III. 공개SW프로젝트 결과보고서

프로젝트명 : CodeBrainer

교과목명	공개SW프로젝트
담당교수	송수환
팀 장	안승원(2023113581)
팀 원	이승규(2023113571)
	김채령(2023113569)
	성시연(2023110482)

시연 영상	https://www.youtube.com/watch?v=yyNfE34gkDg&feature=youtu.be
배포 URL	https://code-brainer.vercel.app/ *해당 프로젝트는 배포 후 테스트까지 완료했으며, 서버 등의 오류로 문제가 발생하였을 시 연락주시면 즉시 복구하겠습니다. seungwon6218@naver.com
테스트계정	<admin> ID: admin, pw: admin1234 <일반 사용자> ID: ahnseungwon, pw: an13882322@

□ 결과보고서 요약

프로젝트 추진배경 및 필요성	기존 코딩 학습 플랫폼은 정답 판정 중심이라 학습자가 문제 풀이 기술에만 치중하게 되며, 최근 생성형 AI의 발전으로 스스로 생각해서 문제를 풀기보다 무분별한 LLM 사용으로 창의력과 사고력의 저하가 우려됩니다. 코딩 학습을 위해서는 단기적인 편리함보다는 장기적인 알고리즘 사고력 확장이 필요합니다. 따라서 정답 코드 대신 사고 과정을 돕는 단계별 힌트와 피드백을 제공하여 사용자 스스로 학습하고 문제를 해결할 수 있는 새로운 학습 환경 구축이 필요합니다.
프로젝트 최종목적 및 세부 목표	최종 목적: 단계별 힌트와 피드백을 통해 사용자의 힌트 의존도 감소와 문제 해결 관련 사고 능력 강화 세부 목표: 1. 사고 프레임 기반 힌트 시스템: 단순 정답이 아닌 '문제 이해 → 접근법 → 구현'의 단계적 사고를 유도하는 힌트 제공. 2. AI 코드 분석 및 피드백: 사용자 코드의 효율성과 가독성을 분석하여 개선점을 제안. 3. 개인화 학습 환경: 학습 로그 분석을 통한 맞춤형 복습 문제 추천 및 모의고사 모드 구현.
기대효과	1. 자기 주도적 학습 역량 강화 2. 효율적인 코딩 습관 형성 3. 학습 몰입도 및 편의성 증대
키워드 (Keyword) (국문)	코딩테스트, 단계별 힌트, 자기주도학습, 코드 분석, 생성형 AI
키워드 (Keyword) (영문)	Coding Test, Step-by-step Hint, Self-directed Learning, Code Analysis, Generative AI

목 차

제 1 장 프로젝트 과제의 필요성

제 1 절 과제 개요 및 추진 배경

1. 현황 및 문제의식

현대 소프트웨어 산업에서 대다수의 기업들이 알고리즘 역량 검증을 위해 코딩 테스트를 채택하였습니다. 이에 따라 백준, 프로그래머스 등 다양한 Online Judge 플랫폼이 활성화되었으나, 대부분은 제출된 코드의 정답 여부만을 판단하는 결과 중심의 평가 방식에 머물러 있습니다. 이는 학습자에게 문제 해결을 위한 논리 사고 과정을 돕기보다는 정답 코드 자체에 집착하게 만듭니다.

또한, 기존의 코딩 학습은 문제 확인(웹 브라우저), 코드 작성(로컬 IDE), 실행 및 디버깅(터미널), 정답 제출(웹 브라우저) 등 여러 환경을 오가며 이루어집니다. 이러한 UI/UX의 분리는 학습자의 몰입을 방해하며, 특히 입출력 예시를 확인하기 위해 창을 번갈아 띄우는 등의 번거로움이 있습니다.

이에 정답에 집착하는 것이 아닌 사고 과정 중심의 학습이 필요하다고 판단하였고, AI가 정답을 바로 알려주는 것을 차단하여 학습자가 스스로 논리적 단계를 밟아 나갈 수 있도록 돕는 시스템을 제안합니다.

본 프로젝트는 별도의 프로그램 설치 없이 웹상에서 코딩, 실행, 채점, 디버깅이 가능한 올인원(All-in-One) 환경을 제공합니다.

알고리즘 문제는 '문제 이해 → 추상화 및 모델링 → 알고리즘 설계 → 구현 → 최적화'의 일련의 과정을 거쳐야 합니다. 이러한 사고의 단계를 시스템적으로 가이드하여, 사용자가 올바른 문제 해결 습관을 형성하도록 유도하도록 합니다. 더하여, 정답을 맞히면 끝이 아닌, 사용자의 코드를 분석하여 피드백을 제공함으로써 실질적인 코딩 능력을 향상시킬 솔루션을 제공합니다.

이 제안의 유효성을 검토하기 위해 저희는 20대 초중반의 사람들에게 설문을 진행하였습니다.[1)]

과반수가 넘는 사용자가 현재 코딩 학습 플랫폼에 불편함을 느끼고 있었습니다. 이유는 ‘IDE가 없어 온라인 IDE를 써야 한다는 점’, ‘백준 문제를 다시 보려고 문제 들어가면 내가 쓴 코드가 초기화 된다는 점’, ‘어떤 테스트케이스에서 틀렸는지 모른다는 점’, ‘ 문제에 봉착했을 때마다 인터넷을 찾아보거나 AI를 쓰면 통째로 답을 받는다는 점’, ‘힌트가 없는 문제를 풀다가 막히면 직접 찾아보는 것에 불편하다는 점’ 등이 있었습니다.[사진 1]

코딩 학습을 위해 생성형 AI를 사용하는 이용자의 과반수가 고민 없이 AI를 사용할 경우 코딩 능력이 향상되지 않음을 느꼈고[도표 1], 사용하지 않는 이용자의 과반수는 이 단점 때문에 AI를 사용하지 않았습니다.[도표 2]

이 결과들은 CodeBrainer 프로젝트가 이용자들이 편리하고 효율적으로 코딩 학습을 할 수 있게 도와주어 기존의 문제를 해결할 수 있음을 반증합니다.

---

1) 공개SW 8팀 설문조사
https://docs.google.com/forms/d/e/1FAIpQLSeYNfplNEXx2vz97aCIT-bhAQMTt6awqxb6RrqY7jR5v_Hrsw/viewform?usp=dialog

사진 1. 설문조사 - 코딩 학습 플랫폼 사용 중 불편한 점 또는 바라는 점

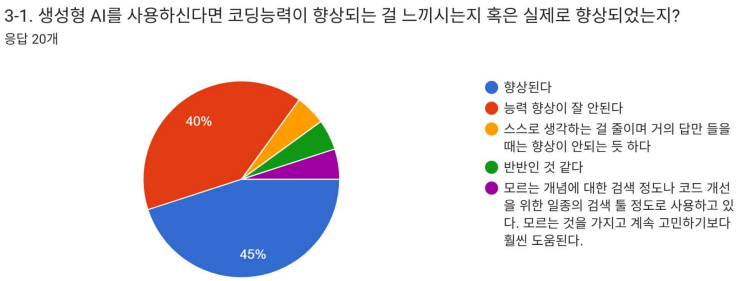

도표 1. 설문조사 - AI가 코딩 능력 향상에 도움이 되는지

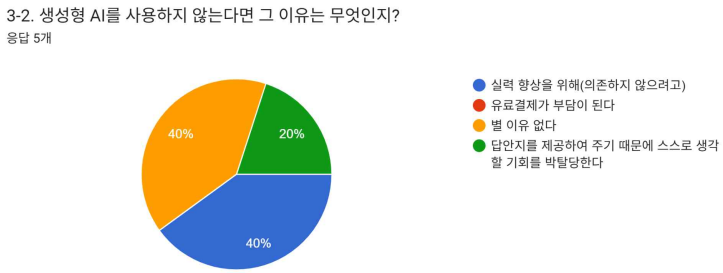

도표 2. 설문조사 - AI를 사용하지 않는 이유

2. 과제 필요성

ChatGPT, Claude 등 고성능 LLM의 등장은 코딩 진입 장벽을 낮추었으나 심각한 학습 저해 요인이 되고 있습니다. 학습자가 문제에 봉착했을 때 고민 없이 AI에게 도움을 요청하는 일이 발생합니다. AI는 바로 정답 코드를 주거나, 사용자가 알고리즘만 요청하더라도 핵심 알고리즘을 바로 제시해버립니다. 더하여 핵심을 파악하지 못한 사용자에게 너무 세부적인 정보를 과다하게 제공합니다.[사진 2] 이에 따라 학습자는 문제에 따른 알고리즘을 차근차근 생각하고 사고를 확장할 기회를 놓치게 됩니다. 이를 그대로 사용하는 방식은 단기적으로 편할지 모르나, 장기적으로는 알고리즘 사고력과 문제 해결 능력을 저하시키는 심각한 부작용을 초래합니다.

현재 코딩 문제를 풀 수 있는 타 플랫폼은 문제 화면과 제출 화면이 분리되어 있어 학습

효율이 떨어지며, 제출 화면에서 입출력 예시가 보이지 않아 다른 곳에 복사해두고 확인하며 코드를 작성해야 합니다. 이를 해결하려면 문제 창을 작게 하고, 코드 작성 창을 새로 띄워야 합니다. 이는 문제 한두 개만 간단하게 풀 때에도 사전 환경설정을 매번 해야 하는 불편함을 줍니다.

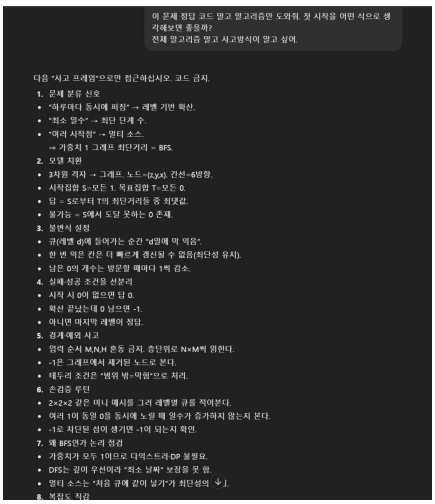

사진 2. AI에게 도움을 요청한 사용자와 답변

사진 3. 타 플랫폼의 분리된 UI/UX - 문제화면

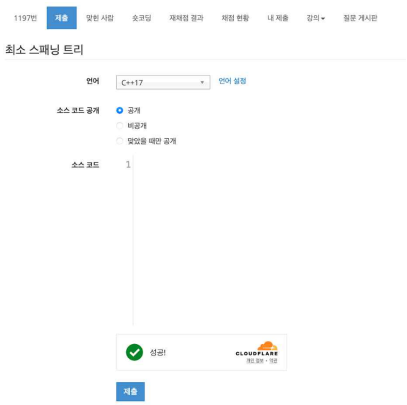

사진 4. 타 플랫폼의 분리된 UI/UX - 제출 화면

제 2 절 과제 목적 및 내용

1. 과제 목적

본 프로젝트의 최종 목적은 LLM과 프롬프트 엔지니어링 기술을 활용하여, 학습자의 코드를 실시간으로 분석하고, 정답을 직접 알려주는 대신 논리적 힌트를 제공하는 사고력 중심의 코딩테스트 자기주도 학습 플랫폼, CodeBrainer를 구축하는 것입니다. 이를 통해 학습자는 AI 기술을 단순한 정답 자판기가 아니라 튜터로 활용하여 사용자의 자립적인 문제 해결 능력을 강화하고, 실전 대비 역량을 기르고자 합니다.

2. 과제 내용

가. 안정적인 서비스 환경 구축

a. React(TypeScript)와 Spring Boot를 기반으로 직관적인 UI와 안정적인 서버를 구축하고, 회원가입/로그인 기능을 통해 사용자 관리 시스템을 구현했습니다.

b. Docker Compose 기반의 컨테이너 환경에서 RabbitMQ와 Redis를 활용한 비동기 처리를 도입하여 대용량 트래픽에도 유연하게 대응합니다.

나. 올인원(All-in-One) 웹 IDE 및 샌드박스 채점

a. Monaco Editor를 내장하여 별도 설치 없이 웹상에서 코딩과 디버깅이 가능하며, Judge0 샌드박스를 통해 보안 격리된 환경에서 안전하게 코드를 채점합니다.

다. AI 기반 단계별 힌트 및 리팩토링

a. AI API를 활용해 문제 난이도에 따라 최대 4단계의 점진적 힌트를 제공하여 학습자의 사고력을 확장시킵니다.

b. 정답 제출 후에는 시간/공간 복잡도 분석 및 코드 품질 개선을 위한 리팩토링 가이드를 자동으로 제공합니다.

라. 데이터 기반 개인화 학습 및 검증

a. 수집된 문제 데이터와 사용자 풀이 로그(성공률, 힌트 의존도 등)를 분석하여 맞춤형 복습 문제를 추천하고 학습 성과를 시각화합니다.

b. 실제 사용자 대상 설문조사를 통해 서비스의 학습 효과와 목표 달성 여부를 검증했습니다.

3. 과제 수행 방법

구 분 (Category)	기술 스택 (Technology Stack)	상세 설명 (Description)
Frontend	React, TypeScript	타입 안정성을 확보한 SPA(Single Page Application) 구현
Backend	Spring Boot	전체 시스템을 조율하는 오케스트레이터(Orchestrator) 역할
Data Collection	Python	BOJ 및 Solved.ac의 알고리즘 문제/메타 데이터 크롤링
Database	Supabase(PostgreSQL)	문제, 테스트케이스, 힌트, 사용자 로그 등 데이터 저장
Sandbox	Judge0 Worker	Docker 컨테이너 내부에서 사용자 코드를 격리하여 실행 및 채점
Infrastructure	Docker Compose	멀티 컨테이너 구조의 마이크로서비스 환경 구축
Message Queue	RabbitMQ	채점 요청의 비동기 처리 및 대용량 트래픽 분산
Cache	Redis	데이터 캐싱 및 메시지 큐 보조 역할 수행
AI Engine	Gemini 2.5 Flash	LangChain 기반 프롬프트 체인을 활용한 단계별 힌트 생성
DevOps	CI/CD, Prometheus,	컨테이너 기반 배포 자동화 및 실시간 서버/리소스 모

	Railway	니터링

제 3 절 과제 범위

1. 사업적 측면

가. 핵심 타겟 사용자

a. SW 개발 직군 취업 준비생: 코딩테스트 합격을 목표로 효율적인 학습과 실전 감각 배양이 필수적인 구직자.

b. 컴퓨터공학 및 관련 학과 전공생: 자료구조 및 알고리즘 수업의 보조 학습 도구와 자기 주도적 실습 환경이 필요한 대학생.

c. 알고리즘 입문자: 초기 진입 장벽을 낮추고, 1:1 과외와 유사한 수준의 밀착 가이드가 필요한 비전공자 및 초심자.

나. 서비스 제공 범위

a. 플랫폼 환경: 소스 코드 작성 및 디버깅에 최적화된 PC Web(데스크톱) 환경을 주력으로 지원하며, 접근성이 뛰어난 반응형 웹 디자인을 적용함.

b. 상용 수준의 웹 서비스 구현:

b-1. 사용자 관리: JWT 기반의 보안 인증을 적용한 회원가입/로그인 및 세션 관리.

b-2. 학습 환경: 별도의 설치가 필요 없는 웹 IDE 기반의 문제 풀이 및 샌드박스 채점 시스템.

b-3. 학습 관리(LMS): 개인별 학습 로그 분석, 취약점 시각화 차트, 오답 노트 기능을 포함한 마이페이지 대시보드.

b-4. 게이미피케이션: 사용자 간 학습 동기를 부여하는 티어(Tier) 제도 및 풀이 완료 목록 시각화 시스템 도입.

다. 수익 모델

a. B2C (Freemium 모델): 기본 문제 풀이와 채점 서비스는 무료로 개방하여 사용자 트래픽(Traffic)을 확보하고, 심층 AI 코드 분석, 맞춤형 모의고사 추천, 기업별 기출 분석 등 고도화된 기능은 월 구독 형태의 유료 서비스로 제공.

b. B2B/B2G: 대학 소프트웨어 학과의 실습 보조 플랫폼이나 기업의 신입 사원 직무 교육 및 채용 연계형 테스트 솔루션으로 커스터마이징하여 라이선스 판매 또는 Software as a Service 형태로 공급.

2. 연구적 측면

가. 정답 유출 방지 및 맞춤형 힌트 생성 로직 최적화 (AI Control)

a. 프롬프트 엔지니어링(Constraint-Based Prompt Engineering): 생성형 AI가 불필요한 사담(Chit-chat)이나 정답 코드를 직접 노출하는 현상을 원천 차단하기 위해, System Prompt에 엄격한 페르소나와 출력 제약 조건을 부여하는 로직을 연구함.

b. Context-Aware 힌트 생성 알고리즘: 사용자의 현재 코드 상태와 문제의 요구사항을 비교 분석하여, 정답을 알려주는 대신 사고의 방향성만 제시하는 문제별 맞춤형 힌트 생성 메커니즘을 구현함.

나. 학습 로그 분석 기반의 개인화 추천 알고리즘 (Data Analysis)

a. 사용자 행동 데이터 마이닝: 문제 풀이 성공 여부, 힌트 사용 빈도, 오답 유형 등의 비정형 로그 데이터를 수집·분석하여 사용자의 취약점(Weakness)을 정량적으로 식별하는 모델을 설계함.

b. 맞춤형 복습 추천 시스템: 분석된 취약점 데이터를 바탕으로, 사용자가 부족한 알고리즘 유형을 보완할 수 있는 적절한 난이도의 복습 문제를 선별하여 추천하는 개인화 추천 알

고리즘을 구현함.

다. 정형화된 테스트 케이스 자동 생성 기술 (Automated Testing)

a. LLM Tuning을 통한 출력 형상 제어: 자연어 처리 모델(LLM)이 채점 시스템에서 즉시 사용 가능한 표준화된 형식(JSON 등)으로 테스트 케이스를 생성하도록 Fine-tuning 및 Few-shot Prompting 기법을 적용함.

b. 데이터 무결성 검증: AI가 생성한 입출력 데이터가 문제의 논리적 조건에 부합하는지 검증하고, 파싱(Parsing) 오류를 최소화하는 테스트 데이터 생성 파이프라인을 구축함.

제 2 장 프로젝트 목적 및 내용

제 1 절 프로젝트의 최종 목적

기존의 정답 확인 위주의 단순 문제 은행 방식을 벗어나, 학습자가 문제 정의부터 해결까지의 전 과정을 주도적으로 설계하고, 이 과정에서 메타인지 능력을 극대화할 수 있는 차세대 지능형 코딩 교육 플랫폼의 표준 모델을 제시하는 것입니다.

제 2 절 프로젝트의 세부 목표

가. 마이크로서비스 지향 아키텍처 및 안정적인 서비스 환경 구축

Frontend & Backend 고도화: React(TypeScript)를 도입하여 타입 안정성을 확보한 직관적인 UI를 구현하고, Spring Boot 기반의 오케스트레이터 백엔드를 구축하여 유기적이고 안정적인 서비스 흐름을 설계했습니다. 또한 JWT 기반의 인증 시스템을 통해 안전한 회원가입 및 로그인 기능을 구현했습니다.

고가용성 비동기 처리 및 캐싱 전략: 대용량 트래픽 상황에서도 유연한 대응이 가능하도록 RabbitMQ 메시지 큐를 도입하여 채점 요청을 비동기로 처리함으로써 서버 부하를 효율적으로 분산시켰습니다. 아울러 Redis를 캐시 레이어로 활용하여 데이터 조회 속도를 최적화했습니다.

DevOps 인프라 및 실시간 모니터링: Docker Compose를 활용해 모든 서비스를 컨테이너화하여 배포 편의성을 극대화했습니다. 운영 측면에서는 Prometheus와 Railway를 연동하여 서버 리소스(CPU/Memory) 및 API 상태를 실시간으로 시각화하고 모니터링하는 관제 시스템을 구축했습니다.

나. 올인원(All-in-One) 웹 IDE 및 보안 샌드박스 채점 시스템

Web IDE 기반 통합 개발 환경: VS Code와 동일한 코어인 Monaco Editor를 내장하여 구문 강조, 자동 완성 등 실제 IDE 수준의 개발 경험을 웹상에서 제공합니다. 이를 통해 별도의 설치 없이 코딩과 디버깅이 가능하도록 하여 불필요한 화면 전환을 제거하고 사용자 편의성을 강화했습니다.

Sandbox Security 및 격리 기술: 채점 엔진인 Judge0 Worker를 Docker 컨테이너 내부에서 실행하여, 사용자 코드가 호스트 시스템에 영향을 주지 못하도록 완벽하게 격리(Isolation)했습니다. 이를 통해 무한 루프나 시스템 콜 등 보안 위협으로부터 안전한 채점 환경을 보장합니다.

다. LangChain 기반 지능형 튜터링 및 자동 리팩토링 시스템

Multi-Model Strategy & 단계별 스캐폴딩: Gemini 2.5 Flash를 LangChain의 프롬프트 체인으로 연결하여 비용 효율적이고 정확도 높은 힌트를 생성합니다. 힌트는 문제 난이도(BRONZE~PLATINUM)와 풀이 상태에 따라 1단계(개념) → 4단계(코드 모듈)로 점진적으로 제공되어 사고력을 확장시킵니다.

게이미피케이션(Gamification): 힌트 열람 시 티어 포인트(Tier Point)를 차감하는 페널

티 시스템을 적용하여, 무분별한 힌트 의존을 방지하고 스스로 문제를 해결하려는 동기를 부여합니다.

심층 코드 분석: 정답 제출 후 의미론적 분석을 수행하여 단순 정답 여부를 넘어 시간/공간 복잡도를 진단합니다. 또한 변수 명명 규칙 준수, 중복 코드 제거 등 구체적인 개선안을 제시하는 자동 리팩토링 가이드를 제공합니다.

라. 데이터 파이프라인 구축을 통한 개인화 학습 관리(LMS) 및 검증

Data Collection & Pipeline: Python 크롤러를 통해 BOJ/Solved.ac의 방대한 알고리즘 문제 데이터를 수집하고, Supabase(PostgreSQL)에 문제, 테스트케이스, 로그 데이터를 구조화하여 저장했습니다.

Personalized Recommendation & Visualization: 사용자의 성공률, 힌트 의존도, 풀이 시간 등 학습 로그를 정량적으로 분석하여, 취약점을 보완할 수 있는 맞춤형 복습 문제를 추천하고 학습 성과를 시각화합니다.

Effectiveness Validation: 기능 구현에 그치지 않고 실제 사용자를 대상으로 설문조사를 실시하여, 해당 기능들이 학습자의 자기주도적 학습 능력 향상과 목표 달성에 실질적으로 기여했음을 정량적/정성적으로 검증했습니다.

제 3 절 프로젝트의 개발 및 연구 내용

기능	사용 기술	사용 계기
가. MSA 기반 컨테이너 인프라	Docker Compose, Spring Boot, React	서비스 간 결합도를 낮추어 유지보수성과 확장성을 높이고, 각 모듈(FE/BE/AI/채점)의 독립적 배포를 위함
나. 단계별 스마트 힌트 알고리즘	LangChain, Gemini 2.5 Flash	단계별 스마트 힌트 알고리즘 정답 코드만 알려주는 기존 LLM의 한계를 극복하고, 사고 과정을 유도하기 위함
다. 비동기 샌드박스 채점 시스템	Judge0, RabbitMQ, Redis	비동기 샌드박스 채점 시스템 채점 요청 폭주(Spike) 시 서버 부하를 분산하고, 사용자 코드를 격리하여 보안성을 확보하기 위함
라. 테스트케이스 자동 생성	LLM Prompt Engineering, JSON Parsing	테스트케이스 자동 생성 채점에 필요한 정형화된 입출력 데이터를 확보하고, 수작업 생성의 비효율성을 개선하기 위함
마. 하이브리드 데이터 모델링	PostgreSQL, JSONB Type	하이브리드 데이터 모델링 문제 정보의 무결성과 비정형 AI 로그 데이터의 유연한 저장을 동시에 충족하기

		위함
바. 올인원(All-in-One) Web IDE	Monaco Editor, React Split Pane	문제 확인, 코딩, 실행이 분리된 기존 환경의 화면 전환 불편함을 해소하고 몰입도를 높이기 위함
사. 개인화 학습 지표 시각화	Data Aggregation, Visualization Lib	단순 문제 풀이 기록을 넘어, 성장 추이와 약점을 시각적으로 제공하여 학습 동기를 부여하기 위함
아. 학습 효과 검증	설문조사 (정량/정성 분석)	구현된 기능이 실제 학습자의 자기주도적 학습 능력 향상에 기여했는지 객관적으로 검증하기 위함

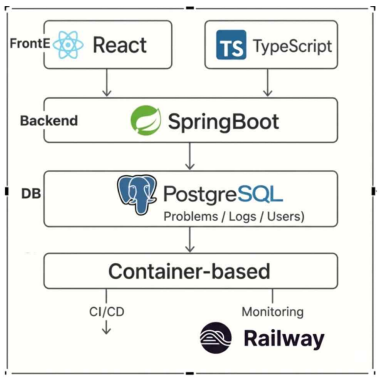

그림 1. 프로젝트 아키텍처 다이어그램

가. 시스템 아키텍처 및 인프라

서비스 간 결합도를 낮추어 유지보수성과 확장성을 높이기 위해 Docker Compose, Spring Boot, React를 활용한 마이크로서비스 아키텍처(MSA)를 구축했습니다. 프론트엔드는 React를 사용하여 사용자 친화적인 UI를 구성하였으며, 백엔드는 Spring Boot를 기반으로 안정적인 API 서버를 구현했습니다. 이들 모듈과 더불어 AI 및 채점 서비스 등을 각각 독립적인 컨테이너로 분리하고 Docker Compose로 관리함으로써, 각 모듈의 독립적인 배포가 가능하도록 설계하여 전체 시스템의 유연성을 극대화했습니다.[그림 1]

나. LLM 기반 단계별 힌트 생성 알고리즘

기존 LLM이 정답 코드만 바로 알려주어 학습 효과를 저해하는 한계를 극복하고자, LangChain, Gemini 2.5 Flash를 활용하여 사고 과정을 유도하는(Socratic Method) 알고리즘을 구현했습니다. LangChain을 이용해 대화의 맥락을 유지하면서, System Prompt에 튜터 역할을 부여하여 알고리즘적 사고를 돕는 힌트만 생성하도록 제약했습니다. 또한 응답 속도가 빠른 Gemini 2.5 Flash를 사용하는 하이브리드 전략을 통해 비용 효율성과 답변의 정확도를 동시에 확보했습니다.

a. 난이도별 힌트 제공 전략

Level 1. Bronze 3 이하
제공 힌트 수: 최대 1개
제공 전략: 언어의 기초 문법이나 연산자 활용법에 초점을 맞춥니다.
힌트 내용: 문제 해결의 중심이 되는 기초 아이디어(Key Idea)만을 텍스트로 설명합니다.
생성 예시 (백준 2588번):
"몫(//)과 나머지(%) 연산자를 활용하여 입력된 정수의 자릿수(일의 자리, 십의 자리, 백의 자리)를 수학적으로 분리하는 것이 핵심입니다."
Level 2. Bronze 2 ~ Silver 3
제공 힌트 수: 최대 2개
제공 전략: 문제를 추상화하여 특정 자료구조(그래프, 스택 등)로 치환하는 모델링 과정을 설명합니다.
힌트 내용:
Hint 1: 문제 핵심 개념 및 모델링 가이드를 제공합니다.
Hint 2: 구체적인 알고리즘 명칭과 작동 논리를 텍스트로 설명하되, 코드는 제공하지 않습니다.
생성 예시 (백준 1085번):
"사람을 정점(Vertex), 친구 관계를 간선(Edge)으로 모델링하여 최단 거리를 구하는 문제입니다. 모든 정점 쌍의 거리를 구해야 하므로 3중 반복문을 사용하는 플로이드-워셜 알고리즘을 적용하세요."
Level 3. Silver 2 ~ Gold 3
제공 힌트 수: 최대 3개
제공 전략: 알고리즘을 유추하도록 돕고 이를 실제로 구현하도록 돕습니다.
힌트 내용:
Hint 1: 문제 핵심 개념을 통한 알고리즘 유추를 돕습니다.
Hint 2: 구체적인 알고리즘 명칭과 작동 논리를 텍스트로 설명하되, 코드는 제공하지 않습니다.
Hint 3: 구체적인 예시와 함께 슈도코드로 설명합니다.
생성 예시 (백준 1006번):
"원형 구조를 선형으로 변환하기 위해 4가지 케이스로 나누어 DP를 수행해야 합니다. dp[i][state] 점화식을 세울 때, 현재 열을 덮는 3가지 경우(단독, 수직, 수평)를 고려하여 최솟값을 갱신합니다."
Level 4. Gold 2 이상
제공 힌트 수: 최대 4개 (1~3단계 + 핵심 코드 모듈)
제공 전략: 구현 난이도가 매우 높아 논리만으로는 해결이 어려운 경우, 핵심 테크닉이 담긴 코드 모듈을 제공합니다.
힌트 내용:
Hint 1: 문제 핵심 개념을 통한 알고리즘 유추를 돕습니다.
Hint 2: 구체적인 알고리즘 명칭과 작동 논리를 텍스트로 설명하되, 코드는 제공하지 않습니다.
Hint 3: 구체적인 예시와 함께 슈도코드로 설명합니다.
Hint 4: 전체 정답 코드가 아닌, 구현의 중심이 되는 핵심 함수나 테크닉만을 모듈 형태로 제공하여 사용자가 나머지를 완성하도록 유도합니다.
생성 예시 (백준 1006번):

"4가지 선형 케이스를 처리하는 핵심 DP 함수 solve_case의 구현 패턴은 다음과 같습니다. + 모듈 제공"

b. 프롬프트 엔지니어링 및 제약

AI가 교육적 목적을 벗어나지 않도록 System Prompt에 엄격한 제한 조건을 설정했습니다. 사용자가 요청한 힌트 단계에 맞춰 정보의 노출 수위를 조절하며, 하위 난이도 문제에서 상위 단계의 힌트를 제공하는 것을 금지했습니다. 불필요한 한자어나 모호한 표현을 배제하여 힌트의 가독성을 높였습니다.

다. Judge0를 활용한 샌드박스 채점 환경 구축

채점 요청이 폭주(Spike)하는 상황에서도 서버 부하를 효과적으로 분산하고 보안성을 확보하기 위해 Judge0, RabbitMQ, Redis를 도입했습니다. 사용자 코드는 Judge0를 통해 격리된 샌드박스 환경에서 안전하게 실행되며, 메모리 및 시간 제한 정책이 엄격하게 적용됩니다. 대량의 제출 요청은 RabbitMQ 메시지 큐에 적재되어 비동기적으로 처리되며, Redis를 캐시 및 상태 저장소로 활용하여 클라이언트가 폴링(Polling) 방식으로 결과를 빠르게 확인할 수 있는 구조를 구현했습니다.

라. 채점 위한 테스트케이스 구성

채점에 필요한 정형화된 입출력 데이터를 확보하고 수작업 생성의 비효율성을 개선하기 위해 LLM Prompt Engineering과 JSON Parsing 기술을 적용했습니다. LLM에게 문제 조건을 입력하고 답변 형식을 JSON으로 강제하여 구조화된 테스트케이스를 추출합니다. 생성된 데이터 중 포맷이 맞지 않는 예외 케이스는 파싱 로직을 통해 자동으로 필터링하거나 수정하여, 즉시 채점 시스템에 투입 가능한 고품질의 데이터를 확보했습니다.

마. 데이터베이스 모델링 및 최적화

문제 정보의 무결성과 비정형 AI 로그 데이터의 유연한 저장을 동시에 충족하기 위해 PostgreSQL과 JSONB Type을 활용한 모델링을 했습니다. 문제, 테스트케이스, 사용자 정보 등 정형 데이터는 PostgreSQL의 관계형 테이블에 저장하여 데이터 무결성을 보장했습니다. AI가 생성하는 긴 데이터는 docker에 저장하고 DB에는 경로를 저장하여 간접적으로 데이터를 관리하고 조회할 수 있도록 설계했습니다.

바. 올인원(All-in-One) Web IDE

기존 코딩 학습 환경에서 문제 확인, 코딩, 실행 창이 분리되어 발생하는 화면 전환의 불편함을 해소하고 학습 몰입도를 높이기 위해 Monaco Editor와 React Split Pane을 도입했습니다. VS Code와 동일한 코어인 Monaco Editor를 내장하여 웹상에서도 실제 IDE 수준의 코드 편집 경험을 제공하며, React Split Pane을 통해 사용자가 자신의 편의에 맞춰 화면 레이아웃(문제창, 에디터, 터미널)을 자유롭게 조절할 수 있는 통합 환경을 구현했습니다.

사. 개인화 학습 지표 시각화

단순한 문제 풀이 기록을 넘어 사용자의 성장 추이와 약점을 시각적으로 제공하여 학습 동기를 부여하기 위해 Data Aggregation 기술과 Visualization Lib을 활용했습니다. 사용자별 풀이 로그를 집계(Aggregation)하여 일자별 티어 증가율, 난이도별 정답률, 알고리즘 유형별 취약점 등을 분석하고, 이를 직관적인 차트와 그래프로 시각화하여 마이페이지에 제공함으로써 자기주도적 학습 계획 수립을 지원했습니다.

아. 학습 효과 검증

구현된 기능들이 실제 학습자의 자기주도적 학습 능력 향상에 기여했는지 객관적으로 검증하기 위해 설문조사(정량/정성 분석)[2]를 실시했습니다. 실제 사용자를 대상으로 문제 해결 시간 단축 여부, 힌트 기능의 유용성, 리팩토링 가이드의 효과 등을 묻는 설문을 진행하고, 수집된 데이터를 정량적 수치와 정성적 피드백으로 분석하여 프로젝트의 실효성을 입증했습니다.

[설문조사 내용]
Q1. 기존에 사용해본 온라인 코딩 플랫폼이 무엇인가요?
Q2. 저희 플랫폼을 사용한 경험은 총 몇 회입니까?
Q3. CodeBrainer를 사용하기 전에, 코딩 문제를 풀 때 힌트를 얻기 위해 서칭 시(생성형 AI 사용 포함) 정답 코드를 너무 쉽게 얻게 되어 깊이 고민할 기회를 놓치고 있다고 느끼신 적이 있습니까?
Q4. 문제 설명부터 힌트, 코드 작성, 실행 결과까지 한 화면에 통합된 CodeBrainer의 UI/UX가 학습에 대한 몰입도와 집중력을 높이는 데 효과가 있었나요? (타 플랫폼 대비)
Q5. 다른 코딩 학습 플랫폼(예: 백준, 프로그래머스 등등)과 비교했을 때, 기존 플랫폼이 제공하지 않는 저희 플랫폼의 'AI 힌트'는 어땠나요?
Q6. 다음 플랫폼 기능/요소가 학습 동기 부여 및 이해도 증진에 미친 영향을 평가해 주세요.
Q7. 다른 플랫폼과 비교했을 때, 저희 플랫폼이 차별화되었다고 생각하는 점은 무엇입니까?
Q8. 제공되는 AI 피드백 수준과 만족도는 어땠나요?
Q9. 기존 플랫폼 방식에 비해 CodeBrainer가 장기적인 알고리즘 사고력 및 문제 해결 역량을 기르는 데 더 효과적이라고 생각하시나요?
Q10. 저희 플랫폼을 사용하신 뒤, 얼마나 만족하셨나요?
Q11. 저희 플랫폼의 개선이 필요하다고 생각하는 부분이나 기타 자유 의견을 남겨주세요.

제 4 절 프로젝트의 주요 결과물

가. 프론트엔드 스토리보드

[그림 5]부터 [그림 22]는 최종 구현된 시스템의 핵심 기능 시연 화면 및 사용자 시나리오입니다. 코딩 테스트 준비와 알고리즘 학습은 텍스트 입력 비중이 높고 정밀한 코드 편집이 요구되는 작업입니다. 이에 본 팀은 모바일 환경보다는 PC 환경에서의 사용성에 주안점을 두어, 접근성이 뛰어난 반응형 웹 플랫폼으로 서비스를 구축하였습니다.

- 메인 페이지[사진 5]
로그인 혹은 회원가입을 하여 코딩 학습을 시작하도록 합니다.

- 회원가입 페이지[사진 6]
아이디 중복확인, 사용자 이름과 비밀번호 입력, 비밀번호 확인 절차를 거쳐 회원가입을 진행합니다. 이 때 계정이 있는 회원은 로그인 화면으로 넘어갈 수 있도록 합니다.

---

2) 공개 SW 8팀 최종 설문조사
https://docs.google.com/forms/d/e/1FAIpQLSdW09UXXb1fiOFBOW0npYjemjhh05S-wKlURybN-luTRXI2UQ/viewform?usp=dialog

- **로그인 페이지[사진 7]**
아이디와 비밀번호를 입력하여 로그인합니다. 로그인 성공 시, 로그인 된 상태로 메인 화면으로 돌아갑니다.

- **문제 풀기 기능[사진 8~17]**
문제 목록 탭에 들어가 전체 문제를 볼 수 있습니다[사진 8]. 다른 목록을 누르면 알고리즘별로도 문제를 보고, 풀고 싶은 알고리즘을 특화하여 학습할 수 있습니다 [사진 9].
문제 하나를 클릭하면 문제와 제한 사항 및 예시 입출력과 함께 소스코드 작성과 컴파일, 제출을 같이 할 수 있도록 합니다[사진 10].
힌트를 보고 싶으면 힌트 탭을 클릭합니다[사진 11]. 힌트의 오남용을 막기 위해 시간 제한을 두어, 사용자가 진짜 필요할 때 힌트를 사용할 수 있도록 합니다[사진 12]. 제공되는 힌트는 문제와 함께 DB에 저장되어 있고, 문제 난이도에 따라 1개에서 최대 4개까지 생성되어 있습니다[사진 13].
소스코드 작성 시, 자동완성 기능과 컴파일 에러가 날 위치에 표시를 해두어 사용자에게 편의성을 줍니다[사진 14]. 사용자가 선택한 언어와 작성하는 언어가 다르면 제출을 여러 번 해야 하는 불편함을 주기에 언어를 감지하는 기능을 제공하여 중복 제출의 번거로움을 줄입니다[사진 15]. 컴파일 시 제한 사항을 잘 맞추었는지 시각화해주고, 정답인지 아닌지 표시하는 기능이 구현되어 있습니다[사진 17]. 제출된 코드가 정답일 때, AI 코드 리뷰를 통해 사용자가 더 좋은 코드를 작성할 수 있도록 돕습니다[사진 16].

- **모의고사 모드[사진 18~20]**
모의고사 탭에 들어가면 세 난이도의 모의고사를 풀어볼 수 있습니다[사진 18]. 제한 시간과 제출 및 정답 여부를 실시간으로 확인할 수 있습니다[사진 19]. 모의고사 결과를 알려줍니다[사진 20].

- **개인 페이지[사진 21, 22]**
활동 그래프를 통해 꾸준히 문제를 풀도록 돕고 문제 풀이 통계를 시각화해줍니다[사진 21]. 힌트 사용량 추이를 통해 성장을 가시화하고 시도한 문제 저장을 통한 AI 추천 문제를 제공합니다[사진 22].

사진 5. 메인

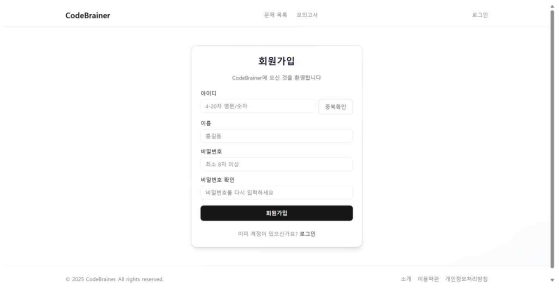

사진 6. 회원가입

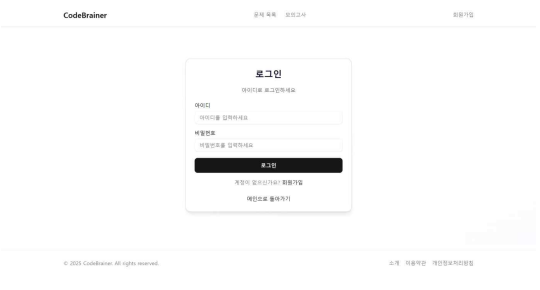

사진 7. 로그인

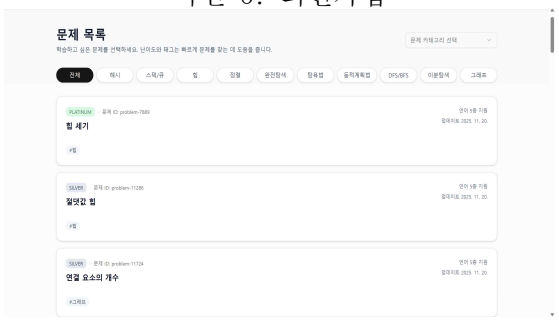

사진 8. 문제 목록

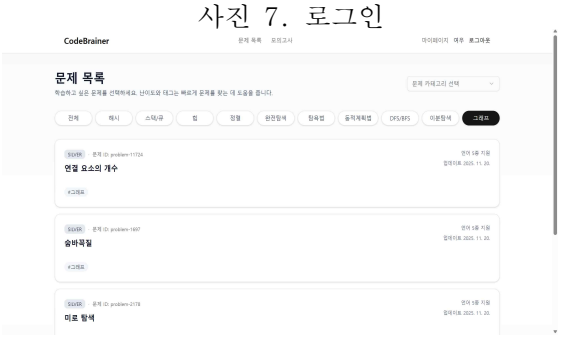

사진 9. 알고리즘별 문제 목록 조회

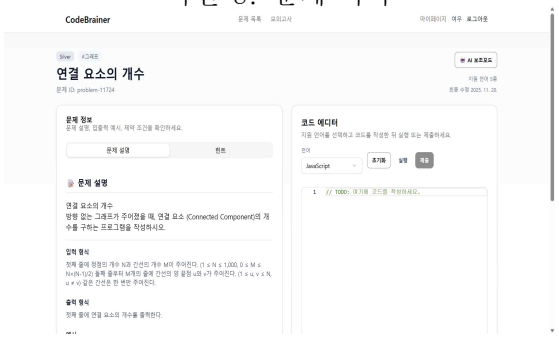

사진 10. 문제 세부 화면

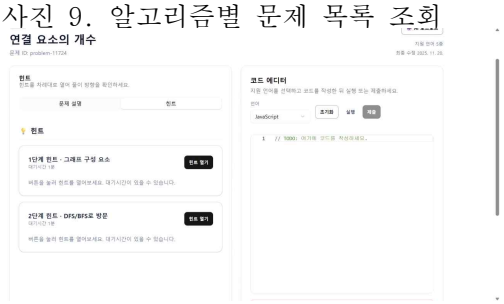

사진 11. 힌트 화면

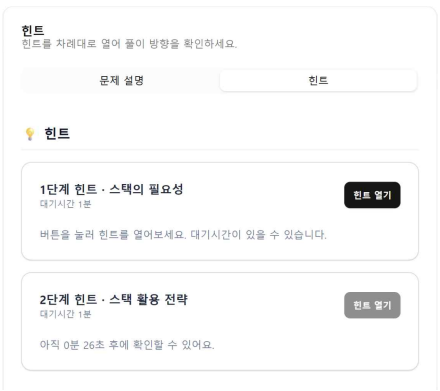

사진 12. 힌트 대기 화면

사진 13. 힌트 제공 화면

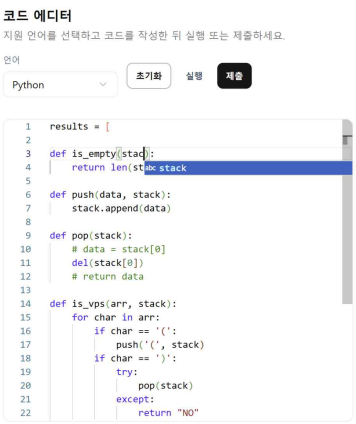

사진 14. 컴파일 오류 시 표시, 자동완성

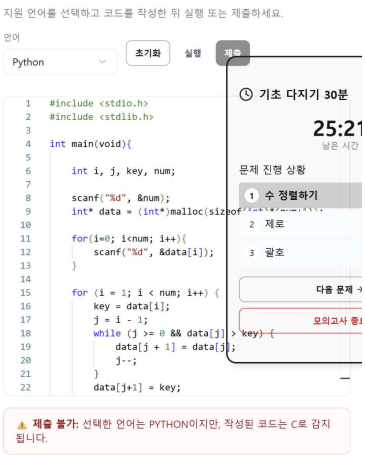

사진 15. 언어 감지

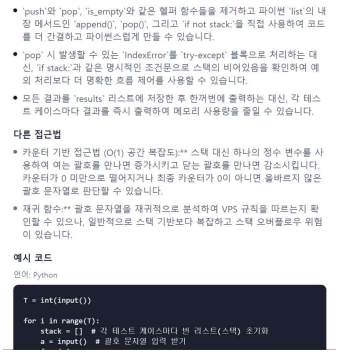

사진 16. 코드 리뷰

사진 17. 결과 확인 기능

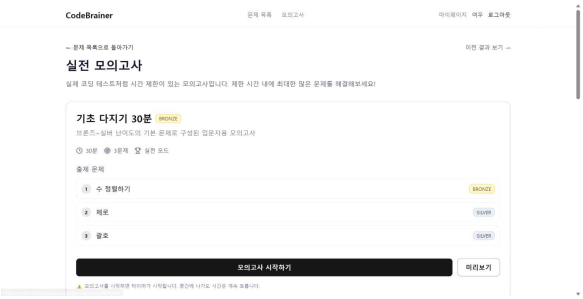

사진 18. 모의고사 모드

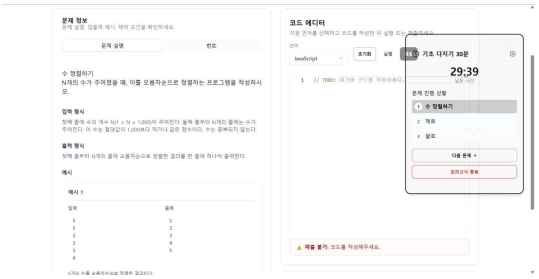

사진 19. 모의고사 모드 세부

사진 20. 모의고사 결과 페이지

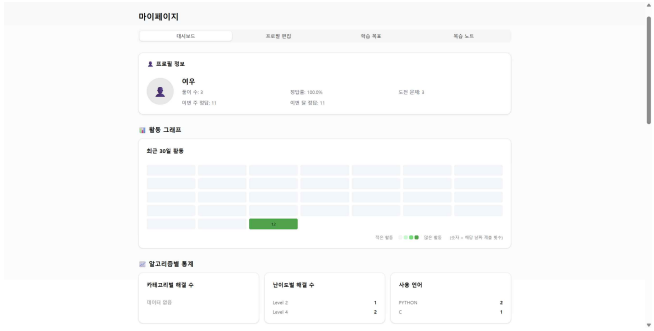

사진 21. 마이페이지1

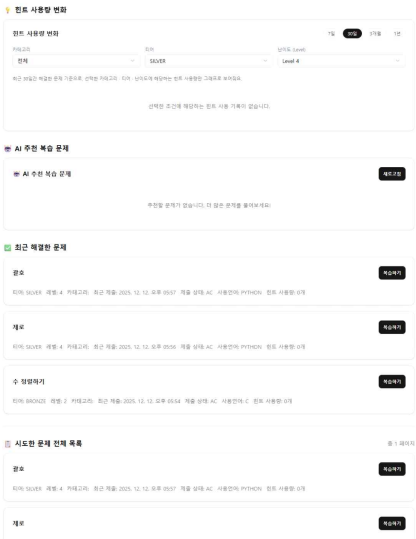

사진 22. 마이페이지2

나. 백엔드 API 엔드포인트

프론트엔드와 백엔드 간의 표준화된 API 규칙을 확립하기 위해, REST API 규칙에 따라 API를 개발했습니다.

회원가입 시 이메일 중복 체크, 아이디 중복 확인을 통해 unique성을 보장하고, 비밀번호를 BCrypt로 해싱해서 암호화한 채로 DB에 저장하여 보안을 유지합니다. 로그인 시 아이디를 조회하고 로그인 후 JWT 토큰을 생성하고 프론트엔드에 토큰을 저장해둡니다. local stroage 기반으로 사용자 정보를 읽고 정보 사용하여 페이지 새로고침해도 정보가 유지되도록 하고 서버 요청없이 접근할 수 있게 합니다. 서버에서 토큰을 검증해서 사용자를 인증합니다.

REST API 규격에서 PUT, DELETE, PATCH를 사용하지 않고 GET과 POST를 이용하여 API를 구성했습니다. HTTP 요청을 처리하는 리소스를 줄이고, 프로젝트를 단순하고 일관성 있게 개발할 수 있게 하기 위함입니다.

다. 문제 생성

실전과 유사한 양질의 학습 환경을 제공하기 위해 백준 온라인 저지(BOJ) 및 Solved.ac의 검증된 문제 유형을 벤치마킹하여 학습 데이터셋을 구성했습니다. 단순한 데이터 수집에 그치지 않고, 핵심 알고리즘 로직과 난이도 기준은 유지하되 문제의 지문과 배경 상황(Context)을 재구성(Reconstruction)하고 입출력 조건을 변형하여 학습 효율을 높였습니다. 또한, 모든 콘텐츠는 플랫폼 UI에 최적화된 마크다운(Markdown) 포맷으로 정규화하여 DB에 적재함으로써, 사용자에게 일관된 가독성을 제공하고 AI가 문제의 의도를 명확히 파악할 수 있도록 설계했습니다.

라. 힌트 생성

기존 생성형 AI가 초심자의 수준을 고려하지 않고 불필요한 예외 처리나 과도하게 복잡한 최적화 기법을 먼저 제시하여 학습 난이도를 불필요하게 높이는 문제를 해결하고자 했습니다. 이를 위해 프롬프트 엔지니어링을 수행하여, AI가 정답 코드는 물론 학습 단계에 맞지 않는 난해한 기술적 조언을 출력하는 것을 원천 차단했습니다. 대신 문제 해결에 필수적인 핵심 논리를 문제 이해 → 알고리즘 접근 → 구현 가이드의 순서로 점진적으로 제시하도록 설계하여, 사용자가 자신의 수준에 맞춰 단계적으로 사고를 확장해 나갈 수 있도록 유도했습니다.

마. 테스트케이스 생성

사용자 코드의 견고성을 검증하기 위해 LLM을 활용하여 경계값 분석(Boundary Value Analysis) 및 오버플로우(Overflow) 유발 등 엣지 케이스 위주의 테스트 데이터를 생성했습니다. 생성형 AI 특성상 발생하는 자연어 혼입이나 포맷 불일치 문제를 해결하기 위해, AI의 답변 형식을 JSON으로 강제하고 정규식을 활용한 후처리 파이프라인을 구축했습니다. 이를 통해 AI 출력물에서 순수한 입출력 데이터만을 파싱하고 유효성을 검증한 뒤 데이터베이스에 적재함으로써 채점 시스템의 신뢰성을 확보했습니다.

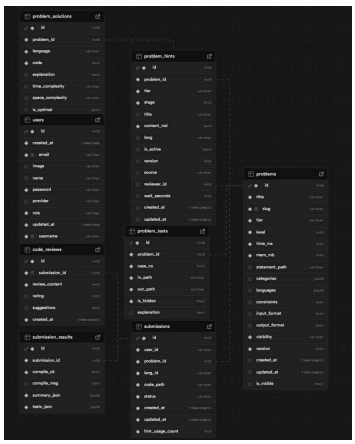

사진 23. 데이터베이스

바. DB 구성[사진 23]

users 릴레이션은 플랫폼을 이용하는 모든 사용자의 계정 정보를 관리합니다. 각 사용자는 고유한 id를 기본키(Primary Key)로 가지며, 로그인에 필요한 email과 username은 unique 제약 조건을 설정하여 데이터 무결성을 보장했습니다. 특히 password 속성은 보안상의 이유로 평문이 아닌 암호화된 문자열로 저장하여 데이터베이스가 유출되더라도 계정 정보를 보호할 수 있도록 설계했습니다. 또한, role 속성을 통해 사용자를 'USER'와 'ADMIN'으로 구분하고, 이에 따라 관리자 페이지 접근 권한 등을 제어하는 역할 기반 접근 제어(RBAC)를 구현했습니다.

학습 콘텐츠의 핵심인 problems 릴레이션은 알고리즘 문제의 제목, 난이도(tier), 시간 및 메모리 제한 등 필수 메타데이터를 저장합니다. 이때 문제의 카테고리나 지원 언어 목록과 같이 가변적인 데이터는 jsonb 타입을 사용하여 스키마 변경 없이도 유연하게 확장할 수 있도록 구성했습니다. 이와 연동된 problem_hints 릴레이션은 본 프로젝트의 핵심 기능인 단계별 힌트 시스템을 뒷받침합니다. problem_id를 외래키로 참조하며, stage 속성을 통해 힌트의 단계를 구분하고 wait_seconds를 통해 힌트 열람 대기 시간을 설정함으로써, 학습자가 점진적

으로 사고를 확장할 수 있는 로직을 데이터베이스 수준에서 지원합니다.
사용자의 문제 풀이 활동은 submissions와 submission_results 두 개의 릴레이션으로 분리하여 관리합니다. submissions 릴레이션은 사용자가 제출한 코드의 경로(code_path), 언어, 판정 상태(status) 등 핵심 정보와 함께 hint_usage_count를 기록하여 학습 분석의 기초 데이터를 제공합니다. 반면, 채점 과정에서 발생하는 방대한 로그 데이터는 submission_results 릴레이션에 별도로 저장하여 시스템 성능을 최적화했습니다. 특히 tests_json 속성을 jsonb 타입으로 설정하여, 테스트 케이스마다 상이한 실행 결과와 메모리 사용량, 런타임 에러 메시지 등을 유연하게 적재하고 고속으로 조회할 수 있게 하였습니다.
마지막으로 code_reviews 릴레이션은 AI가 생성한 코드 분석 결과를 저장합니다. submission_id를 외래키로 하여 특정 제출 기록과 1:1로 매핑되며, AI가 작성한 상세 리뷰(review_content), 개선 제안(suggestions), 그리고 코드 품질 점수(rating)를 영구적으로 보존합니다. 이를 통해 사용자는 언제든지 자신의 과거 제출 내역에 대한 피드백을 다시 확인하고 복습할 수 있습니다. 또한 problem_tests 릴레이션에서는 is_hidden 속성을 두어 일반 예제와 채점용 히든 케이스를 구분함으로써 채점 시스템의 보안성과 공정성을 확보했습니다.

사. Docker 구성

백엔드 오케스트레이터 역할을 수행하는 Spring Boot(Java)와 AI 힌트 생성 및 데이터 수집을 담당하는 Python 모듈이 유기적으로 연동되는 Polyglot(다언어) 아키텍처를 채택했습니다. 이를 단일 컨테이너 환경에서 효율적으로 배포하고 운영하기 위해 Multi-stage Build 전략을 기반으로 Docker 이미지를 구성했습니다.
빌드 프로세스의 첫 단계인 Build Stage에서는 maven:3.9.9-eclipse-temurin-21 이미지를 기반으로 소스 코드를 컴파일하고 패키징합니다. 이 과정에서 dependency:go-offline 명령어를 선행하여 의존성 라이브러리를 레이어에 캐싱함으로써, 소스 코드 변경 시 재빌드 시간을 단축시켰습니다. 이후 테스트 단계를 건너뛰고(-DskipTests) 신속하게 프로덕션용 JAR 파일을 생성하도록 설정하여 CI/CD 파이프라인의 효율성을 높였습니다.
실제 애플리케이션이 구동되는 Runtime Stage는 경량화된 python:3.11-slim 이미지를 베이스로 채택했습니다. 이는 Python 기반의 LangChain 및 크롤링 라이브러리 구동 환경을 우선 확보하기 위함이며, 여기에 openjdk-21-jre를 추가로 설치하여 Java 기반의 Spring Boot 애플리케이션이 실행될 수 있는 하이브리드 런타임 환경을 완성했습니다. Python 환경은 가상 환경(venv)을 통해 격리하였으며, BeautifulSoup4와 lxml(크롤링), LangChain 및 OpenAI SDK(AI 모델 연동) 등 필수 라이브러리만을 선별적으로 설치하여 이미지 크기를 최적화했습니다.
최종적으로 빌드 단계에서 생성된 app.jar만을 런타임 컨테이너로 복사하여 불필요한 빌드 도구와 소스 코드를 제거함으로써 보안성과 스토리지 효율성을 확보했습니다. 컨테이너는 8080 포트를 통해 외부와 통신하며, Java 엔트리포인트를 통해 오케스트레이터가 실행되면서 내부적으로 Python 모듈을 호출하는 구조로 동작합니다.

아. 설문 조사 결과

본 프로젝트의 실효성을 검증하기 위해 실제 코딩테스트 학습 경험이 있는 사용자 대상으로 CodeBrainer 플랫폼 이용 후 설문조사를 실시했습니다. 설문 결과는 프로젝트의 기획 의도인 사고력 증진과 학습 효율성 향상이 실제 사용자에게 유효하게 작용했음을 정량적으로 입증하고 있습니다.
1. 기존 학습 환경의 문제점 검증
‘기존 검색 시 정답 코드를 너무 쉽게 얻어 깊은 고민의 기회를 놓쳤다고 느낀 적이 있는

가?[도표 3]' 에 응답자의 78.6%가 기존의 단순 정답 제공 방식으로 인해 학습 기회를 상실한 경험이 있다고 응답했습니다. 이는 정답 중심의 학습이 사고력 저하를 유발한다는 본 프로젝트의 문제 정의가 실제 학습자의 니즈와 일치함을 보여줍니다.

2. 핵심 솔루션(AI 힌트 및 피드백)의 유효성 검증

타 플랫폼 대비 AI 힌트의 유용성[도표 5] 및 장기적 사고력 증진 효과[도표 9] 분석 결과, 모든 응답자가 CodeBrainer의 단계별 힌트 시스템이 코딩 학습에 유의미한 도움을 주었다고 평가했습니다. 또한, 92.9%의 사용자가 기존 방식보다 본 플랫폼이 장기적인 알고리즘 사고력을 기르는 데 더 효과적이라고 답변했습니다. 이는 단순히 정답을 알려주는 것이 아니라, 사고의 과정을 유도하는 방식이 교육적으로 주효했음을 시사합니다.

3. 통합 UI/UX의 사용자 경험 평가

통합된 UI/UX가 학습 몰입도에 미친 영향[도표 4]에 대해 응답자 전원(100%)이 통합된 Web IDE 환경이 학습 몰입도와 집중력을 높이는 데 효과적이었다고 응답했습니다. 이는 불필요한 화면 전환을 제거한 올인원(All-in-One) 설계 전략이 사용자 편의성을 성공적으로 개선했음을 입증합니다.

4. 플랫폼 차별성 및 종합 만족도

타 플랫폼 대비 가장 차별화된 점을 묻는 질문[도표 7]에서 AI 힌트를 통해 스스로 해결하게 유도하는 방식이 85.7%로 압도적인 1위를 차지했습니다. 이어 간편한 UI/UX(57.1%)와 상세한 AI 피드백(57.1%)이 뒤를 이었습니다.

종합 만족도 분석 결과, AI 피드백 수준[도표 8]과 플랫폼 전반적인 만족도[도표 10]에서 모든 사용자가 5점 만점에 4점 이상의 높은 점수를 부여했습니다. 특히 부정적인 평가가 단 한 건도 없었다는 점은 모델의 완성도가 높음을 방증합니다.

5. 결론 설문조사 결과를 종합해 볼 때, CodeBrainer는 정답 의존도를 낮추고 자기주도적 문제 해결 능력을 강화한다는 프로젝트의 핵심 목표를 성공적으로 달성하였으며, 기능적 우수성과 교육적 효과 모두 사용자에게 긍정적인 평가를 받았다고 할 수 있습니다.

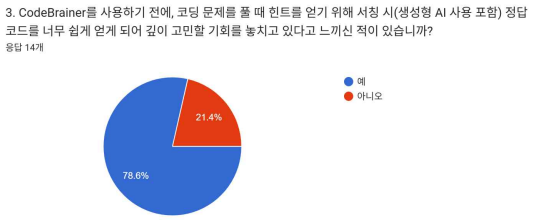

도표 3. 기존 학습의 문제점 검증

도표 4. 프로젝트의 UI/UX 편의성 및 몰입도

도표 5. AI 솔루션의 유효성 평가

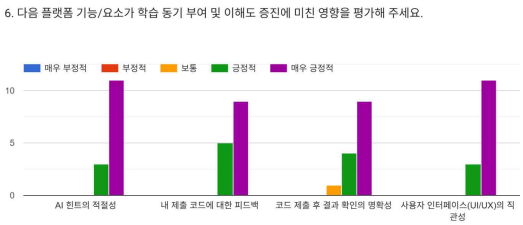

도표 6. 교육적 효과 및 역량 강화도 평가

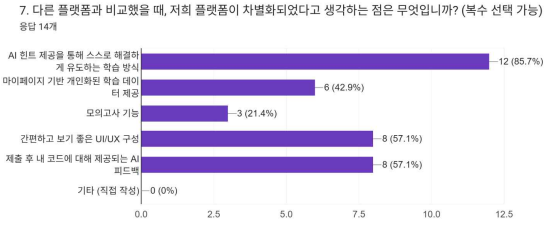

도표 7. 플랫폼 경쟁력 및 만족도

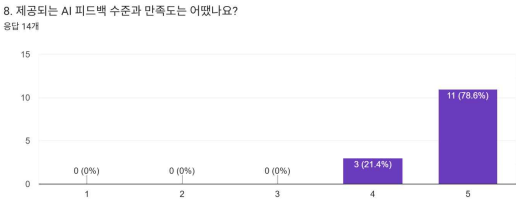

도표 8. AI 솔루션의 만족도

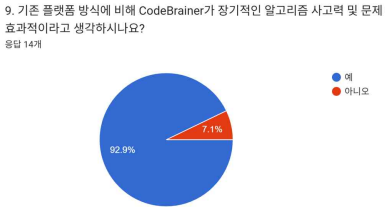

도표 9. 교육적 효과 평가

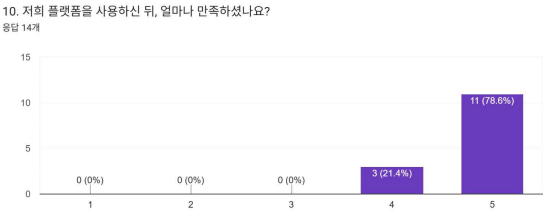

도표 10. 플랫폼 만족도

제 3 장 추진 일정

제 1 절 추진전략 및 방법

1. 추진 목표

자기주도적 알고리즘 학습을 위한 지능형 튜터링 플랫폼의 성공적인 구현을 목표로 합니다. 이를 위해 2주 단위의 스프린트를 수행하여, 웹 IDE와 샌드박스 채점 시스템 등 핵심 기능을 갖춘 MVP(Minimum Viable Product)를 조기에 확보하고 기술적 검증을 완료하였습니다. 이후 프롬프트 엔지니어링 기술을 고도화하여 단순 정답 제공이 아닌 학습자의 사고 흐름을 유도하는 단계별 스마트 힌트 시스템을 통합함으로써 교육적 차별성을 확보하고자 합니다. 기술적으구조를 도입하여, 대규모 채점 요청이 발생하는 상황에서도 중단 없는 고가용성 서비스를 실로는 Docker Compose 기반의 마이크로서비스 아키텍처와 RabbitMQ 및 Redis를 활용한 비동기 처리 현하는 것을 핵심 목표로 삼고 있습니다.

2. 추진 전략 및 방법

완성도 높은 결과물을 도출하기 위해 기획부터 검증까지 체계적인 단계별 개발 프로세스를 수립하였으며, 애자일(Agile) 문화에 기반한 협업 전략을 채택하여 개발 생산성과 소프트웨어 품질을 극대화하였습니다.

가. 단계별 개발 추진 전략

1단계: 요구사항 분석 및 아키텍처 설계

현황 분석: 백준(BOJ), 프로그래머스 등 기존 코딩테스트 플랫폼의 사용자 경험(UX)을 심층 분석하여, '결과 중심의 평가'와 '파편화된 학습 환경'이라는 주요 문제를 도출했습니다.

설계: 이를 해결하기 위해 '과정 중심의 학습'이라는 핵심 가치를 설정하고, 시스템의 유연성과 확장성을 보장하기 위해 프론트엔드, 백엔드, AI, 채점 엔진이 독립적으로 동작하는 구조를 설계했습니다. 또한, 데이터의 무결성과 조회 성능을 고려하여 DB를 확정하였습니

다.
2단계: 프로토타입 구축 및 기술 검증 (Prototyping)
기술 검증(PoC): React(TypeScript) 기반의 프론트엔드 UI 골격과 Spring Boot 백엔드 서버의 기본 구조를 신속하게 구축하여 기술 스택의 적합성을 검증했습니다.
핵심 연동 테스트: 웹 기반 개발 환경의 핵심인 Monaco Editor와 Docker 기반 샌드박스 간의 통신 및 코드 실행 흐름을 우선적으로 구현하여, 기술적 난이도가 높은 기능의 리스크를 사전에 식별하고 제거했습니다.
3단계: 핵심 기능 구현 및 고도화 (Core Implementation)
기능 개발: Web IDE와 Judge0 채점 서버를 연동하여 실시간 코드 실행 및 채점 환경을 완성하고, LangChain 기반의 단계별 AI 힌트 시스템 및 사용자 로그 분석을 통한 개인화 복습 시스템 등 핵심 비즈니스 로직을 구현했습니다.
성능 최적화: 대규모 트래픽 발생 시에도 안정적인 서비스를 제공하기 위해 RabbitMQ를 도입하여 채점 요청을 비동기로 처리하고, Redis 캐싱을 적용하여 데이터 조회 속도를 획기적으로 개선했습니다.
4단계: 사용자 피드백 기반 검증 및 최적화 (Validation)
베타 테스트 및 설문조사: 기능 구현 완료 후, 실제 학습자를 대상으로 베타 테스트를 진행하고 설문조사를 실시했습니다. 학습 효율성, 힌트의 유용성, UI 편의성 등 다양한 지표를 정량적/정성적으로 수집했습니다.
피드백 반영: 수집된 사용자 피드백을 분석하여 UI/UX를 개선하고, 발견된 버그를 수정하여 서비스의 완성도를 최종적으로 높였습니다. 이를 통해 본 프로젝트가 목표한 '자기주도적 학습 능력 향상'을 실질적으로 달성했음을 검증했습니다.

나. 협업 및 형상 관리 전략

GitHub Flow 기반의 코드 품질 관리
안정적인 지속적 배포(CD)를 위해 GitHub Flow 브랜치 전략을 도입했습니다. main 브랜치는 언제나 배포 가능한 상태(Deployable)를 유지하며, 모든 기능 개발과 버그 수정은 별도의 feature 브랜치에서 진행했습니다.
기능 구현이 완료된 코드는 팀원들의 코드 리뷰를 거친 뒤에만 merge함으로써, 잠재적인 오류를 방지하고 코드 스타일을 통일하여 유지보수성을 확보했습니다.
실시간 커뮤니케이션 및 체계적인 문서화
문서화: Notion을 메인 협업 도구로 활용하여 기획서, 회의록, 기능 정의서 등 프로젝트의 모든 산출물을 체계적으로 기록하고 공유하여 팀원 간 정보 비대칭을 해소했습니다.
실시간 소통: Discord를 활용하여 개발 이슈와 진행 상황을 실시간으로 공유하고, 긴급한 버그나 이슈 발생 시 즉각적으로 대응할 수 있는 핫라인 체계를 구축하여 협업 효율을 높였습니다.

제 2 절 추진체계

1. 프로젝트 계획

세부 개발 내용	세부 추진일정(주)										비 고
	1	2	3	4	5	6	7	8	9	10	
프로젝트 기획 및 요구사항 분석, 아키텍처 설계											기획/설계
MSA 인프라 구축											인프라
Web IDE 구현 및 Judge0 샌드박스 연동											큰 틀 구현
알고리즘 문제 데이터 크롤링 및 DB 구축											데이터
관리자 페이지 및 문제 관리 시스템 개발											백오피스
LangChain 기반 AI 힌트 및 리팩토링 로직 구현											AI Core
개인화 대시보드 및 학습 추천 알고리즘 구현											FE/데이터
테스트케이스 생성 기능 구현											데이터
통합 테스트 및 최종 배포											마무리
분기별 진척률(%)	50%					100%					

2. 프로젝트 실행

개발 과정의 불확실성을 최소화하고 효율성을 극대화하기 위해, 전체 일정을 2주 단위의 단기 개발 주기로 나누어 체계적으로 수행하였습니다. 초기 단계에서는 기술적 난이도가 높은 웹 IDE와 샌드박스 채점 시스템을 우선적으로 구현하여 핵심 기능의 타당성을 검증하는 데 주력했습니다. 이를 통해 AI 힌트 생성 및 리팩토링 기능과 같은 고도화 작업을 단계적으로 통합하는 점진적 개발 전략을 취하여 프로젝트의 리스크를 효과적으로 관리했습니다. 또한, 매주 정해진 시간에 진행 상황과 기술적 장애 요소를 공유하는 데일리 미팅을 통해 팀원 간의 업무 흐름을 긴밀하게 동기화했습니다.

소프트웨어 품질 관리와 안정적인 협업을 위해서는 GitHub Flow 전략을 엄격히 준수하여 실행했습니다. 배포의 안정성을 보장하기 위해 메인 브랜치는 항상 실행 가능한 상태로 유지하였으며, 모든 기능 개발과 버그 수정은 독립된 기능 브랜치에서 진행했습니다. 기능 구현이 완료된 후에는 팀원 간의 상호 코드 리뷰와 승인 절차를 거친 뒤에만 코드를 병합함으로써 잠재적인 오류를 사전에 차단하고 코드 스타일의 일관성을 유지했습니다.

더불어 효율적인 커뮤니케이션 환경을 조성하기 위해 Notion을 메인 워크스페이스로 활용하여 기획서, 기능 정의서 등 프로젝트의 모든 산출물을 중앙에서 관리하여 정보의 비대칭을 해소했습니다. 아울러 Discord와 GitHub를 연동한 실시간 알림 체계를 구축하여, 코드 변경 사항이나 이슈 발생 시 즉각적으로 상황을 인지하고 대응할 수 있는 기민한 협업 문화를 정착시켜 프로젝트를 완수했습니다.

3. 품질 관리

가. 샌드박스기술을 통한 보안 품질 확보에 주력했습니다. 불특정 다수의 사용자가 제출하는 임의의 코드는 무한 루프나 시스템 파일 접근 등 서버에 치명적인 위협이 될 수 있습니다. 이를 방지하기 위해 채점 엔진인 Judge0를 Docker 컨테이너 내부의 격리된 환경에서 실행하도록 구성하여, 악성 코드가 호스트 시스템에 영향을 미치지 못하도록 물리적·논리

적 차단벽을 구축했습니다.

나. 프론트엔드 연동 전 Postman을 활용하여 모든 API 엔드포인트에 대한 다양한 요청 및 응답 시나리오를 시뮬레이션함으로써, 통합 과정에서 발생할 수 있는 인터페이스 오류를 사전에 차단했습니다.

다. AI 생성 데이터에 대한 자동화된 품질 검증 파이프라인을 구축했습니다. LangChain을 통해 알고리즘 문제의 테스트케이스를 생성할 때, LLM의 출력 형식을 엄격한 JSON 포맷으로 강제하여 파싱 오류를 최소화했습니다. 만약 파싱에 실패하거나 데이터베이스 저장 과정에서 오류가 발생할 경우, 시스템이 자동으로 원인을 분석하고 재시도를 수행하는 로직을 탑재하여 학습 콘텐츠의 데이터 무결성을 지속적으로 유지했습니다.

4. 프로젝트 모니터링 및 제어

가. 원활한 협업을 위해 정기 회의 및 자유로운 의사 교환으로 소통의 어려움이 없게 하였습니다. 매주 정기적인 온·오프라인 회의를 통해 각자의 개발 진척도를 공유하였으며, 특히 프론트엔드와 백엔드 연동 과정에서 빈번하게 발생하는 에러나 데이터 형식 불일치 문제를 즉각적인 토론을 통해 해결했습니다. 또한 카카오톡과 디스코드 단체방을 활용하여 개발 중 긴급한 오류나 이슈가 발생했을 때 즉시 상황을 전파하고 공동 대응하는 협업 체계를 유지했습니다.

나. 서버 로그 모니터링을 통한 선제적 장애 대응을 수행했습니다. AWS EC2 서버 내의 Spring Boot 애플리케이션 로그를 주기적으로 분석하여, API 요청 실패(500 Error 등)의 근본 원인을 추적하고 해결했습니다. 또한, 로컬 개발 환경과 배포 환경의 로그를 비교 분석하는 디버깅 과정을 거쳐 시스템의 안정성을 지속적으로 확보했습니다.

제 4 장 프로젝트의 활용 방안 및 기대효과

제 1 절 기대효과 및 수익성

1. 기대효과

저희의 프로젝트는 교육적, 사회적, 사용자 경험 측면에서 유의미한 가치를 창출할 수 있는 잠재력이 있습니다. 우선 교육적 측면에서는 학습자가 AI의 단계별 힌트를 통해 알고리즘의 원리를 스스로 깨우치는 자기주도적 학습 능력을 배양할 수 있을 것으로 기대됩니다. 시각화된 성장 지표는 지속적인 동기 부여를 돕게 합니다. 사회적으로는 시공간과 비용의 제약 없는 AI 튜터 시스템을 통해 고비용의 사교육 의존도를 낮추고, 누구나 양질의 코딩 교육을 받을 수 있는 환경을 조성하여 교육 격차 해소에 기여할 가능성이 있습니다. 사용자 경험 측면에서는 오류 분석 시간을 획기적으로 단축시키고, 단일 화면에서 모든 학습 과정이 이루어지는 통합 환경을 통해 학습 효율성과 몰입도를 극대화할 수 있을 것으로 기대됩니다.

2. 수익성

비즈니스 모델 측면에서는 B2C와 B2B 시장 모두에서 수익을 창출할 수 있는 다양한 가능성을 열어두고 있습니다. 일반 사용자에게는 기본 기능을 무료로 제공하되, 심층 코드 분석이나 맞춤형 커리큘럼과 같은 고도화된 기능을 'Freemium' 모델로 제공하여 부분 유료화를 시도할 수 있습니다. 또한, 대학의 실습 보조 도구나 기업의 채용 및 직무 교육 플랫폼으로 솔루션을 커스터마이징하여 공급하는 B2B/B2G 비즈니스 모델로의 확장 가능성도 검토해 볼 수 있습니다.

제 2 절 기술적 측면

기술적으로는 에듀테크(Edu-Tech) 분야에서 범용 LLM을 특수 도메인에 최적화하여 적용하는 실질적인 노하우를 확보했다는 점에서 의의가 있습니다. 이번 프로젝트를 통해 축적된 프롬프트 엔지니어링 및 튜닝 기술은 향후 수학이나 과학 등 다른 교육 분야로 AI 튜터 시스템을 확장할 때 핵심적인 기반 기술로 활용될 수 있을 것입니다. 또한, 컨테이너 기반의 샌드박스 환경 구축 경험은 향후 고성능 자동 평가 시스템이나 클라우드 기반의 개발 환경 서비스로 발전시킬 수 있는 기술적 토대가 될 것입니다.

제 3 절 경제, 산업적 측면

경제 및 산업적 관점에서는 인적 자원 낭비를 줄이고 교육의 질적 성장을 유도할 수 있을 것으로 예상됩니다. 기업 및 교육 기관에서 코드 리뷰와 멘토링에 투입되는 막대한 리소스를 AI 자동화 시스템이 보조하거나 대체함으로써, 운영 비용을 절감할 수 있는 가능성을 제시합니다. 더 나아가, 단순 암기식 코딩 교육 시장을 논리적 사고력 중심의 시장으로 전환하는 계기를 마련하여, 장기적으로 소프트웨어 산업 전반에 걸쳐 인재들의 역량을 상향 평준화하는 데 긍정적인 영향을 미칠 수 있을 것으로 기대됩니다.

제 4 절 활용 방안

CodeBrainer 플랫폼은 다양한 사용자 층을 대상으로 폭넓게 활용될 수 있는 유연성을 갖추고 있습니다. 취업이나 이직을 준비하는 개인 학습자에게는 실전 감각을 익히고 역량을 강화하는 보조 도구로써 기능할 수 있습니다. 대학이나 전문 교육기관에서는 자료구조 및 알고리즘 수업의 실습 플랫폼으로 도입되어, 과제 채점의 효율성을 높이고 즉각적인 피드백을 제공하는 교육 보조재로 활용될 수 있습니다. 또한, 개발자 커뮤니티나 스터디 그룹에서는 AI 분석 결과를 공유하고 토론하는 소셜 러닝의 장으로 활용되어, 상호 학습을 촉진하는 플랫폼으로 자리 잡을 수 있을 것입니다.

제 5 장 참여인력

제 1 절 업무 분담

[표 3] 참여인력 및 담당분야

NO.	성명	소속			담당분야	참여도(%)
		학과	학번	학년		
1	안승원	컴퓨터AI학부	2023113581	3	frontend/기획	25
2	이승규	컴퓨터AI학부	2023113571	3	backend	25
3	김채령	컴퓨터AI학부	2023113569	3	backend	25
4	성시연	컴퓨터AI학부	2023110482	3	backend/AI	25

제 2 절 팀원 별 수행 성과

안승원:
1. 채점 서버 구현
Judge0 기반 멀티 언어 코드 실행 시스템 구축
Orchestrator 백엔드 서비스 설계 및 구현
코드 제출 → 채점 → 결과 반환 E2E 파이프라인 완성
2. 문제 크롤링 및 테스트케이스 구축
백준 문제 크롤링 스크립트 개발
정답 코드 및 테스트케이스 생성 시스템 구축
문제 데이터 관리 자동화
3. AI 힌트 생성 및 API 연동
Gemini AI 기반 코드 리뷰 시스템 구현
오답 시 AI 힌트 생성 기능 개발
알고리즘 카테고리별 맞춤 피드백 시스템
4. 데이터베이스 구축 및 관리
Supabase 클라우드 DB 설계 및 연동
Supabase Storage 통합
연결 풀 및 성능 최적화
5. 서버 연동 및 배포
Railway/Vercel 클라우드 배포
CloudAMQP RabbitMQ 메시지 큐 설정
CORS 및 환경 변수 관리
6. 테스트케이스 및 코드 수정
코드 제출 검증 로직 구현
버그 수정 및 안정화
보안 취약점 패치
7. 전반적인 프로젝트 기획 및 관리
시스템 아키텍처 설계
팀 협업 및 일정 관리
최종 통합 테스트 및 배포
8. 프론트엔드 전반 구현
난이도별/알고리즘별/기업별 문제 페이지
모의고사 시스템 (세트 선택, 타이머, 결과)
문제 풀이 워크스페이스
마이페이지 및 제출 내역

이승규:
1. 사용자별 데이터 기반 마이페이지 기능 구현 (프론트엔드 & 백엔드)
데이터베이스 최적화 및 인덱싱
프론트엔드 클라이언트 모듈 개발
2. 문제 DB 관리
문제 구조화 및 메타데이터 정리
3. 프로젝트 오류 검출 보조
백엔드/프론트엔드 디버깅
통합 테스트 지원 및 설문 진행

4. 테스트케이스 생성 및 이중점검
.in/.out 파일 생성
자동화 스크립트 작성
이중점검 프로세스
5. 정답 코드 생성 및 이중점검
Judge0 검증
테스트케이스 점검
6. 자료조사 및 PPT 제작과 발표
기술 스택 조사
발표 자료 제작
문서화 작업

김채령:
1.데이터베이스 구축: User 엔티티 설계, JPA Repository 구현, PostgreSQL 연결 설정, 자동 스키마 생성
2.회원가입: 이메일/아이디 중복 체크, 비밀번호 암호화(BCrypt), 유효성 검증, REST API 구현
3.로그인: 이메일/아이디 자동 판단, 비밀번호 검증, JWT 토큰 생성 및 반환, REST API 구현
4.관리자 콘솔: 대시보드 통계 조회, 사용자 목록 조회(페이지네이션), 통계 계산 로직
5.테스트케이스 관리: 테스트케이스 DB 입력

성시연:
1. 문제 데이터 크롤링 및 테스트케이스 자동화
크롤러 개발: Python과 BeautifulSoup을 활용해 BOJ/Solved.ac의 문제 본문, 제약조건, 입출력 형식을 파싱하는 자동화 스크립트 개발.
테스트케이스 시스템 구축: 크롤링 된 예제 데이터와 LLM이 생성한 추가 케이스를 .in / .out 파일로 정규화하여 Storage에 저장하는 파이프라인 구현.
데이터 정합성 관리: 문제 메타데이터와 실제 파일 경로(Storage) 간의 매핑 로직을 자동화하여 수작업 관리 소요 제거.
2. AI 기반 힌트/피드백 생성 및 API 연동
Gemini AI 연동: Google Gemini API를 활용하여 사용자의 오답 코드에 대해 정답을 직접 알려주는 대신 논리적 오류를 지적하는 코드 리뷰 시스템 구현.
단계별 힌트 로직: 알고리즘 난이도(Tier)와 카테고리에 맞춰 1~4단계로 점진적인 힌트를 제공하는 프롬프트 엔지니어링 및 생성 로직 개발.
맞춤형 피드백 제공: 문제 유형(DP, 그리디, 그래프 등)에 따라 차별화된 학습 가이드를 제공하는 자동화 API 구축.
3. 데이터베이스 및 스토리지 인프라 관리
데이터 관리 유틸리티: 중복 데이터 제거, 누락 필드 복구 등 데이터 품질 유지를 위한 유지보수 SQL 및 스크립트 실행.
4. 안정화 및 디버깅: 운영 중 발생하는 연결 타임아웃, 컨테이너 네트워크 이슈 등을 모니터링하고 해결.
5. 프로젝트 관리 및 문서화
보고서 및 산출물 작성: 최종 결과 보고서, 기술 명세서 및 매뉴얼

제 6 장 참고문헌

AI가 학습에 끼치는 악영향:
Georgios P. Georgiou. (n.d.). ChatGPT produces more "lazy" thinkers: Evidence of cognitive engagement decline(arxiv preprint(2025)). n.p..

judge0 활용 방안: Code Execution (Judge0) . (n.d.). https://ce.judge0.com/.