디자인 명세서

코드 탄소 배출량 측정 시스템

Team 6

김선우, 설현원, 이찬구, 이현민, 조재범, 한수민

[여기에 입력]

디자인 명세서

코드 탄소 배출량 측정 시스템

Team 6

김선우, 설현원, 이찬구, 이현민, 조재범, 한수민

Instructor: 이은석

Document Date: 19 Nov, 2023

Faculty: SungKyunKwan University

[여기에 입력]

Contents

1. Purpose 1
1.1 Readership 1
1.2 Scope 1
1.3 Objective 1
1.4 Document Structure 2
2. Introduction 3
2.1 Objectives 3
2.2 Applied Diagrams 3
2.2.1 Used Tools 3
2.2.2 Diagram Types 3
2.2.2.1 Context Diagram 3
2.2.2.2 Use Case Diagram 4
2.2.2.3 Class Diagram 4
2.2.2.4 Sequence Diagram 5
2.2.2.5 Entity Relationship Diagram 5
2.2.3 Project Scope 5
2.2.4 References 6
3. System Architecture - Overall 7
3.1 Objectives 7
3.2 System Organization 7
3.2.1 System Diagram 8
3.3 Use Case Diagram 8
4. System Architecture - Frontend 9
4.1 Objectives 9
4.1.1 User 9
4.1.2 Carbon Analysis 11
4.1.3 Ranking 12
5. System Architecture - Backend 14
5.1 Objectives 14
5.2 Subcomponents 15
5.2.1 User Management System 15
5.2.2 User Ranking System 16
5.2.3 Carbon Analysis System 17
6. Protocol Design 19
6.1 Objectives 19

6.2 Axios 19
6.3 HTTP 19
6.4 Authentication 20
6.4.1 Login 20
6.5 User Ranking 21
6.6 Carbon Calculating 22
7. Database Design 23
7.1 Objectives 23
7.2 ER Diagram 23
7.2.1 User 24
7.2.2 Carbon 24
7.2.3 Sample 25
7.3 Relational Schema 25
7.4 SQL DDL 26
7.4.1 User 26
7.4.2 Carbon 26
7.4.3 Sample 26
8. Testing Plan 27
8.1 Objectives 27
8.2 Testing Policy 27
8.2.1 Development Testing 27
8.2.2 Release Testing 28
8.2.3 User Testing 28
9. Development Plan 29
9.1 Objectives 29
9.2 Development Environment 29
9.2.1 Github 29
9.2.2 Github Action 29
9.2.3 AWS EC2 30
9.2.4 VSCode 30
9.2.5 JavaScript 30
9.3 Frontend Environment 31
9.3.1 React 31
9.3.2 Next.js 31
9.4 Backend Environment 32
9.4.1 Express.js 32
9.4.2 MySQL 32

9.5 Constrains 33
9.6 Assumptions and Dependencies 33
10. Supporting Information 34
10.1 Software Requirements Specification 34
10.2 Document History 34

List of Figures

3.1 Overall System Architecture 7
3.2 System Diagram 8
3.3 Use Case Diagram 8
4.1 Class Diagram - User 9
4.2 Sequence Diagram - User 10
4.3 Class Diagram - Carbon 11
4.4 Sequence Diagram - Carbon 12
4.5 Class Diagram - Ranking 12
4.6 Sequence Diagram - Ranking 13
5.1 Overall Backend Architecture 14
5.2 Class Diagram – User Management System 15
5.3 Sequence Diagram – User Management System 15
5.4 Class Diagram – User Ranking System 16
5.5 Sequence Diagram – User Ranking System 16
5.6 Class Diagram – Carbon Analysis System 17
5.7 Sequence Diagram – Carbon Analysis System 18
6.1 Table of Login Request 20
6.2 Table of Login Response 20
6.3 Table of User Ranking Request 21
6.4 Table of User Ranking Response 21
6.5 Table of Carbon Calculating Request 22
6.6 Table of Carbon Calculating Response 22
7.1 ER Diagram, Entity, User, Carbon, Sample 23
7.2 User ER Diagram 24
7.3 Carbon ER Diagram 24
7.4 Sample ER Diagram 25
7.5 Relational Schema 25
7.6 User SQL schema 26
7.7 Carbon SQL schema 26
7.8 Sample SQL schema 26
9.1 Github Logo 29
9.2 Github Action Logo 29
9.3 AWS EC2 Logo 30

9.4 VSCode Logo 30
9.5 JavaScript Logo 30
9.6 React Logo 31
9.7 Next.js Logo 31
9.8 Express.js Logo 32
9.9 MySQL Logo 32

1

Purpose

1.1. Readership

본 문서는 시스템 개발자(Team 6)와 소프트웨어 공학 개론의 교수, 조교 학생을 위해 작성되었다. 시스템 개발자는 크게 Frontend, Backend 개발자로 나뉜다.

1.2. Scope

본 소프트웨어 제품은 Java 코드의 탄소 배출량 측정 웹 플랫폼이다. 본 제품은 사용자의 Java 코드의 탄소 배출량을 측정하고 이를 시각적으로 표시하는 기능을 포함한다. 본 제품의 목적은 환경 친화적 개발을 통하여 소프트웨어 개발 생태계의 지속 가능성을 높이고자함에 있다.

1.3. Objective

본 디자인 명세서의 주요 목적은 코드 탄소 배출량 측정 시스템의 기술적 설계(design) 에 대한 설명이다. 이 문서는 시스템 개발의 기반이 되는 Frontend, Backend, Database 측면에서 설계를 정의한다. 모든 설계는 앞서 제작된 Software Requirements Specification 의 내용을 기반으로 작성되었다.

1.4. Document Structure

1. Purpose: 본 문서의 목적 및 문서의 구조에 대해 설명한다.
2. Introduction: 본 시스템의 전반적 구조, frontend, backend, database design 에 사용된 diagram 에 대해 설명한다.
3. System Architecture – Overall: 본 프로젝트의 전반적인 시스템 구성에 대해 설명한다.
4. System Architecture – Frontend: Frontend 시스템의 구조, 속성 및 기능과 각 구성 요소의 관계를 설명한다.
5. System Architecture – Backend: Backend 시스템의 구조, 속성 및 기능과 각 구성 요소의 관계를 설명한다.
6. Protocol Design: Frontend application 과 server 가 어떤 프로토콜로 상호 작용하는지 설명한다.
7. Database Design: 시스템의 데이터베이스의 구조와 관계성에 대해 설명한다.
8. Testing Plan: 시스템의 결함과 오류를 찾아낼 수 있는 테스트 계획을 설명한다.
9. Development Plan: 시스템의 구현 계획 및 개발 언어, 도구 등에 대한 개발 환경을 설명한다.
10. Supporting Information: 본 문서의 작성 및 수정 기록을 설명한다.

2

Introduction

2.1. Objectives

이번 장에서는 overall architecture, frontend, backend, database design 에 사용된 diagram 들에 대해 설명한다. Diagram 을 도식한 tool, 그리고 각 diagram 의 목적과 문법에 대해 기술하고, 마지막으로 개발 범위를 간단히 명시한다.

2.2. Applied Diagrams

2.2.1. Used Tools

웹 기반 디자인 툴인 Figma 를 통해 기본적인 도형이나 아이콘을 생성할 수 있는 도구로 다이어그램을 그릴 수 있다.

2.2.2. Diagram Types

2.2.2.1. Context Diagram

Context diagram 은 높은 추상도를 가진 Data Flow Diagram 의 일종이다. 시스템의 overall architecture 를 알기 쉽게 요약하기 때문에, 세부 개발 정보를 필요로 하지 않는 stakeholder 들이 주요 대상으로 작성된다. Context diagram 이 표시한 overall system architecture 에서는 크게 다음 두 가지 정보를 기대할 수 있다.

1. 시스템과 외부 entity 간의 관계
2. 시스템을 구성하는 architectural model

본 문서에서는 코드 탄소 배출량 측정 시스템의 overall architecture 를 기술하기 위해 사용됐다. 해당 시스템의 overall architecture 가 client-server model 로 정의됨과 함께 View, Server, Model entity 간의 관계를 확인할 수 있다.

2.2.2.2. Use Case Diagram

Use case Diagram 은 시스템이 제공해야 하는 기능과 서비스를 기술하는 UML 의 behavioral diagram 이다. 특히, use case diagram 은 시스템의 기능들을 시스템과 외부 entity 의 상호작용으로서 설명하기 때문에, 내부 로직보다는 외부 관점에서의 기능을 표현하는데 그 목적이 있다.

본 문서에서는 코드 탄소 배출량 측정 시스템이 제공하는 서비스와 기능을 기술하기 위해 사용됐다. 타원은 use case 로서 시스템이 제공하는 기능이다. use case 들을 둘러싸는 직사각형은 시스템이 제공하는 기능의 범위가 직사각형 내부의 use case 들임을 명시한다. 사람 기호는 actor 로서 해당 시스템에서는 "user"와 "system"와 같은 actor 를 제시했다. 마지막으로 직선은 actor 와 use case 의 관계를 표시한다.

2.2.2.3. Class Diagram

Class diagram 은 시스템을 구성하는 클래스와 각 속성들을 식별하고, 클래스 간의 관계를 표현하는 UML 의 structural diagram 이다. 시스템의 정적인 구조를 식별할 필요가 있는 개발자들이 주요 대상 중 하나이다.

본 문서에서는 시스템 frontend 에 위치한 class "user", "carbon", "ranking"과 backend 에 위치한 subcomponent "user management system", "carbon management system", "ranking management system"을 기술하기 위해 사용됐다. Diagram 의 직사각형들은 각 class 를 의미하고, 첫번째 칸은 class 의 이름을, 두번째 칸은 class 의 속성을, 마지막 칸은 class 의 method 를 포함한다. 다음의 규칙이 적용되어 기술된다.

1. 속성과 method 는 접근제어자가 접두사로서 붙는다.
   (public: +, private: -, protected: #, package: ~)
2. Colon(:)뒤에 type 이 표시된다.
3. Guillemet(<<>>)으로 interface, 추상 클래스 여부가 표시된다.

마지막으로 class 간 직선은 class 간의 관계를 의미한다. 실선으로 연관관계, 속이 빈 화살표로 상속 관계 등을 표현할 수 있다.

Class diagram 은 개념, 명세, 구현 단계를 거치며, 코드 개발에 직접적인 영향을 미친다. 특히 객체지향 프로그래밍언어를 사용할 때, class diagram 가 큰 도움이 된다. 그러나, 실제로 구현될 소스코드와 일부 차이가 있을 수 있으며, 기술된 요소의 해석 또한 달라질 수 있음을 미리 알린다.

2.2.2.4. Sequence Diagram

Sequence diagram(혹은 event diagram)은 시스템의 object 들이 서로 어떻게 상호작용하는지 시간 순서로 표현하는 UML 의 behavioral diagram 이다. 특히, object 들이 상호작용할 때 주고받아야 하는 message 를 명확히 알 수 있기 때문에, 특정 시스템 event 의 scenario 역할을 한다.

본 문서에서는 시스템 frontend 와 backend 에서 사용되는 class 들의 scenario 를 기술하기 위해 사용됐다. 예를 들어 Frontend part 에서는 user class 의 user, server, client 간 상호작용이 기술됐다.

Diagram 의 최하단에 각 객체가 표시되며, 각 객체의 위로 수직선 lifeline 이 표시된다. 수직선의 점선은 시간의 흐름을 나타내며, 상호작용하는 구간은 점선이 아닌 직사각형 막대로 대체된다. 마지막으로 각 객체의 lifeline 사이 화살표는 상호작용하는데 주고받는 message 를 의미한다. (이 때, 화살표는 자신의 lifeline 으로 회귀할 수 있으며, 이런 표시는 해당 객체가 작업을 자체적으로 처리했음을 의미한다.)

2.2.2.5. Entity Relationship Diagram

ER Diagram 은 시스템이 사용하는 데이터를 entity 라는 단위로 구체화하여 기술하는 diagram 이다. 특히, entity 와 그들의 attribute, 그리고 entity 간의 relationship 이 중점으로 기술된다. 본 문서에서는 코드 탄소 배출량 측정 시스템 데이터베이스의 논리적 구조를 설명하는데 사용됐다. 시스템에서 식별된 entity 로는 "user", "carbon", "sample"이 있다.

본 문서에서 ER diagram 은 UML class notation 으로 작성되었으며, entity 간 직선은 각각 다음의 관계를 의미한다.

0…1: zero or one | 1: one and only one | zero or more: 0…* | one or more 1…*

2.2.3. Project Scope

본 문서에서 설계하는 코드 탄소 배출량 측정 시스템은 사용자들이 java 소스코드의 탄소배출량을 쉽게 측정해 볼 수 있는 기능을 제공한다. 또한, 사용자들은 제출한 소스코드에 상응하는 환경적 효과를 확인할 수 있다. 사용자들은 Runtime 을 기반으로 한 복잡한 탄소배출량 계산식을 몰라도, 웹 페이지에 소스코드를 입력하는 것만으로 탄소배출량을 측정할 수 있는 것이다. 궁극적으로는 본 시스템을 통해, java 개발자들이 친환경적 소스코드 개발을 하는데 도움을 주는 것이 목표다.

2.2.4. References

- IEEE Std 830-1998 IEEE Recommended Practice for Software Requirements
- Specifications, In IEEE Xplore Digital Library
- https://www.uml-diagrams.org/
- https://www.omg.org/spec/UML/

3

System Architecture - Overall

3.1. Objectives

이번 장에서는 프로젝트 전반적인 시스템 구성에 대해 설명한다.

3.2. System Organization

이 서비스는 client-server 모델을 사용하며, 사용자에게 보여지는 view 에서는 사용자와의 모든 상호 작용이 이루어진다. Client 와 server 는 Http 통신을 통해 JSON 기반의 데이터를 주고 받는다. Server 에서는 MySQL 기반의 데이터 베이스에서 데이터를 생성, 수정, 확인, 삭제의 CRUD 작업을 진행하고, 해당 데이터를 JSON 형식을 통해 client 에게 전달한다. 전체적인 과정에 대해 설명하면, 페이지에 접속 시 기본적으로 전체 사용자 중 상위 5 명에 대한 정보를 client 가 server 에 자동적으로 요청한다. 그리고 사용자가 본인의 이름 정보를 입력하면, 중복 체크를 통해 기존에 있는 회원이면 로그인을, 없는 회원이면 회원 추가를 해준다. 사용자가 코드 제출을 하면, server 는 서버 정보, 탄소 배출량, 예시 데이터를 반환하고 사용자 정보를 저장한다. 이 때, 만약 로그인이 되어있지 않다면, 익명 이라는 이름으로 데이터가 저장된다.

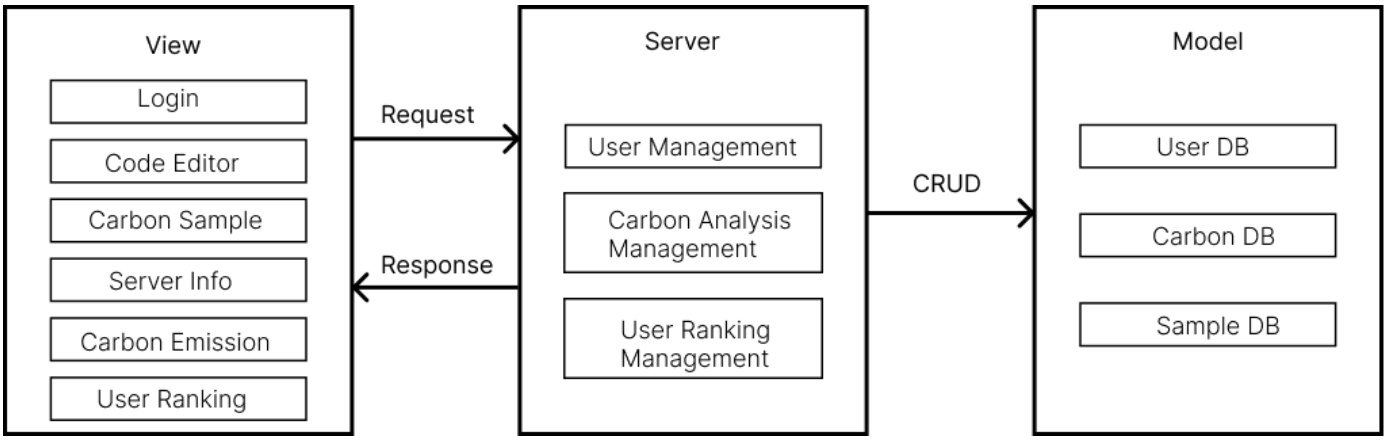

Figure 3.1: Overall System Architecture

3.2.1. System Diagram

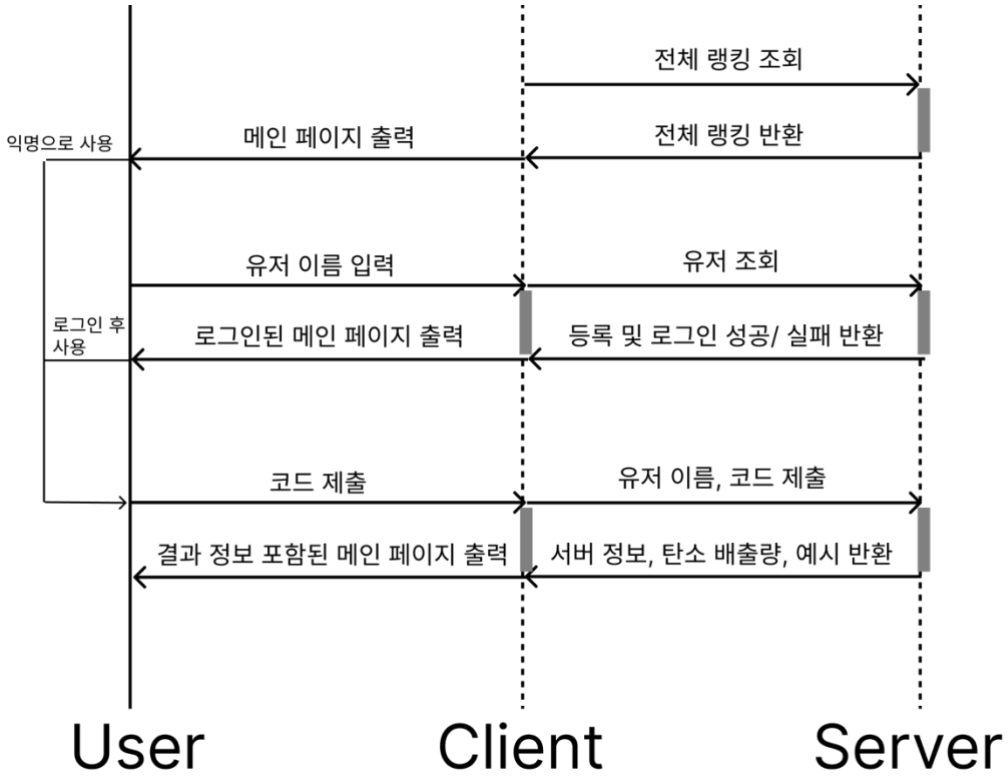

Figure 3.2: System Diagram

3.3. Use Case Diagram

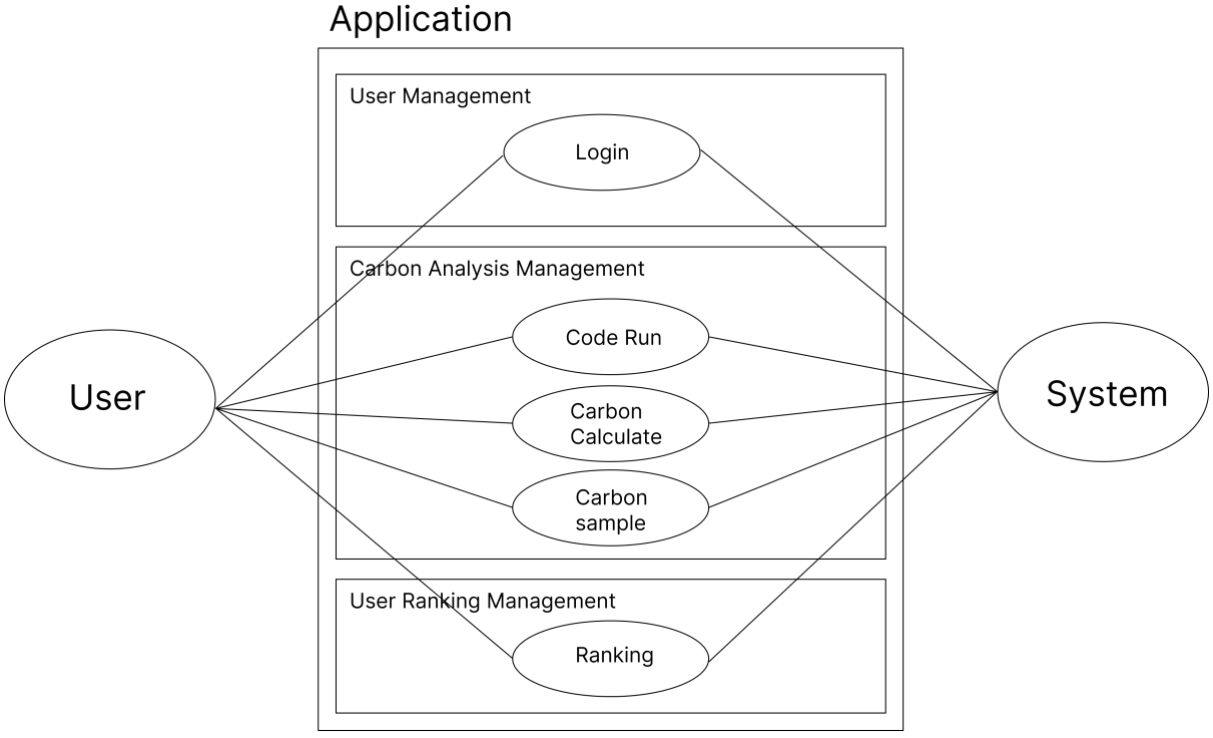

Figure 3.3: Use Case Diagram

4

System Architecture - Frontend

4.1. Objectives

이번 장에서는 프론트엔드 시스템의 구조, 속성 및 기능을 설명하고 코드 탄소 배출량 측정 시스템에서 각 구성 요소의 관계를 설명한다.

4.1.1. User

Objectives

name: 사용자의 이름

Methods

createID()

Class Diagram

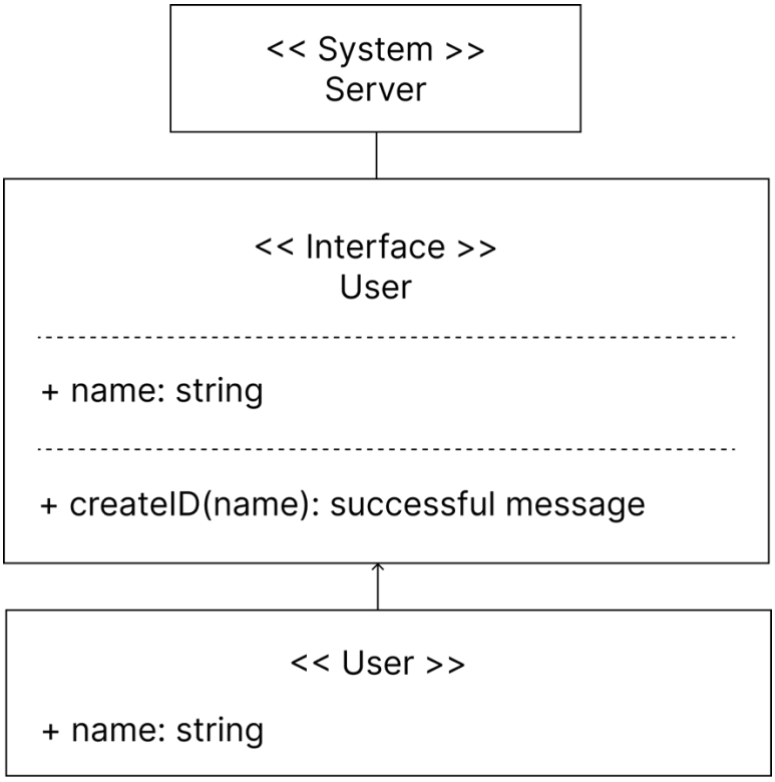

Figure 4.1: Class Diagram - User

Sequence Diagram

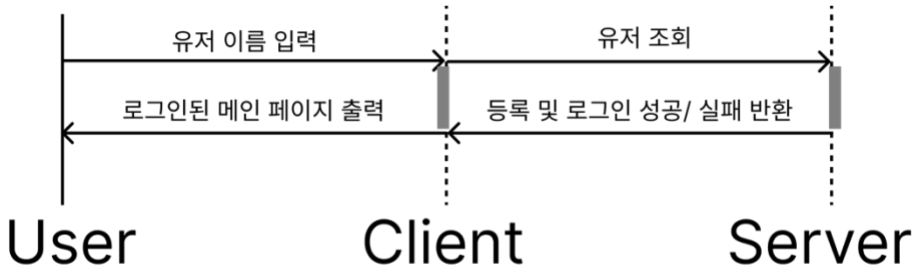

Figure 4.2: Sequence Diagram - User

4.1.2. Carbon Analysis

Objectives

name: 사용자의 이름

code: 사용자 제출 코드

Methods

carbonCalculate()

Class Diagram

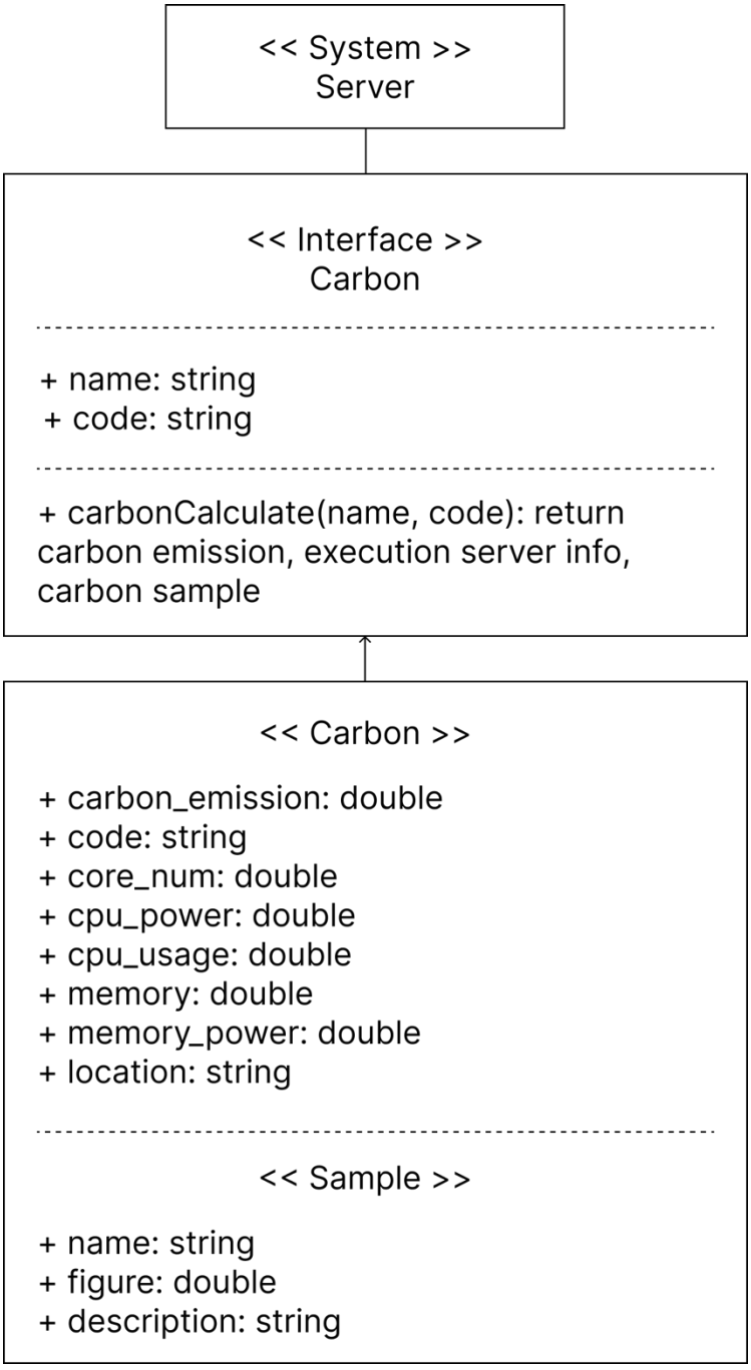

Figure 4.3: Class Diagram - Carbon

Sequence Diagram

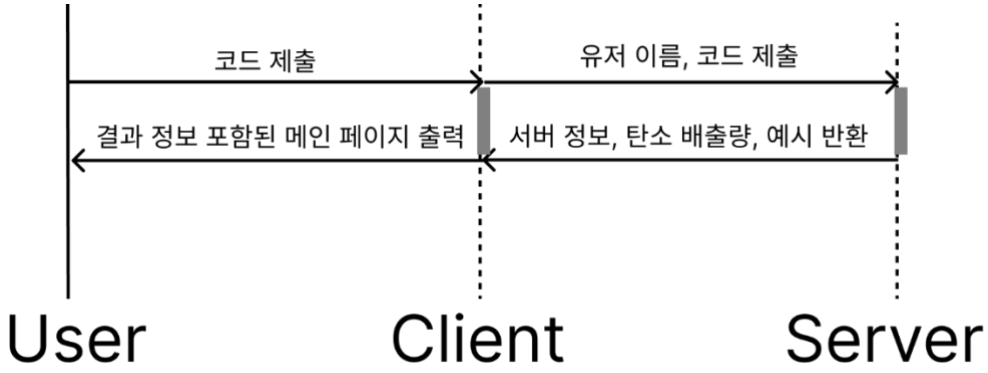

Figure 4.4: Sequence Diagram - Carbon

4.1.3. Ranking

Methods

getRanking()

Class Diagram

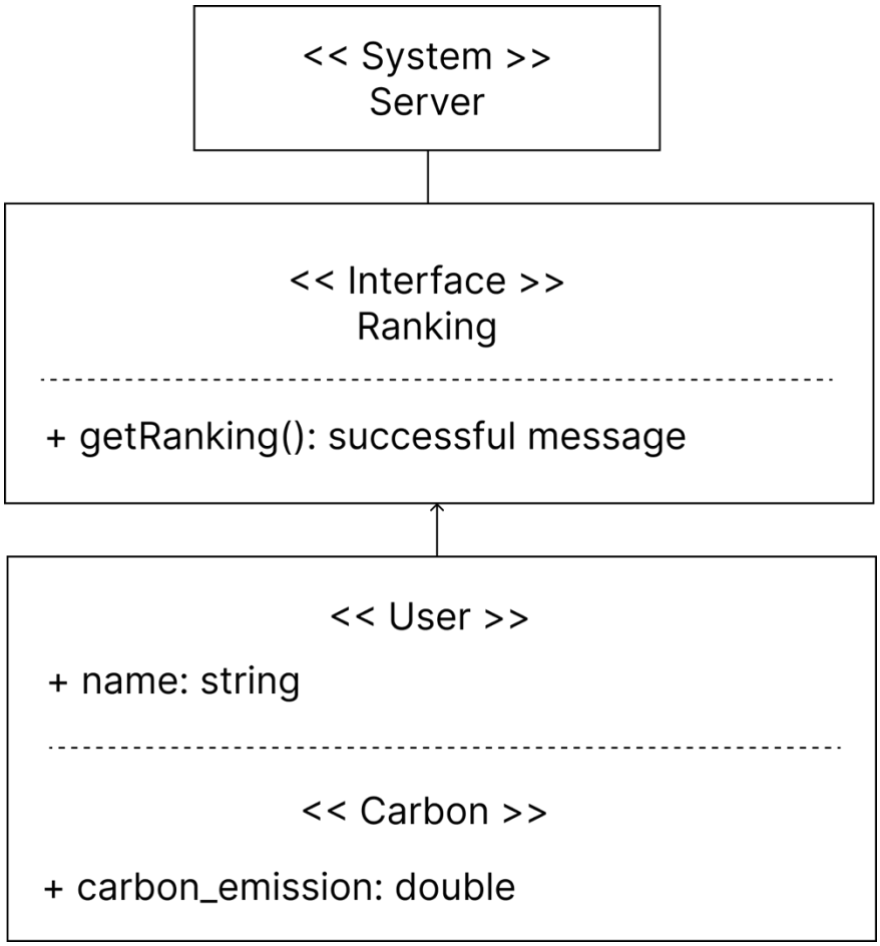

Figure 4.5: Class Diagram - Ranking

Sequence Diagram

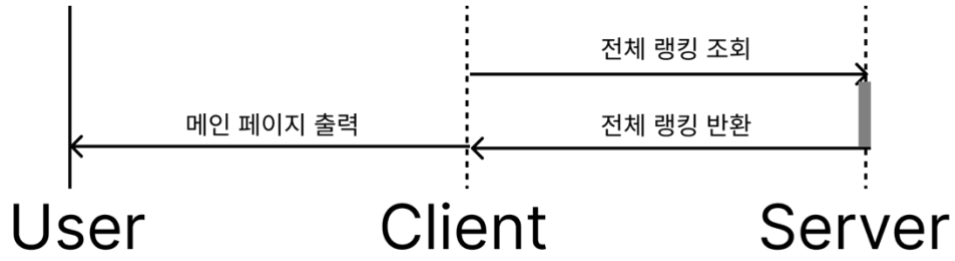

Figure 4.6: Sequence Diagram - Ranking

5

System Architecture – Backend

5.1. Objectives

이번 장에서는 백엔드 시스템의 구조, 속성 및 기능을 설명하고 코드 탄소 배출량 측정 시스템에서 각 구성 요소의 관계를 설명한다.

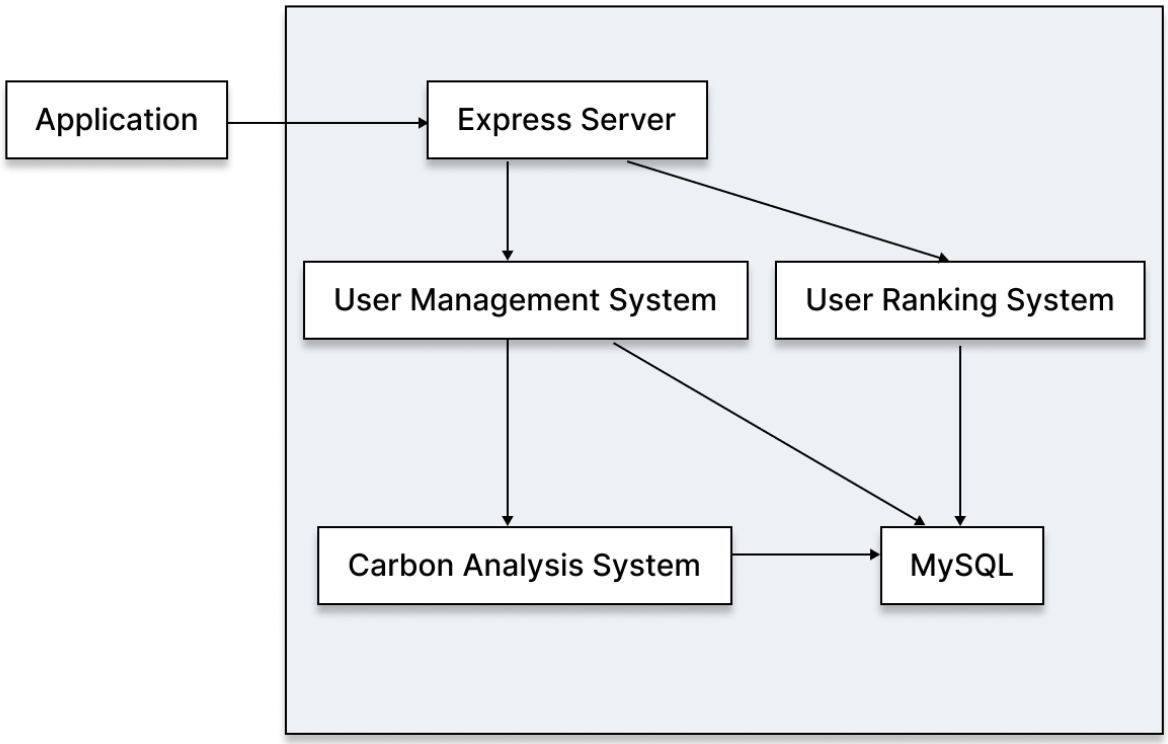

Figure 5.1: Overall Backend Architecture

사용자가 Java 코드의 탄소 배출량 측정을 할 수 있도록 도와주며, 궁극적으로는 환경 친화적 개발을 통해 소프트웨어 개발 생태계의 지속 가능성을 높일 수 있도록 유도하는 본 시스템의 전반적인 구조는 위와 같다. front-end 로부터 요청이 들어오면 Express Server 에서 해당 요청을 처리하는 function 을 실행한다. back-end 에서 처리하는 요청으로는 Java code 에 대한 탄소배출량 측정, 상위 5 명(탄소배출량 내림차순)의 사용자 랭킹 처리, 탄소배출량 결과에 따른 예시 데이터 반환 등이 있다. 또한, 데이터베이스로는 MySQL 을 사용한다.

5.2. Subcomponents

5.2.1. User Management System

Class Diagram

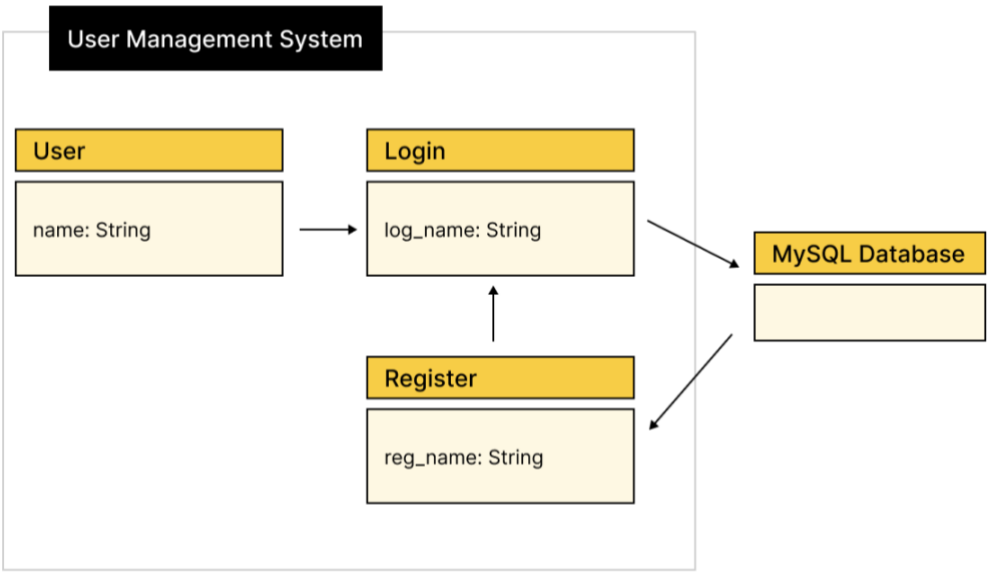

Figure 5.2: Class Diagram – User Management System

- Register class: 입력 받은 name 이 중복되지 않았는지 확인하며, 중복되지 않았다면 입력 받은 정보를 Database 에 새로 추가한다.
- Login class: 입력 받은 name 이 Database 상에 존재하는지 확인하고, 정보가 일치한다면 해당 class 의 정보를 불러온다.

Sequence Diagram

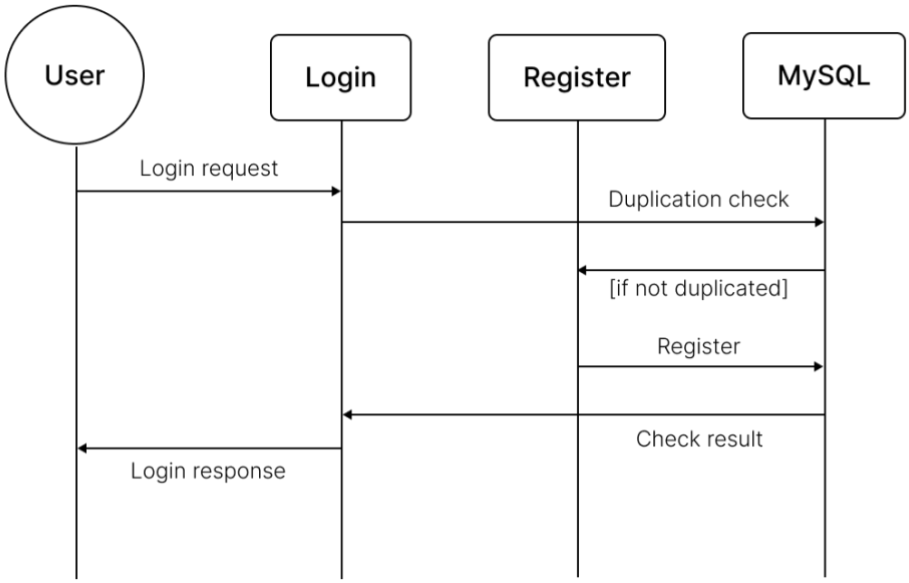

Figure 5.3: Sequence Diagram – User Management System

5.2.2. User Ranking System

Class Diagram

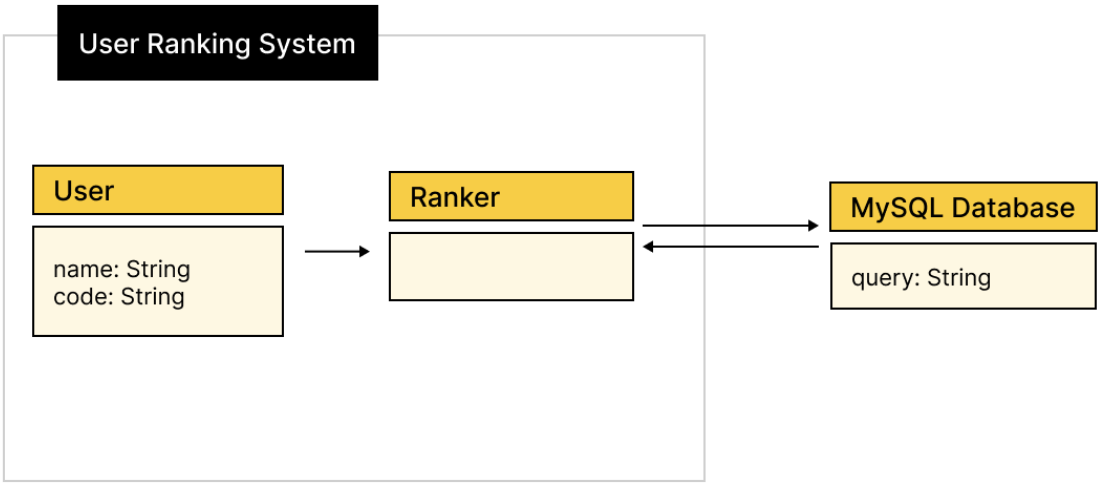

Figure 5.4: Class Diagram – User Ranking System

- Ranker class: Database 로부터 상위 5 명(탄소배출량 내림차순)의 사용자 이름과 각 탄소배출량 정보를 불러온다.

Sequence Diagram

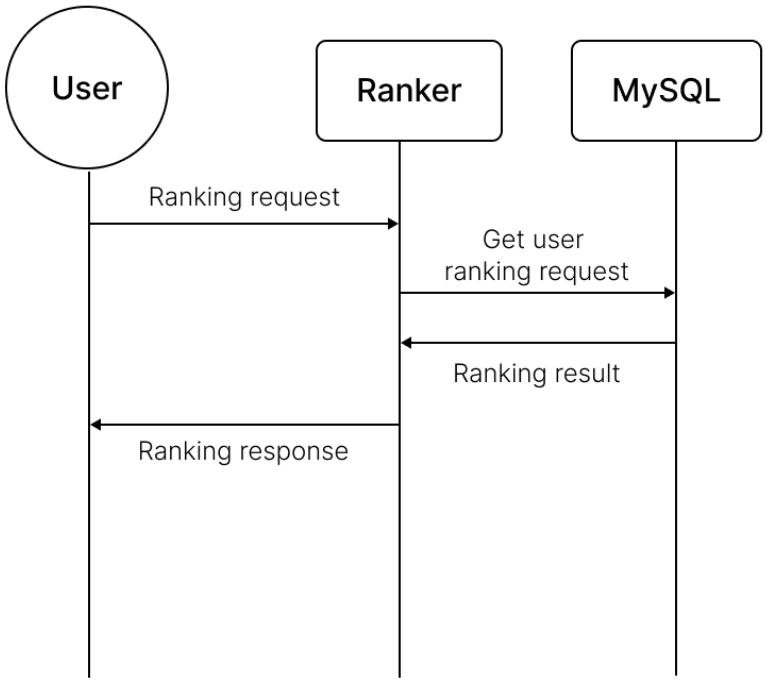

Figure 5.5: Sequence Diagram – User Ranking System

5.2.3. Carbon Analysis System

Class Diagram

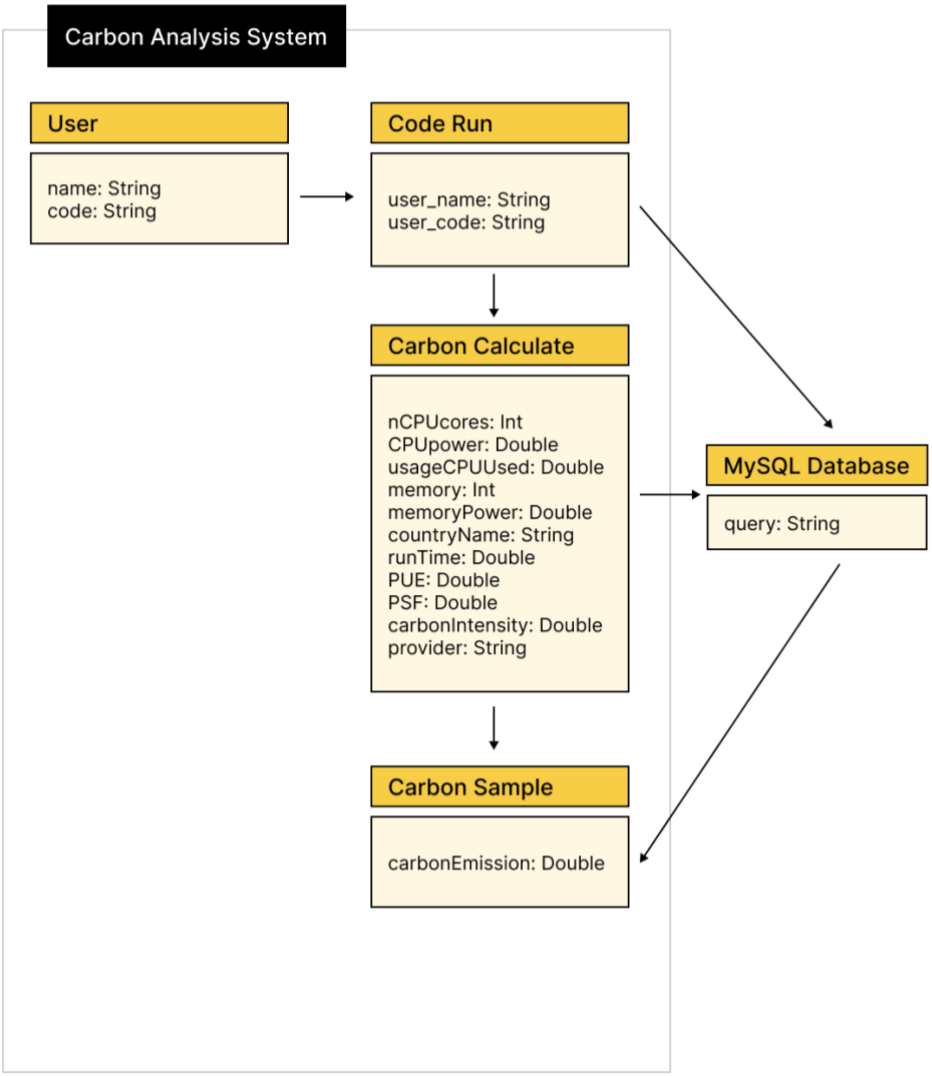

Figure 5.6: Class Diagram – Carbon Analysis System

- Code Run class: 사용자의 이름과 사용자가 작성한 코드 전체를 String 의 형태로 Database 에 저장한다. 또한, 사용자가 작성 코드를 실제로 실행하고, 해당 실행한 서버 정보 및 기타 탄소배출량 계산에 필요한 값을 반환한다.
- Carbon Calculate class: 실행 서버 정보와 실행 시간 등 탄소배출량 계산에 필요한 값을 Code Run Class 로부터 받아 탄소배출량 공식에 맞게 탄소배출량을 구한다. 도출된 탄소배출량을 Database 에 저장한다.
- Carbon Sample class: 사용자의 코드에 따른 탄소배출량 값을 Carbon Calculate clsss 로부터 받고, 사전에 저장되어 있던 sample 정보(ex. 탄소배출량 1g 당 매연 10g)를 Database 로부터 받아온다. 이후, 탄소배출량에 비례하는 sample 값과 그에 따른 정보를 반환한다.

Sequence Diagram

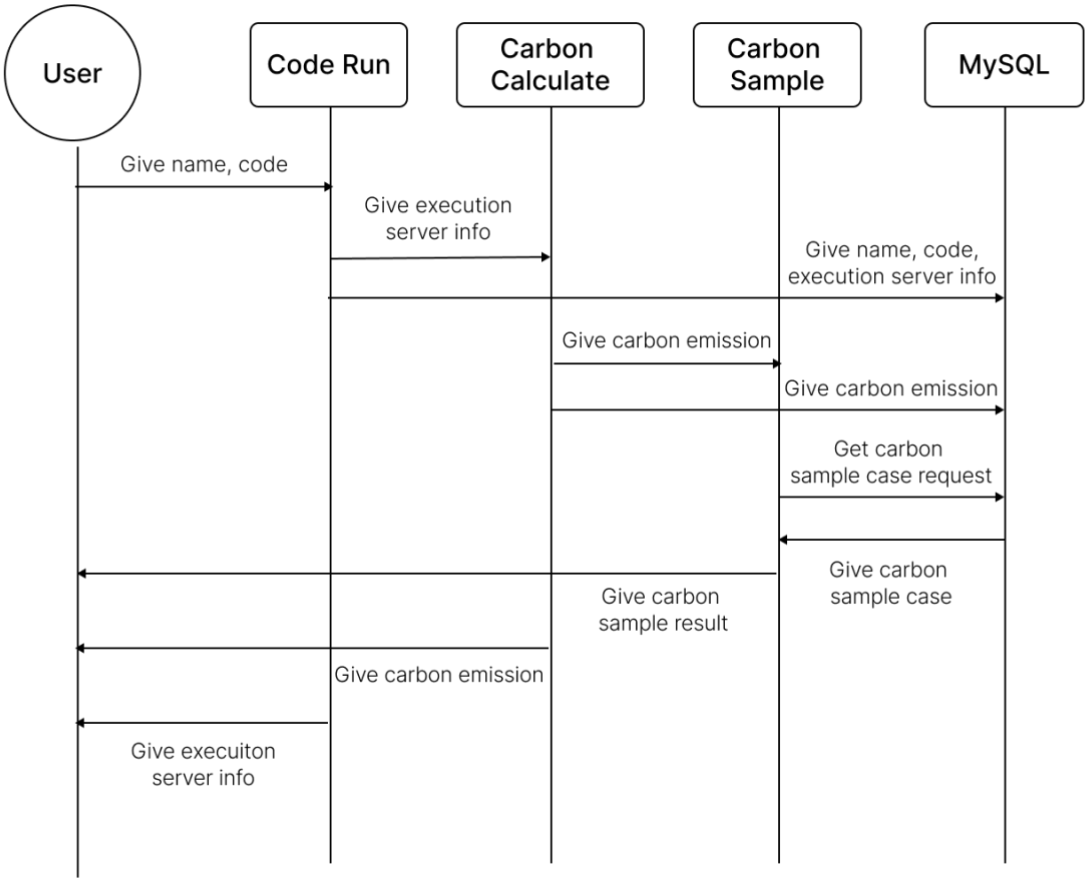

Figure 5.7: Sequence Diagram – Carbon Analysis System

6

Protocol Design

6.1. Objectives

이번 장에서는 front-end application 과 서버가 어떤 프로토콜로 상호작용하는 지 서술한다. 또한, 각 인터페이스가 어떻게 정의되어 있는지 서술한다.

6.2. Axios

Axios 는 브라우저 또는 Node.js 에서 HTTP 요청을 처리하기 위한 JavaScript 라이브러리이다. Axios 는 Promise 기반의 API 를 제공하여 비동기 작업을 처리한다는 점이 Axios 의 가장 큰 특징이다. 또한, 자동으로 JSON 데이터를 변환하는 기능을 제공한다. 이에 따라 서버로부터 수신한 JSON 데이터를 자바스크립트 객체로 변환하여 사용자에게 더 편리한 형태로 제공할 수 있다. 이외에도 요청을 취소하는 기능 또한 제공한다.

6.3. HTTP

HTTP (Hypertext Transfer Protocol)는 월드 와이드 웹에서 데이터를 전송하기 위한 응용 계층 프로토콜이다. GET, POST, PUT, DELETE 등 다양한 메소드를 제공하며, URI(Uniform Resource Identifier)를 통해 리소스를 식별한다. 또한, client-server model 을 따른다.

6.4. Authentication

6.4.1. Login

Request

Attribute	Detail	
Protocol	HTTP	
Method	GET	
Parameter	name (required)	User name

Figure 6.1: Table of Login Request

Response

Attribute	Detail	
Protocol	HTTP	
Success Code	200 OK	
Failure Code	HTTP error code = 400(Bad Request, overlap)	
Success Response Body	Message	Success message
Failure Response Body	Message	Failure Message

Figure 6.2: Table of Login Response

6.5. User Ranking

Request

Attribute	Detail
Protocol	HTTP
Method	GET

Figure 6.3: Table of User Ranking Request

Response

Attribute	Detail	
Protocol	HTTP	
Success Code	200 OK	
Failure Code	HTTP error code = 400(Bad Request, overlap)	
Success Response Body	Message	Success message
	Result	Top 5 user names (in descending order of carbon emissions) and each carbon emission
Failure Response Body	Message	Failure Message

Figure 6.4: Table of User Ranking Response

6.6. Carbon Calculating

Request

Attribute	Detail	
Protocol	HTTP	
Method	POST	
Request Body	code	User code
	name	User name

Figure 6.5: Table of Carbon Calculating Request

Response

Attribute	Detail	
Protocol	HTTP	
Success Code	200 OK	
Failure Code	HTTP error code = 400(Bad Request, overlap)	
Success Response Body	Message	Success Message
	Carbon Emission	calculated carbon emission value
	Execution Server Info	server environment in which user code was executed
	Carbon Sample	sample values proportional to carbon emission value and their descriptions
Failure Response Body	Message	Failure Message

Figure 6.6: Table of Carbon Calculating Response

7

Database Design

7.1. Objectives

이번 장에서는 시스템 데이터 구조와 이러한 구조가 데이터베이스로 어떻게 구현되었는지 서술한다. 우선, ER Diagram(Entity Relationship diagram)을 통해 entity 와 그 관계를 식별한다. 이후, 관계형 schema 및 SQL DDL(Data Definition Language)을 작성한다.

7.2. ER Diagram

본 어플리케이션 시스템은 User, Carbon, Sample 총 3 가지 entity 로 이루어져 있다. ER-Diagram 은 entity 간의 관계, 그리고 entity 와 attribute 의 관계를 diagram 으로 설명한다. 각 entity 의 primary key 는 밑줄로 표시되어 있다. 각 entity 마다 대응되는 개수는 entity 를 연결하는 선 주변에 표기되어 있어 확인 가능하다.

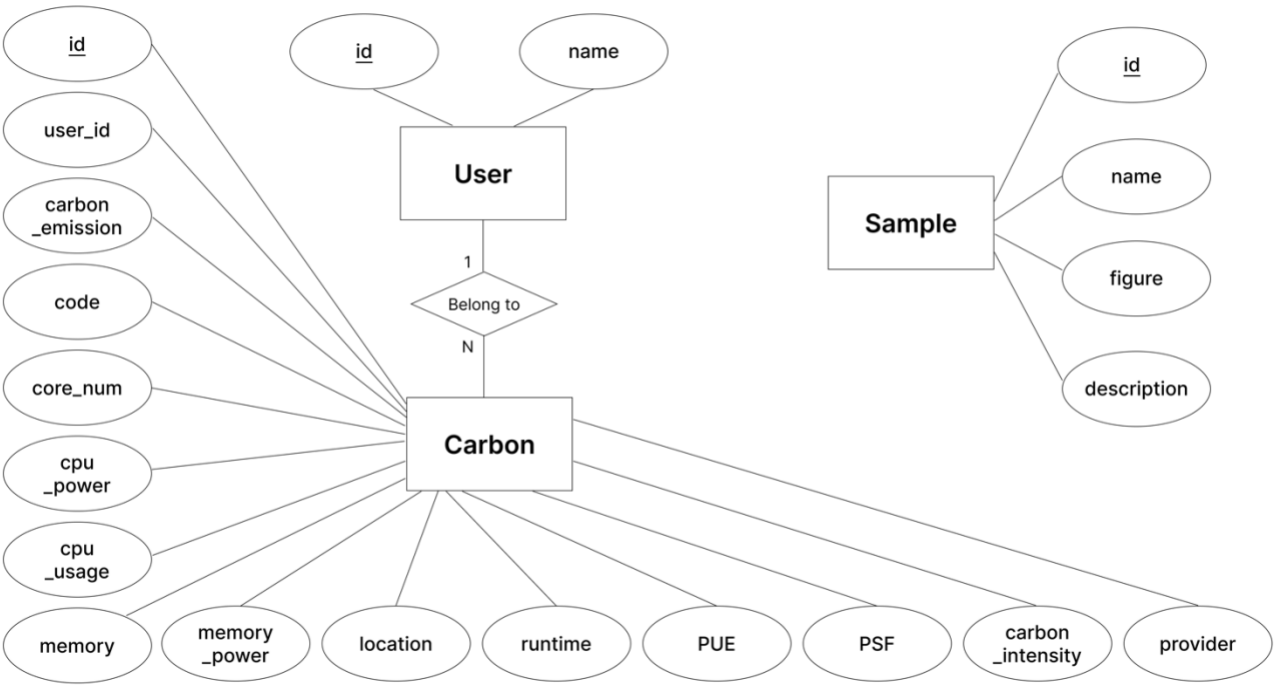

Figure 7.1: ER Diagram, Entity, User, Carbon, Sample

7.2.1. User

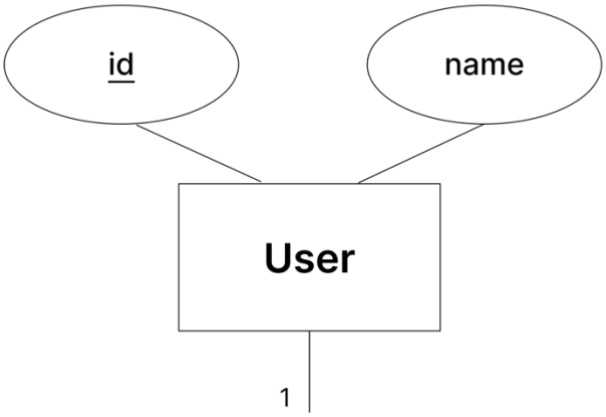

Figure 7.2: User ER Diagram

User Entity 는 application 사용자에 해당한다. User Entity 는 id, email 를 attribute 로 가진다. id 가 primary key 이며, name 은 unique 제약사항을 지닌다. 또한, User Entity 는 Carbon Entity 와 1:N 관계를 지니며, 한 명의 사용자마다 1 개 이상의 Carbon 데이터를 가지고 있다.

7.2.2. Carbon

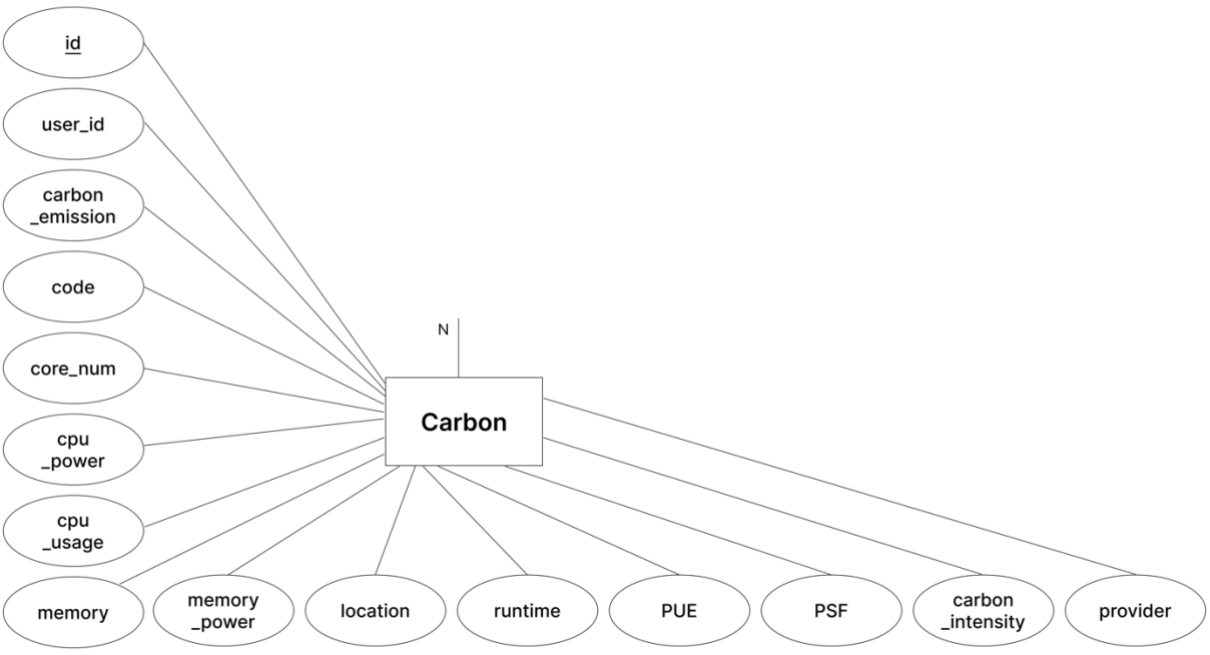

Figure 7.3: Carbon ER Diagram

Carbon Entity 는 사용자가 제출한 Java code 와 그에 대한 탄소배출량, 서버 실행 환경을 저장한다. id 를 primary key 로 가지고 있다. 또한, Carbon Entity 는 User Entity 와 N:1 관계를 지니며, Carbon Entity 가 User Entity 의 id 를 foreign key 로 참조해 접근한다.

7.2.3. Sample

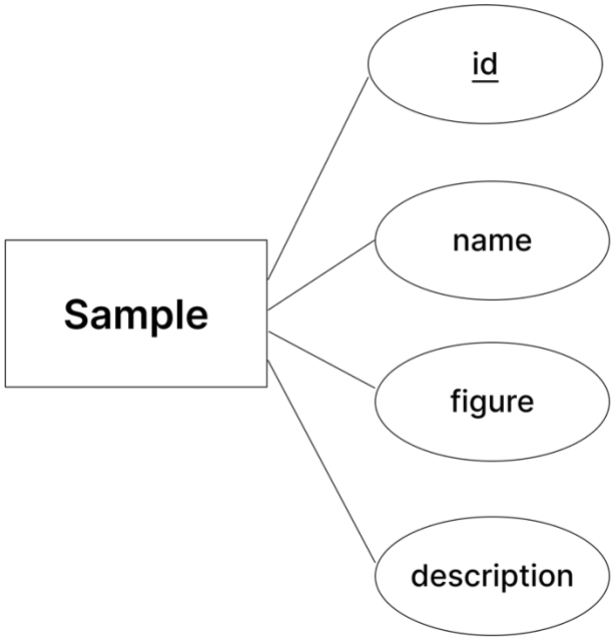

Figure 7.4: Sample ER Diagram

Sample Entity 는 탄소배출량 결과에 따른 예시 데이터를 지니고 있다. id 를 primary key 로 지니며, name, figure, description 을 attribute 로 가지고 있다.

7.3. Relational Schema

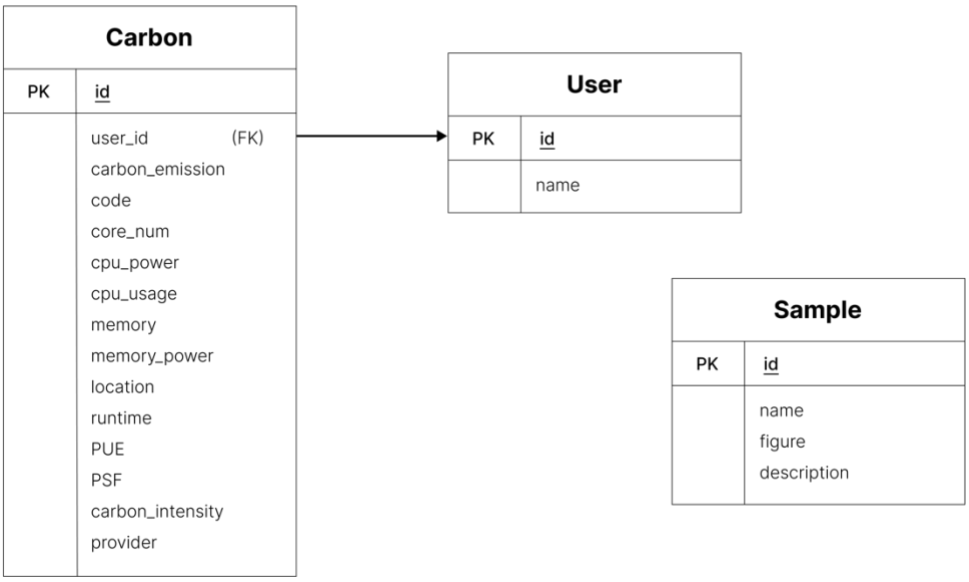

Figure 7.5: Relational Schema

7.4. SQL DDL

7.4.1. User

```sql
CREATE TABLE `tb_user` (
    `id` bigint NOT NULL AUTO_INCREMENT,
    `name` varchar(10) NOT NULL,
    PRIMARY KEY (`id`),
    UNIQUE KEY `tb_user_UN` (`name`)
) ENGINE=InnoDB AUTO_INCREMENT=9 DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci;
```

Figure 7.6: User SQL schema

7.4.2. Carbon

```sql
CREATE TABLE `tb_carbon` (
  `id` bigint NOT NULL AUTO_INCREMENT,
  `user_id` bigint NOT NULL,
  `carbon_emission` double NOT NULL,
  `code` varchar(5000) CHARACTER SET utf8mb4 COLLATE utf8mb4_0900_ai_ci NOT NULL,
  `core_num` double NOT NULL,
  `cpu_power` double NOT NULL,
  `cpu_usage` double NOT NULL,
  `memory` double NOT NULL,
  `memory_power` double NOT NULL DEFAULT '0.3725',
  `location` varchar(100) NOT NULL,
  `runtime` double NOT NULL,
  `PUE` double NOT NULL,
  `PSF` double NOT NULL,
  `carbon_intensity` double NOT NULL,
  `provider` varchar(100) NOT NULL,
  PRIMARY KEY (`id`),
  KEY `tb_carbon_FK` (`user_id`),
  CONSTRAINT `tb_carbon_FK` FOREIGN KEY (`user_id`) REFERENCES `tb_user` (`id`) ON DELETE CASCADE ON UPDATE CASCADE
) ENGINE=InnoDB AUTO_INCREMENT=22 DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci;
```

Figure 7.7: Carbon SQL schema

7.4.3. Sample

```sql
CREATE TABLE `tb_sample` (
    `id` bigint NOT NULL AUTO_INCREMENT,
    `name` varchar(100) NOT NULL,
    `figure` varchar(100) NOT NULL,
    `description` varchar(100) NOT NULL,
    PRIMARY KEY (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci;
```

Figure 7.8: Sample SQL schema

8

Testing Plan

8.1. Objectives

이번 장에서는 개발한 소프트웨어의 결함과 오류를 찾아내어 수정함으로써 개발된 애플리케이션이 사용자에게 불편함을 주지 않도록 목표하는 내용을 설명한다. 앞서 말한 결함들을 감지하는 방법은 여러 단계의 테스트를 통해 이루어 지고 각 테스트 단계에 대한 설명과 과정을 포함 하고 있다.

8.2. Testing Policy

8.2.1 Development Testing

본 테스트 과정은 개발하는 과정에서 소프트웨어를 직접 테스트하는 과정이다. 과정은 아래의 순서로 진행된다.

1. Unit testing

소프트웨어의 각 함수와 개체들을 테스트한다. 독립적으로 기능하는 유닛 단위의 테스트를 목적으로 한다. 본 소프트웨어에서는 아래와 같은 요소들이 평가 지표가 될 수 있다.

- 백엔드-프론트엔드의 연결
  - 통신간 데이터의 손실량 , 데이터의 전송시간
- 탄소 배출량 계산
  - 코드의 컴파일 시간, 코드의 에러 -예외 핸들링

2. Component testing

위 과정에서 각각의 요소들을 모두 테스트 하였다면, 이 요소들의 상호작용이 정상적으로 이루어지며 애플리케이션 내에서 잘 작동하는지 테스트한다. 본 소프트웨어에서는 아래와 같은 요소들이 평가 지표가 될 수 있다.

- html 인터페이스 상호작용
  - 메인 인터페이스와 서브인터페이스의 이동, 데이터 전송
- 서버 - 클라이언트 상호작용
  - 탄소배출량 계산의 (기준 서버 환경)에 대한 데이터 전송, 탄소배출량에 따른 환경 척도 계산

3. System testing

애플리케이션 내부 사항이 아닌 외부의 Non-functional 요소를 테스트한다. 본 소프트웨어에서는 아래와 같은 요소들이 평가 지표가 될 수 있다.

- 웹 인터페이스
    - 크롬, 파이어폭스, 사파리 등의 다양한 브라우저 사용
- 단말기 스펙
    - 서버-클라이언트 환경의 엔드 노드 단말기의 유형 혹은 성능

8.2.2 Release Testing

이 테스트 과정은 출시 직전의 소프트웨어를 예상 사용자가 직접 테스트하는 단계이다. 소프트웨어가 요구사항에 맞게 제대로 구현됐는지 사용에 불편한 점과 개선될 점은 없는지 피드백을 수용하며 전체적으로 테스트한다. 따라서 테스트하기 위한 애플리케이션의 알파, 베타버전을 제작 해야하고 ,배포 전 최종 테스트라고 볼 수 있기 때문에, 기능적으로는 결함이 없어야 하는 것이 전제된다.

객관성을 위해 테스터는 개발과정을 모르는 상태로 테스트하는 block-box testing 이 권장되고, 여러 오버로드를 유발하거나, 다양한 환경에서의 테스트를 통해 실사용에서의 성능 한계를 알아 볼 수 있다.

8.2.3. User Testing

이 테스트 과정은 완성된 배포 버전의 애플리케이션을 실 사용자들이 직접 애플리케이션을 사용함으로써 제품의 완성도와 만족도를 측정하고, 다음 업데이트 혹은 새로운 파생 제품 제작에 대한 사전데이터를 얻을 수 있다. 테스트 집단에 대해 다음과 같다.

- Alpha testing
    - 어느 정도 개발과 관련이 있는 테스터 집단이다. 본 수업에 참여한 다른 조원들이 가장 적합한 대상이다. 이를 통해 개발 관련 피드백을 받을 수 있다.
- Beta testing
    - 일반적인 사용자인 테스터 집단이다. 이전 명세서에서 나타낸 예상 사용자에 해당하는 인물들이 적합하다. 이를 통해 일반적인 피드백을 기대할 수 있다.

9

Development Plan

9.1. Objectives

이번 장에서는 시스템의 개발 환경 및 기술에 대하여 설명한다.

9.2. Development Environment

9.2.1 Github

Figure 9.1: Github Logo

GitHub 는 코드 호스팅 및 협업 플랫폼으로, 개발자들이 소스 코드를 관리하고 협업하는 데 사용된다. Git 을 기반으로 하며, 버전 관리, 이슈 트래킹, 협업 기능을 제공하여 효율적인 소프트웨어 개발을 지원한다. 따라서 본 소프트웨어의 개발 및 버전 관리를 위해 사용할 것이다.

9.2.2. Github Action

Figure 9.2: Github Action Logo

GitHub Actions 은 GitHub 에서 제공하는 자동화 서비스로, 코드를 빌드, 테스트, 배포하는 워크플로우를 자동화할 수 있다. 저장소 내에서 이벤트가 발생할 때 특정 작업을 수행하도록 구성할 수 있으며, CI/CD(Continuous Integration/Continuous Deployment) 프로세스를 지원하여 개발 효율성을 향상시킨다. 따라서 코드 수정에 따른 자동 배포를 위해 사용할 것 이다.

9.2.3. AWS EC2

Figure 9.3: AWS EC2 Logo

Amazon EC2(Elastic Compute Cloud)는 Amazon Web Services(AWS)에서 제공하는 클라우드 컴퓨팅 서비스로, 가상 서버를 실행할 수 있다. 다양한 운영 체제 및 인스턴스 유형을 지원하며, 필요에 따라 확장이 가능하고 유연한 컴퓨팅 리소스를 제공한다. 따라서 프론트엔드 및 백엔드의 서버 배포를 위해 사용할 것이다.

9.2.4. VSCode

Figure 9.4: VSCode Logo

Visual Studio Code(VSCode)는 Microsoft 에서 개발한 무료이며 오픈 소스의 경량 코드 편집기이다. 다양한 언어 지원, 풍부한 확장성, 통합 터미널 등의 기능을 제공하여 개발자들이 효율적으로 코드를 편집하고 디버깅할 수 있다. 따라서 프로그래밍 작업 툴로 사용할 것이다.

9.2.5. JavaScript

Figure 9.5: JavaScript Logo

JavaScript 는 클라이언트 측 웹 개발을 위한 스크립트 언어로, 웹 페이지를 동적으로 제어하고 상호 작용할 수 있게 한다. 이벤트 처리, DOM 조작, 비동기 통신과 같은 기능으로 사용되며, 최초에는 웹 브라우저에서 실행되었지만 현재는 Node.js 를 통해 서버 측에서도 실행 가능하다. 따라서 프론트 및 백엔드의 언어로 사용할 것이다.

9.3. Frontend Environment

9.3.1. React

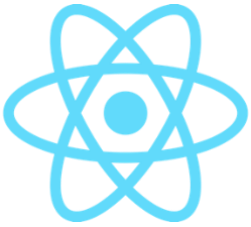

Figure 9.6: React Logo

React 는 Facebook 에서 개발한 JavaScript 라이브러리로, 사용자 인터페이스를 만들기 위한 선언적이고 효율적인 방법을 제공한다. 컴포넌트 기반 아키텍처를 기반으로 하며, 가상 DOM 을 활용하여 성능을 최적화한다. 따라서 프론트엔드 웹 페이지 구현을 위해 사용할 것이다.

9.3.2. Next.js

Figure 9.7: Next.js Logo

Next.js 는 React 기반의 웹 애플리케이션을 빠르고 간편하게 개발할 수 있도록 도와주는 프레임워크이다. 서버 사이드 렌더링(SSR)과 정적 사이트 생성(SSG)을 지원하여 성능 최적화와 검색 엔진 최적화(SEO)를 강화한다. 따라서 프론트엔드 웹 페이지 구현 시 프레임워크로 사용할 것이다.

9.4. Backend Environment

9.4.1. Express.js

Figure 9.8: Express.js Logo

Express.js 는 Node.js 를 위한 빠르고 유연한 웹 애플리케이션 프레임워크로, 간결한 코드로 웹 서버를 만들고 라우팅 및 미들웨어를 지원한다. RESTful API 와 웹 애플리케이션을 쉽게 구축할 수 있도록 돕는다. 따라서 백엔드 구현 프레임워크로 사용할 것이다.

9.4.2. MySQL

Figure 9.9: MySQL Logo

MySQL 은 오픈 소스의 관계형 데이터베이스 관리 시스템(RDBMS)으로, 다양한 응용 프로그램에서 데이터 저장 및 관리에 사용된다. SQL(Structured Query Language)을 사용하여 데이터베이스를 쿼리하고 조작할 수 있으며, 안정성과 성능면에서 널리 사용되고 있다. 따라서 본 소프트웨어의 데이터베이스로 사용할 것이다.

9.5. Constraints

본 시스템은 이 문서에 언급된 내용들에 기반하여 디자인되고 구현될 것이다. 이를 위한 세부적인 제약 사항을 다음과 같다.

- 시스템은 Window 7 이상, Linux Kernel 5.0 이상, 또는 macOS 10.0 이상의 운영체제에서 실행되어야 한다. 또한, 시스템 사용을 위해 웹 어플리케이션의 원활한 동작과 최소 10Mbps 의 인터넷 속도가 요구된다.
- 시스템은 다수의 사용자가 동시에 코드를 입력하고, 그에 따른 탄소 배출량 계산 결과와 실행 서버 정보를 각각 반환 받을 수 있도록 동작해야 한다. 이때, 사용자는 일반 유저와 관리자로 구분된다.
- 사용자가 입력하는 코드는 Java 로 제한되며, GPU 를 사용하지 않고 실행 가능해야 한다.
- 사용자 코드를 입력하고, 그에 따른 탄소배출량 계산 결과를 확인하는데 걸리는 시간은 3 분 내로 완료되어야 한다.
- 탄소 배출량 계산 결과와 코드 실행 후 실행한 환경 정보는 1 분 이내로 계산 시스템에 의해 도출되고, 이를 나타낼 수 있어야 한다. 또한, 위 데이터는 10 초 이내에 데이터베이스에 저장되어야 한다.
- 최소 300 명의 사용자가 동시에 접속하더라도, 시스템은 부드럽고 끊김없이 동작해야 한다.
- 사용자가 코드를 수정 후, 다시 시스템에 입력할 때, 즉시 업데이트된 코드에 따른 탄소 배출량 계산 결과와 실행 서버 정보를 실시간으로 새롭게 반환해야 한다.
- 사용자의 회원가입 요청은 10 초 이내로 완료되어야 하며, 새로운 사용자 정보는 10 초 이내로 데이터베이스에 저장되어야 한다.
- 사용자의 로그인/로그아웃 요청은 5 초 이내로 완료되어야 한다.

9.6. Assumptions and Dependencies

본 문서의 모든 시스템은 데스크탑 환경에서 기반하여 디자인 및 구현되었다고 가정하며 작성되었다. 또한 윈도우 10, 11 혹은 macOS 14 기반의 OS 환경을 기반으로 하여 작성되었으며 따라서 다른 OS 나 조건을 만족하는 않는 환경에서 시스템의 지원은 보장할 수 없다.

10

Supporting Information

10.1. Software Requirements Specification

소프트웨어 요구사항 명세서 IEEE 권장 사항 (IEEE Recommend Practice for Software Requirements Specifications, IEEE-Std-830)에 따라 작성되었다.

10.2. Document History

Date	Version	Description	Writer
2023-11-19	V1.00	Purpose, System Architecture-Frontend, Formatting	김선우
2023-11-19	V1.00	Testing Plan	설현원
2023-11-19	V1.00	Introduction	이찬구
2023-11-19	V1.00	System Architecture-Overall	이현민
2023-11-19	V1.00	Development Plan	조재범
2023-11-19	V1.00	System Architecture-Backend, Protocol Design, Database Design	한수민