 제출물로써 간주하여 제출한다.

2.3.7 Result Section

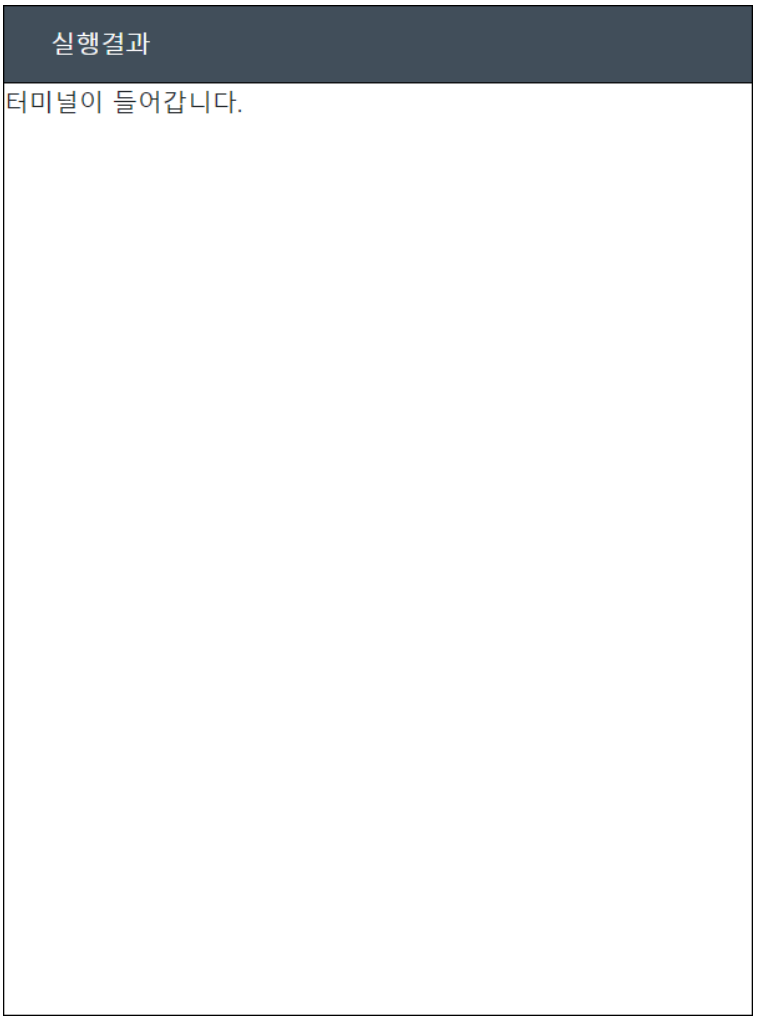

Figure 2.16 Result Section

결과 섹션의 경우 실행 결과, 채점 결과, 과거 제출 코드, Code Diff, 제출 결과, 코드 설명, 관련 자료를 출력한다.

1) 실행 결과의 경우 사용자가 입력한 코드를 돌렸을 때의 출력 값을 제공한다. 이때 에러가 나지 않을 경우에만 정상적으로 출력한다.

2) 실행 결과의 경우 중 사용자가 입력한 코드를 돌렸을 때 에러가 날 경우, 정상 출력값을 제공하지 아니하고 에러 메시지와 에러 메시지가 발생한 위치를 출력한다. 또한 이 경우 코드 에디터 섹션의 코드 에디터 영역에 에러가 난 라인을 빨간색 하이라이트하여 사용자가 즉각 알아볼 수 있도록 한다. 또한 빨간색으로 하이라이트 된 부분을 클릭하면 그 아래에 초록색으로 에러 메시지를 하이라이트하여 보여준다.

3) 채점 결과의 경우 사용자가 작성한 코드로 돌린 테스트케이스에서 각각의 배점을 고려하여 채점한 사용자의 최종 점수를 출력한다. 또한 이에 따른 pass/fail여부를 기준에 맞추어 결정한 뒤 출력한다. 채점 시에는 사용자에게 보여주지 않은 비공개 테스트케이스까지 모두 테스트하며, 공개 테스트테이스에 대하여 fail했을 경우에는 이에 대한 정보를 제공한다. 반대로 비공개 테스트케이스에 대하여 fail했을 경우에는 정보를 제공하지 않으며, 점수에만 반영됨을 원칙으로 한다.

4) 과거 제출 코드의 경우 최대 세 번의 다시 풀기가 가능하며 이때 다시 푼 세 번의 코드를 불러올 수 있게 한다.

5) Code Diff의 경우 정답 코드와의 비교를 통해 Diff를 확인하고 이를 표시해 준다. 또한 코드 에디터 섹션의 코드 입력 영역을 좌로 확대하여 표시하였을 경우에는 그 차이를 빨간색/초록색으로 하이라이트하여 표시한다.

6) 제출 결과의 경우 코드의 채점에 대한 결과를 출력한다. 이 과정에서 평가 지표로 표절률, 기능성, 효율성, 가독성 등이 포함되며 각각은 특정 라이브러리를 통해 평가한다.

7) 코드 설명의 경우 OpenAI Codex API를 활용하여, 사용자가 작성한 코드에 대한 설명을 출력한다.

8) 관련 자료의 경우 해당 문제에 연관된 문제, 자료 영상, 학습 자료등을 선별하여 사용자에게 보여준다.

3. Components

3.1 Front-end

여기에서는 프론트 엔드를 구성하는 각 요소들의 구조, 속성 및 기능에 관해 설명하고, 이들간의 관계를 설명한다.

3.1.1 Main

모든 컴포넌트를 포함하는 최상위 컴포넌트이다. 문제와 테스트케이스 컴포넌트(Left), 코드 에디터와 다중 기능 컴포넌트(Center), 결과 컴포넌트(Right)를 하위 컴포넌트로 가지며 대부분의 속성을 여기에서 관리한다.

- **State**
    - **lecture** : 모든 lecture를 저장하는 변수
    - **selectedLecture** : 현재 선택된 lecture의 인덱스를 저장하는 변수
    - **assignment** : 현재 선택된 lecture의 모든 assignment를 저장하는 변수
    - **selectedAssignment** : 현재 선택된 assignment의 인덱스를 저장하는 변수
    - **problem** : 현재 선택된 assignment의 모든 problem을 저장하는 변수
    - **selectedProblem** : 현재 선택된 problem의 인덱스를 저장하는 변수
    - **code** : 현재 코드 에디터에 있는 code를 저장하는 변수
    - **rightSection** : 현재 결과 컴포넌트에서 출력하는 내용이 실행결과인지, 채점결과인지, 제출결과인지 구분하는 변수

3.1.2 Header

사이트 최상단에 위치한 헤더 컴포넌트이다. 홈 아이콘, 타이틀(강의 타이틀, 과제 타이틀), 마감일 컴포넌트를 하위 컴포넌트로서 갖는다.

- **State**
    - **lectureTitle** : 현재 선택된 lecture의 제목을 저장하는 변수
    - **assignmentTitle** : 현재 선택된 assignment의 제목을 저장하는 변수
    - **deadline** : 현재 선택된 assignment의 마감일을 저장하는 변수

3.1.3 Problem&Testcase(Left)

사이트 왼쪽에 위치한 문제, 테스트케이스 컴포넌트이다. 문제, 테스트케이스 컴포넌트를 하위 컴포넌트로서 갖는다.

- **State**
    - **ProblemTitle** : 현재 선택된 problem의 제목을 저장하는 변수
    - **ProblemContent** : 현재 선택된 problem의 내용을 저장하는 변수
    - **ProblemConstraint** : 현재 선택된 problem의 제약사항을 저장하는 변수

3.1.4 CodeEditor&MultiFunction(Center)

사이트 중앙에 위치한 코드에디터, 다중 기능 컴포넌트이다. 코드에디터, 다중기능(불러오기 버튼, 리셋 버튼, 복사 버튼, 다운로드 버튼, 실행 버튼, 채점 버튼, 제출 버튼) 컴포넌트를 하위 컴포넌트로서 갖는다.

3.1.5 Terminal(Right)

사이트 우측에 위치한 결과 컴포넌트이다. 실행결과, 채점결과, 제출결과 컴포넌트를 하위 컴포넌트로서 갖는다.

3.2 Back-end

3.2.1 Lecture

강의 내용에 관한 Class이다. 강의에서 출제되는 Assignment Class를 Attribute로 가진다.

- **Attributes**
    - **lecture_id** : 해당 lecture의 id 및 강의 번호 이다.
    - **lecture_title** : 해당 lecture의 강의 제목이다.
    - **assignment** : 해당 강의에서 출제되는 과제 목록이다.
- **Methods**
    - **getAssignment(lecture_id)** : 해당 lecture에서 출제된 모든 assignment의 정보를 출력한다.
- **Sequence Diagram**

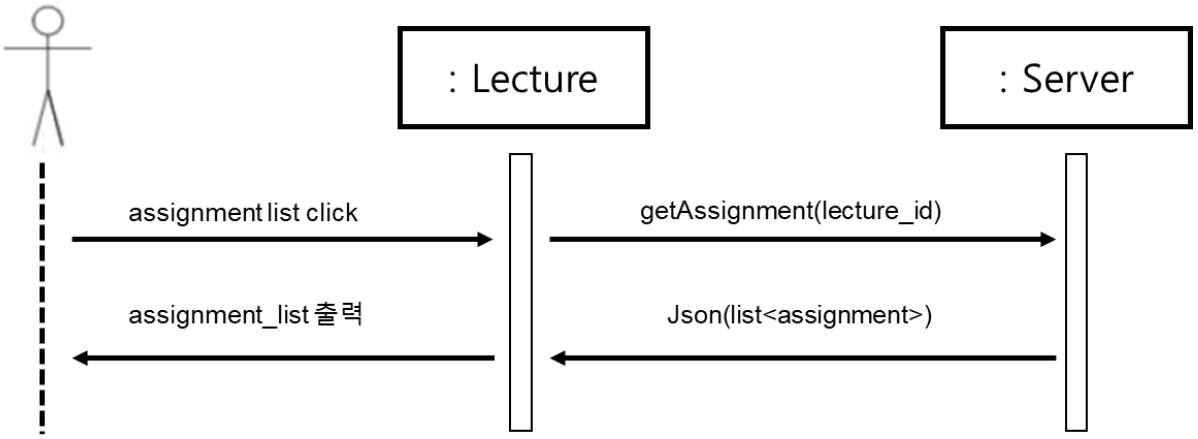

Figure 3.1 Sequence Diagram - lecture

3.2.2 Assignment

강의에서 출제된 과제에 관한 Class이다. 해당 Assignment하위에 학생들이 풀어야하는 문제인 Problem Class를 Attribute로 가진다.

- **Attribute**
  - **assignment_id** : 해당 assignment의 id 및 과제 번호 이다.
  - **assignment_title** : 해당 assignment의 과제 제목이다.
  - **problem**: 해당 과제에서 학생들이 풀어야하는 문제 목록이다.
  - **deadline**: 해당 과제의 마감 기한이다.
- **Methods**
  - **getProblems(assignment_id)** : 해당 assignment에서 학생들이 풀어야하는 모든 문제 정보를 가져온다.
- **Sequence Diagram**

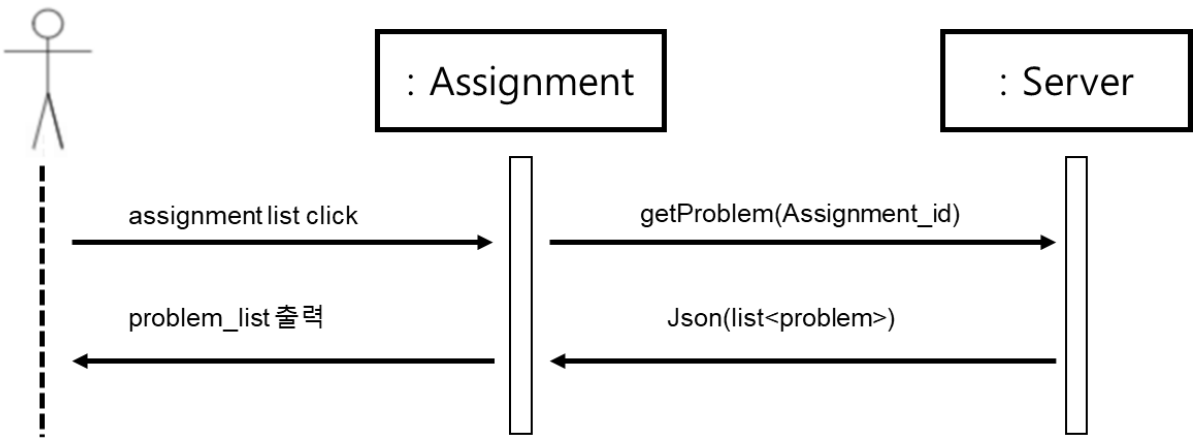

Figure 3.2 Sequence Diagram - assignment

3.2.3 Problem

학생들이 풀어야하는 문제에 대한 정보를 가진 Class이다. 정답 Code에 해당하는 Solution, 채점을 위해 사용하는 Test Case 등의 객체를 Attribute로 가진다.

- **Attribute**
  - **problem_id**: 해당 problem의 id 및 문제 번호 이다.
  - **problem_title**: 해당 problem의 제목이다.
  - **description**: 해당 문제의 설명이다. 학생이 풀어야할 문제의 요구 사항들을 설명한다.
  - **restriction**: 해당 문제를 풀 때 고려할 사항이나, 제한 사항을 명시한다. 입력 값은 범위를 주로 가진다.
  - **time_limit**: 하나의 test case를 실행시키는데 허용되는 최대 시간이다. 단위는 초 단위이다. 이 시간을 넘어서 실행되는 경우 오답 처리한다.
  - **memory_limit**: code의 최대 용량을 의미한다.

- **reference**: 해당 문제를 풀기 위해 필요한 지식을 가진 web page 주소이다. 채점 후 공개하여 추가 학습을 용이하게 한다.
- **test case**: 해당 문제의 test case이다. input과 output의 속성을 가지며, ishidden이라는 속성으로 공개/비공개 test case로 나뉜다.
- **solution**: 해당 문제의 정답 code이다. 채점 후 학생 code와 비교할 때 사용된다.

- **Methods**
  - **runTestCase(testcase_idx)** : problem의 test case중 testcase_idx에 해당하는 test case를 실행하여 실행결과와 정답을 비교하여 정답 여부를 출력한다.

- **Sequence Diagram**

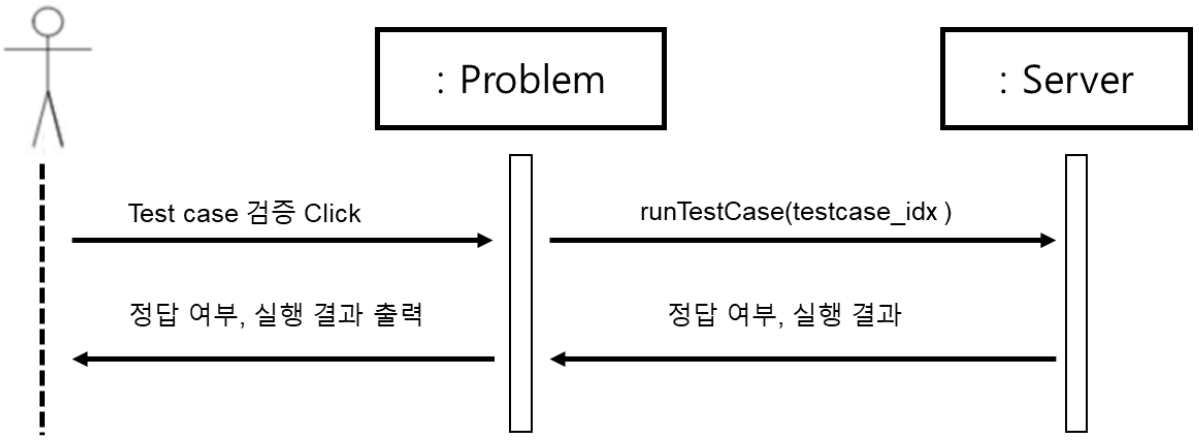

Figure 3.3 Sequence Diagram - problem

3.2.4 Code

학생들이 문제를 풀고 작성한 코드에 대한 정보를 가진 Class이다.

- **Attribute**
    - **code_id**: 해당 코드의 id이다.
    - **user_code**: 해당 코드의 내용이다.
    - **created_date**: 해당 코드가 데이터베이스에 생성된 날짜이다.
    - **modified_date**: 해당 코드가 변경된 날짜를 나타낸다.
    - **type**: 해당 코드가 단순 저장된 상태인지, 제출된 상태인지를 나타낸다.
    - **idx**: 해당 코드가 몇 번째 저장본인지를 나타낸다.
- **Methods**
    - **runCode(user_code)**: Code class의 user_code에 담긴 코드를 실행하여 정상적으로 코드가 실행된 경우 표준 출력을 반환하고, 에러가 발생하여 정상적으로 실행되지 못한 경우 에러에 대한 설명과 에러 발생 라인 정보를 반환한다.
    - **grade_code**(): 문제에 등록된 모든 테스트케이스를 실행하여 각각의 결과를 프론트에 전달한다.

- **submit_code()**: Code class의 user_code에 담긴 코드를 기반으로 코드 제출 시 코드에 대한 평가를 진행한다. 평가 결과로 표절률, 테스트케이스 통과 여부, 코드 효율, 코드 가독성에 대한 지표를 반환한다.
- **save_code**(): 유저가 작성한 코드를 데이터베이스에 추가 또는 수정 한다.
- **download_code**(): 유저가 작성한 코드를 유저의 pc에 파일로 저장한다.

- **Sequence Diagram**

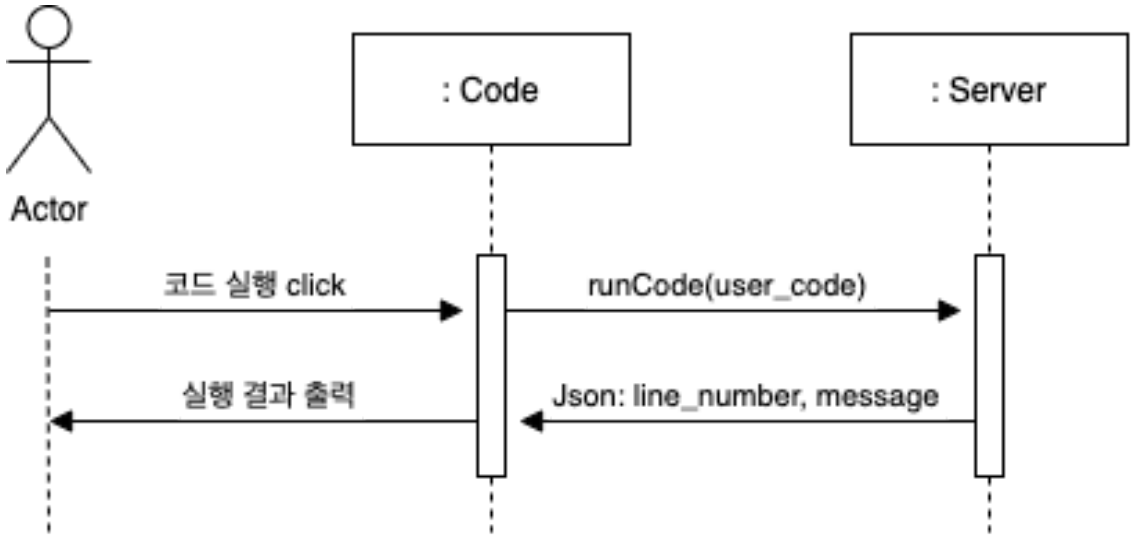

Figure 3.4 Sequence Diagram - runCode()

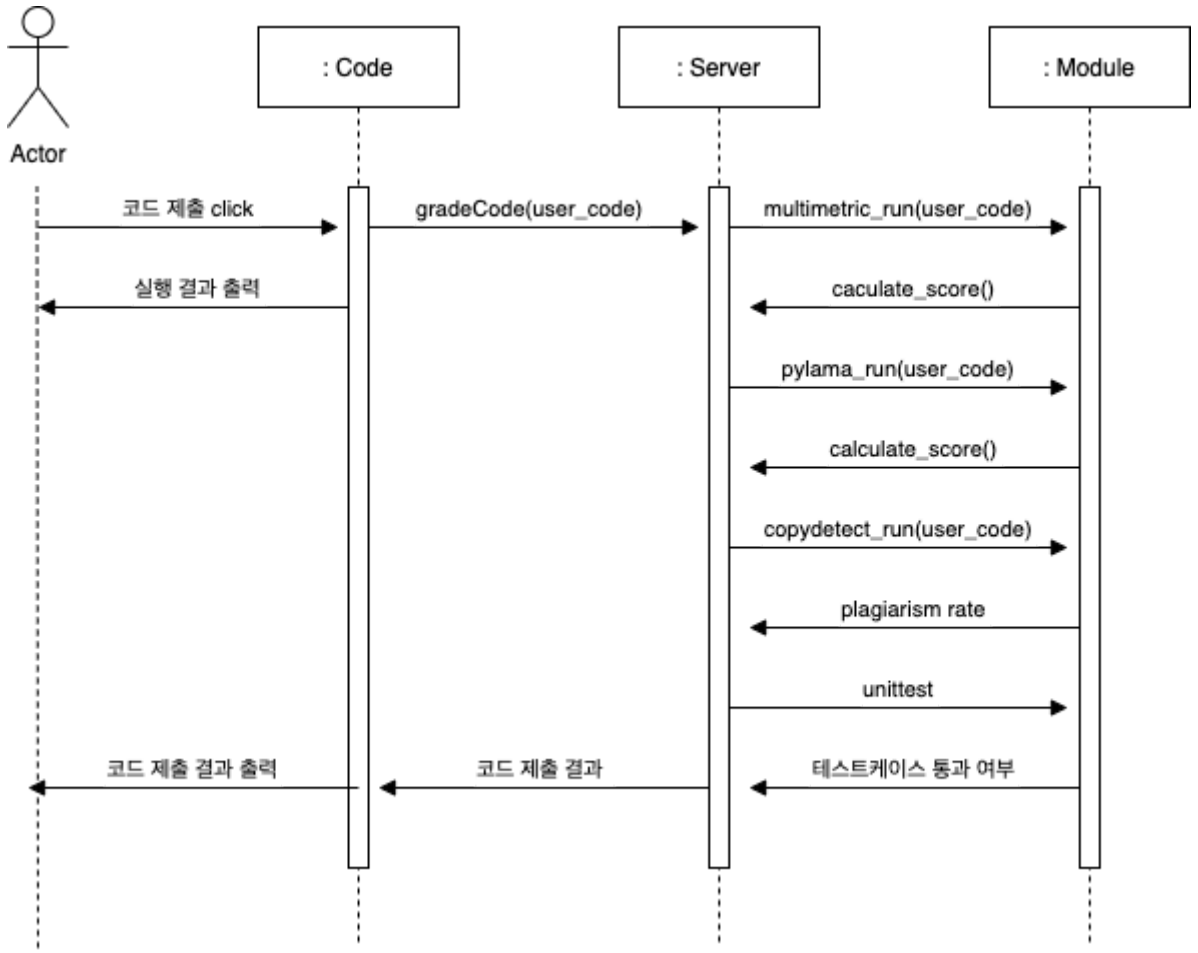

Figure 3.5 Sequence Diagram - submitCode()

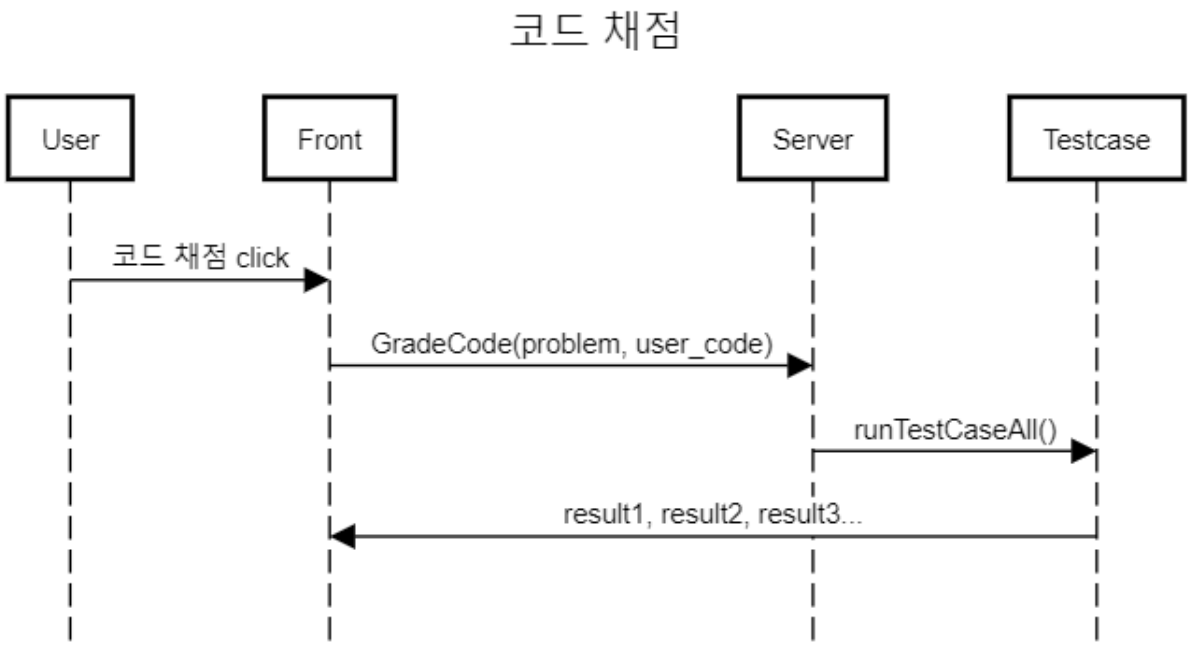

Figure 3.6 Sequence Diagram - submitCode()

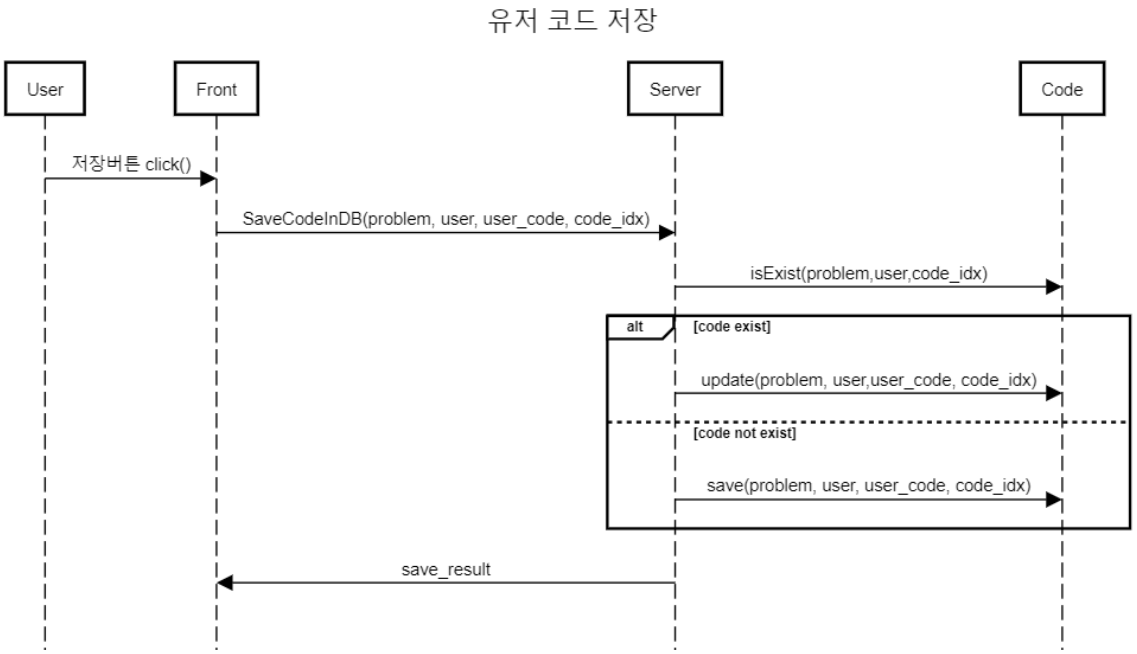

Figure 3.7 Sequence Diagram - saveCode()

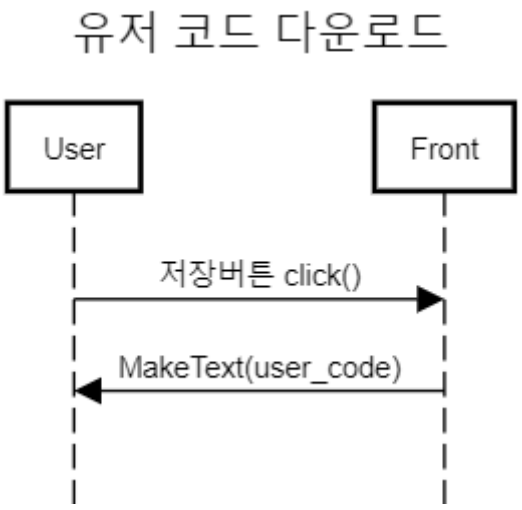

Figure 3.8 Sequence Diagram - downloadCode()

4 References

이 디자인 명세서는 다음과 같은 자료를 참고하여 작성되었다.

- Team 2, 2022 spring, Software Design Document, SKKU
- Team 3, 2022 spring, Software Design Document, SKKU
- https://yuan-lu.weebly.com/uploads/3/9/3/7/39371931/designdoc-2.pdf