GPTeacher

Software Design Specification

소프트웨어공학개론 2 조

김형진, 박혜인, 서민경, 신재욱, 이승민

2024-12-1

CONTENTS

List of Figures

1

Introduction

본 문서가 예상하는 독자, 다루는 내용의 범위와 목표, 그리고 각 챕터에 대한 간략한 내용에 대해 설명한다.

1.1 Readership

본 문서의 독자는 다음과 같다. 본 시스템의 개발자들(team2)이며, 필요에 따라 본 시스템의 사용자를 포함한다. 본 시스템의 개발자는 front-end와 back-end 개발자, 그리고 시스템의 교육 내용 및 기능에 대한 개발자를 포함한다.

1.2 Scope

본 문서에서는 *GPTeacher*의 설계 범위와 디자인을 정의하고 테스팅 및 개발 계획을 세운다. 주요 기능인 힌트 제공과 기본적인 변형 문제 및 관심사와 관련된 변형 문제 제공과 관련된 설계 및 디자인을 포함한다.

1.3 Objective

본 문서의 주요 목적은 본 시스템 *GPTeacher*의 설계 구조와 구현 방법에 대한 설명이다. 전반적인 시스템의 구조에 대해 설명하고, front-end, back-end, 그리고 protocol과 database 각각에서의 측면에서 정의한다.

1.4 Document Structure

1) Preliminary: 본 문서의 독자, 범위, 목표 및 전체적인 문서의 구조에 대해 설명한다.

2) Introduction: system design에 사용된 다양한 다이어그램의 유형과 목적에 대해 설명하고, 프로젝트의 범위와 본 문서 작성을 위해 참고한 자료 등에 대해 설명한다.

3) System Architecture – Overall: 전체적인 시스템의 구성에 대해 context diagram, process model, use case diagram을 통해 설명한다.

4) System Architecture – Frontend: Frontend의 목표와 관련 컴포넌트에 대해 sequence

diagram, data flow diagram을 통해 설명한다.

5) System Architecture – Backend: Backend의 목표와 하위 구성요소에 대해 sequence diagram을 통해 설명한다.

6) Protocol Design: Frontend와 Backend 사이의 통신에 사용되는 protocol과 interface에 대해 설명한다.

7) Database Design: Entity-Relationship diagram 및 다양한 entity, 그리고 relational schema를 통해 데이터베이스 디자인의 목표와 이와 관련된 설명을 제공한다.

8) Testing Plan: Testing policy와 그에 따른 test case 및 pseudocode에 관해 설명한다.

9) Development Plan: 개발 환경 및 도구, 제약사항 및 전제 조건 등에 대하여 설명한다.

2

Overview

2.1. Objectives

System Design에 사용된 Diagram과 Reference, Project Scope를 설명한다.

2.2. Applied Diagrams

2.2.1. Used tools

Diagram을 그리기 위하여 시스템 diagram 제작에 특화된 draw.io를 활용하였다.

2.2.2. Diagrams

2.2.2.1. Use Case Diagram

시스템 사용자, 서버와 같은 ACTOR가 실제로 사용할 수 있는 작업이나 기능에 관하여 서술한 Diagram으로, 해당 Interactions을 한눈에 묘사한다.

2.2.2.2. Sequence Diagram

시스템의 여러 Object들이 시간에 따라 Interaction하는 flow를 보여준다. 이를 통해 시스템의 주요 Prosses가 구분된 시간에 따라 어떤 순서로 진행되는지 묘사한다.

2.2.2.3. Context Diagram

개발하고자 하는 LLM 기반 문제 풀이 시스템과 Interaction할 수 있는 여러 외부 시스템들이 어떤 것들이 있는지 묘사한다.

2.2.2.4. Entity Relationship Diagram

데이터베이스에서 서로 관계되어 있는 Entity들이 어떤 관계를 어떻게 맺고 있는지 묘사하고, 어떤 Attributes를 가지고 있는지 구조화해 표현한다.

2.2.3. Project Scope

GPTeacher는 컴퓨터 공학 비전공자를 대상으로 코딩을 쉽고 재미있게 학습할 수 있도록

돕는 문제 풀이 기반 웹 서비스로, 학습자가 개인화된 학습 및 산출물을 생성할 수 있도록 LLM을 활용한다. 학습자는 LLM이 생성하는 문제 힌트를 통해 문제 풀이에 도움을 받을 수 있고, 역시 LLM이 생성하는 변형 문제 및 관심사 맞춤형 문제를 통하여 자신이 원하는 주제의 원하는 알고리즘에 대한 최종 프로그램을 생성할 수 있도록 돕는다.

2.2.4. References

IEEE Std 1016.1-1993 IEEE Guide to Software Design Descriptions

Draw.io https://app.diagrams.net/

Baekjoon Online Judge https://www.acmicpc.net/

LeetCode https://leetcode.com/

3

System Architecture - Overall

3.1 Objectives

이 챕터에서는 시스템의 전체적인 구조에 대해 설명하며, 이후 문서에서 다루는 프론트엔드, 백엔드, 데이터베이스 설계의 세부 내용을 요약한다. 또한 context diagram, process model, use case diagram을 통해 시각화 한다.

3.2. System Organization

이 시스템은 Client-Server모델을 기반으로 설계되었으며, React를 사용하여 javascript 기반으로 구축된 프론트엔드 애플리케이션이 사용자와의 모든 상호작용을 담당한다. 프론트엔드 애플리케이션과 백엔드 애플리케이션은 JSON 기반의 HTTP 통신을 통해 데이터를 주고받는다. 백엔드 애플리케이션은 python 기반의 Django 프레임워크로 구현되었으며, 프론트엔드로부터 요청을 받아 LLM 및 데이터베이스와 상호작용하고, 다시 프론트엔드에 응답을 전달한다.

데이터베이스는 PostgreSQL을 사용하며, Django를 통해 작업이 이루어진다. 데이터의 백업 및 복구 작업은 PostgreSQL DBMS에 직접 접근하여 수행된다. 백엔드 애플리케이션은 사용자가 요청한 작업을 실행하고 결과를 반환하며, 결과 데이터는 데이터베이스에 저장된다.

3.2.1 Frontend

프론트엔드는 사용자 인터페이스 UI를 제공하며, 사용자가 시스템과 상호작용할 수 있도록 설계되었다. MVC 패턴과 Client-Server 패턴을 기반으로 설계되어 UI와 로직이 분리되어 있으며, 사용자의 입력을 받아 처리하고, 백엔드와 통신하여 데이터를 가져온다.

주요 페이지는 다음과 같다.

- 회원가입/로그인 페이지
- 메인 페이지
- 프로필 관리 페이지
- 학습현황 페이지

• 문제 풀이 페이지
• 제출 페이지

3.2.2 Backend

백엔드는 모듈화 된 구조로 구성되어 기능별로 독립적으로 작동한다. RESTful API를 통해 프론트엔드와 통신하여 프론트엔드의 요청을 처리하고, 데이터베이스에서 필요한 정보를 가져와 데이터 처리를 하며, LLM과 상호작용하며 사용자 맞춤 서비스를 제공한다.
주요 기능은 다음과 같다.
• 계정 관리 시스템: 회원가입, 로그인, 프로필 관리
• 힌트 제공 시스템
• 문제 제공 시스템
• 문제 변형 시스템
• 맞춤형 문제 생성 시스템
• 문제 채점 시스템
• 코드 리뷰 시스템

3.2.3 Database

데이터베이스는 시스템의 핵심 데이터를 저장하고 관리하는 역할을 한다.
주요 entity와 relationship은 다음과 같다.
• User entity
• Problem entity
• Testcase entity
• Algorithm entity
• Hint entity
• Usersubmit entity
→ User entity와 Problem entity는 N:M의 관계를 가진다.
→ Problem entity와 Algorithm entity는 N:M의 관계를 가진다.
→ Problem entity와 Hint entity는 1:M의 관계를 가진다.
→ Problem entity와 Testcase entity는 1:M의 관계를 가진다.

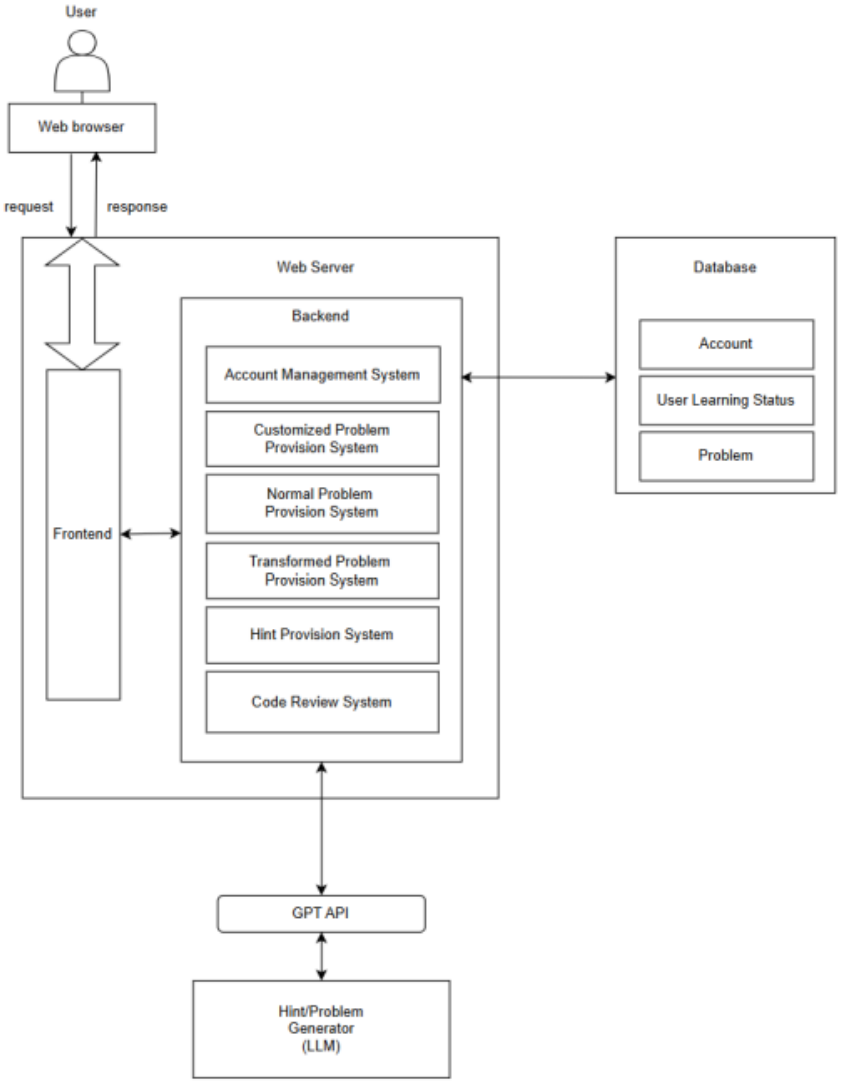

Figure 3.1: Overall System Architecture

3.3 System Diagram

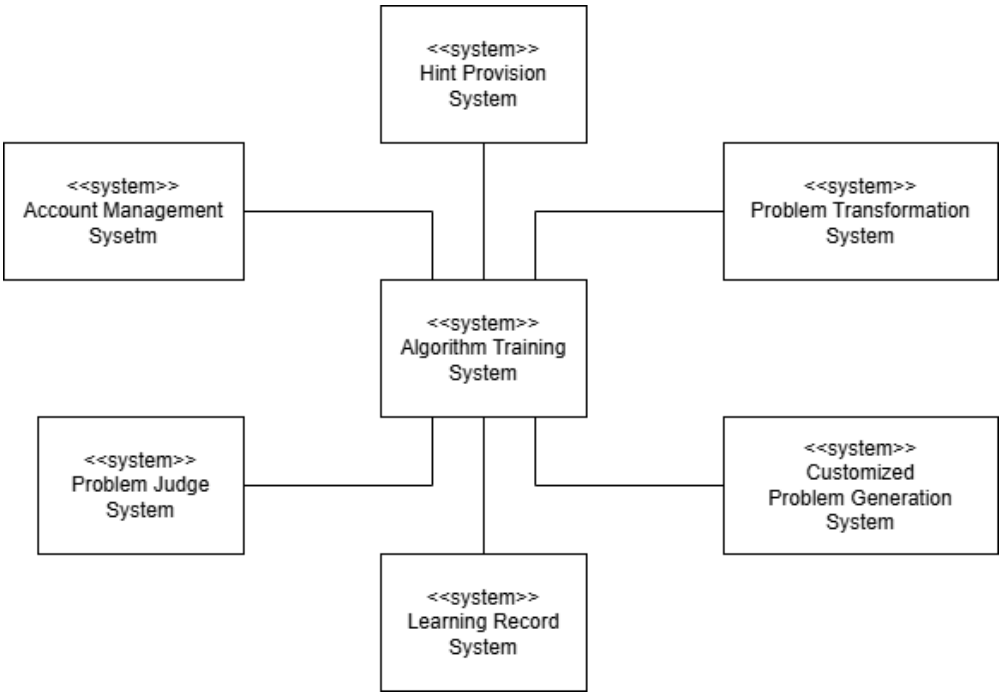

Figure 3.2: Context Model

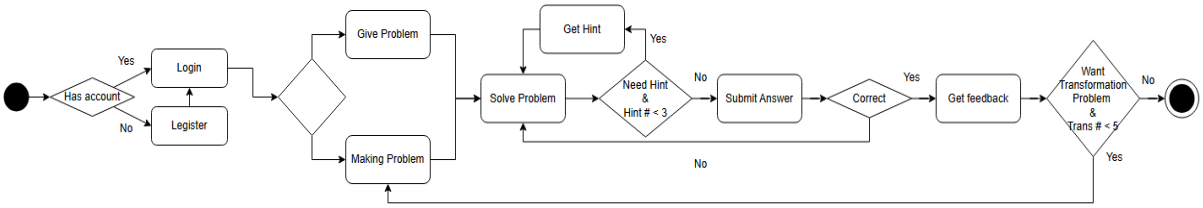

Figure 3.3: Process Model

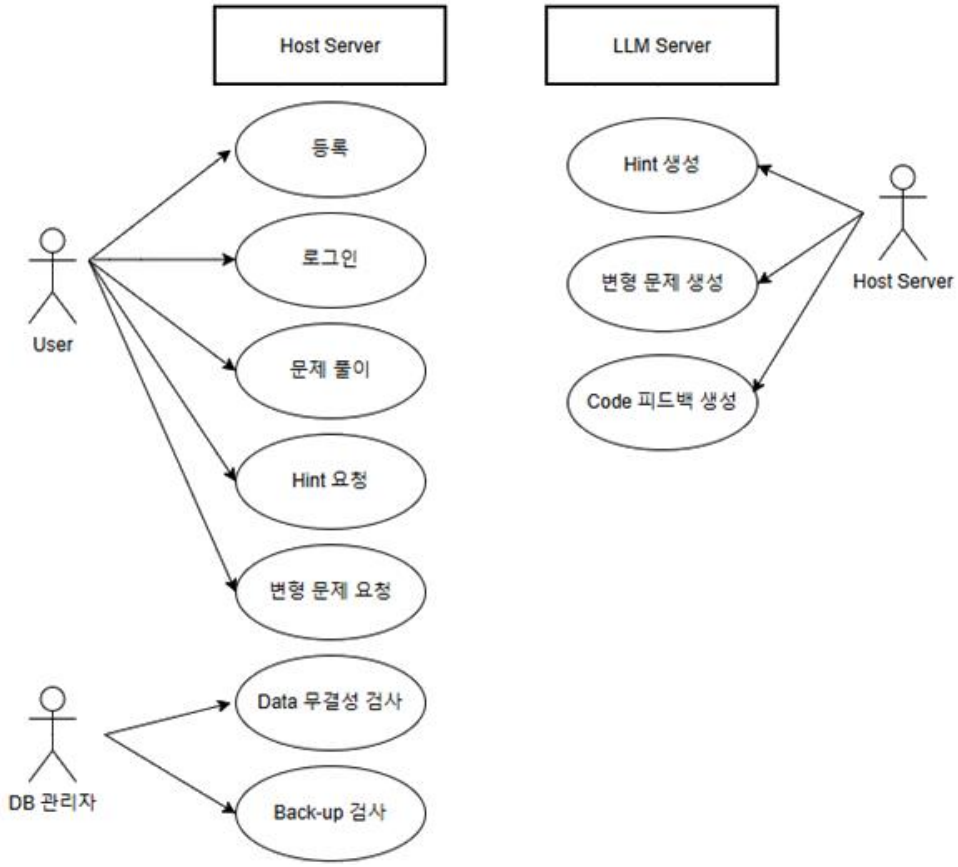

Figure 3.4: Use Case Diagram

4

System Architecture - Frontend

4.1 Objectives

이 장에서는 프론트엔드 시스템의 구조, 속성 및 기능을 설명하고 각 구성 요소의 관계를 설명한다.

유저에게는 다음과 같이 총 6개의 화면이 제공된다

회원가입/로그인 페이지, 문제를 선택하거나 맞춤형 문제를 생성할 수 있는 메인 페이지, 프로필 확인/변경 페이지, 학습 현황 확인 페이지, 문제 풀이 페이지, 제출결과 페이지.

각 화면의 관계와 구체적인 구성 방식은 Figma를 통해 제공한다. 시스템의 Frontend는 MVC pattern과 client-server pattern을 따른다.

4.1.1 Front-End Architecture

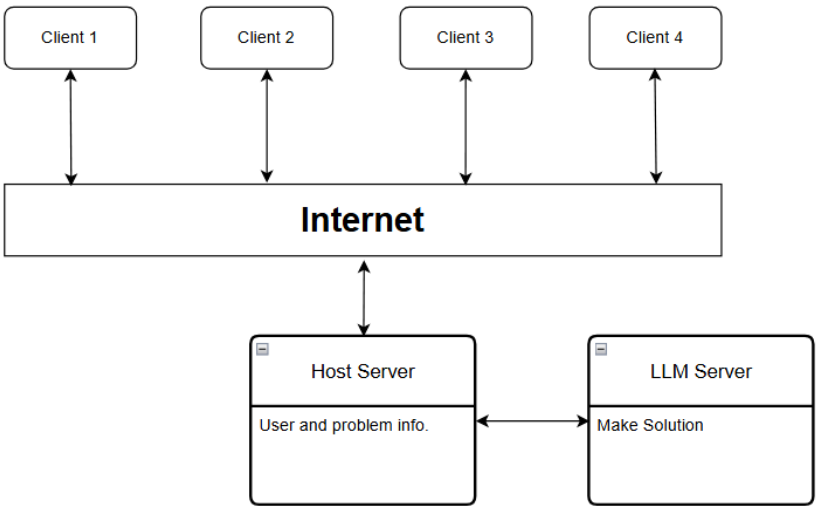

Figure 4.1: Architecture of Client and Server

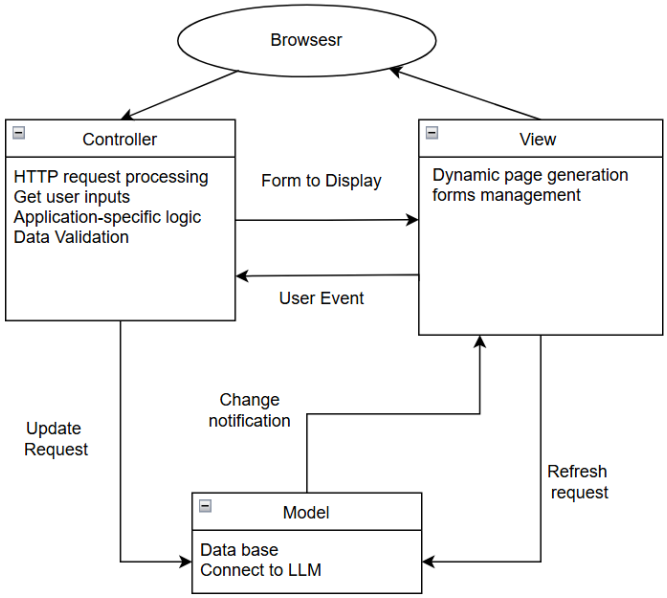

Figure 4.2: VMC Pattern

4.2 Pages

4.2.1 회원가입/로그인 페이지

사용자는 해당 view에서 로그인 또는 회원가입을 할 수 있다. 사용자는 등록을 위해 아이디, e-mail 주소, 비밀번호, 이름, 핸드폰 번호 필수적으로 입력해야 한다. 관심분야는 가입시에 입력하지 않아도 된다. 로그인 시에는 사용자는 아이디와 비밀번호를 입력하고 서버는 정보를 확인해 로그인 성공 여부를 판단한다.

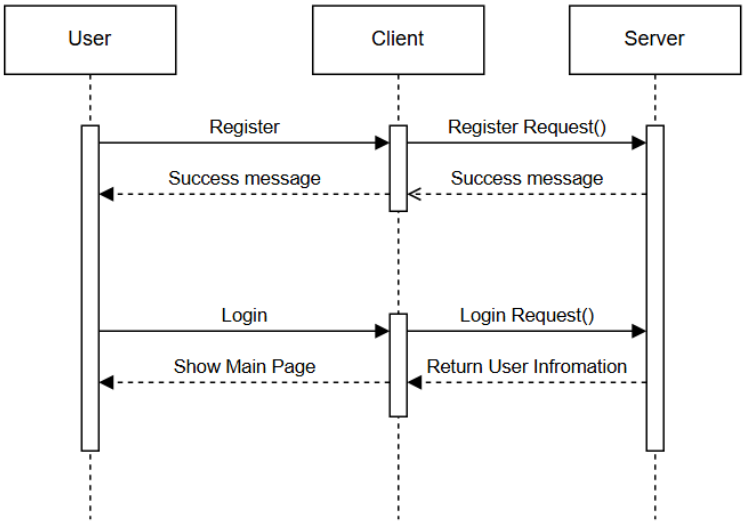

Figure 4.3: Sequence Diagram – Register/Login

4.2.2 메인 페이지

사용자는 해당 view에서 프로필 확인, 변경을 위한 페이지로 이동하거나, 학습 이력 페이지로 이동할 수 있다. 또는 data base에 존재하는 문제를 선택하여 풀거나, 자신의 관심사에 맞는 새로운 문제를 생성할 수 있다. 문제 생성시 반드시 유저의 관심분야 정보가 있어야 하며 관심분야 정보가 없는 경우 프로필 변경창에서 입력하라는 알림이 표시된다. 사용자는 LLM을 사용한 생성 기능을 하루 20번까지 사용 가능하다. 서버는 client의 LLM 생성 기능 요청 시 database를 통해 해당 유저의 LLM 생성 기능 사용 횟수를 조회한다. 문제 선택 혹은 맞춤형 문제 생성 시 문제 풀이 페이지로 이동하며, 프로필 메뉴 또는 학습 이력 확인 메뉴를 클릭 시 해당 페이지로 이동한다.

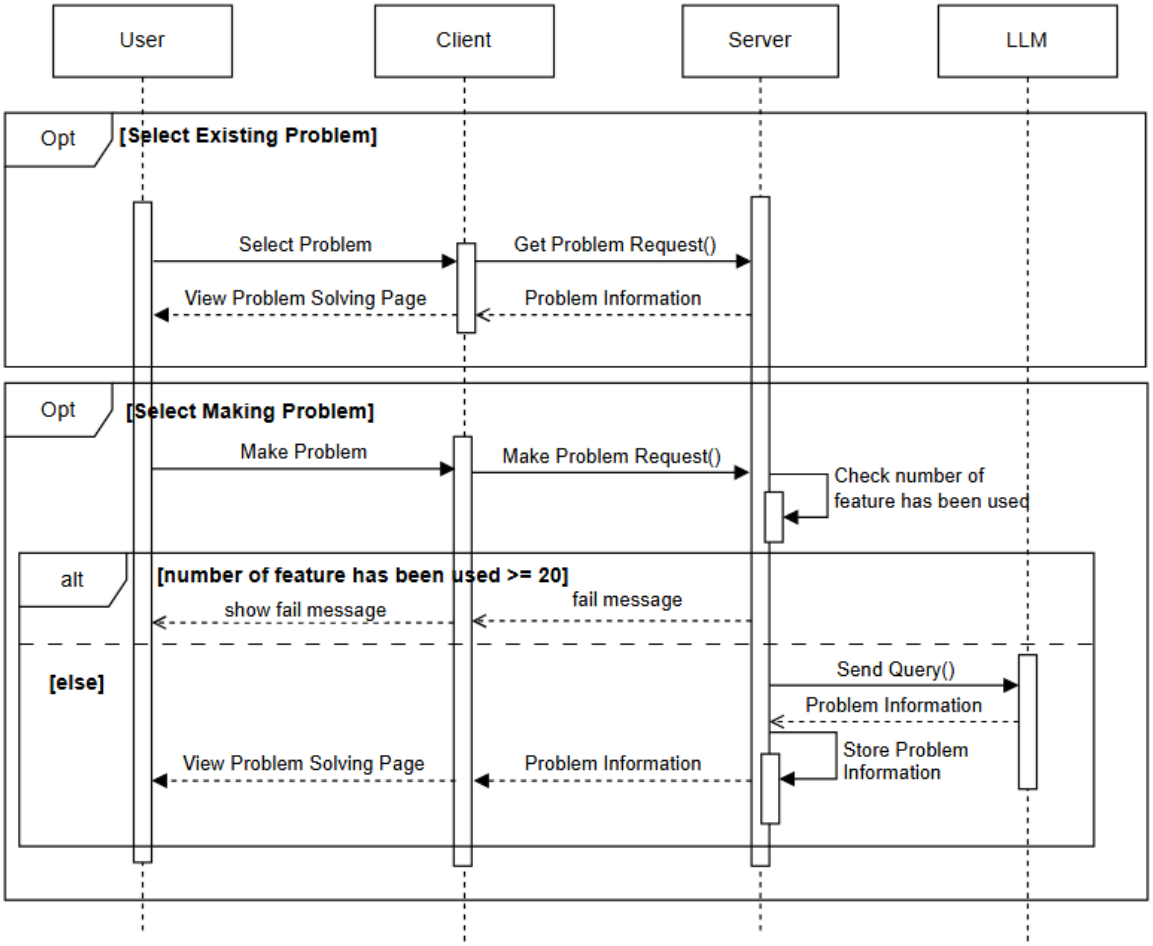

Figure 4.4: Sequence Diagram – Select Problem

4.2.3 프로필 확인/변경 페이지

사용자는 해당 view에서 이름, 이메일, 핸드폰 번호, 관심 분야에 관한 항목을 변경할 수 있다.

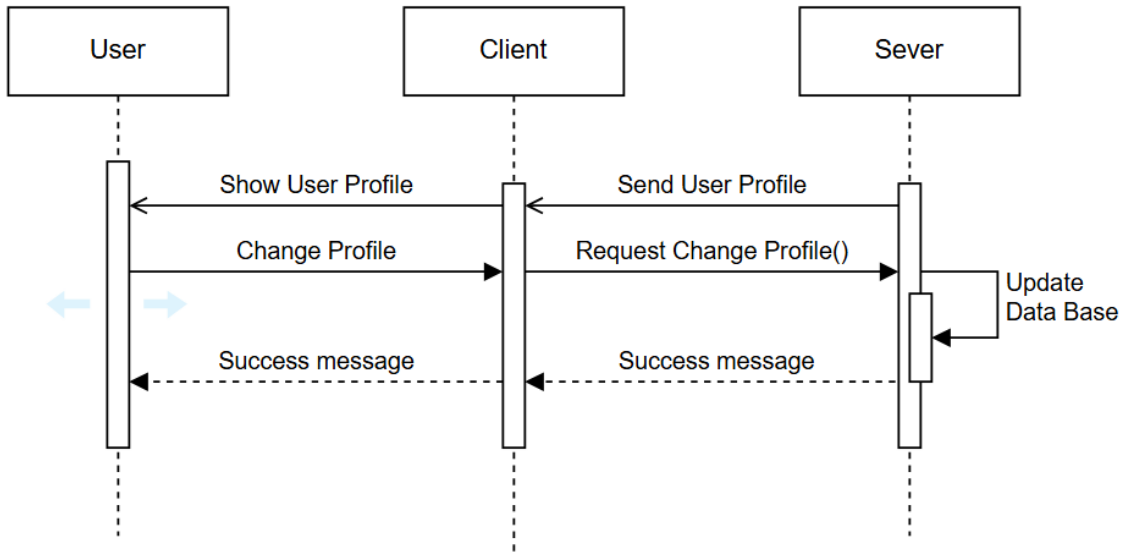

Figure 4.5: Sequence Diagram – Change Profile

4.2.4 학습 현황 페이지

사용자는 해당 view에서 지금까지 풀었던 문제와, 자신이 제출한 코드, 제출결과를 확인할 수 있다. 제출 결과에는 정답여부, 메모리, 실행시간, 제출시간이 포함된다. 사용자는 특정 기록을 클릭함으로써 해당 문제풀이 페이지로 이동 가능하며 이렇게 이동한 경우 문제 풀이 페이지의 코드 입력창에는 자동으로 저장 되어있던 코드가 표시된다.

4.2.5 문제풀이 페이지

사용자는 해당 view에서 문제에 대한 힌트를 제공받거나, 문제에 대한 정답을 작성하여 제출하고 피드백을 제공받는다. 문제를 풀던 도중 페이지에서 나갈 경우에도 작성 중인 코드는 학습 기록에 저장된다.

문제풀이 도중 사용자가 힌트제공 버튼을 누를 경우 LLM을 통해 문제에 대한 힌트를 제공한다. 힌트는 3단계로 표시되며, 1단계는 문제의 keyword에 강조표시, 2단계는 강조된 keyword에 대한 자세한 설명, 3단계는 문제에 적용 가능한 알고리즘, 자료구조를 설명한

다. 사용자가 힌트 제공 기능을 사용하면 우선 사용자가 하루 최대 LLM 생성 기능 사용 횟수를 초과하였는지 확인한다. 가용할 경우 해당 문제에 대한 힌트가 데이터 베이스에 저장되어 있는지 확인한다. 저장되어 있지 않다면 LLM 생성 기능을 통해 문제에 대한 힌트를 생성한다. 이 때 문제에 대한 세 단계 힌트를 모두 생성하여 데이터 베이스에 저장한다. 이후 사용자가 요청할 때마다 단계별로 힌트를 공개한다.

코드를 제출하는 경우 제출결과 페이지로 이동한다. 이후 사용자가 변형문제 제공을 원하는 경우 LLM 생성 기능을 사용하여 해당 문제와 비슷한 변형문제가 제공된다. 제공된 변형문제는 데이터베이스에 저장되며 문제당 최대 5개의 변형문제 생성이 가능하다. 사용자들이 요청 시 변형문제 제공되며 해당 문제에 대해 다른 사용자가 이미 생성한 변형문제가 있는 경우 해당 문제를 제공한다. 변형문제 대한 변형문제는 생성 불가능하다. 변형문제 생성 기능 역시 LLM 생성 기능으로 카운트된다.

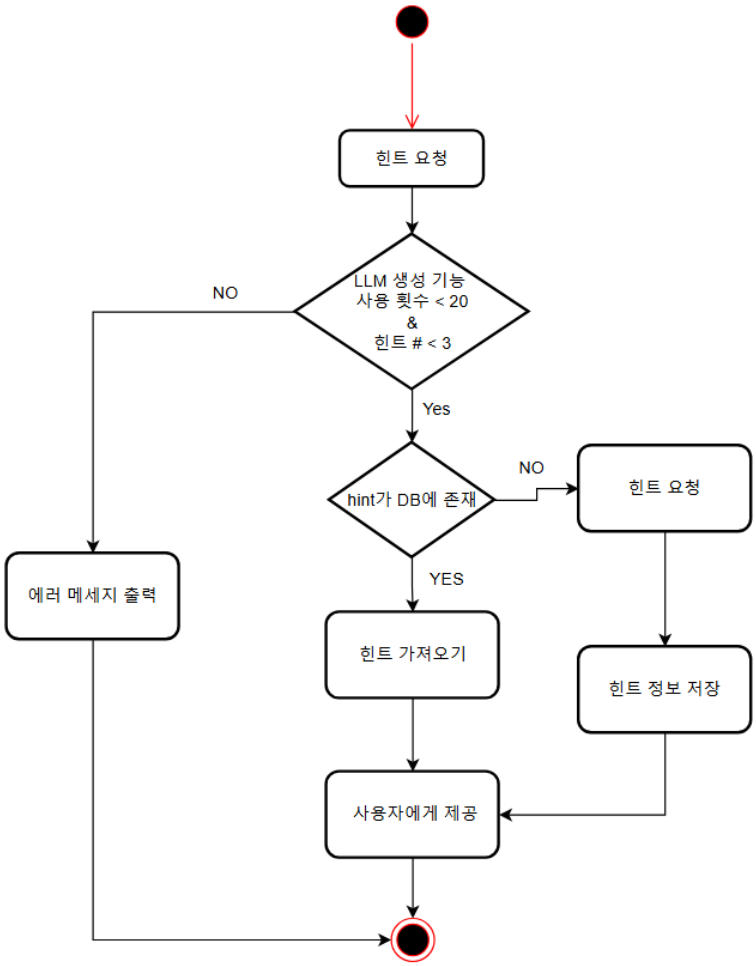

Figure 4.6: Process Diagram – 힌트제공

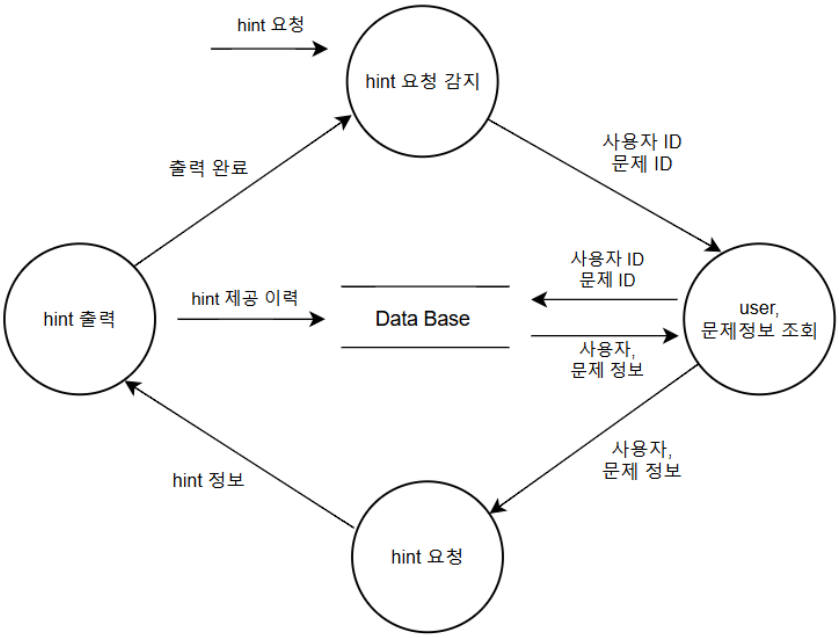

Figure 4.7: Data Flow Diagram – 힌트 제공

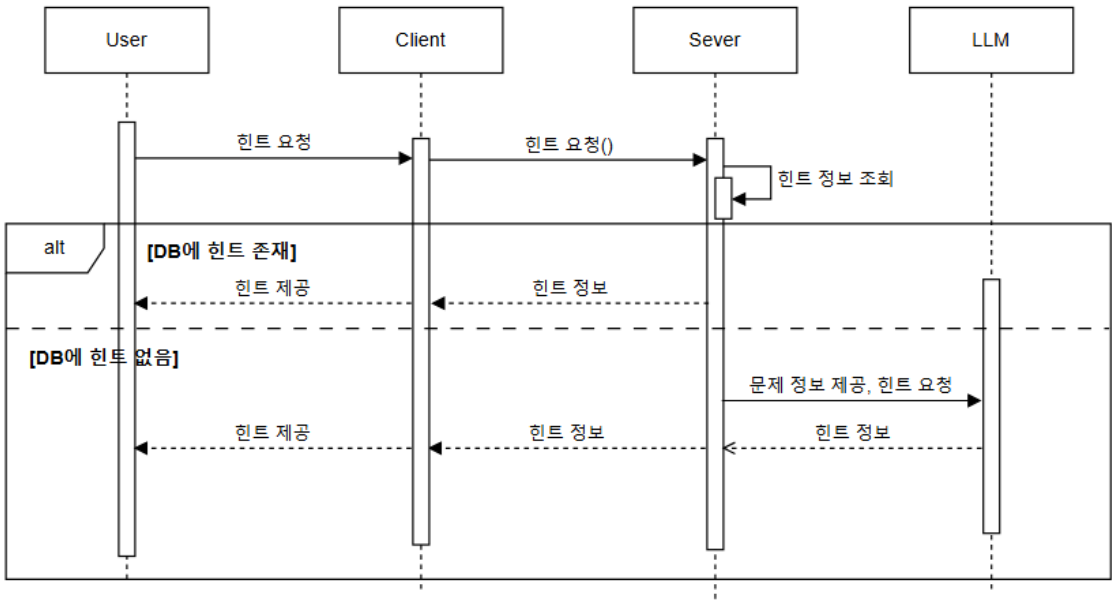

Figure 4.8: Sequence Diagram – 힌트 제공

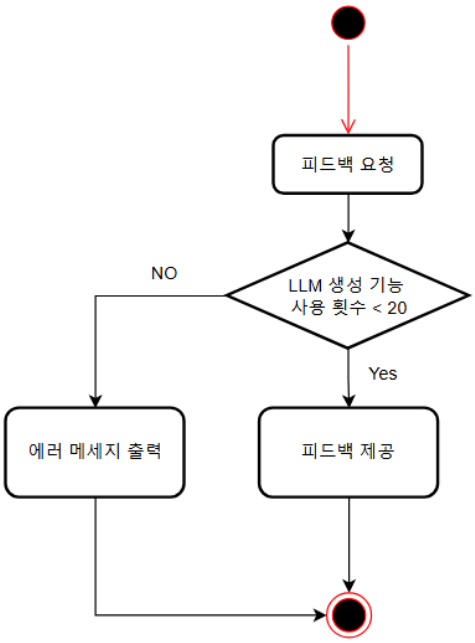

Figure 4.9: Process Diagram – 피드백 제공

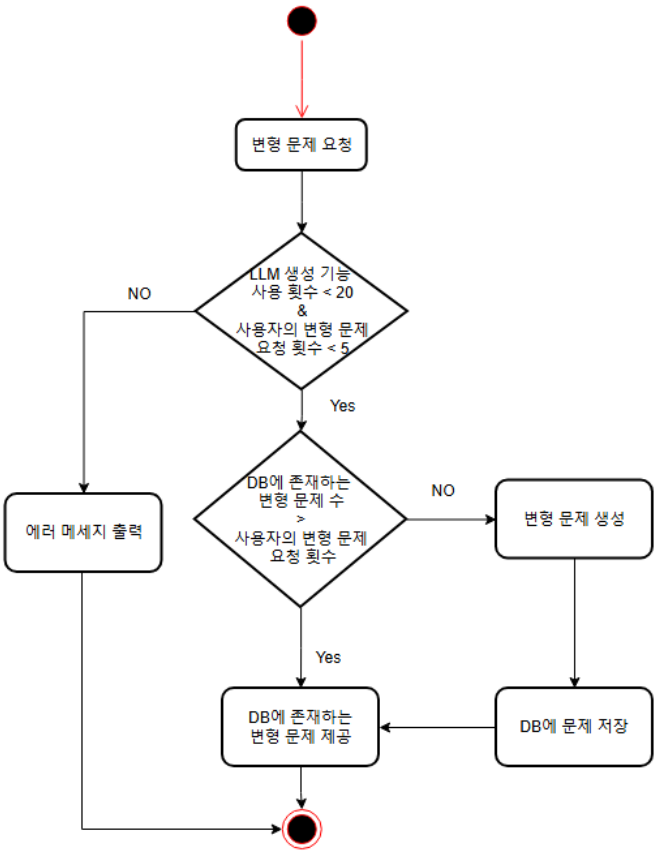

Figure 4.10: Process Diagram – 변형 문제 제공

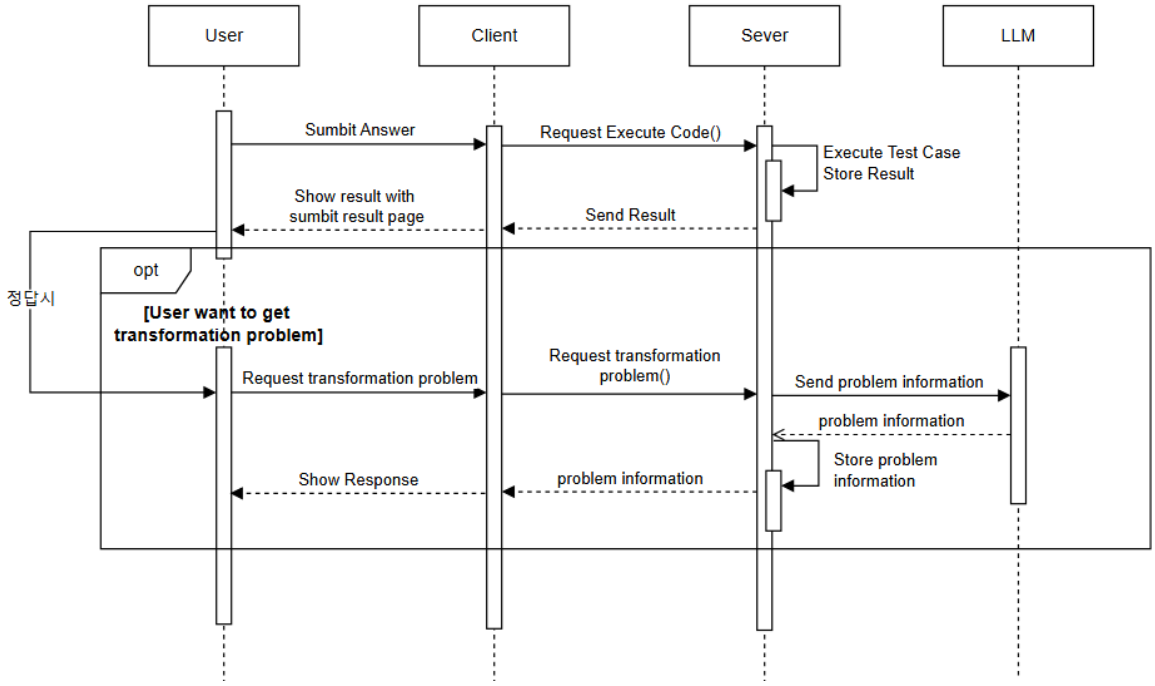

Figure 4.11: Sequence Diagram – 변형 문제 제공

4.2.6 제출 결과 페이지

채점 결과, 실행 결과가 화면에 표시되며, 제출한 코드, 채점 결과, 메모리 사용량, 실행 시간, 제출 시간이 데이터 베이스에 기록된다. 정답이 맞은 경우 사용자에게 LLM을 사용한 피드백 제공을 원하는지 묻는다. 원하는 경우 피드백 내용이 사용자에게 표시된다. 피드백 제공 기능 역시 LLM 생성 기능으로 카운트된다.

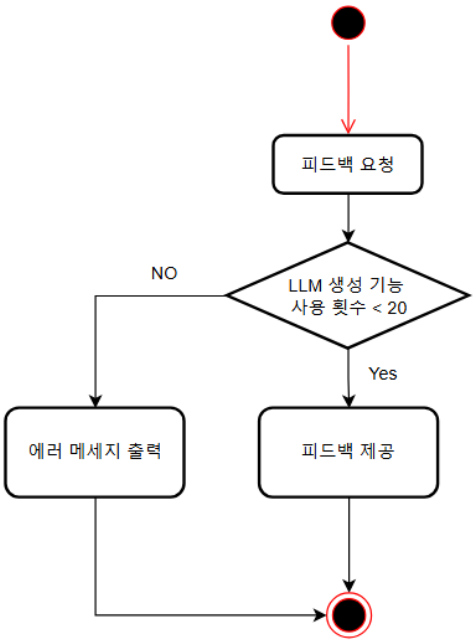

Figure 4.12: Process Diagram – 피드백 제공

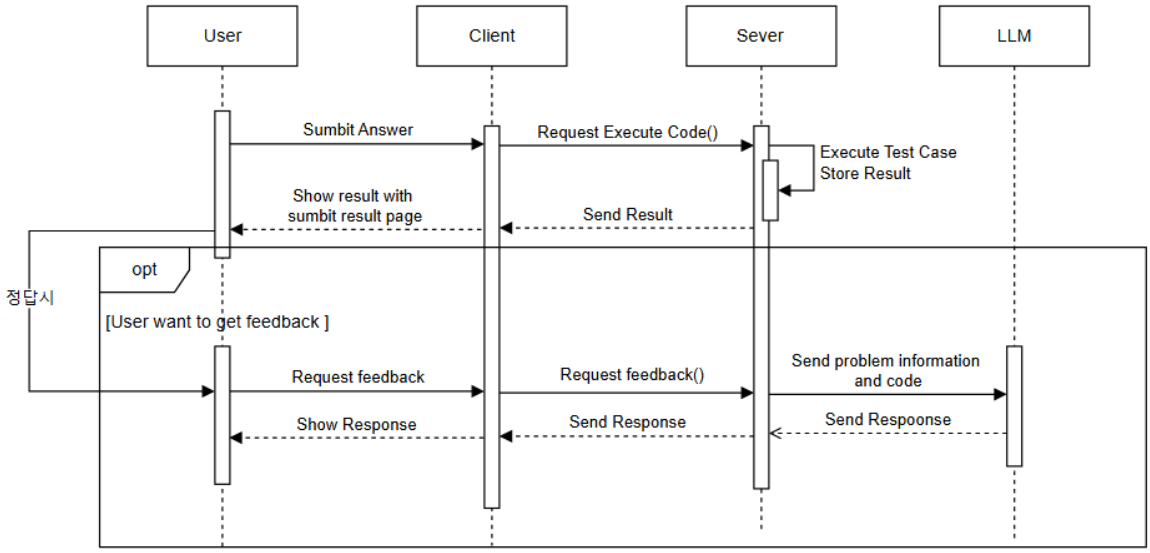

Figure 4.13: Sequence Diagram – 피드백 제공

5

System Architecture - Backend

5.1 Objectives

5장에서는 GPTeacher 시스템의 Backend 구조에 대해서 설명한다.

5.2 Architecture

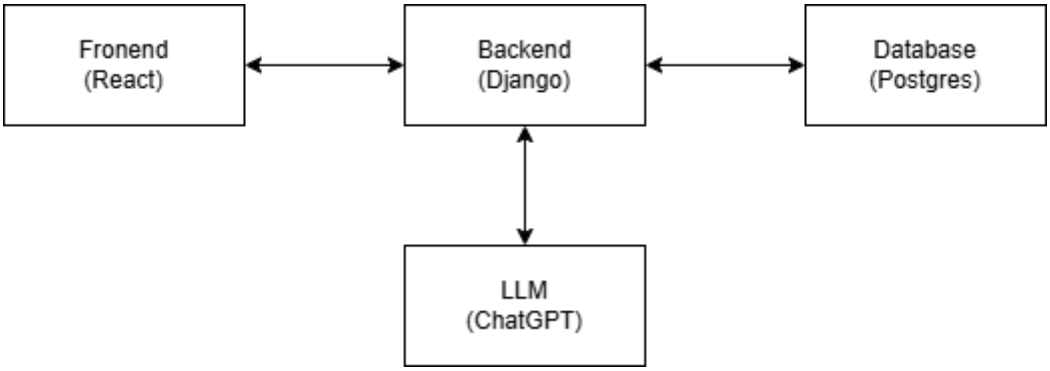

Figure 5.1: Backend Architecture

Backend는 Frontend로부터 요청을 받는다.
그 후 Database, LLM과 상호작용 후 Frontend에 응답을 전달한다.

5.3 Subcomponents

5.3.1 Account Management System

5.3.1.1 Signup

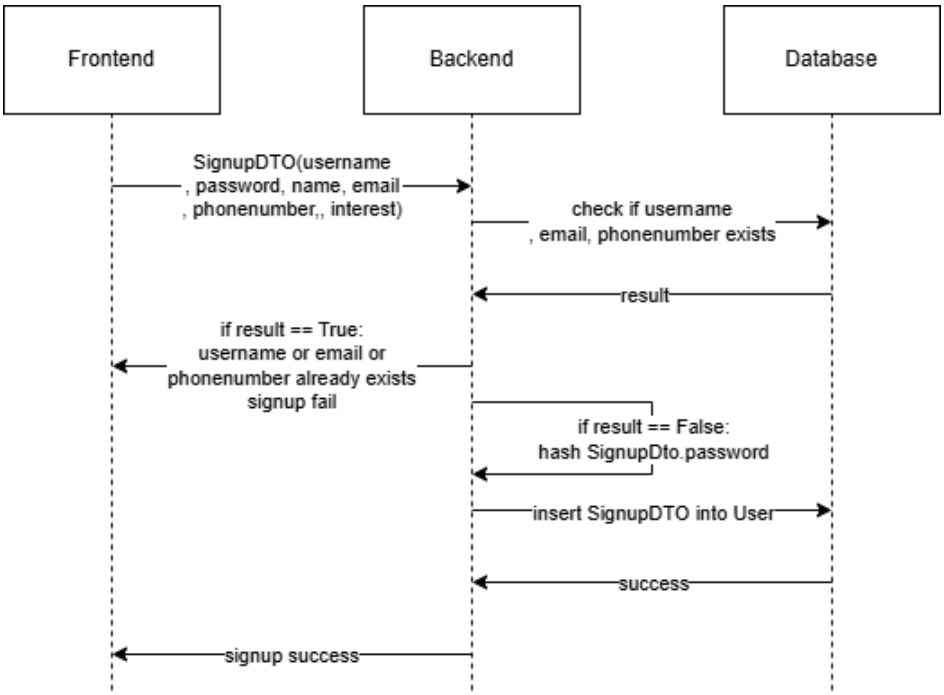

Figure 5.2: Signup Sequence Diagram

Frontend로부터 SignupDTO(username, password, name, email, phonenumber, interest)를 받는다.
username, email, phonenumber가 중복되는지 확인한다.
중복된다면 Frontend에 회원가입 실패를 전달한다.
중복되지 않는다면 비밀번호를 해싱한 후 SignupDTO를 Database의 User 테이블에 추가한다. 이후 Frontend에 회원가입 성공을 전달한다.

5.3.1.2 Login

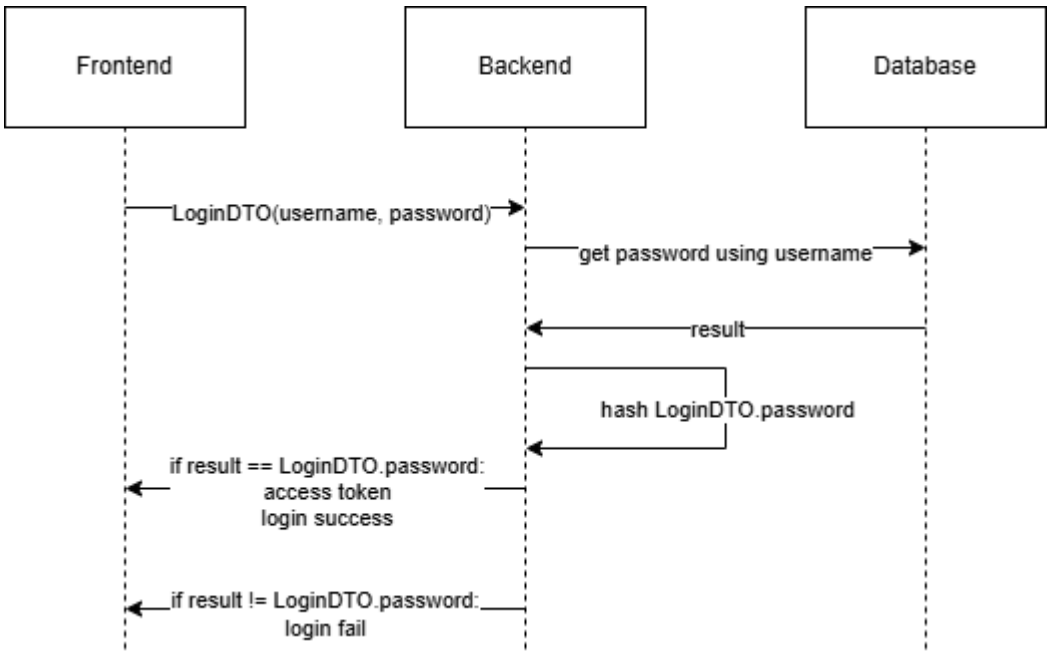

Figure 5.3: Login Sequence Diagram

Frontend 로부터 LoginDTO(username, password)를 받는다.
username 으로 Database 에서 사용자의 password 를 가져온다.
LoginDTO.password 를 해싱한다.
해싱한 password 와 Database 에서 가져온 password 가 같은지 확인한다.
같다면 Frontend 에 로그인 성공과 함께 access token 을 전달한다.
다르다면 Frontend 에 로그인 실패를 전달한다.

5.3.1.3 Edit Profile

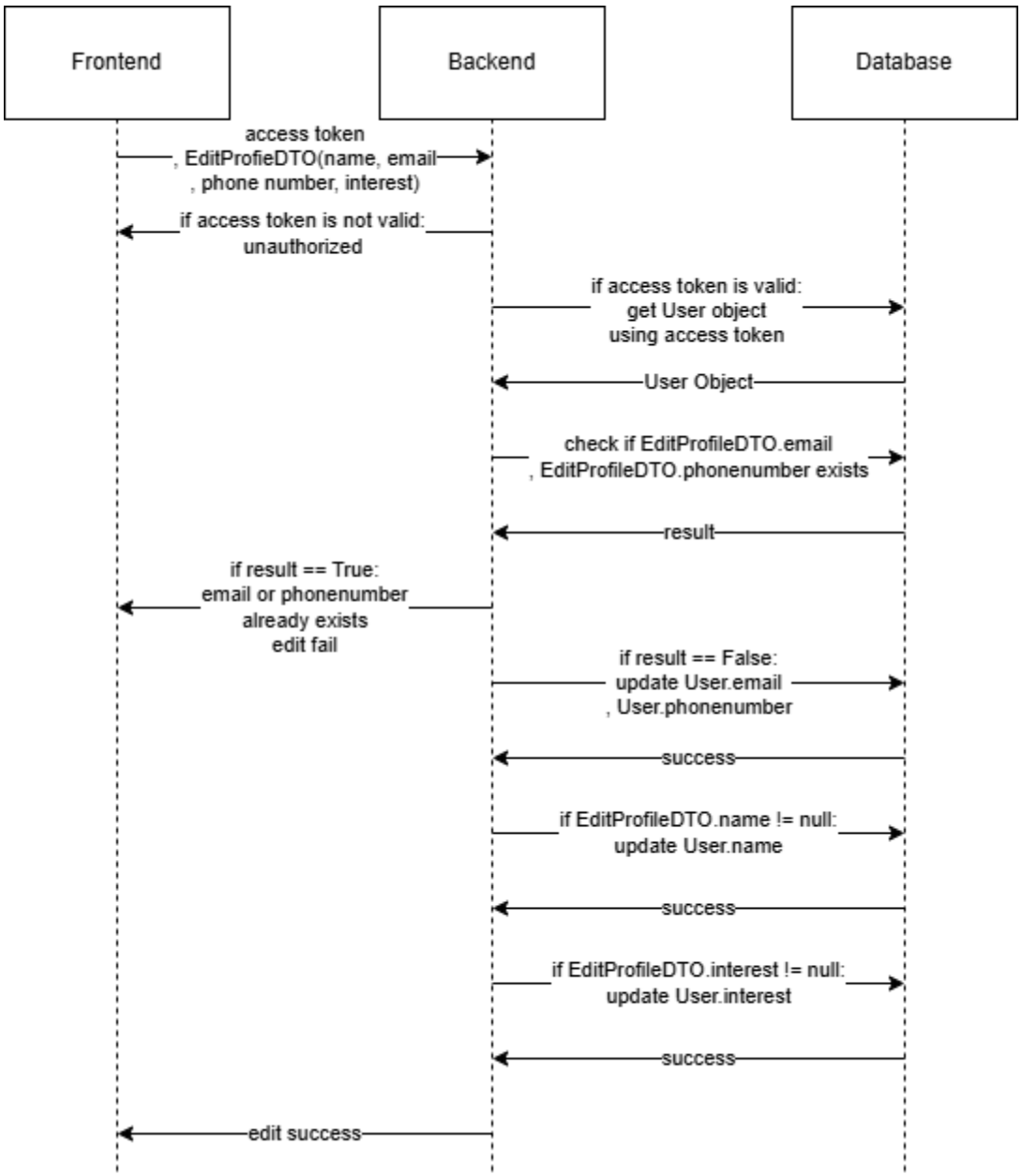

Figure 5.4: Edit Profile Sequence Diagram

Frontend 로부터 accesstoken 과 EditProfileDTO(name, email, phonenumber, interest)를 받는다.
access token 이 valid 하지 않다면 Frontend 에 unauthorized 를 전달한다.
access token 이 valid 하다면 access token 을 사용하여 User object 를 가져온다.
EditProfile.DTO.email, EditProfileDTO.phonenumber 들이 중복되는지 확인한다.

만약 중복된다면, Frontend 에 edit fail 을 전달한다.
만약 중복되지 않는다면, User.email, User.phonenumber 를 업데이트한다.
만약 EditProfileDTO.name 이 null 이 아니라면, User.name 을 업데이트한다.
만약 EditProfileDTO.interets 가 null 이 아니라면, User.interest 를 업데이트한다.
Frontend 에 edit success 를 전달한다.

5.3.2 Hint Provision System

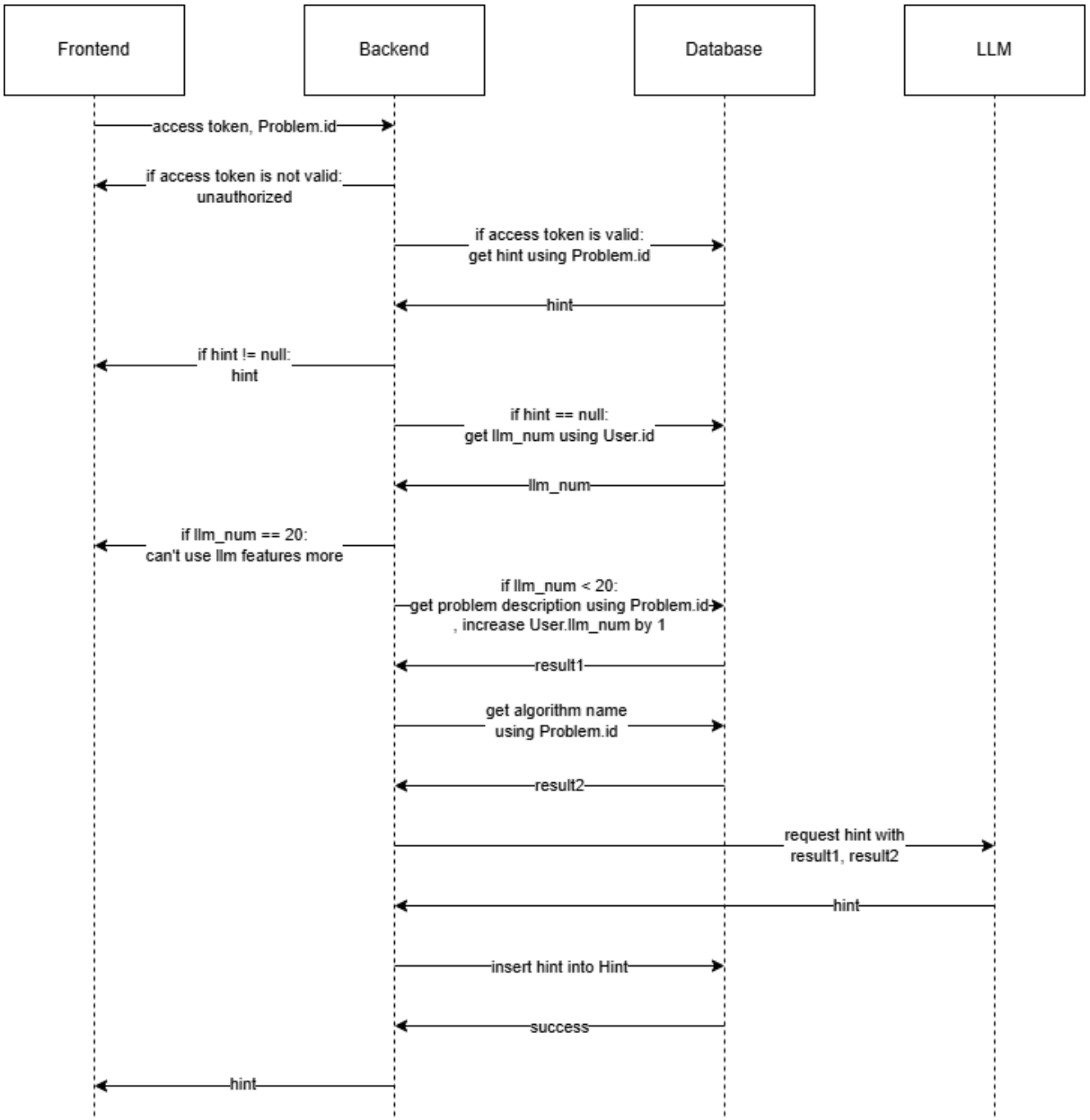

Figure 5.5: Hint Provision Sequence Diagram

Frontend로부터 accesstoken과 Problem.id를 받는다.
access token이 valid 하지 않다면 Frontend에 unauthorized를 전달한다.
access token이 valid 하다면 Problem.id로 Database에서 hint를 가져온다.
만약 힌트가 null이 아니라면 Frontend에 hint를 전달한다.
만약 힌트가 null이라면 다음 단계를 진행한다.
User.id로 Database에서 llm_num을 가져온다.
만약 llm_num이 20이라면 Frontend에 can't use llm features more를 전달한다.
만약 llm_num이 20보다 작다면 다음 단계를 진행한다.
Problem.id로 Database에서 problem의 description을 가져오고, User.llm_num의 값을 1 증가시킨다.
Problem.id로 Database에서 algorithm name을 가져온다.
description과 algorithm name으로 LLM에 hint를 요청하고 받아온다.
hint를 Database의 Hint 테이블에 저장한다.
Frontend에 hint를 전달한다.

5.3.3 Problem Provision System

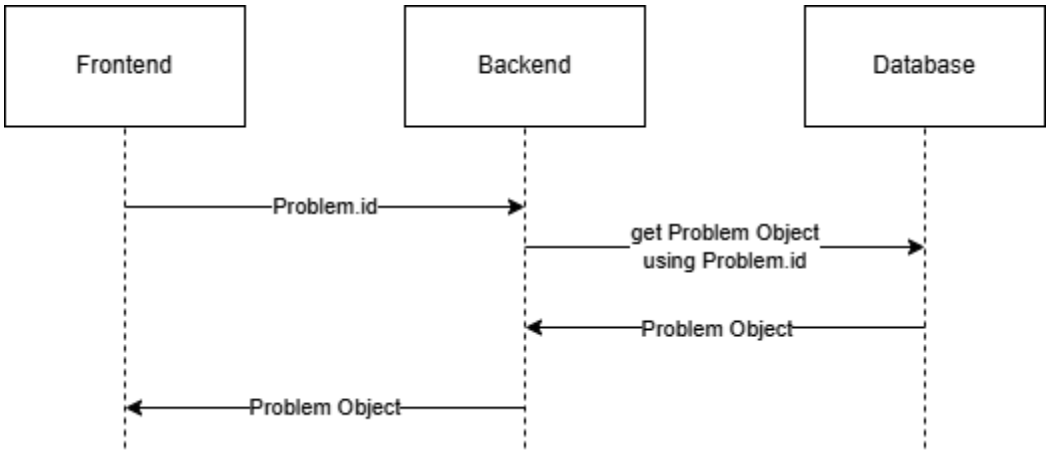

Figure 5.6: Problem Provision Sequence Diagram

Frontend 로부터 Problem.id 를 받는다.
Problem.id 로 Database 에서 Problem Object 를 가져온다.
Frontend 에 Problem Object 를 전달한다.

5.3.4 Problem Transformation System

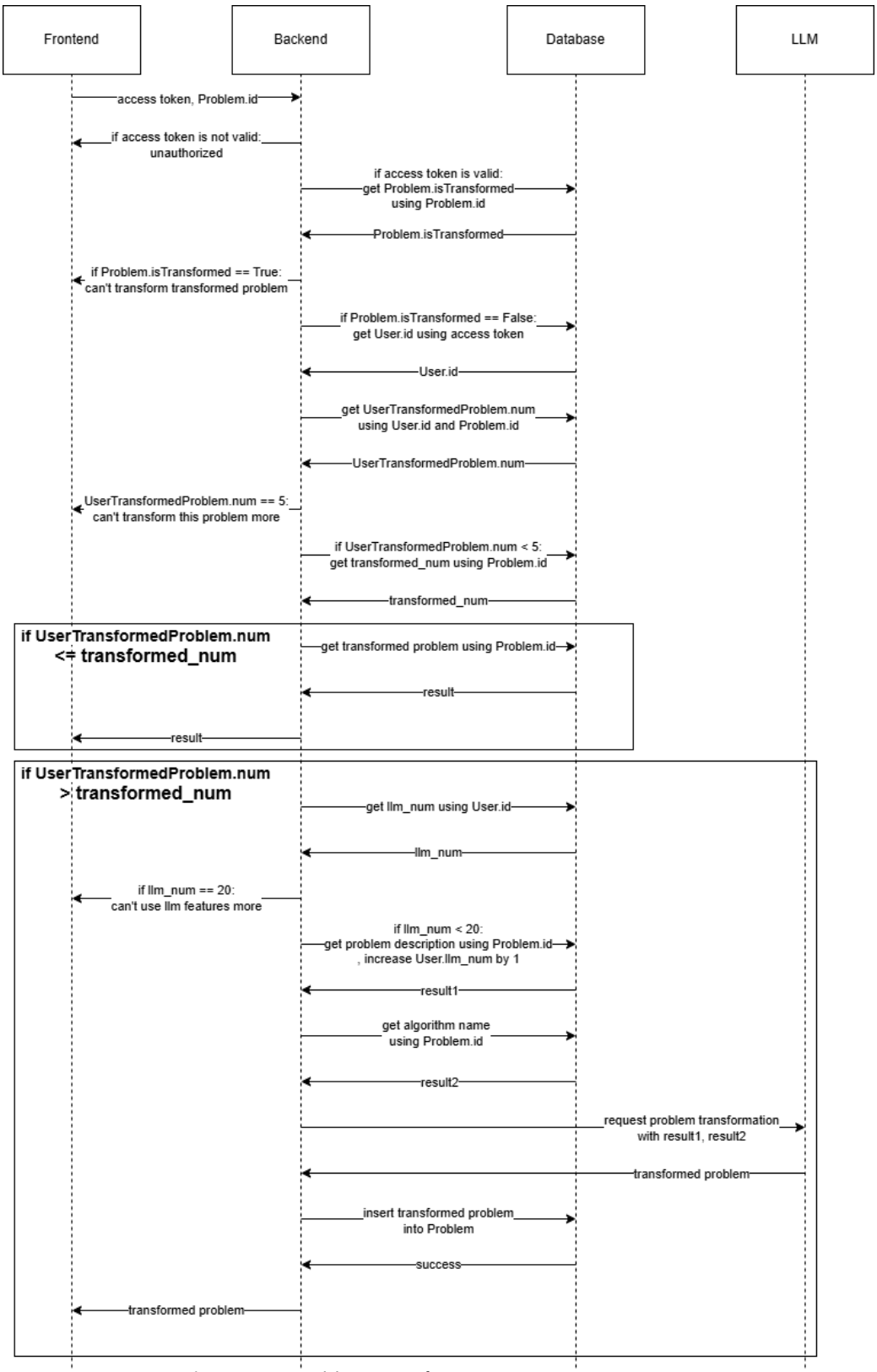

Figure 5.7: Problem Transformation Sequence Diagram

Frontend로부터 access token과 Problem.id를 받는다.
access token이 valid 하지 않다면 Frontend에 unauthorized를 전달한다.
access token이 valid 하다면 Problem.id로 Database에서 Problem.isTransformed를 가져온다.
만약 Problem.isTransformed가 True라면 Frontend에 can't transform transformed problem을 전달한다.
만약 Problem.isTransformed가 False라면 access token으로 User.id를 가져온다.
User.id와 Problem.id로 Database에서 UserTransformedProblem.num을 가져온다.
만약 UserTransformedProblem.num이 5라면 Frontend에 can't transform this problem more를 전달한다.
만약 UserTransformedProblem.num이 5보다 작다면 Problem.id로 Database에서 transformed_num을 가져온다.
만약 UserTransformedProblem.num이 transformed_num보다 작거나 같다면 Problem.id로 Database에서 transformed problem을 가져온다음 Frontend에 전달한다.
만약 UserTransformedProblem.num이 transformed_num보다 크다면 다음 단계를 진행한다.
User.id로 Database에서 llm_num을 가져온다.
만약 llm_num이 20이라면 Frontend에 can't use llm features more를 전달한다.
만약 llm_num이 20보다 작다면 다음 단계를 진행한다.
Problem.id로 Database에서 problem의 description을 가져오고, User.llm_num의 값을 1 증가시킨다.
Problem.id로 Database에서 algorithm name을 가져온다.
description과 algorithm_name으로 LLM에 problem transformation을 요청하고 받아온다.
transformed problem을 Database의 Problem 테이블에 저장한다.
Frontend에 transformed problem을 전달한다.

5.3.5 Customized Problem Generation System

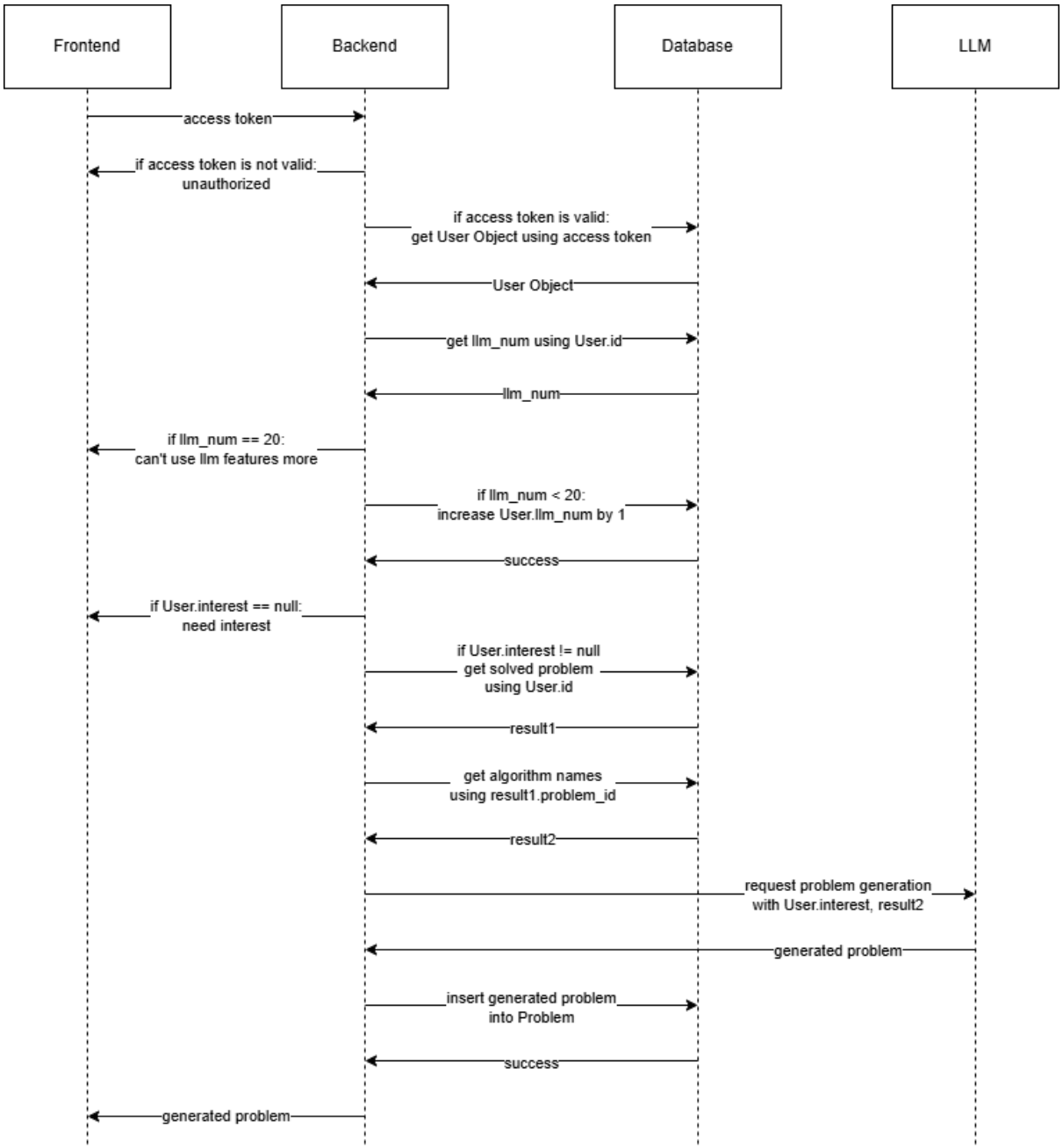

Figure 5.8: Customized Problem Generation Sequence Diagram

Frontend로부터 access token을 받는다.
access token이 valid 하지 않다면 Frontend에 unauthorized를 전달한다.
access token이 valid 하다면 access token을 사용하여 User object를 가져온다.
User.id로 Database에서 llm_num을 가져온다.
만약 llm_num이 20이라면 Frontend에 can't use llm features more를 전달한다.

만약 llm_num이 20보다 작다면 llm_num의 값을 1 증가시킨다.
만약 User.interest == null 이라면 Frontend에 need interest를 전달한다.
만약 User.interest != null 이라면 다음 단계를 진행한다.
User.id로 Database에서 solved problem을 가져온다.
result1에 있는 모든 problem_id로 Database에서 algorithm_name을 가져온다.
User.interest와 result2에 있는 모든 algorithm_name으로 LLM에 problem generation을 요청하고 받아온다.
generated problem을 Database의 Problem 테이블에 저장한다.
Frontend에 generated problem을 전달한다.

5.3.6 Problem Judge System

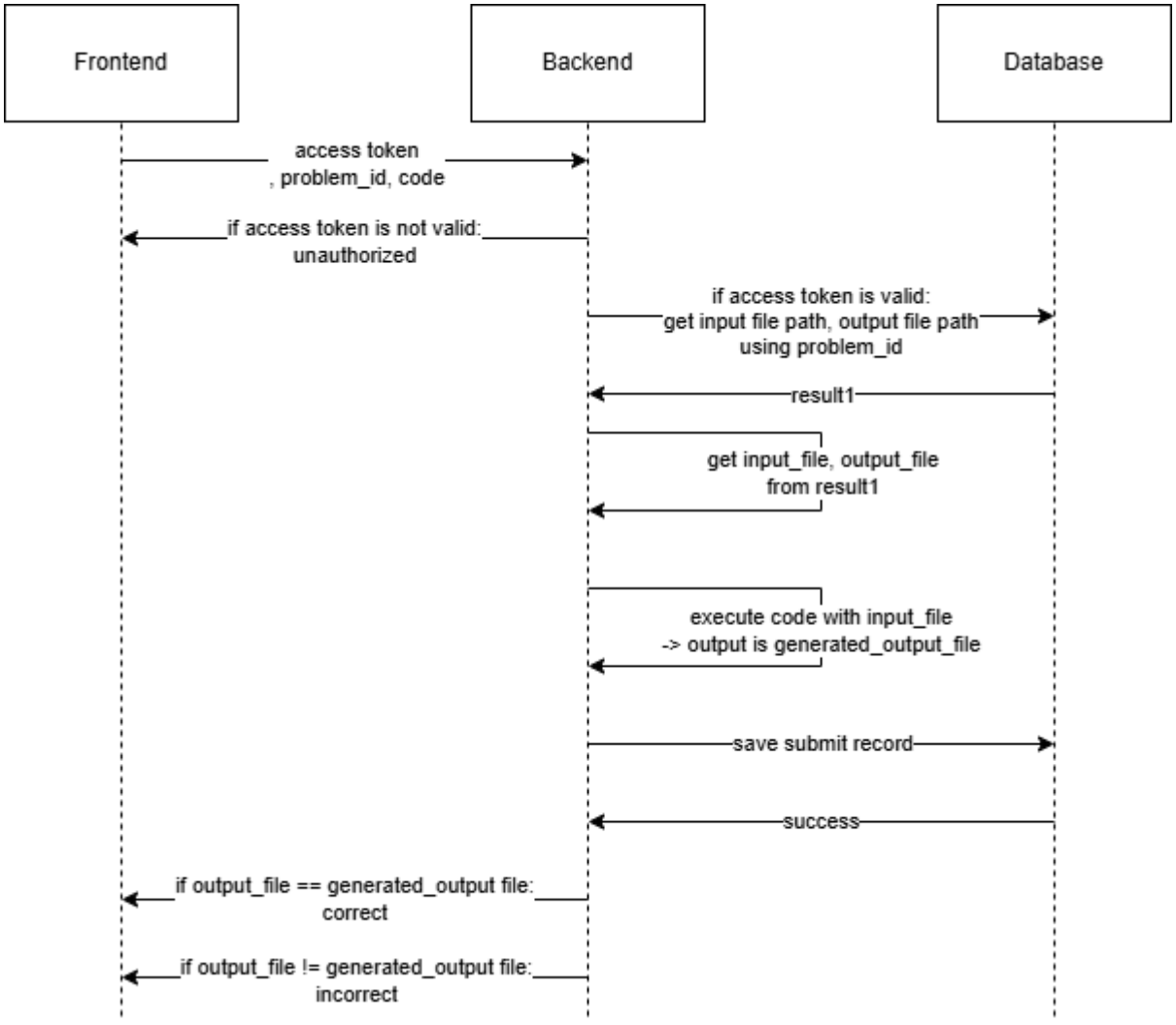

Figure 5.9: Problem Judge Sequence Diagram

Frontend로부터 access token, problem_id, 사용자의 code를 받는다.
access token이 valid 하지 않다면 Frontend에 unauthorized를 전달한다.
access token이 valid 하다면 다음 단계를 진행한다.
problem_id를 사용하여 Database에서 input file path와 output file path를 가져온다.
result1을 사용하여 input_file과 output_file을 가져온다.
사용자의 code를 input_file과 함께 실행하고, generated_output_file을 생성한다.
제출 기록을 Database에 저장한다.
output_file과 generated_output_file이 같은지 확인한다.
같다면 Frontend에 correct를 전달한다.
다르다면 Frontend에 incorrect를 전달한다.

5.3.7 Code Review System

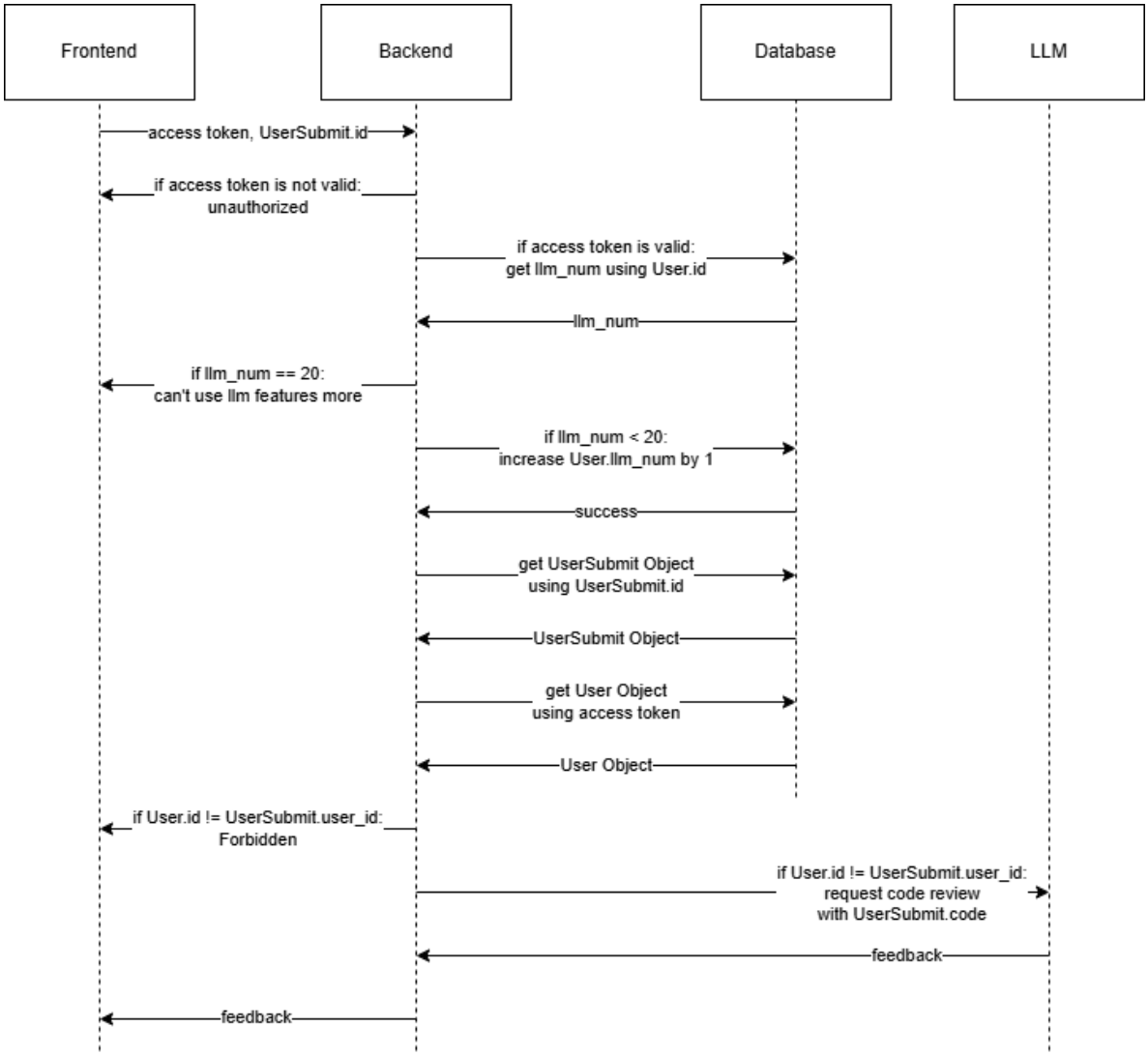

Figure 5.10: Code Review Sequence Diagram

Frontend 로부터 access token, UserSubmit.id 를 받는다.
access token 이 valid 하지 않다면 Frontend 에 unauthorized 를 전달한다.
access token 이 valid 하다면 User.id 로 Database 에서 llm_num 을 가져온다.
만약 llm_num 이 20 이라면 Frontend 에 can't use llm features more 를 전달한다.
만약 llm_num 이 20 보다 작다면 llm_num 의 값을 1 증가시킨다.
UserSubmit.id 를 사용해서 Database 에서 UserSubmit Object 를 가져온다.
access token 을 사용해서 User Object 를 가져온다.
만약 User.id != UserSubmit.user_id 라면 Frontend 에 Forbidden 을 전달한다.
만약 User.id == UserSubmit.user_id 라면 다음 단계를 진행한다.
사용자의 code 로 LLM 에 code review 를 요청하고 feedback 을 받아온다.
Frontend 에 feedback 을 전달한다.

6

Protocol Design

6.1 Objectives

이 절에서는 프론트엔드와 백엔드 사이에 사용되는 통신 프로토콜과 통신에서 사용되는 인터페이스에 대해 기술한다.

6.2 AJAX (Asynchronous JavaScript and XML)

AJAX는 javaScript와 XML을 이용한 비동기적 정보 교환 기법을 의미한다. AJAX는 XMLHttpRequest객체를 통해 서버에 HTTP요청을 보내고, JSON, XML, HTML 또는 일반 텍스트와 같은 다양한 형식으로 필요한 데이터를 받아온다. 서버와 웹 페이지 간의 통신이 비동기적으로 이루어지기 때문에 전체 페이지를 로드 할 필요 없이 필요한 데이터만 교환하여 응답 속도가 빠르다. 또한 브라우저는 서버에 보낸 요청에 대한 응답을 기다리지 않고 다른 작업을 수행할 수 있다. 이러한 특성으로 인해 사용자 경험을 향상시킬 수 있다. 이 프로젝트에서는 서버와 웹페이지가 XML이 아닌 JSON형식으로 데이터를 교환한다.

6.3 HTTPS (Hyper Text Transfer Protocol over Secure Socket Layer)

HTTPS는 클라이언트와 서버 간의 데이터를 안전하게 전송하기 위해 사용되는 통신 프로토콜이다. HTTP(Hyper Text Transfer Protocol)에 보안 기능을 추가한 것으로, TLS 프로토콜을 사용해 데이터를 암호화하여 민감한 데이터를 안전하게 보호한다.

6.4 JWT (JSON Web Token)

JWT은 웹에서 사용되는 JSON형식의 표준화된 토큰이다. 주로 인증과 정보 교환에 사용되어 클라이언트와 서버 간의 정보를 안전하게 전달할 수 있게 한다. JWT는 토큰 자체에

인증 정보를 포함하고 있기 때문에 서버가 사용자 정보를 저장할 필요가 없어 무상태(stateless) 환경에서도 동작할 수 있으며, 서명을 통해 데이터의 변조를 방지해 데이터의 무결성을 보장한다.

6.5 REST API (Representational State Transfer API)

REST API는 클라이언트와 서버 간의 통신을 제공하는 인터페이스이다.

본 시스템에서 사용하는 REST API는 다음의 설계 가이드를 따른다.

<REST API 설계 가이드>

1) Protocol

- api 통신 시 사용하는 프로토콜 정보를 표시한다.
- 본 시스템의 api는 HTTPS 프로토콜을 사용한다.
- HTTPS로 표기한다.

2) Method

- 리소스에 대한 행위를 나타낸다.
- 다음의 methods를 리소스의 목적에 따라 사용한다.
  - POST: 새 리소스 생성
  - GET: 특정 리소스 조회
  - PUT: 기존 리소스 전체 업데이트
  - PATCH: 기존 리소스의 일부를 업데이트
  - DELETE: 기존 리소스 삭제
  - HEAD: 리소스의 메타데이터만 조회

3) URI

- URI는 리소스를 나타낸다.
- 리소스는 시스템에서 관리되는 객체나 데이터를 나타낸다.
- 리소스는 일반적으로 복수형 명사로 표현된다.
- 리소스는 일반적으로 소문자로 표현된다.
- 여러 단어로 이루어진 리소스는 –(hyphen)으로 구분한다.
- 리소스 간의 계층 관계를 /(slash)로 나타낸다.
- 리소스에는 _(underbar) 가 아닌 -(dash)를 사용한다.

- 컨트롤 리소스는 예외적으로 동사를 허용한다.

4) Header

- Content Type
  - 전송되는 데이터의 타입을 표기한다.
  - 본 시스템에서는 JSON형식으로 데이터를 주고받는다.
  - application/json으로 표기한다.
- Authorization
  - 사용되는 토큰을 표기한다.
  - 본 시스템에서는 토큰으로 JWT를 사용한다.
  - Bearer <JWT>로 표기한다.
  - Authorization Header에 JWT를 포함해 요청을 전달함으로써 CSRF공격을 방어할 수 있다.

5) Body

- Request Body
  - 클라이언트가 서버에 요청 시 보낼 데이터를 표기한다.
- Response Body
  - 클라이언트가 서버의 응답 수신 시 받을 데이터를 표기한다.

6) Status code

- 2xx: 성공 응답
  - 200 OK: 요청 정상 처리
  - 201 Created: 새 리소스 생성
  - 204 No Content: 요청 정상 처리, 응답 데이터 없음
- 4xx: 실패 응답
  - 400 Bad Request: 잘못된 요청
  - 401 Unauthorized: 인증 실패
  - 403 Forbidden: 서버가 요청을 이해했으나 거부하거나 접근이 허용되지 않음
  - 409 Conflict: 리소스 충돌
- 5xx: 서버 오류
  - 500 Internal Server Error: 서버 오류 발생

6.6 Authentication

6.6.1 Register

Request

Attribute	Detail	
Protocol	HTTPS	
Method	POST	
URI	/api/auth/signup	
Request body	Name	User name
	ID	User ID
	Password	User password
	Email	User email
	Phone Number	User phone number
	Interest (Optional)	User Interest
Content Type	application/json	

Figure 6.1: Register Request

Response

Attribute	Detail	
Protocol	HTTPS	
Success Code	201 Created	
Failure Code	400 Bad Request or 409 Conflict or 500 Internal Server Error	
Success Response Body	Message	Success message
Failure Response Body	Message	Failure message

Figure 6.2: Register Response

6.6.2 Login

Request

Attribute	Detail	
Protocol	HTTPS	
Method	POST	
URI	/api/auth/login	
Request body	ID	User ID
	Password	User password
Content Type	application/json	

Figure 6.3: Login Request

Response

Attribute	Detail	
Protocol	HTTPS	
Success Code	200 OK	
Failure Code	400 Bad Request or 500 Internal Server Error	
Success Response Body	Message	Success message
	Access Token	JWT
Failure Response Body	Message	Failure message

Figure 6.4: Login Response

6.6.3 Logout

Request

Attribute	Detail
Protocol	HTTPS
Method	POST
URI	/api/auth/logout
Authorization	Bearer <JWT>

Figure 6.5: Logout Request

Response

Attribute	Detail	
Protocol	HTTPS	
Success Code	200 OK	
Failure Code	400 Bad Request or 401 Unauthorized or 500 Internal Server Error	
Success Response Body	Message	Success message
Failure Response Body	Message	Failure message

Figure 6.6: Logout Response

6.6.4 Profile Update

Request

Attribute	Detail	
Protocol	HTTPS	
Method	PATCH	
URI	/api/auth/profile	
Request body	Name (Optional)	User name
	ID (Optional)	User ID
	Password (Optional)	User password
	Email (Optional)	User email
	Phone Number (Optional)	User phone number
	Interest (Optional)	User Interest
Content Type	application/json	
Authorization	Bearer <JWT>	

Figure 6.7: Profile Update Request

Response

Attribute	Detail	
Protocol	HTTPS	
Success Code	200 OK	
Failure Code	400 Bad Request or 401 Unauthorized or 409 Conflict or 500 Internal Server Error	
Success Response Body	Message	Success message
Failure Response Body	Message	Failure message

Figure 6.8: Profile Update Response

6.7 Problem Provision

6.7.1 Problem

Request

Attribute	Detail
Protocol	HTTPS
Method	GET
URI	/api/problems/{problem_id}
Authorization	Bearer <JWT>

Figure 6.9: Problem Request

Response

Attribute	Detail	
Protocol	HTTPS	
Success Code	200 OK	
Failure Code	400 Bad Request or 401 Unauthorized or 500 Internal Server Error	
Success Response Body	Message	Success message
	Problem	Problem information
Failure Response Body	Message	Failure message

Figure 6.10: Problem Response

6.7.2 Transformed Problem

Request

Attribute	Detail
Protocol	HTTPS
Method	GET
URI	/api/problems/{problem_id}/transformation
Authorization	Bearer <JWT>

Figure 6.11: Transformed Problem Request

Response

Attribute	Detail	
Protocol	HTTPS	
Success Code	200 OK or 201 Created	
Failure Code	400 Bad Request or 401 Unauthorized or 403 Forbidden or 500 Internal Server Error	
Success Response Body	Message	Success message
	Problem	Transformed problem information
Failure Response Body	Message	Failure message

Figure 6.12: Transformed Problem Response

6.7.3 Customized Problem

Request

Attribute	Detail	
Protocol	HTTPS	
Method	POST	
URI	/api/problems/customization	
Request Body	Algorithm	Algorithm chosen by the user
Content Type	application/json	
Authorization	Bearer <JWT>	

Figure 6.13: Customized Problem Request

Response

Attribute	Detail	
Protocol	HTTPS	
Success Code	201 Created	
Failure Code	400 Bad Request or 401 Unauthorized or 403 Forbidden or 500 Internal Server Error	
Success Response Body	Message	Success message
	Problem	Customized problem information
Failure Response Body	Message	Failure message

Figure 6.14: Customized Problem Response

6.8 Hint

Request

Attribute	Detail
Protocol	HTTPS
Method	GET
URI	/api/problems/{problem_id}/hint
Authorization	Bearer <JWT>

Figure 6.15: Hint Request

Response

Attribute	Detail	
Protocol	HTTPS	
Success Code	200 OK or 201 Created	
Failure Code	400 Bad Request or 401 Unauthorized or 403 Forbidden or 500 Internal Server Error	
Success Response Body	Message	Success message
	Hint	Hint for the problem
Failure Response Body	Message	Failure message

Figure 6.16: Hint Response

6.9 Judge

Request

Attribute	Detail	
Protocol	HTTPS	
Method	POST	
URI	/api/problems/{problem_id}/judgement	
Request body	Code	Code written by the user
Content Type	application/json	
Authorization	Bearer <JWT>	

Figure 6.17: Judge Request

Response

Attribute	Detail	
Protocol	HTTPS	
Success Code	201 Created	
Failure Code	400 Bad Request or 401 Unauthorized or 403 Forbidden or 500 Internal Server Error	
Success Response Body	Message	Success message
	Result	Result for the user code
Failure Response Body	Message	Failure message

Figure 6.18: Judge Response

6.10 Code Review

Request

Attribute	Detail	
Protocol	HTTPS	
Method	POST	
URI	/api/problems/{problem_id}/code-review	
Request body	Code	Code written by the user
Content Type	application/json	
Authorization	Bearer <JWT>	

Figure 6.19: Code Review Request

Response

Attribute	Detail	
Protocol	HTTPS	
Success Code	200 OK	
Failure Code	400 Bad Request or 401 Unauthorized or 403 Forbidden or 500 Internal Server Error	
Success Response Body	Message	Success message
	Feedback	Feedback for the user code
Failure Response Body	Message	Failure message

Figure 6.20: Code Review Response

7

Database Design

7.1 Objectives

7장에서는 GPTeacher 시스템의 데이터베이스 구조에 대해서 설명한다. 먼저 Entity Relationship Diagram을 통해 여러 Entity과 그들의 관계에 대해 식별한다. 그 후 실제 데이터베이스 구현을 위한 구체적인 Relational Schema를 제공한다.

7.2 Entity Relationship Diagram

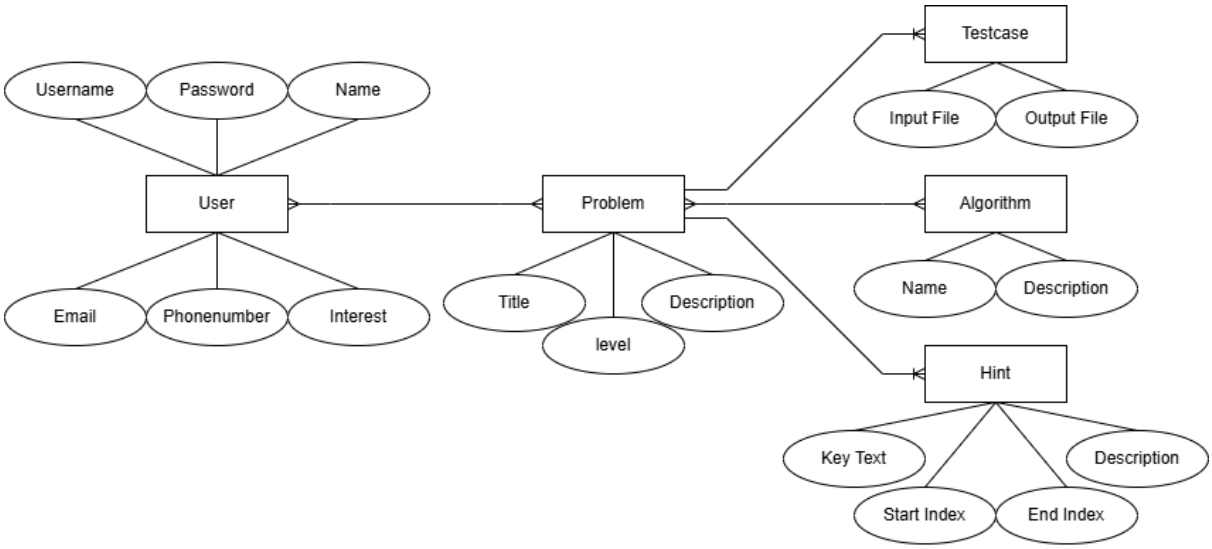

Figure 7.1: Entity Relationship Diagram

시스템은 User, Problem, Testcase, Algorithm, Hint 의 5 개의 entity 들로 이루어져있다. 각각의 entity 들의 속성, 다른 entity 와의 관계를 표시한다.

7.2.1 User Entity

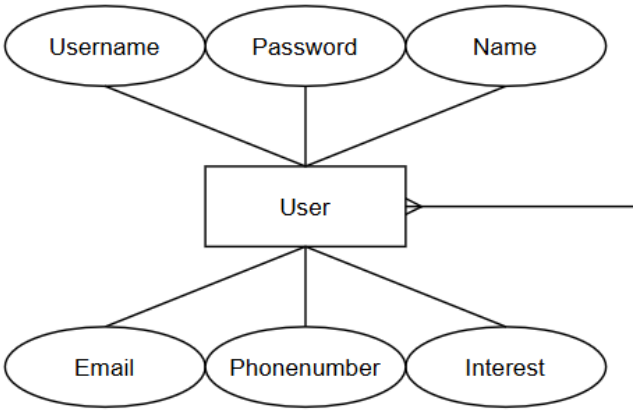

Figure 7.2: User Entity

User Entity 는 시스템 사용자이다. Username, Password, Name, Email, Phonenumber, Interest 의 속성을 가지며, 모두 사용자가 회원가입 시 입력하는 정보이다. Interest 의 경우에는 필수 입력사항은 아니다.
User Entity 와 Problem Entity 는 N:M 의 관계를 가진다. 한 명의 사용자는 다양한 문제를 풀 수 있고, 하나의 문제는 여러 사용자가 풀 수 있다.

7.2.2 Problem Entity

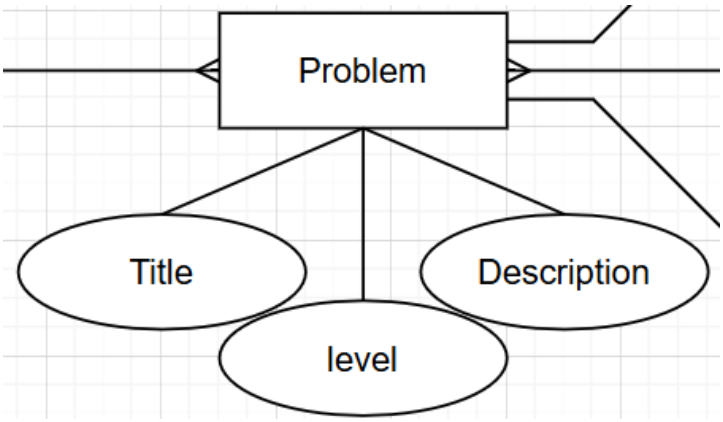

Figure 7.3: Problem Entity

Problem Entity 는 사용자에게 기본적으로 제공되는 문제 또는 사용자 맞춤형으로 생성 및 변형되는 문제이다. Title 에 문제의 제목, Description 에 문제의 설명, level 에 문제의 난이도가 저장된다.
User Entity 와 Problem Entity 는 N:M 의 관계를 가진다. 한 명의 사용자는 다양한 문제를 풀 수 있고, 하나의 문제는 여러 사용자가 풀 수 있다.
Problem Entity 와 Algorithm Entity 는 N:M 의 관계를 가진다. 하나의 문제는 여러

알고리즘으로 분류될 수 있고, 하나의 알고리즘은 여러 문제에 적용될 수 있다.
Problem Entity 와 Hint Entity 는 1:M 의 관계를 가진다. 하나의 문제에는 여러 개의 힌트가 있을 수 있지만, 하나의 힌트는 하나의 문제만을 위한 것이다.

7.2.3 Testcase Entity

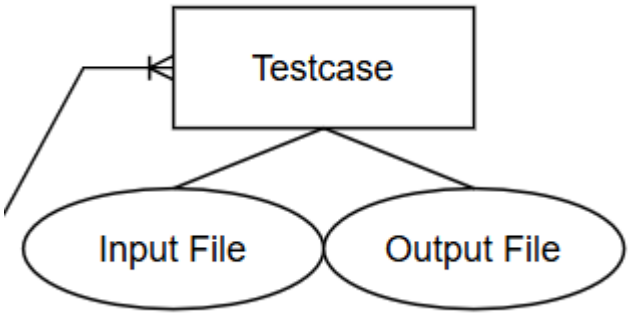

Figure 7.4: Testcase Entity

Testcase Entity 는 특정 문제에 대한 테스트케이스이다. Input File 과 Output File 로 이루어져 있다.
Problem Entity 와 Testcase 는 1:M 의 관계를 가진다. 하나의 문제는 여러 테스트케이스를 가질 수 있으며(테스트케이스의 조건이 나누어져 있는 경우), 하나의 테스트 케이스는 하나의 문제만을 위한 것이다.

7.2.4 Algorithm Entity

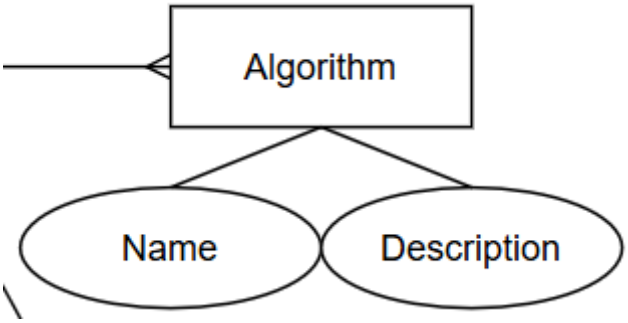

Figure 7.5: Algorithm Entity

Algorithm Entity 는 문제 해결을 위한 여러 방법이다. Name 에 알고리즘의 이름, Description 에 알고리즘의 설명이 저장된다.
Problem Entity 와 Algorithm Entity 는 N:M 의 관계를 가진다. 하나의 문제는 여러 알고리즘으로 분류될 수 있고, 하나의 알고리즘은 여러 문제에 적용될 수 있다.

7.2.5 Hint Entity

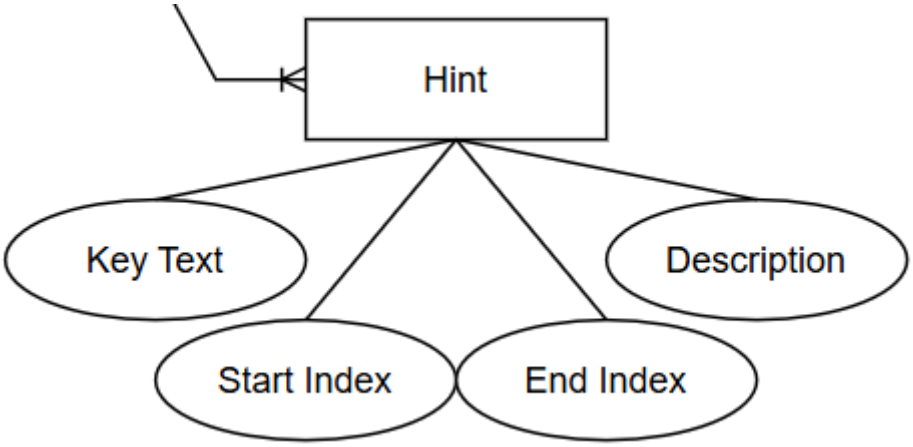

Figure 7.6: Hint Entity

Hint Entity 는 특정 문제에 대한 힌트이다. Key Text 에는 문제의 설명에서 뽑힌 핵심 키워드 또는 문장이 저장되며 Start Index 와 End Index 에는 Key Text 의 위치가 저장된다. Description 에는 힌트에 대한 설명이 저장된다.
Problem Entity 와 Hint Entity 는 1:M 의 관계를 가진다. 하나의 문제에는 여러 개의 힌트가 있을 수 있지만, 하나의 힌트는 하나의 문제만을 위한 것이다.

7.3 Relationship Schema

Figure 7.7: Relational Schema

전체적인 Relational Schema 이다.

7.3.1 User Table

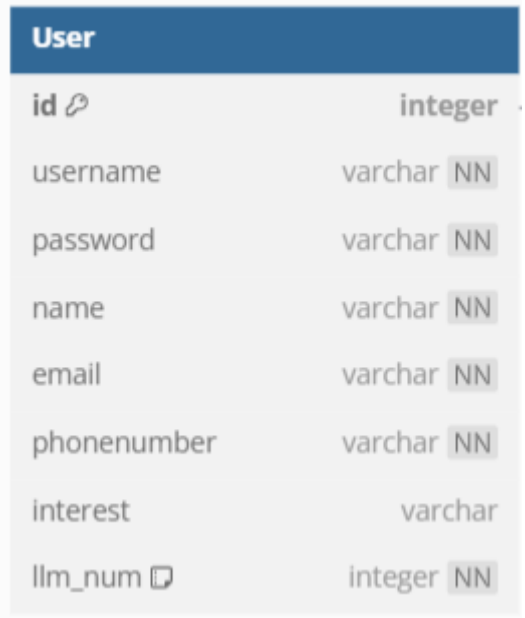

Figure 7.8: User Table

id를 고유식별번호로 가지며 primary key의 역할을 한다.
username은 로그인 시 아이디의 역할을 한다.
username, email, phonenumber은 유일한 필드이다.
password는 pbkdf2로 해싱하여 저장한다.
llm_num은 하루에 사용한 llm 기능의 횟수이며, 20이 되면 그 유저는 llm기능을 사용할 수 없다. 모든 유저의 llm_num은 자정에 0으로 초기화된다.

7.3.2 Problem Table

Figure 7.9: Problem Table

id를 고유식별번호로 가지며 primary key의 역할을 한다.
level은 난이도이다. 'easy', 'medium', 'hard' 3개의 경우가 있다.
transformed_num은 문제가 변형된 횟수이다. Default는 0이며 값이 5가 될 경우 그 문제의 변형은 더 이상 불가능하다.
isTransformed는 변형된 문제라면 True, 기본 문제라면 False이다.
parent_problem_id는 변형된 문제라면, 어떤 문제를 변형했는지 저장한다.

7.3.3 Testcase Table

Figure 7.10: Testcase Table

id를 고유식별번호로 가지며 Primary Key의 역할을 한다.
problem_id는 Problem Table의 id를 Foreign Key로 참조하며, Problem과 Testcase의 1:N 관계를 나타낸다.
input_filepath에는 input file의 경로가, output_filepath에는 output file의 경로가 저장된다.

7.3.4 Algorithm Table

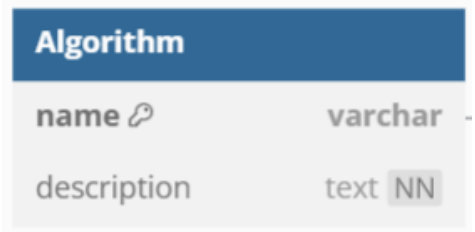

Figure 7.11: Algorithm Table

name은 Primary Key의 역할을 한다.

7.3.5 Hint Table

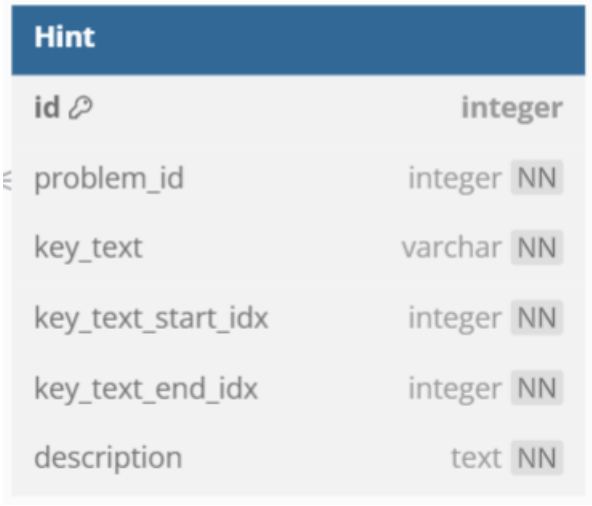

Figure 7.12: Hint Table

id를 고유식별번호로 가지며 primary key의 역할을 한다.
problem_id는 Problem Table의 id를 Foreign Key로 참조하며, Problem과 Hint의 1:N 관계를 나타낸다.

7.3.6 UserTransformedProblem Table

Figure 7.13: UserTransformedProblem Table

사용자의 문제 변형 횟수를 저장하는 테이블이다.
user_id와 problem_id는 복합키로 설정되어 Primary Key의 역할을 한다.
user_id는 User Table의 id를, problem_id는 Problem Table의 id를 Foreign Key로 참조하며 User와 Problem의 N:M 관계를 나타낸다.

7.3.7 UserSubmit Table

Figure 7.14: UserSubmit Table

사용자의 문제 풀이 제출을 저장하는 테이블이다.
id를 고유식별번호로 가지며 Primary Key의 역할을 한다.
user_id는 User Table의 id를, problem_id는 Problem Table의 id를 Foreign Key로 참조하며 User와 Problem의 N:M 관계를 나타낸다.
solved 필드에 사용자가 맞은 문제인지, 틀린 문제인지를 저장한다(맞은 문제인 경우 True, 틀린 문제인 경우 False).
code에는 사용자가 제출한 코드를 저장한다.
memory_usage와 execution_time에는 각각 메모리 사용량과 실행 시간을 저장하며, 맞았을 경우에만 저장한다.
timestamp에는 제출한 시각을 저장한다.

7.3.8 ProblemAlgorithm Table

Figure 7.15: ProblemAlgorithm Table

문제가 어떤 알고리즘으로 분류되는지를 저장하는 테이블이다.
problem_id와 algorithm_name은 복합키로 설정되어 Primary Key의 역할을 한다.
problem_id는 Problem Table의 id를, algorithm_name은 Algorithm Table의 name을 Foreign Key로 참조하며 Problem과 Algorithm의 N:M 관계를 나타낸다.

7.4 SQL Query for Creating Tables

7.4.1 User Table

```
CREATE TABLE User (
    id SERIAL PRIMARY KEY,
    username VARCHAR NOT NULL UNIQUE,
    password VARCHAR NOT NULL,
    name VARCHAR NOT NULL,
    email VARCHAR NOT NULL UNIQUE,
    phonenumber VARCHAR NOT NULL,
    interest VARCHAR,
    llm_num INTEGER NOT NULL DEFAULT 0
);
```

7.4.2 Problem Table

```
CREATE TABLE Problem (
   id SERIAL PRIMARY KEY,
   title VARCHAR NOT NULL,
   description TEXT NOT NULL,
   level VARCHAR NOT NULL,
   transformed_num INTEGER NOT NULL DEFAULT 0,
   isTransformed BOOLEAN NOT NULL,
   parent_problem_id INTEGER NOT NULL
);
```

7.4.3 Testcase Table

```
CREATE TABLE Testcase (
   id SERIAL PRIMARY KEY,
   problem_id INTEGER NOT NULL REFERENCES Problem(id) ON DELETE CASCADE,
   input_filepath VARCHAR NOT NULL UNIQUE,
   output_filepath VARCHAR NOT NULL UNIQUE
);
```

7.4.4 Algorithm Table

```
CREATE TABLE Algorithm (
   name VARCHAR PRIMARY KEY,
   description TEXT NOT NULL
);
```

7.4.5 Hint Table

```
CREATE TABLE Hint (
    id SERIAL PRIMARY KEY,
    problem_id INTEGER NOT NULL REFERENCES Problem(id) ON DELETE CASCADE,
    key_text VARCHAR NOT NULL,
    key_text_start_idx INTEGER NOT NULL,
    key_text_end_idx INTEGER NOT NULL,
    description TEXT NOT NULL
);
```

7.4.6 UserTransformedProblem Table

```
CREATE TABLE UserTransformedProblem (
    user_id INTEGER NOT NULL,
    problem_id INTEGER NOT NULL,
    num INTEGER NOT NULL DEFAULT 0,
    PRIMARY KEY (user_id, problem_id),
    FOREIGN KEY (user_id) REFERENCES User(id) ON DELETE CASCADE,
    FOREIGN KEY (problem_id) REFERENCES Problem(id) ON DELETE CASCADE
);
```

7.4.7 UserSubmit Table

```
CREATE TABLE UserSubmit (
    id SERIAL PRIMARY KEY,
    user_id INTEGER NOT NULL REFERENCES User(id) ON DELETE CASCADE,
    problem_id INTEGER NOT NULL REFERENCES Problem(id) ON DELETE CASCADE,
    solved BOOLEAN NOT NULL,
```

```
    code TEXT NOT NULL,
    memory_usage INTEGER,
    execution_time INTEGER,
    timestamp TIMESTAMP NOT NULL
);
```

7.4.8 ProblemAlgorithm Table

```
CREATE TABLE ProblemAlgorithm (
    problem_id INTEGER NOT NULL,
    algorithm_name VARCHAR NOT NULL,
    PRIMARY KEY (problem_id, algorithm_name),
    FOREIGN KEY (problem_id) REFERENCES Problem(id) ON DELETE CASCADE,
    FOREIGN KEY (algorithm_name) REFERENCES Algorithm(name) ON DELETE CASCADE
);
```

8

Testing Plan

8.1 Objectives

본 Section에서는 Testing policy을 설명하고, 그에 따른 Test Case 및 Pseudocode를 서술하고자 한다. 이러한 Test를 통하여 개발 과정에서 defect를 발견하고, 개발된 System이 실제로 System Requirement를 충족하고 있는지 검사하는데 목적이 있다.

8.2 Testing Policy

8.2.1 Development Testing

Development Testing은 개발자가 작성한 code에 defect가 존재하는지 최대한 확인하고 fix 하기 위해 수행한다. 개발 단계에서 개발자가 작성한 code가 의도에 따라 input에 따른 output을 산출하는지 확인한다. 개발은 CI/CD로 진행하며, 테스트는 개발 팀 내부적으로 자동화된 Test Code를 통해 진행하거나 PDB debugger를 활용해 진행한다. 먼저 unit testing을 진행하고, Component testing을 아래 사항에 대해 진행하며, 이후 system testing을 진행한다. 위 3가지 Testing은 필요에 따라 inter-leaved 될 수 있다.

8.2.1.1 System Component

회원 가입, log-in/logout, 프로필 정보 수정, 단계별 Hint 제공, 변형 문제 제공, 맞춤형 문제 제공–관심 분야 관련 문제, Code feedback

8.2.2 Release Testing

Release Testing은 개발된 Software System이 원활하게 release될 수 있는지 확인하기위해 수행하며, 고객의 환경에서 System이 원활히 동작하는지 확인한다. 서비스에 대한 functional requirement 뿐만 아니라 Performance, Reliability, Security, Dependability 등을 검사한다. Test는 개발 팀 외부적으로 진행하며, 이를 위해 교내에서 테스터를 모집해 다양한 환경에서 System이 잘 release될 수 있는지 아래 항목을 검사할 계획이다.

8.2.2.1 Performance

시스템은 사용자의 회원가입, 로그인, 로그아웃 요청을 1초 이내에 처리한다. 또한, 사용자가 문제를 조회할 때는 5초 이내로 처리되며, 문제 풀이 제출과 기록 저장도 5초 이내에 완료된다. 제출한 소스코드에 대한 채점은 1분 이내로 처리하고, 사용자가 요청한 힌트 제공은 10초 이내, 변형 문제 제공은 20초 이내에 이루어진다. 코드 리뷰와 최적화 요청 역시 10초 이내에 처리된다. 시스템은 초당 최대 500개의 트랜잭션을 처리한다.

8.2.2.2 Reliability

데이터베이스에 저장되는 모든 고객의 학습 데이터는 무결성을 유지하며, 하루 단위로 백업한다. 모든 서비스는 웹사이트에서 충돌 없이 동작해야 하며, 오류가 발생할 경우 자동으로 복구된다. 오류로부터 복구되는 시간은 오류의 종류에 따라 최대 1~5분 이내로 한다. 서비스 배포 시 자동화된 정기적 Back-to-back test를 수행하여 모든 기능이 정상적으로 동작함을 항상 확인한다. 시스템 하드웨어의 사용량을 모니터링하고, 일정 사용량을 넘지 않도록 서비스 사용 부하를 조절한다.

8.2.2.3 Availability

시스템은 사용자가 필요할 때 항상 접근 가능하고 안정적으로 작동해야 하며, 24시간 언제나 사용 가능해야 한다. 장애가 발생하면 복구시간은 5분 이내로 한다. 일반적인 유지보수 및 업데이트가 필요할 경우, 시스템을 최대한 중단하지 않도록 해야 한다. 시스템은 최대 1만 명의 사용자의 정보를 저장해야 할 수 있어야 한다. 시스템은 최대 3만 개의 알고리즘 문제를 제공해야 한다. 시스템은 최대 1000명의 사용자의 동시 접속을 지원하고 요청을 처리해야 한다. 이를 위해 서버는 시스템의 안정적인 운영을 위해 2.0GHz, 4 코어 이상의 CPU, 32GB 이상의 메모리, 5Gbps 이상의 대역폭, 100GB 이상의 데이터베이스 용량이 필요하다. 또한 사용자는 시스템을 안정적으로 이용하기 위해 웹 브라우저를 실행할 수 있는 환경, 1.0GHz, 1 코어 이상의 CPU, 4GB 이상의 메모리, 1Mbps 이상의 대역폭을 갖추고 있어야 한다.

8.2.2.4 Security

모든 데이터의 전송은 HTTPS를 통해 암호화한다. 사용자는 회원가입 시 핸드폰 번호 인증과 이메일 인증을 진행해야 하며, 비밀번호 설정 시 최소 길이와 영문, 숫자, 특수문자를 포함해야 한다. 사용자는 로그인 후에만 시스템의 서비스를 이용할 수 있으며, 다른 사용자의 정보를 열람할 수 없다. 비공개로 설정된 소스코드는 열람할 수 없으며, 1분에 최대 5회까지만 소스코드를 제출할 수 있다. 사용자는 시스템 데이터베이스에 직접적으로 접근할 수 없다.

8.2.2.5 Maintainability

데이터 테이블 간 고유키는 한 개만 존재해야 하며, 고유키들은 외래키로 연결되어 있어야 한다. 중복되거나 상이한 데이터는 존재해서는 안 되며, 데이터는 3NF를 유지하여 모듈성을 높인다. 문제 풀이, 피드백 제공, 키워드 제공 등의 주요 기능은 별도의 소프트웨어로 만들어 관리하고, 이들의 기능은 결합되지 않도록 한다. 각 소프트웨어 간의 인터페이스와 데이터베이스와 소프트웨어 간의 인터페이스는 모두 정의되어 명세화 되어야 한다. 데이터 접근 방식과 소프트웨어에 사용되는 알고리즘은 nlog(n) 미만의 시간 복잡도를 가져야 하며, scalability를 만족해야 한다. 코드 구조는 객체지향 방식으로 일반화되어야 하며, 코드는 Sub-system별로 별도의 파일로 관리한다.

8.2.3 User Testing

User Testing은 Client가 실제로 소프트웨어를 사용하면서 System의 기능들이 원활하고 정확하게 동작하는 것을 보장하기 위하여 수행한다. User가 System의 기능들을 사용하였을 때 기능적, 비기능적 Requirement가 잘 충족되는지 확인하고, 잘 충족되지 않는 부분에 대한 feedback을 받아 User Experience를 개선해 System의 Quality를 상승시킨다.

8.2.4 Scenario testing

테스트 케이스는 scenario testing 방식으로 디자인한다. 아래는 Testing target이 되는 scenario를 구체적으로 서술한 것이다. 고객이 사용할 시스템의 주요 기능과 interaction에 대한 testing을 정상/비정상 case를 나누어 수행하며, output이 예측한 output과 일치하는지, 실행 시간이 허용 이내인지 검사한다.

8.2.4.1 회원 가입

회원 가입 기능은 등록되지 않은 사용자가 시스템에 처음 접속하여 필요한 정보를 입력하고, 본인 인증 절차를 완료한 후 회원가입을 마무리하는 프로세스를 테스트한다. 사용자가 이름, 핸드폰 번호, 이메일, 아이디, 비밀번호, 관심 분야를 입력하고, 해당 정보가 중복되지 않으며, 형식에 맞게 입력되었는지 확인한다. 본인 인증 절차가 정상적으로 진행되며, 최종적으로 사용자가 회원가입을 완료할 수 있어야 한다. 형식에 맞지 않는 정보를 입력했을 경우 "형식에 맞지 않습니다. 다시 입력해주세요." 라는 메시지를 출력한다. 이미 등록된 사용자의 경우 "이미 등록된 사용자입니다. 로그인 페이지로 돌아갑니다."를 출력한다.

8.2.4.2 Log-In/Logout

로그인과 로그아웃 기능을 테스트하여 사용자가 올바른 아이디와 비밀번호로 로그인하

고, 시스템에 접근할 수 있는지 확인한다. 로그인에 성공하면 "로그인 성공"을 출력한다. 로그인 시 입력한 정보가 정확하지 않으면 "입력한 정보가 올바르지 않습니다. 다시 한 번 확인하여 주세요." 라는 오류 메시지가 표시되고, 사용자는 로그인 페이지에 머물러야 한다. 로그인 후 사용자가 시스템에서 로그아웃을 할 때 정상적으로 로그아웃이 이루어져야 하며, 로그아웃 완료 시 "로그아웃 완료"를 출력한다. 로그아웃 후에는 사용자가 시스템에 다시 접근할 수 없고, 로그인 페이지로 Re-Direction된다.

8.2.4.3 문제 제출

사용자가 문제 제출 버튼을 클릭해 푼 문제를 제출한다. 제출된 코드는 사용자의 데이터베이스에 저장된다. 정상적으로 제출된 경우 "제출 완료" 메시지가 표시되고, 코드가 비어 있어 정상적으로 제출되지 않은 경우 "코드를 먼저 작성해 주세요." 메시지가 표시되어야 한다.

8.2.4.4 프로필 정보 수정

프로필 정보 수정 기능을 테스트하여 사용자가 자신의 개인정보를 수정할 수 있는지 확인한다. 사용자는 프로필에서 이름, 이메일, 핸드폰 번호, 관심 분야 등을 수정할 수 있어야 하며, 수정된 정보가 시스템에 반영되어야 한다. 수정 후, 변경 사항이 저장되고, 사용자가 다시 시스템에 로그인할 때 수정된 정보가 정상적으로 나타나야 한다.

8.2.4.5 단계별 Hint 제공

단계별 Hint 제공 기능을 테스트하여 사용자가 문제 풀이 중에 힌트를 요청하고, 단계별로 제공되는 힌트가 문제 해결에 도움이 되는지 확인한다. 사용자가 1단계 힌트를 요청하면 문제의 주요 키워드나 문장이 강조되고, 추가적인 힌트를 요청할 경우, 점차적으로 더 자세한 설명이 제공되어야 한다. 세 번째 힌트까지 제공되며, 그 이후에는 "더 이상 제공될 수 있는 힌트가 없습니다. 관련된 내용을 다시 학습하세요." 라는 메세지를 학습자에게 표시하여 추가 힌트를 제공하지 않음을 사용자에게 알리고 관련된 자료구조 및 알고리즘에 관련된 강의 링크를 제공한다.

8.2.4.6 변형 문제 제공

변형 문제 제공 기능을 테스트하여 사용자가 문제 풀이 후 부족한 부분을 보충하기 위해 변형 문제를 요청할 수 있는지 확인한다. 사용자가 문제 풀이를 완료하고 변형 문제 요청 버튼을 누르면, 시스템은 기존 문제를 바탕으로 새로운 맞춤형 문제를 제공해야 한다. 학습자가 문제 풀이를 완료하지 않고 버튼을 누르는 경우 "아직 문제 풀이를 완료하지 않았습니다. 문제 풀이를 완료한 후, 요청하여 주시기 바랍니다."라는 메세지를 학습자에

게 표시한다. 5번 이상 변형 문제를 요청했을 경우, "더 이상 변형할 수 없습니다. 다른 문제를 시도해보세요."라는 메세지를 학습자에게 표시하고, 다른 문제를 시도하도록 유도한다.

8.2.4.7 맞춤형 문제 제공–관심 분야 관련 문제

맞춤형 문제 제공 기능을 테스트하여 사용자가 등록한 관심 분야를 기반으로 맞춤형 문제를 제공받을 수 있는지 확인한다. 시스템은 학습자에게 알고리즘 관련하여 선호하는 주제를 선택하거나 입력하도록 요청한다. 사용자가 관심 알고리즘을 선택 또는 입력하고, 프로필에 있는 관심 분야를 바탕으로 맞춤형 문제를 요청하면, 시스템은 맞춤형 문제를 생성하여 제공한다. 사용자가 관심 분야가 등록되어 있지 않으면, "학습자의 관심 분야가 등록되어 있지 않습니다. 먼저 관심 분야를 선택하여 등록해주시기 바랍니다."라는 메시지를 학습자에게 표시한다.

8.2.5 Test Case & Pseudocode

위에 서술한 각 scenario에 대한 Test Case & Pseudocode를 서술한다.

각 scenario에 대한 Performance Test는 아래의 pseudocode로 진행한다.

Algorithm 1 Check Execution Time

```
Input: start_time, max_time, function_name
elapsed_time ← current_time() - start_time
if elapsed_time >max_time then
    Print "[경고] function_name 함수가 max_time초 내에 완료되지 않았습니다. (소요 시간: elapsed_time:.2f초)"
end if
Return elapsed_time
```

Figure 8.1: Performance Test Pseudocode

8.2.5.1 회원 가입

8.2.5.1.1 Test Case

Description: 사용자가 자신의 정보를 바탕으로 본 서비스에 회원 가입한다.

Pre-condition: 사용자가 시스템에 등록되지 않은 상태이다.

<CASE 1 (정상 가입)>

Input: name="이승민", phone_Number="123-4567-8910", email="lsm&gmail.com",

id="lsm", password="12341234", domain_Interested = ["Movie", "Actor"]

Steps: 회원가입 페이지로 이동 -> 미등록 확인 -> 모든 정보를 위와 같이 입력 -> 입력된 전화번호로 본인 인증을 진행 -> 완료 버튼 클릭 -> 사용자 정보를 데이터베이스에 저장.

Output: "회원가입이 완료되었습니다."

<CASE 2 (전화번호 형식 오류)>

Input: name="이승민", phone_Number="123-가나다라-8910", email="lsm&gmail.com", id="lsm", password="12341234", domain_Interested = ["Movie", "Actor"]

Steps: 회원가입 페이지로 이동 -> 미등록 확인 -> 모든 정보를 위와 같이 입력 -> 오류 메시지 출력

Output: "형식에 맞지 않습니다. 다시 입력해주세요."

8.2.5.1.2 Pseudocode

Algorithm 2 User Registration

```
Input: name, phone_number, email, id, password, domain_interested
if is_registered(id) is false then
    if is_valid_format(phone_number, email, password) then
        send_verification(phone_number)
        save_user_to_db(name, phone_number, email, id, password, domain_interested)
        Print "회원가입이 완료되었습니다."
    else
        Print "형식에 맞지 않습니다. 다시 입력해주세요."
    end if
else
    Print "이미 등록된 사용자입니다. 로그인 페이지로 돌아갑니다."
end if
check_execution_time(start_time, 1, "register")
```

Figure 8.2: User Registration Pseudocode

8.2.5.2 Log-In/Logout

8.2.5.2.1 Test Case

Description: 사용자가 서비스에 Log-In 하고 Log-Out 한다.

Pre-condition: Log-In을 위해서 사용자가 서비스에 가입되어 있어야 한다.

<CASE 1 (정상 로그인)>

Input: id="lsm", password="12341234"

Steps: 로그인 화면에서 id, password 입력 -> 등록 확인 -> 비밀번호 확인 -> Log-In 성공

Output: "로그인 성공"

<CASE 2 (미등록 로그인)>

Input: id="lsm", password="12341234444444"

Steps: 로그인 화면에서 id, password 입력 -> 미등록 확인 -> 오류 메시지 출력 -> 로그인 페이지로 Re-direction

Output: "입력한 정보가 올바르지 않습니다. 다시 한번 확인하여 주세요."

<CASE 3 (정상 로그아웃)>

Input: id="lsm"

Steps: 로그아웃 버튼 클릭 -> 로그인 확인 -> 로그아웃 -> 로그인 페이지로 Re-direction

Output: "로그아웃 완료"

8.2.5.2.2 Pseudocode

Algorithm 3 User Login

```
1: Input: id, password
2: if is_registered(id) then
3:     if check_password(id, password) then
4:         Print "로그인 성공"
5:     else
6:         Print "입력한 정보가 올바르지 않습니다. 다시 한번 확인하여 주세요."
7:         redirect_to_login()
8:     end if
9: else
10:    Print "입력한 정보가 올바르지 않습니다. 다시 한번 확인하여 주세요."
11:    redirect_to_login()
12: end if
13: check_execution_time(start_time, 1, "login")
```

Figure 8.3: User Login Pseudocode

Algorithm 4 User Logout

```
1: if is_logged_in(id) then
2:     Print "로그아웃 완료"
3:     redirect_to_login()
4: end if
5: check_execution_time(start_time, 1, "logout")
```

Figure 8.4: User Logout Pseudocode

8.2.5.3 프로필 정보 수정

8.2.5.3.1 Test Case

Description: 사용자가 자신의 프로필 정보(특히, 관심사)를 수정한다.

Pre-condition: 사용자가 Log-In 된 상태여야 한다.

<CASE 1 (관심 분야 업데이트)>

Input: id="lsm", domain_interested = "Economics"

Steps: 프로필 버튼 클릭 -> 관심사 수정 버튼 클릭 -> 로그인 확인 -> 새로운 관심사 입력 -> 정보 업데이트

Output: "정보가 업데이트 되었습니다."

<CASE 2 (관심 분야 형식 오류)>

Input: : id="lsm", domain_interested = 111%@!@#

Steps: 프로필 버튼 클릭 -> 관심사 수정 버튼 클릭 -> 로그인 확인 -> 새로운 관심사 입력 -> 형식 오류 확인 -> 오류 메시지 출력

Output: "형식에 맞지 않습니다. 다시 입력해주세요."

8.2.5.3.2 Pseudocode

Algorithm 5 Update Profile

```
1: Input: id, domain_interested
2: if is_logged_in(id) then
3:     if is_valid_format(domain_interested) then
4:         update_user_interest(id, domain_interested)
5:         Print "정보가 업데이트되었습니다."
6:     else
7:         Print "형식에 맞지 않습니다. 다시 입력해주세요."
8:     end if
9: else
10:    Print "로그인 후에만 프로필을 수정할 수 있습니다."
11: end if
12: check_execution_time(start_time, 1, "update profile")
```

Figure 8.5: Update Profile Pseudocode

8.2.5.4 문제 제출

8.2.5.4.1 Test Case

Description: 사용자가 문제 풀이 후 작성한 코드를 제출한다.

Pre-condition: 작성한 코드는 비어 있지 않아야 한다.

<CASE 1 (코드 제출)>

Input: id="lsm", problem_id = 1, code = "print("hello_world ₩n")"

Steps: 제출 버튼 클릭 -> 제출 코드가 비어 있는지 확인 -> 서버에 제출 -> 제출된 코드 저장

Output: "제출 완료"

<CASE 2 (NULL 코드 제출)>

Input: id="lsm", problem_id = 1, code = NULL

Steps: 제출 버튼 클릭 -> 제출 코드 NULL 확인 -> 오류 메시지 출력

Output: "코드를 먼저 작성해 주세요."

8.2.5.4.2 Pseudocode

Algorithm 6 Submit Code Process

```
1: procedure SubmitCode(id, problem_id, code)
2:     if code is NULL or empty then
3:         print "코드를 먼저 작성해 주세요."
4:         return
5:     end if
6:     response ← SendToServer(user_id=id, problem_id=problem_id,
   code=code) Exception e
7:     print "제출에 실패했습니다."
8:     return
9:     if response.status = "success" then
10:        print "제출 완료"
11:    else
12:        print "제출에 실패했습니다."
13:    end if
14: end procedure
```

Figure 8.6: Submit Code Process Pseudocode

8.2.5.5 단계별 Hint 제공

8.2.5.5.1 Test Case

Description: 사용자가 단계적으로 단계별 Hint 기능을 사용한다.

Pre-condition: 사용자가 같은 문제에 대해 문제 힌트 기능을 3회 이하 사용한 상태이다. 사용자가 LLM 생성 기능을 하루 20회 이하 사용한 상태이다.

<CASE 1 (2단계 요청)>

Input: id="lsm", problem_id = 1

Steps: 힌트 요청 버튼 클릭 -> LLM 횟수제한 검사 -> 힌트 단계 (2단계) 식별 -> 힌트 단계 및 문제 정보를 LLM에 전달하고, 해당 힌트 생성 요청 -> 생성된 힌트를 받아 사용자에게 출력

Output: 생성된 2단계 힌트. Ex) "수의 범위가 명확하게 정의되어 있어, 배열이나 해시맵을 사용해 효율적으로 처리할 수 있습니다."

<CASE 2 (4회 이상 요청)>

Input: id="lsm", problem_id = 1

Steps: 힌트 요청 버튼 클릭 -> LLM 횟수제한 검사 -> 힌트 단계 범위 (1~3단계) 초과 -> 오류 메시지 출력 -> 관련 강의 및 자료 링크 탐색 -> 해당 링크 출력

Output: "더 이상 제공될 수 있는 힌트가 없습니다. 관련된 내용을 다시 학습하세요.", 관련 링크. Ex) www.xxxx.xxxx

<CASE 3 (LLM 기능 20회 이상 요청)>

Input: id="lsm", problem_id = 1

Steps: 힌트 요청 버튼 클릭 -> LLM 횟수제한 검사 -> 오류 메시지 출력

Output: "오늘 LLM 생성 기능을 최대(20회)로 사용하셨습니다. 내일 다시 시도해 주세요."

8.2.5.5.2 Pseudocode

Algorithm 7 Request Hint

```
1: Input: id, problem_id
2: llm_usage ← get_llm_usage(id)
3: if llm_usage >= 20 then
4:     Print "오늘 LLM 생성 기능을 최대(20회)로 사용하셨습니다. 내일 다시 시도해 주세요."
5:     Return
6: end if
7: current_step ← calculate_hint_step(id, problem_id)
8: if current_step ≤ 3 then
9:     hint ← generate_hint(problem_id, current_step)
10:    Return hint
11: else
12:    link ← search_related_link(problem_id)
13:    Print "더 이상 제공될 수 있는 힌트가 없습니다. 관련된 내용을 다시 학습하세요."
14:    Return link
15: end if
16: check_execution_time(start_time, 1, "request hint")
```

Figure 8.7: Request Hint Pseudocode

8.2.5.6 변형 문제 제공

8.2.5.6.1 Test Case

Description: 사용자가 문제에 대한 변형 기능을 사용한다.

Pre-condition: 사용자가 해당 문제를 이미 풀었고, 같은 문제에 대해 변형 문제 제공 기능을 5회 이하 요청했다. 사용자가 LLM 생성 기능을 하루 20회 이하 사용한 상태이다.

<CASE 1 (변형 문제 생성)>

Input: id="lsm", problem_id = 1

Steps: 사용자가 변형 문제 생성 버튼 클릭 -> LLM 횟수제한 검사 -> 문제 풀이 완료 인식 -> 5회 이하 요청 인식 -> LLM에 해당 문제 정보 및 변형 요청 전달 -> LLM이 생성한 변형 문제를 사용자에게 출력하고, 해당 문제의 테스트 코드 및 케이스를 호스트

서버에 저장

Output: 생성된 변형 문제, 해당 문제의 테스트 코드 및 테스트 케이스

Ex)

-원래 문제

```
def add_numbers(a, b):
        """두 수를 더한 결과를 반환합니다."""
        return
```

-변형 문제

```
def multiply_numbers(a, b):
        """두 수를 곱한 결과를 반환합니다."""
        return
```

-테스트 코드 및 케이스

```
assert multiply_numbers(2, 3) == 6
assert multiply_numbers(0, 5) == 0
assert multiply_numbers(-1, 4) == -4
```

<CASE 2 (사용자가 문제 풀이 미완료)>

Input: id="lsm", problem_id = 1

Steps: 사용자가 변형 문제 생성 버튼 클릭 -> LLM 횟수제한 검사 -> 문제 풀이 미완료 인식 -> 오류 메시지 출력

Output: "아직 문제 풀이를 완료하지 않았습니다. 문제 풀이를 완료한 후, 요청하여 주시기 바랍니다."

<CASE 3 (변형 문제 5회 초과 요청)>

Input: id="lsm", problem_id = 1

Steps: 사용자가 변형 문제 생성 버튼 클릭 -> LLM 횟수제한 검사 -> 문제 풀이 완료 인식 -> 5회 초과 요청 인식 -> 오류 메시지 출력

Output: "더 이상 변형할 수 없습니다. 다른 문제를 시도해보세요."

<CASE 4 (LLM 기능 20회 이상 요청)>

Input: id="lsm", problem_id = 1

Steps: 사용자가 변형 문제 생성 버튼 클릭 -> LLM 횟수제한 검사 -> 오류 메시지 출력

Output: "오늘 LLM 생성 기능을 최대(20회)로 사용하셨습니다. 내일 다시 시도해 주세요."

8.2.5.6.2 Pseudocode

Algorithm 8 Generate Modified Problem

```
Input: id, problem_id
llm_usage ← get_llm_usage(id)
if llm_usage >= 20 then
    Print "오늘 LLM 생성 기능을 최대(20회)로 사용하셨습니다. 내일 다시 시도해 주세요."
    Return
end if
request_count ← calculate_request_count(id, problem_id)
if request_count ≤ 5 then
    if is_problem_solved(id, problem_id) then
        modified_problem ← create_modified_version(problem_id)
        save_modified_problem(modified_problem)
        Return modified_problem
    else
        Print "아직 문제 풀이를 완료하지 않았습니다. 문제 풀이를 완료한 후, 요청하여 주시기 바랍니다."
    end if
else
    Print "더 이상 변형할 수 없습니다. 다른 문제를 시도해보세요."
end if
check_execution_time(start_time, 1, "generate modified problem")
```

Figure 8.8: Generate Modified Problem Pseudocode

8.2.5.7 맞춤형 문제 제공–관심 분야 관련 문제

8.2.5.7.1 Test Case

Description: 사용자가 맞춤형 종합 문제 제공 기능을 사용한다.

Pre-condition: 사용자가 Log-In 되어 있고, LLM과 호스트 서버가 연결되어 있는 상태이며, 학습자의 관심 분야가 학습자의 개인정보에 등록되어 있어야 한다. 사용자가 LLM 생성 기능을 하루 20회 이하 사용한 상태이다.

<CASE 1 (정상 생성)>

Input: id="lsm", want_alg = ["파일_입출력"]

Steps: 사용자가 맞춤형 문제 생성 버튼 클릭 -> LLM 횟수제한 검사 -> 로그인 확인 -> 관심 분야 등록 확인 -> 사용자에게 정의된 여러 알고리즘들 중 선택 요청 -> 사용자가 관심 알고리즘을 선택하고, 이 외에 관심있는 알고리즘을 추가적으로 입력 -> 해당 정보들과 학습자의 관심 분야를 바탕으로 LLM에게 맞춤형 문제 및 테스트코드 & 테스트케이스 생성 요청 -> 생성된 문제를 사용자에게 출력 -> 호스트 서버에 테스트코드 & 테스트 케이스 저장

Output: 생성된 맞춤형 문제, 해당 문제의 테스트 코드 및 테스트 케이스

Ex) 관심 분야가 영화인 경우

- 맞춤형 문제

```
import json

def print_movie_titles(filename):
    """영화 목록 파일에서 제목을 출력하는 함수"""
    return
```

-테스트 코드

```
def test_print_movie_titles():
    # 테스트용 영화 목록을 movies.json 파일로 저장
    movies = [ {'title': '인터스텔라', 'year': 2014}, {'title': '어벤져스: 엔드게임', 'year': 2019}, {'title': '기생충', 'year': 2019} ]
    with open('movies.json', 'w', encoding='utf-8') as file:
        json.dump(movies, file, ensure_ascii=False, indent=4)
```

```
# 출력 결과를 확인하기 위한 테스트
import io
import sys
captured_output = io.StringIO()
sys.stdout = captured_output
print_movie_titles('movies.json')
# 원래 출력으로 돌려놓기
sys.stdout = sys.__stdout__
# 예상된 출력과 비교
expected_output = "인터스텔라₩n어벤져스: 엔드게임₩n기생충₩n"
assert captured_output.getvalue() == expected_output
```

-테스트 케이스

INPUT:

movies.json 파일

OUTPUT:

인터스텔라

어벤져스: 엔드게임

기생충

<CASE 2 (관심 분야 미등록)>

Input: id="lsm", want_alg = ["파일_입출력"]

Steps: 사용자가 맞춤형 문제 생성 버튼 클릭 -> LLM 횟수제한 검사 -> 로그인 확인 -> 관심 분야 미등록 확인 -> 오류 메시지 출력

Output: "학습자의 관심 분야가 등록되어 있지 않습니다. 먼저 관심 분야를 선택하여 등록해주시기 바랍니다."

<CASE 3 (LLM 기능 20회 이상 요청)>

Input: id="lsm", want_alg = ["파일_입출력"]

Steps: 사용자가 맞춤형 문제 생성 버튼 클릭 -> LLM 횟수제한 검사 -> 로그인 확인 -> 오류 메시지 출력

Output: "오늘 LLM 생성 기능을 최대(20회)로 사용하셨습니다. 내일 다시 시도해 주세요."

8.2.5.7.2 Pseudocode

Algorithm 9 Generate Custom Problem

```
Input: id, want_alg
llm_usage ← get_llm_usage(id)
if llm_usage >= 20 then
    Print "오늘 LLM 생성 기능을 최대(20회)로 사용하셨습니다. 내일 다시 시도해 주세요."
    Return
end if
if is_logged_in(id) then
    if is_domain_registered(id) then
        domain_interested ← find_client_domain_interested(id)
        want_alg.add(choose_want_alg(id))
        custom_problem ← create_custom_problem(domain_interested, want_alg)
        save_custom_problem(custom_problem)
        Return custom_problem
    else
        Print "학습자의 관심 분야가 등록되어 있지 않습니다. 먼저 관심 분야를 선택하여 등록해주시기 바랍니다."
    end if
else
    Print "로그인 후에만 맞춤형 문제를 제공받을 수 있습니다."
end if
check_execution_time(start_time, 1, "generate custom problem")
```

Figure 8.9: Generate Custom Problem Pseudocode

9

Development Plan

9.1 Objectives

9장에서는 GPTeacher 시스템의 개발 환경 및 도구, 제약사항, 전제 조건 및 의존성에 대해 설명한다.

9.2 Frontend

9.2.1 React

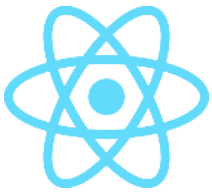

Figure 9.1: React Logo

React는 프론트엔드 개발을 위한 자바스크립트 라이브러리이다. Single Page Application(SPA)으로 개발하여 UI를 빠르고 효율적으로 렌더링할 수 있으며 이를 통해 사용자에게 문제, 힌트, 피드백을 빠르게 제공할 수 있다. 또한 컴포넌트 기반 아키텍처를 통해 여러 컴포넌트들을 개발하고 재사용할 수 있다.

9.2.2 HTML(HyperText Markup Language)

Figure 9.2: HTML Logo

HTML은 웹 페이지의 구조와 내용을 정의하는 마크업 언어이다. 제목, 문단, 이미지, 링크, 버튼 등 웹 페이지의 구성 요소들을 정의하기 위해 사용한다.

9.2.3 CSS(Cascading Style Sheet)

Figure 9.3: CSS Logo

CSS는 HTML과 같은 마크업 언어로 작성된 웹 페이지가 실제로 표현되기 위한 스타일과 디자인을 정의하는 스타일 시트 언어이다. 웹 페이지의 여러 시각적 요소를 꾸밀 수 있고 이를 통해 사용자의 경험을 개선할 수 있다.

9.3 Backend

9.3.1 Django

Figure 9.4: Django Logo

파이썬을 사용한 웹 프레임워크이다. 사용자 인증과 같은 기능이 내장되어 있어 빠르게 개발할 수 있고, Admin 기능을 통해 데이터베이스의 데이터를 쉽게 수정하고 관리할 수 있다. 또한 Object Relational Mapping(ORM)을 통해 SQL 쿼리 작성 없이 데이터베이스와 상호작용할 수 있다.

9.3.2 PostgreSQL

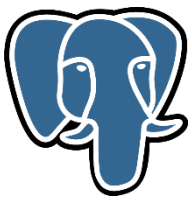

Figure 9.5: PostgreSQL Logo

PostgreSQL은 Relational Database Management System(RDBMS)의 한 종류이다. 다양한 데이터 타입을 지원하며 오픈 소스로 제공되기 때문에 무료로 사용할 수 있다. 또한 Django의 ORM과도 매우 잘 호환된다.

9.4 ChatGPT

Figure 9.6: ChatGPT Logo

ChatGPT 는 Generative Pre-trained Transformer(GPT) 기반의 대화형 인공지능 서비스이다. 문제에 대한 힌트 제공, 변형 문제 및 맞춤형 문제 제공, 사용자의 코드에 대한 피드백을 제공하기 위해 사용한다.

9.5 Version Control

9.5.1 Git

Figure 9.7: Git Logo

Git은 버전 관리 시스템이다. Branch 기능을 사용해 여러 팀원들이 동시에 여러 기능을 병렬 개발할 수 있고, Merge와 Rebase 기능을 통해 개발한 여러 기능들을 통합할 수 있다.

9.5.2 Github

Figure 9.8: Github Logo

Github은 Git을 기반으로 한 웹 플랫폼이다. Git의 기능을 온라인으로 쉽게 사용할 수 있으며 버전 관리와 협업을 더 효율적으로 관리할 수 있다.

9.6 Constraints

이 시스템은 아래 제약 사항들을 고려하여 개발해야 한다.

- 기존에 충분히 검증된 기술 및 프로그래밍 언어들을 사용한다.
- 사용자가 쉽고 편리하게 이용할 수 있도록 개발한다.
- ChatGPT를 사용하는 기능 이외의 기능들은 오픈소스 소프트웨어를 사용하거나 직접 개발한다.
- ChatGPT의 새로운 버전을 통한 시스템 업그레이드를 고려하여 개발해야 한다.
- 시스템의 개발 및 유지 보수 비용을 고려하여 개발해야 한다.
- 시스템 성능 향상을 위해 소스코드를 최적화해야 한다.

9.7 Assumptions and Dependencies

이 시스템은 PC 또는 노트북, Window 10 또는 11의 운영체제, Chrome, Firefox, Safari, Edge의 웹 브라우저에서의 사용을 가정하고 개발된다. 이외의 환경에서의 안정적인 시스템 지원은 보장할 수 없다.

10

Supporting Information

10.1 Document History

Document History			
Date	**Edit**	**Description**	**Writer**
11-19	Add contents	Add chapter 3,4,5,6,7,8	All
11-26	Update contents	Update chapter 3,4,5,6,7,8	All
11-29	Add contents	Add chapter 1,2,9,10	All
11-30	Organizing	Final Document	All