로고, 상징, 폰트, 그래픽이(가) 표시된 사진

자동 생성된 설명

디자인 명세서

과목: 소프트웨어공학개론

담당교수: 이은석

작성자: 김다한, 김민재, 이동신, 진서영, 최지애, 한종승

제출일: 2024년 12월 1일

폰트, 텍스트, 화이트, 로고이(가) 표시된 사진

자동 생성된 설명

**Contents**

**1 Preface**

1.1 Readership 1

* 1. Scope 1
  2. Objective 1
  3. Document Structure 2

1. **Introduction**

2.1 Objectives 3

2.2 Applied Diagrams 3

2.2.1 Used Tools 3

2.2.2 Use Case Diagram 3

2.2.3 Sequence Diagram 4

2.2.4 Class Diagram 4

2.2.5 Context Diagram 4

2.2.6 Entity Relationship Diagram 4

2.2.7 Project Scope 5

2.2.8 References 5

**3 System Architecture - Overall**

3.1 Objectives 6

3.2 System Organization 6

3.2.1. System Diagram 7

3.3 Use Case Diagram 8

**4 System Architecture - Frontend**

4.1 Objectives 9

4.1.1. Profile 9

4.1.2. Problem solving 11

4.1.3. Code Mate 12

**5 System Architecture - Backend**

5.1 Objectives 14

5.2 Overall Architecture 14

5.3 Components 15

5.3.1 User Management System 15

5.3.2 Coding Test System 16

5.3.3 User Vulnerability Analyzing System 18

5.3.4 Question Generating System 20

**6 Protocol Design**

6.1 Objectives 22

6.2 TLS and HTTPS 22

6.3 AJAX 22

6.4 CSRF and CSRF Token 22

6.5 Authentication 23

6.5.1 Register 23

6.5.2 Login 24

6.5.3 Password Reset 25

6.6 Level Test 26

6.7 Coding Test 27

6.8 User Vulnerability Analysis 28

6.9 Coding Learning Report 29

6.11 Admin Page 30

**7 Database Design**

7.1 Objectives 31

7.2 ER Diagram 31

7.2.1 User 33

7.2.2 Problem 33

7.2.3 User Coding History 34

7.2.4 Test Case 34

7.2.5 Test Case 35

7.2.5 Average Result 35

7.3 Relational Schema 36

7.4 SQL DDL 36

7.4.1 User 36

7.4.2 Problem 37

7.4.3 User Coding History 37

7.4.4 Test Case 37

7.4.5 User Diagnostics 38

7.4.5 Average Result 38

**8 Testing Plans**

8.1 Objectives 39

8.2 Testing Policy 39

8.2.1 Development Testing 39

8.2.2 Release Testing 40

8.2.3 User Testing 40

8.2.4 Testing Case 41

**List of Figures**

3.1 Overall System Architecture 6

3.2 Context Diagram – Overall 7

3.3 User Case Diagram 8

4.1 Class diagram – Profile 10

4.2 Sequence diagram – Profile 10

4.3 Class diagram - Problem Solving 11

4.4 Sequence diagram - Problem Solving 12

4.5 Class diagram - Code Mate 13

4.6 Sequence diagram - Code Mate 13

5.1 Overall Architecture 14

5.2 Class Diagram - User Management System 15

5.3 Sequence Diagram - User Management System 16

5.4 Class Diagram - Coding Test System 16

5.5 Sequence Diagram - Coding Test System 17

5.6 Class Diagram – User Vulnerability Analyzing System 18

5.7 Sequence Diagram – User Vulnerability Analyzing System 19

5.8 Class Diagram – Question Generating System 20

5.9 Sequence Diagram – Question Generating System 21

6.1 Table of Register Request 23

6.2 Table of Register Response 23

6.3 Table of Login Request 24

6.4 Table of Login Request 24

6.5 Table of Password Request 25

6.6 Table of Password Request 25

6.7 Table of Level Test Request 26

6.8 Table of Level Test Request 26

6.9 Table of Coding Test Request 27

6.10 Table of Coding Test Request 27

6.11 Table of User Vulnerability Analysis Request 28

6.12 Table of User Vulnerability Analysis Request 28

6.13 Table of Coding Learning Report Request 29

6.14 Table of Coding Learning Report Request 29

6.15 Table of Admin Page Request 30

6.16 Table of Admin Page Request 30

7.1 ER Diagram 32

7.2 ER Diagram, Entity, User 33

7.3 ER Diagram, Entity, Problem 33

7.4 ER Diagram, Entity, User Coding History 34

7.5 ER Diagram, Entity, Test Case 34

7.6 ER Diagram, Entity, User Diagnostics 35

7.7 ER Diagram, Entity, Average Result 35

7.8 Relational Schema 36

7.9 user sql schema 36

7.10 problem sql schema 37

7.11 coding history sql schema 37

7.12 test case sql schema 37

7.13 user diagnostics sql schema 38

7.14 average result sql schema 38

1

**Preface**

본 디자인 명세서는 예상 독자와 문서의 구조, 그리고 각 단원에 대해 설명한다. 그리고 문서의 버전과 각 버전에서 만들어진 변경 사항에 대해 요약한다.

**1.1 Readership**

본 문서는 다음과 같이 독자를 정의한다. 주요 독자는 CODOC의 기획자인 4조 구성원들이며, 소프트웨어공학개론 수업의 교수, 조교, 참여 학생 또한 독자가 될 수 있다.

**1.2 Scope**

CODOC은 코딩테스트 준비 온라인 플랫폼으로 사용자 코드의 취약점 분석, 실시간 피드백, 진단서 생성 등의 기능을 제공하며, 다양한 프로그래밍 언어를 지원한다. 사용자가 자신의 실력을 객관적으로 평가하고 문제 해결 능력을 향상시킬 수 있도록 돕고, 학습 동기를 부여하는 것을 목표로 한다.

**1.3 Objective**

해당 디자인 명세서의 주요 목적은 CODOC 시스템의 기술적 설계에 대한 설명이다. 이 문서는 실습 프로그램의 구현의 기반이 되는 소프트웨어 Front-End, Back-End, Dtatbase 측면에서의 설계를 정의한다. 모든 설계는 앞서 제작된 소프트웨어 요구사항 명세서를 기반으로 작성되었다.

**1.4 Document Structure**

1) Preface: 본 문서의 목적, 예상 독자 및 문서의 구조에 대해 설명한다.

2) Introduction: 본 문서를 작성하는데 사용된 도구들과 다이어그램들, 참고 자료들에 대해 설명한다.

3) Overall System Architecture: 시스템의 전체적인 구조를 Context Diagram, Use-case Diagram, Sequence Diagram을 이용하여 서술한다.

4) System Architecture - Frontend: Frontend 시스템의 구조를 Class Diagram, Sequence Diagram을 이용하여 서술한다.

5) System Architecture - Backend: Backend시스템의 구조를 Sequence Diagram, Class Diagram을이용하여 서술한다.

6) Protocol Design: 클라이언트와 서버의 커뮤니케이션을 프로토콜 디자인을 서술한다.

7) Database Design: 시스템의 Database Requirements를 기반으로 ER Diagram, Relation Schema를 이용하여 시스템의 데이터베이스 디자인을 서술한다.

8) Testing Plan: 시스템을 위한 테스트 계획을 서술한다.

9) Development Plan: 시스템의 구현 계획 및 구현을 위한 개발 도구, 라이브러리 등의 개발 환경을 설명한다.

10) Supporting Information: 본 문서의 작성 및 수정 기록을 기술한다.

2

**Introduction**

디자인 명세서는 프로젝트 구현에 있어서 기반이 될 수 있는 설계를 제공한다. 또한 설계는 앞서 제작된 요구사항 명세서에서 명시된 요구사항에 따른다.

**2.1 Objectives**

이번 챕터에서는 제안한 시스템의 설계에 사용된 다이어그램, 툴, 그리고 개발 범위에 대해 설명한다.

**2.2 Applied Diagrams**

**2.2.1 Used Tools**

웹 기반의 다이어그램 작성 프로그램인 draw.io를 이용하여 시각적으로 정보를 정리하고 표현할 수 있다.

**2.2.2 Use Case Diagram**

Use Case Diagram은 UML에서 시스템이 제공하는 기능 단위를 설명하는 도구로, 개발 팀이 시스템의 기능 요구사항을 시각화하는 데 중점을 둔다. 이 다이어그램은 서로 다른 유스 케이스 간의 관계뿐만 아니라 시스템과 상호작용하는 액터 간의 관계를 포함하여 주요 프로세스를 나타낸다. 유스 케이스 다이어그램은 시스템의 고급 기능과 범위를 설명하는 데 유용하게 사용된다.

**2.2.3 Sequence Diagram**

Sequence Diagram은 특정 유스 케이스 또는 그 일부분의 상세한 흐름을 나타내는 도구이다. 이 다이어그램은 서로 다른 객체 간의 호출 관계를 시간 순서에 따라 보여주며, 각 객체에 대한 다양한 호출을 세부적으로 나타낼 수 있다. 시퀀스 다이어그램은 두 차원으로 표현되며, 세로축은 발생 시간의 순서에 따른 메시지/호출 순서를 표시하고, 가로축은 메시지가 전달되는 객체 인스턴스를 나타낸다. 이러한 구조 덕분에 시퀀스 다이어그램은 매우 직관적이며, 각 클래스의 인스턴스를 다이어그램 상단의 박스에 배치하여 객체를 분류할 수 있다.

**2.2.4 Class Diagram**

Class Diagram은 UML에서 다양한 엔티티(사람, 제품, 데이터) 간의 관계를 표현하는 정적 구조 다이어그램이다. 이 다이어그램은 시스템의 정적 구조를 나타내며, 프로그래머가 처리하는 구현 클래스를 주로 보여준다. 구현 클래스 다이어그램은 논리적 클래스 다이어그램처럼 클래스의 구조를 명확히 하여 시스템의 전반적인 아키텍처를 이해하는 데 기여한다.

**2.2.5 Context Diagram**

Context Diagram은 시스템 또는 시스템의 일부와 환경 간의 경계를 정의하는 도구로, 상호작용하는 엔티티를 보여준다. 이 다이어그램은 시스템의 내부 구조에 대한 세부사항 없이 중앙 시스템을 나타내며, 시스템이 상호작용하는 모든 시스템, 환경 및 활동으로 둘러싸여 있다. 컨텍스트 다이어그램의 목적은 전체 시스템 요구사항 및 제약을 개발할 때 고려해야 할 외부 요인과 이벤트에 집중하는 것이다. 가장 효과적인 컨텍스트 다이어그램은 시스템들이 고차원적으로 어떻게 상호작용하는지를 보여주거나, 시스템 간의 논리적 작업과 상호작용을 설명하는 데 사용된다.

**2.2.6 Entity Relationship Diagram**

Entity Relationship Diagram은 시스템의 데이터베이스 구조를 모델링할 때 사용되는 도구로, 고유한 특성을 가진 엔티티와 그들 간의 관계를 네트워크 형태로 보여준다. 이 다이어그램에서 엔티티 집합은 사각형으로, 속성은 타원으로 표현되며, 엔티티 집합과 속성은 선으로 연결된다. 관계 집합은 마름모로 나타내고, 관계 집합의 매핑 형태는 화살표로 표시하여 엔티티 간의 관계를 시각적으로 이해할 수 있도록 돕는다.

**2.2.7 Project Scope**

본 문서에서 설계하는 코딩 테스트 플랫폼 CODOC은 사용자들이 코딩 능력을 체계적으로 향상시킬 수 있도록 지원하기 위해 기획되었다. 해당 플랫폼은 초보자부터 숙련된 개발자까지 다양한 수준의 사용자들이 자신의 실력을 객관적으로 진단하고 그에 맞는 수준별 문제를 경험할 수 있도록 설계되었다. CODOC이 사용자가 코딩 문제 해결 능력을 향상시키는 것뿐만 아니라, 학습 과정을 모니터링하고 발전 상황을 시각적으로 제공하여 지속적인 자기 개선을 유도하는 역할을 수행하길 기대한다.

**2.2.8 References**

IEEE Std 830-1998 IEEE Recommended Practice for Software Requirements Specifications, In IEEEXplore Digital Library

<http://ieeexplore.ieee.org/Xplore/guesthome.jsp>

3

**System Architecture - Overall**

**3.1 Objectives**

이 챕터에서는 프론트 엔드 설계에서 백 앤드 설계에 이르는 프로젝트 어플리케이션의 시스템 구성에 대해 설명한다.

**3.2 System Organizations**

CODOC 서비스는 모델-뷰-컨트롤러(Model-View-Controller, MVC) 패턴을 기반으로 설계되었다. 이 구조는 프론트엔드와 백엔드 간의 명확한 분리를 통해 사용자와의 상호작용을 최적화한다. 프론트엔드는 사용자 가입, 로그인, 문제 검색, 문제 풀이 및 리뷰 작성 기능을 제공하며, 사용자가 문제를 검색할 경우 요청이 백엔드의 컨트롤러로 전달된다. 사용자가 문제를 풀이할 때 작성한 코드는 백엔드에 저장되며 이후 피드백 시스템을 통해 분석 결과가 제공된다. 또한 LLM을 통해 풀이한 문제의 취약점 분석이 이루어지고 그 결과도 사용자에게 전달됩니다. 이러한 구조는 CODOC이 효과적인 코딩 학습 경험을 제공하는 데 기여한다.

**Figure 3.1:** Overall System Architecture

텍스트, 스크린샷, 도표, 직사각형이(가) 표시된 사진

자동 생성된 설명

**3.2.1 System Diagram**

**Figure 3.2:** Context Diagram - Overall

텍스트, 도표, 평행, 라인이(가) 표시된 사진

자동 생성된 설명

**3.3 Use Case Diagram**

**Figure 3.3:** User Case Diagram

텍스트, 도표, 라인, 평행이(가) 표시된 사진

자동 생성된 설명

4

**System Architecture – Frontend**

**4.1 Objectives**

이 장에서는 프론트엔드 시스템의 구조, 속성 및 기능을 설명하고 CODOC 코딩 학습 시스템에서 각 구성 요소 사이의 관계를 설명한다.

**4.1.1 Profile**

사용자의 정보가 포함된 프로필 클래스의 세부 정보이다. 회원가입을 진행하는 사용자는 프로필에 필수 입력 정보를 입력해야 한다.

**Objectives**

email: 사용자의 이메일

nickname: 사용자의 닉네임

password: 사용자의 비밀번호

level: 사용자의 코딩 수준

language: 사용자가 주로 사용하는 코딩 언어

**Methods**

createID()

getID()

level\_test()

**Class Diagram**

**Figure 4.1:** Class diagram – Profile

**텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명**

**Sequence Diagram**

**Figure 4.2:** Sequence diagram – Profile

**텍스트, 스크린샷, 도표, 폰트이(가) 표시된 사진

자동 생성된 설명**

**4.1.2 Problem solving**

사용자의 문제 해결 정보가 포함된 Problem Solving 클래스의 세부 정보이다. 사용자가 자신의 레벨에 적합한 문제를 해결하고 문제 풀이의 결과를 데이터베이스에 저장한다. 해당 정보는 취약점 분석에 사용된다.

**Attributes**

ID: 사용자의 이메일

password: 사용자의 비밀번호

problem: 사용자가 해결할 문제의 내용

code: 사용자가 작성한 답변

category: 사용자가 해결할 문제의 유형

level: 사용자가 해결할 문제의 레벨

answer: 사용자의 답변과 비교해서 정답인지 판단하는 데 사용되는 정답 데이터

**Class Diagram**

**Figure 4.3:** Class diagram - Problem Solving

**텍스트, 스크린샷, 디자인이(가) 표시된 사진

자동 생성된 설명**

**Sequence Diagrams**

**Figure 4.4:** Sequence diagram - Problem Solving

**텍스트, 도표, 라인, 평행이(가) 표시된 사진

자동 생성된 설명**

**4.1.3 Code Mate**

사용자가 문제 해결 과정에서 어려움을 겪으면 현재 작성중인 코드를 Code Mate(LLM)에 전달하여 현재 코드에서 수정 및 추가해야 할 부분을 알려줌으로써 문제 해결에 도움을 받을 수 있다.

**Attributes**

ID: 사용자의 이메일

code: 사용자가 작성 중인 코드

**Class Diagram**

**Figure 4.5:** Class diagram - Code Mate

**텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명**

**Sequence Diagrams**

**Figure 4.6:** Sequence diagram - Code Mate

**텍스트, 스크린샷, 폰트, 도표이(가) 표시된 사진

자동 생성된 설명**

5

**System Architecture - Backend**

**5.1 Objectives**

이 챕터는 back-end 시스템의 구조에 대해서 기술한다.

**5.2 Overall Architecture**

**Figure 5.1:** Overall Architecture

**텍스트, 스크린샷, 도표, 직사각형이(가) 표시된 사진

자동 생성된 설명**

**5.3 Components**

**5.3.1 User Management System**

**Class Diagram**

**Figure 5.2:** Class Diagram - User Management System

**텍스트, 스크린샷, 도표, 디자인이(가) 표시된 사진

자동 생성된 설명**

**Register System:** 입력된 ID가 중복되지 않았는지 확인하고 중복되지 않았다면 입력받은 정보를 Database에 추가한다.

**Login System**: 입력받은 ID와 Password가 Database에 존재하는지 확인하고, 존재한다면 해당 Class의 정보를 가져온다.

**Sequence Diagram**

**Figure 5.3:** Sequence Diagram - User Management System

텍스트, 도표, 평면도, 기술 도면이(가) 표시된 사진

자동 생성된 설명

**5.3.2 Coding Test System**

**Class Diagram**

**Figure 5.4:** Class Diagram - Coding Test System

**텍스트, 스크린샷, 도표, 디자인이(가) 표시된 사진

자동 생성된 설명**

**Problem System:** 사용자의 Level에 맞는 문제를 제공하거나 사용자가 선택한 문제를 제공한다.

**Testcase System:** 해당 문제에 대한 테스트케이스를 통해서 사용자가 작성한 코드가 주어진 Input에 대해서 기대되는 Output을 출력하는지 확인하고 이를 통해서 성공 실패 여부를 출력한다.

**Submission System:** 채점 결과를 제출하고 이에 대한 피드백을 요청한다.

**Feedback System:** 제출한 코드에 대한 피드백을 생성해서 사용자에게 전달한다. 그리고 이 피드백을 DB에 저장한다.

**Sequence Diagram**

**Figure 5.5:** Sequence Diagram - Coding Test System

텍스트, 도표, 라인, 스크린샷이(가) 표시된 사진

자동 생성된 설명

**5.3.3 User Vulnerability Analyzing System**

**Class Diagram**

**Figure 5.6:** Class Diagram – User Vulnerability Analyzing System

**텍스트, 스크린샷, 도표, 소프트웨어이(가) 표시된 사진

자동 생성된 설명**

**Submission System:** 사용자가 풀이한 문제, 코드, 채점 결과를 제출한다.

**Vulnerability Analyzing System**: 사용자의 submission에 대해 LLM이 취약점을 분석하게 한다.

**Coding Learning Report System:** 사용자가 풀이한 모든 문제에 대한 문제 풀이 분석 결과와 추천 문제를 전달한다.

**Coding Learning Note System:** 사용자가 풀이한 문제 중 사용자가 확인하기를 원하는 문제에 대한 문제 풀이 결과, 분석 결과, 참고 문서 링크를 제공한다.

**Sequence Diagram**

**Figure 5.7:** Sequence Diagram – User Vulnerability Analyzing System

**스크린샷, 도표, 텍스트, 디자인이(가) 표시된 사진

자동 생성된 설명**

**5.3.4 Question Generating System**

**Class Diagram**

**Figure 5.8:** Class Diagram – Question Generating System

**텍스트, 스크린샷, 도표, 디자인이(가) 표시된 사진

자동 생성된 설명**

**Provide Problem System:** 사용자가 요청한 문제를 제공한다. DB에 해당 문제가 없을 경우, LLM이 기존 문제를 참고하여 새로운 문제를 만들 수 있게 한다.

**Problem Management System:** 관리자가 문제 관리를 할 수 있도록 LLM이 생성한 문제들과 각각의 문제에 대한 check state를 제공한다. check state가 0인 경우 관리자가 문제를 확인하지 않은(확인할 필요가 있는) 상태이고, 1인 경우 관리자가 문제를 확인한 상태이다.

**Sequence Diagram**

**Figure 5.9:** Sequence Diagram – Question Generating System

스크린샷, 도표, 디자인이(가) 표시된 사진

자동 생성된 설명

6

**Protocol Design**

**6.1 Objectives**

이 챕터는 front-end 애플리케이션과 서버가 어떤 프로토콜로 상호작용하는 지를 기술한다. 또한 각 인터페이스가 어떻게 정의되어 있는지 기술한다.

**6.2 TLS and HTTPS**

TLS(Transport Layer Security)는 웹사이트와 브라우저 사이(또는 두 서버 사이)에 전송되는 데이터를 암호화하여 인터넷 연결을 보호하기 위한 표준 기술이다. HTTPS(HyperText Transfer Protocol over Transport Layer Security)는 월드 와이드 웹 통신 프로토콜인 HTTP를 TLS위에서 패킹하여 강화된 보안을 제공하는 버전이다.

**6.3 AJAX**

AJAX란 비동기 자바스크립트와 XML(Asynchronous JavaScript And XML)을 말한다. 간단히 말하면, 서버와 통신하기 위해 XMLHttpRequest 객체를 사용하는 것을 말한다. JSON, XML, HTML 그리고 일반 텍스트 형식 등을 포함한 다양한 포맷을 주고받을 수 있다. AJAX의 강력한 특징은 페이지 전체를 새로고침하지 않고서도 수행되는 비동기성이다. 이러한 비동기성을 통해 사용자의 Event가 있으면 전체 페이지가 아닌 일부분만을 업데이트 할 수 있게 해준다.

**6.4 CSRF and CSRF Token**

CSRF 공격(Cross Site Request Forgery)은 웹 어플리케이션 취약점 중 하나로 인터넷 사용자(희생자) 가 자신의 의지와는 무관하게 공격자가 의도한 행위(수정, 삭제, 등록 등)를 특정 웹사이트에 요청하게 만드는 공격이다.

CSRF Token CSRF 공격을 방지하기 위해 서버에서 난수 토큰을 생성하여 서버에 요청을 주는 페이지에 부여하여 정당하지 않은 페이지에서 오는 요청을 막는 기법이다.

**6.5 Authentication**

**6.5.1 Register**

**Request**

**Figure 6.1:** Table of Register Request

| Attribute | Detail | |
| --- | --- | --- |
| Protocol | HTTPS | |
| Method | POST | |
| Request Body | Email | User email |
| Name | User name |
| Password | User password |
| CSRF Token | CSRF Token issued by server |

**Response**

**Figure 6.2:** Table of Register Response

|  |  |  |
| --- | --- | --- |
| Attribute | Detail | |
| Protocol | HTTPS | |
| Success Code | 200 OK | |
| Failure Code | HTTP error code = 400 (Bad Request, overlap) | |
| Success Response Body | Session ID | Session ID to access |
| Message | Success message |
| Failure Response Body | Message | Failure message |

**6.5.2 Login**

**Request**

**Figure 6.3:** Table of Login Request

|  |  |  |
| --- | --- | --- |
| Attribute | Detail | |
| Protocol | HTTPS | |
| Method | POST | |
| Request Body | Email | User email |
| Password | User password |
| CSRF Token | CSRF Token issued by server |

**Response**

**Figure 6.4:** Table of Login Request

|  |  |  |
| --- | --- | --- |
| Attribute | Detail | |
| Protocol | HTTPS | |
| Success Code | 200 OK | |
| Failure Code | HTTP error code = 400 (Bad Request, overlap) | |
| Success Response Body | Session Token | Token for ID session |
| Message | Success message |
| Failure Response Body | Message | Failure message |

**6.5.3 Password Reset**

**Request**

**Figure 6.5:** Table of Password Request

|  |  |  |
| --- | --- | --- |
| Attribute | Detail | |
| Protocol | HTTPS | |
| Method | POST | |
| Request Body | Email | User email |
| CSRF Token | CSRF Token issued by server |

**Response**

**Figure 6.6:** Table of Password Request

|  |  |  |
| --- | --- | --- |
| Attribute | Detail | |
| Protocol | HTTPS | |
| Success Code | 200 OK | |
| Failure Code | HTTP error code = 400 (Bad Request, overlap) | |
| Success Response Body | Message | Success message |
| Action | Send password change email |
| Failure  Response Body | Message | Failure message |

**6.6 Level Test**

**Request**

**Figure 6.7:** Table of Level Test Request

|  |  |  |
| --- | --- | --- |
| Attribute | Detail | |
| Protocol | HTTPS | |
| Method | POST | |
| Request Body | Code | User-written code in the text editor | |
| Language | Programming language selected by the user | |
| User ID | Unique identifier of the user | |
| CSRF Token | CSRF Token issued by server | |

**Response**

**Figure 6.8:** Table of Level Test Request

|  |  |  |
| --- | --- | --- |
| Attribute | Detail | |
| Protocol | HTTPS | |
| Success Code | 200 OK | |
| Failure Code | HTTP error code = 400 (Bad Request, overlap) | |
| Success Response Body | Message | Success message |
| Result | Result of code execution and evaluation |
| Failure Response Body | Message | Failure message |

**6.7 Coding Test**

**Request**

**Figure 6.9:** Table of Coding Test Request

|  |  |  |
| --- | --- | --- |
| Attribute | Detail | |
| Protocol | HTTPS | |
| Method | POST | |
| Request Body | Code | User-written code in the text editor |
| Language | Programming language selected by the user |
| Problem ID | Unique identifier for the test problem |
| User ID | Unique identifier of the user |
| CSRF Token | CSRF Token issued by server |

**Response**

**Figure 6.10:** Table of Coding Test Request

|  |  |  |
| --- | --- | --- |
| Attribute | Detail | |
| Protocol | HTTPS | |
| Success Code | 200 OK | |
| Failure Code | HTTP error code = 400 (Bad Request, overlap) | |
| Success Response Body | Message | Success message |
| Result | Test results and detailed feedback |
| Failure Response Body | Message | Failure message |

**6.8** **User Vulnerability Analysis**

**Request**

**Figure 6.11:** Table of User Vulnerability Analysis Request

|  |  |  |
| --- | --- | --- |
| Attribute | Detail | |
| Protocol | HTTPS | |
| Method | POST | |
| Request Body | User ID | Unique identifier of the user |
| Code ID | Identifier for the submitted code |
| CSRF Token | CSRF Token issued by server |

**Response**

**Figure 6.12:** Table of User Vulnerability Analysis Request

|  |  |  |
| --- | --- | --- |
| Attribute | Detail | |
| Protocol | HTTPS | |
| Success Code | 200 OK | |
| Failure Code | HTTP error code = 400 (Bad Request, overlap) | |
| Success Response Body | Message | Success message |
| Result | Vulnerability analysis information |
| Failure Response Body | Message | Failure message |

**6.9** **Coding Learning Report**

**Request**

**Figure 6.13:** Table of Coding Learning Report Request

|  |  |  |
| --- | --- | --- |
| Attribute | Detail | |
| Protocol | HTTPS | |
| Method | POST | |
| Request Body | User ID | Unique identifier of the user |
| Code ID | Identifier for the submitted code |
| CSRF Token | CSRF Token issued by server |

**Response**

**Figure 6.14:** Table of Coding Learning Report Request

|  |  |  |
| --- | --- | --- |
| Attribute | Detail | |
| Protocol | HTTPS | |
| Success Code | 200 OK | |
| Failure Code | HTTP error code = 400 (Bad Request, overlap) | |
| Success Response Body | Message | Success message |
| Result | Evaluation result value |
| Recommended problem list |
| Failure  Response Body | Message | Failure message |

**6.11 Admin Page**

**Request**

**Figure 6.15:** Table of Admin Page Request

|  |  |  |
| --- | --- | --- |
| Attribute | Detail | |
| Protocol | HTTPS | |
| Method | POST | |
| Request Body | Problem ID | Unique identifier of the problem |
| CSRF Token | CSRF Token issued by server |

**Response**

**Figure 6.16:** Table of Admin Page Request

|  |  |  |
| --- | --- | --- |
| Attribute | Detail | |
| Protocol | HTTPS | |
| Success Code | 200 OK | |
| Failure Code | HTTP error code = 400 (Bad Request, overlap) | |
| Success Response Body | Message | Success message |
| Result | Save modified coding problems |
| Failure Response Body | Message | Failure message |

7

**Database Design**

**7.1 Objectives**

이 장에서는 시스템의 데이터 구조와 해당 구조가 데이터베이스에서 어떻게 구현되어 있는지에 대해 설명한다. 먼저 ER 다이어그램을 통해 entity와 그 관계를 식별한다. 그런 다음 관계형 스키마 및 SQL DDL (Data Definition Language)을 작성한다.

**7.2 ER Diagram**

CODOC은 총 5가지 entity로 이루어져 있다: User, Problem, User Coding History, User Diagnostics, Average Result. ER-Diagram은 entity간의 관계, 그리고 entity와 attribute의 관계를 다이어그램으로 설명한다. 각 entity의 primary key는 밑줄로 표시되어 있다. 각 entity마다 대응되는 개수는 entity를 연결하는 선 주변에 표기되어 있어 확인 가능하다.

**Figure 7.1:** ER Diagram

스크린샷, 텍스트, 흑백, 그래픽이(가) 표시된 사진

자동 생성된 설명

**7.2.1 User**

User는 Codoc을 사용하는 사용자를 의미한다. User Entity는 id, email, nickname, password, level, language를 attribute로 가지며, id가 primary key이다. 이 attribute들은 회원 가입 당시 사용자가 기입한 정보로 level과 language는 사용자의 코딩 능력치에 대한 정보를 담고 있다. User은  User Diagnostics와는 1:1, Coding History와는 1:N의 관계를 가진다.

**Figure 7.2:** ER Diagram, Entity, User

**도표, 라인, 텍스트, 원이(가) 표시된 사진

자동 생성된 설명**

**7.2.2 Problem**

Problem Entity는 사용자에게 제공하는 문제가 저장되어 있다. 별도의 id를 Primary Key로 가지고 있으며 prob\_description에 문제 내용이 적혀있다. Problem Entity는 User Coding History, Test Case와는 1:N의 관계를, Average Result와는 1:1의 관계를 가지고 있다.

**Figure 7.3:** ER Diagram, Entity, Problem

**도표, 라인, 스케치, 원이(가) 표시된 사진

자동 생성된 설명**

**7.2.3 User Coding History**

User coding history는 사용자가 문제 풀이 과정에서 코드를 실행해볼 때 마다 제출한 코드와 결과값이 저장된다. 해당 결과들은 사용자의 취약점을 분석할 때 사용된다. User와 Problem 양쪽 모두와 1:N 관계를 가지고 있다.

**Figure 7.4:** ER Diagram, Entity, User Coding History

**텍스트, 도표, 라인, 스크린샷이(가) 표시된 사진

자동 생성된 설명**

**7.2.4 Test Case**

Test Case Entity는 사용자가 문제를 해결할 때 제출한 답안의 정답 여부를 체크하는데 사용된다. 저장된 input 값을 집어넣었을 때 결과 값을 Test Case Entity에 저장된 output값과  비교해서 정답 여부를 판단한다. Problem Entity와 1:N의 관계를 가지고 있다.

**Figure 7.5:** ER Diagram, Entity, Test Case

**도표, 텍스트, 라인, 그림이(가) 표시된 사진

자동 생성된 설명**

**7.2.5 Test Case**

User Diagnostics Entity는 사용자의 취약점을 clean\_code, jsdoc, memory\_use, time, time\_complexity, execution\_num 6가지 데이터로 나누어서 저장하고 있다. 해당 데이터들은 애플리케이션이 사용자에게 취약점 보완을 위한 문제를 추천해 줄 때 사용된다. User Entity와 1:1관계를 가지고 있다.

**Figure 7.6:** ER Diagram, Entity, User Diagnostics**텍스트, 도표, 스크린샷, 라인이(가) 표시된 사진

자동 생성된 설명**

**7.2.5 Average Result**

Average Result Entity에는 Codoc 애플리케이션을 사용하는 모든 사용자들이 특정 문제를 풀이했을 때의 결과를 분석하여 저장하고 있다. 해당 Entity에 저장된 데이터들은 한 사용자가 문제를 풀었을 때 해당 답안의 수준의 상대적으로 어느 정도인지 파악하기 위해 필요하다. Problem Entity와 1:1관계를 가지고 있다.

**Figure 7.7:** ER Diagram, Entity, Average Result**텍스트, 도표, 그림, 스케치이(가) 표시된 사진

자동 생성된 설명**

**7.3 Relational Schema**

**Figure 7.8:** Relational Schema

**스크린샷, 텍스트, 디자인이(가) 표시된 사진

자동 생성된 설명**

**7.4 SQL DDL**

**7.4.1 User**

**Figure 7.9:** user sql schema

**텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명**

**7.4.2 Problem**

**Figure 7.10:** problem sql schema

**텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명**

**7.4.3 User Coding History**

**Figure 7.11:** coding history sql schema

**텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명**

**7.4.4 Test Case**

**Figure 7.12:** test case sql schema

**텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명**

**7.4.5 User Diagnostics**

**Figure 7.13:** user diagnostics sql schema

**텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명**

**7.4.5 Average Result**

**Figure 7.14: average result sql schema**

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

8

**Testing Plans**

**8.1 Objectives**

본 장에서는 CODOC 서비스에 대한 테스트 계획을 설명하며, 개발 테스트, 릴리스 테스트 및 사용자 테스트의 세 가지 주요 하위 그룹을 포함한다. 이러한 테스트의 목적은 애플리케이션의 잠재적 오류와 결함을 감지하고, CODOC의 완성도를 높여 사용자에게 안정적인 시스템을 제공하는 것이다.

**8.2 Testing Policy**

**8.2.1 Development Testing**

Development Testing은 CODOC 시스템의 개발 과정에서 발생할 수 있는 오류를 감지하고 수정하기 위한 단계이다. 이 단계에서는 프론트엔드와 백엔드 간의 충돌이 발생할 수 있으므로, 데이터 흐름 검토 및 코드 리뷰를 통해 소프트웨어의 성능, 신뢰성, 보안을 보장하는 데 집중한다.

**Performance**

* 로그인/로그아웃 요청은 2초 이내에 완료되어야 한다.
* 문제 채점 결과는 2초 이내에 데이터베이스에 저장되고 사용자에게 알려져야 한다.
* 계산된 결과는 3초 내로 데이터베이스에 저장되고 사용자에게 나타낼 수 있어야 한다.
* 사용자의 회원가입 및 계정 정보 저장은 각각 2초 이내로 완료되어야 한다.
* CODOC 시스템은 최대 10,000명의 사용자 계정을 관리할 수 있어야 한다.
* 코드메이트의 답장 시간은 3초 이내에 이루어져야 한다.
* 문제 완료 시 취약점 갱신은 3초 이내에 이루어져야 한다.
* 사용자가 오답 노트를 추가하는 데 소요되는 시간은 3초 이내여야 한다.
* 진단서 작성은 5초 이내에 완료되어야 한다.
* 사용자 레벨 반영은 문제 풀이 후 3초 이내에 이루어져야 한다.
* 문제를 데이터베이스에서 불러오는 시간은 3초 이내여야 한다.
* 새로운 문제 생성 시간은 3초 이내로 설정되어야 한다.

**Reliability**

CODOC 시스템이 오류 없이 안전하게 작동하기 위해서는 하위 구성 시스템이 정상적으로 작동하고, 전체 시스템으로 통합되는 과정에서 오류를 반복적으로 확인하고 수정해야 한다. 이를 위해 개발 단계에서부터 순차적으로 테스트를 진행한다.

**Security**

CODOC 시스템은 사용자 개인정보 보호에 집중해야 하며, 사용자들은 타인의 정보에 접근할 수 없도록 해야 한다. 외부 시스템의 데이터베이스 접근을 차단하고, 사용자 ID 및 비밀번호는 암호화되어야 한다.

**8.2.2 Release Testing**

Release Testing은 CODOC 시스템의 배포를 위한 중요한 단계이다. 소프트웨어의 업데이트된 버전을 테스트하여 최종 상태로 배포되기 전, 작동에 하자가 없는지 확인해야 한다. 배포 과정은 알파 버전에서 시작하여 초기 개발 단계에서 기능이 정상 작동하는지를 확인한 후, 베타 버전을 배포하여 보다 넓은 사용자 그룹의 피드백을 받는다. 이 과정에서 사용자의 다양한 사용 시나리오를 통해 발생할 수 있는 오류를 검토하고, 사용자 요구 사항을 충족하는지 확인하여 최종 제품의 품질을 보장한다.

**8.2.3 User Testing**

User Testing은 실제 사용자가 CODOC 소프트웨어를 사용하는 과정을 통해 진행된다. 수업을 수강하는 학생들과 개발 팀원을 대상으로 베타 버전을 배포하고, 다양한 시나리오를 수행하며 오류가 발생하는지를 검토한다. 참가자들은 시스템을 사용하면서 발생할 수 있는 문제점이나 개선점을 기록하고, 이를 분석하여 시스템 품질 향상을 위한 기초 자료로 활용한다.

**8.2.4 Testing Case**

테스트 케이스는 성능, 신뢰성, 보안에 초점을 맞추어 설계된다. 각 측면에서 최소 5개 이상의 테스트 케이스와 시나리오를 통해 CODOC 소프트웨어를 테스트하고, 그 결과를 평가 보고서로 작성한다. 성능 테스트 케이스는 사용자의 코드 처리 시간, 데이터 저장 시간, 로그인 및 로그아웃 시간을 측정하여 성능 지표를 평가하고, 신뢰성 테스트 케이스는 시스템의 모듈별 기능이 정상 작동하는지 점검한다. 보안 테스트 케이스는 데이터 암호화, 접근 제어 및 취약점 분석의 효과성을 검토하여 시스템의 보안 수준을 평가한다. 이러한 테스트 계획은 CODOC이 안정적이고 신뢰할 수 있는 코딩 학습 환경을 제공하는 데 기여할 것이다.