[CODEMETER]

Design Specification

by

김동한, 안낙균, 유지훈,

임소리, 조민호, 조유지

TEAM 4

1. 서론 **5**
1.1. Readership 5
1.2. Scope 5
1.3. Objective 5
1.4. Document Structure 5
2. 소개 **7**
2.1. Objectives 7
2.2. Applied Diagrams 7
2.2.1. Used Tools 7
2.2.2. Activity Diagram 7
2.2.3. Use Case Diagram 7
2.2.4. Sequence Diagram 7
2.2.5. Class Diagram 8
2.2.6. State Diagram 8
2.2.7. Entity Relationship Diagram 8
2.2.8. Project Scope 8
2.2.9. References 9
3. 시스템 아키텍처 - Overall **10**
3.1. Objectives 10
3.2. System Organization 10
3.2.1. System Diagram 11
3.3. Use Case Diagram 13
4. 시스템 아키텍처 - Frontend **14**
4.1. Objectives 14
4.1.1. Overall Architecture 14
4.1.2. Carbon Calculation 14
5. 시스템 아키텍처 - Backend **16**

5.1. Objectives 16
5.2. Overall Backend Architecture 16
5.3. Subcomponents 17
5.3.1. Main Server (WAS) 17
5.3.2. Java Compiler 18
5.3.3. Java Runner 18
5.3.4. Carbon Calculator 19
5.3.5. Result Converter 19
5.3.6. Database 20
6. 프로토콜 디자인 20
6.1. Objectives 20
6.2. JSON 20
6.3. HTTP 20
6.4. RESTful API 20
6.5. Application Programming Interface 21
6.5.1. Calculate Carbon Emission 21
7. 데이터베이스 디자인 22
7.1. Objectives 22
7.2. ER Diagram 22
7.2.1. Code 23
7.2.2. ExecutionResult 24
7.2.3. Emission 25
7.3. Relational Schema 25
7.4. SQL DDL 26
7.4.1. Code 26
7.4.2. ExecutionResult 26
7.4.3. Emission 26

8. 테스팅 계획 **27**
8.1. Objectives 27
8.2. Testing Policy 27
8.2.1. 단위 테스트(Unit Test) 27
8.2.2. 통합 테스트(Integration Test) 27
8.2.3. 유저 테스트(User Test) 27
9. Development Plan **28**
9.1. Objectives 28
9.2. Frontend Environment 28
9.2.1. JavaScript 28
9.2.2. ReactJS 28
9.2.3. Styled-components 29
9.3. Backend Environment 29
9.3.1. GitHub 29
9.3.2. NestJS 30
9.3.3. MySQL 30
9.4. Constraints 31
9.5. Assumptions and Dependencies 31

1. 서론

1.1. Readership

본 문서는 Java Code의 탄소배출량 측정 도구인 CODEMETER의 개발에 참여한 개발자 및 이를 이해하고 활용하려는 이해관계자들을 대상으로 한다. Java 를 사용하는 개발자들, 그리고 개발 활동이 환경과 탄소 배출량에 대해 미치는 영향에 민감한 조직 및 사용자들이 이 문서를 활용하여 시스템을 이해하고 활용할 수 있다.

1.2. Scope

본 문서는 CODEMETER 시스템의 전반적인 아키텍처와 이를 구성하는 각 컴포넌트에 대한 디테일한 내용을 다룬다. 특히 Java 코드의 탄소 배출량 측정에 중점을 두며, 이러한 기능을 수행하는 시스템의 전반적인 프로세스를 포괄적으로 다룬다.

1.3. Objective

본 문서의 목표는 CODEMETER 시스템의 아키텍처와 각 컴포넌트를 명확하고 상세하게 정의하여, 개발자들이 요구 사항을 명확히 이해하고 개발할 수 있도록 하는 것이다. 이를 위해 Java 코드의 탄소 배출량을 측정하는 기능을 구현하는 데에 필요한 명세를 구체적으로 제공한다.

1.4. Document Structure

본 문서는 다음과 같은 구조로 이루어져 있다.

(1) **서론**: Java 코드의 탄소 배출량을 측정하는 CODEMETER 시스템에 대한 디자인 명세서의 전체적인 내용을 요약해 제공한다. 본 명세서의 독자층을 정의하고, 목표를 소개하여 문서의 전반적인 방향을 제시한다.

(2) **소개**: 본 시스템의 디자인 명세서에서 사용하는 각종 다이어그램의 유형과 목적, 이를 작성하는 데에 도움을 주는 툴, 그리고 프로젝트의 개발 범위와 문서 작성 시에 참고한 자료를 명시한다.

(3) **시스템 아키텍처 - Overall**: CODEMETER 시스템의 전체적인 아키텍처를 소개하며, 주요 컴포넌트 간의 상호 작용을 중심으로 구체적이고 세부적으로 설명한다.

(4) **시스템 아키텍처 - Frontend**: CODEMETER 시스템의 프론트엔드 구조와 속성 및 기능을 설명하고, 각 컴포넌트 간의 관계를 설명한다.

(5) **시스템 아키텍처 - Backend**: CODEMETER 시스템의 백엔드 서버 구조와 이를 구성하는 각 컴포넌트의 동작 방식을 기술한다.

(6) **프로토콜 디자인**: 프론트엔드 애플리케이션과 백엔드 서버가 상호작용하기 위해 설정한 각종 규약 및 API들에 대해 설명한다.

(7) **데이터베이스 디자인**: ER Diagram을 활용하여 CODEMETER 시스템의 데이터 구조를 설명하고, 각 Entity에 대해 자세히 설명한다.

(8) **테스팅 계획**: 본 시스템이 개발되는 과정에서 수행할 테스팅 계획을 설명한다. 이에 대해 각 테스팅 과정이 어떤 목적으로 수행되는지를 명확하게 기술한다.

(9) **개발 계획**: CODEMETER 시스템의 개발 환경 및 기술 스택을 설명하며, 본 시스템을 구현할 때 지켜야 할 세부적인 제약사항과 전제 조건 및 의존성을 기술한다.

2. 소개

2.1. Objectives

본 문서는 CODEMETER 구현의 기반이 될 설계 명세서를 작성한다. 이번 챕터에서는 독자가 설계를 이해할 수 있도록 사용된 다이어그램과 툴에 대해 설명하며, 프로젝트의 개발 범위를 한정한다.

2.2. Applied Diagrams

2.2.1. Used Tools

‘Draw.io’는 UML 표준 다이어그램과 ER 다이어그램 등, 소프트웨어의 설계 과정에서 필요한 각종 다이어그램을 손쉽게 그릴 수 있는 무료 프로그램이다. 본 문서에 포함된 모든 다이어그램은 해당 소프트웨어를 활용하여 작성하였다.

2.2.2. Activity Diagram

Process Model은 개발할 시스템이 시스템의 환경에서 어떻게 사용되는지를 모델링한 것으로, UML에서는 Activity Diagram으로 표현할 수 있다. 개발할 시스템이 Java Compiler 등과 같은 외부 시스템과 어떤 정보를 주고받을 것인지를 보이기 위해 사용하였다.

2.2.3. Use Case Diagram

Use Case Diagram은 사용자 간 상호작용이나 시스템 간 상호작용을 명세화한다. 이는 요구사항과 시스템 간 통신 방식을 확인하기 위해 작성하였다. 다이어그램 상에는 행위자(actor)와 유스 케이스(use case)의 개략적인 관계만 표현되어 있으므로, 각 행위자나 유스 케이스에 대한 보다 상세한 내용은 함께 제시된 표를 참고하면 된다.

2.2.4. Sequence Diagram

Sequence Diagram은 어떠한 유스 케이스 중에 일어나는 상호작용의 순서도(sequence)를 표현한 것이다. Sequence Diagram을 이용하면 행위자와 객체 간에 어떤 상호작용이 어떤 순서대로 일어나는지를 명시적으로 확인할 수 있다. 여러

서브시스템 사이에서 발생하는 데이터 전달 및 동작 등을 시간 순으로 확인할 수 있도록 명세서에 이를 포함하였다.

2.2.5. Class Diagram

Class Diagram은 객체 지향적으로 시스템 모델을 만들 때, 시스템의 클래스(혹은 컴포넌트)와 클래스 간의 관계를 보이기 위해 제작하는 다이어그램이다. 시스템이 처리할 데이터의 구조 및 시스템의 정적인 조직도를 표현하기 위해 작성하였다.

2.2.6. State Diagram

Behavioral Model은 시스템이 실행 중일 때 어떤 동작을 하는지를 모델링하는 것이다. 이 프로젝트에서는 대상 시스템을 주로 이벤트에 의해 동작이 변화하는 시스템(event-based system)으로 보고, 이벤트에 따른 상태 변화를 표현할 수 있는 State Diagram을 활용하여 표현하였다. State Diagram은 시스템의 변화를 유도하는 이벤트와 각 이벤트의 발생으로 인해 도달할 수 있는 상태 및 동작을 보여주는 다이어그램이다.

2.2.7. Entity Relationship Diagram

Entity Relationship Diagram은 데이터 모델링의 한 종류로, 시스템이 처리하는 데이터를 여러 엔티티와 속성, 그리고 엔티티 사이의 관계로 명료화하기 위해 사용한다. 본 명세서에서는 데이터베이스의 논리적 구조를 설계하기 위하여 해당 다이어그램을 활용하였다.

2.2.8. Project Scope

CODEMETER가 수행하는 작업은 다음과 같다.

(1) 사용자로부터 Java 코드를 입력받아 이를 컴파일 및 실행한다. 이때 보안성을 강화하기 위해 실행 시간 제한, 리소스 격리 등을 수행한다.
(2) 사용자의 프로그램 실행 시간과 메모리 사용량을 측정한다.
(3) 측정값을 바탕으로, 서버의 컴퓨팅 환경을 기준으로 코드의 탄소 배출량을 계산한다.
(4) 탄소 배출량을 실생활에서의 탄소 배출량과 비교하여 사용자에게 출력한다.

본 프로젝트에서 개발할 시스템은 (2)~(4)과, (1)에서 컴파일 및 실행을 유도하는 작업을 담당한다. 실제로 컴파일 및 실행, 코어 격리 등을 수행하는 코어 프로그램은 외부 프로그램을 사용할 예정이다.

2.2.9. References

- IEEE Std 830-1998 IEEE Recommended Practice for Software Requirements
- https://github.com/skkuse/2022spring_41class_team1/blob/main/docs
- 프로젝트 요구사항 명세서: https://rb.gy/uzvxe8

3. 시스템 아키텍처 - Overall

3.1. Objectives

이 챕터에서는 시스템 내부에서 프론트엔드와 백엔드가 어떻게 설계, 구성되어 있는지에 대해 설명한다.

3.2. System Organization

먼저 이 시스템은 클라이언트-서버 구조를 가진다. 시스템은 프론트엔드, 백엔드로 구분된다. 프론트엔드 애플리케이션은 사용자와의 상호작용을 담당한다. 프론트엔드와 백엔드는 HTTP를 기반으로 JSON 포맷의 데이터를 주고 받는다. 또한 백엔드는 처리한 데이터를 Database에 저장하는 방식으로 응답을 처리한다. 이때 사용되는 Database는 MySQL이다.

프론트엔드로부터 백엔드가 전달 받는 데이터는 Java code이다. 백엔드는 전달 받은 코드를 바탕으로 탄소 배출량 결과 정보를 계산한다. 이때 백엔드는 전달 받은 Java code를 컴파일 및 실행하고, 실행 정보 (피크 메모리 사용량, 실행 시간)을 계산한다. 이후, 해당 실행 정보를 바탕으로 실생활 사용량을 계산한 뒤, 실행 결과와 함께 프론트엔드에게 반환한다. 또한, Java code, 실행 결과와 실행 정보, 탄소배출량 정보를 DB에 저장한다.

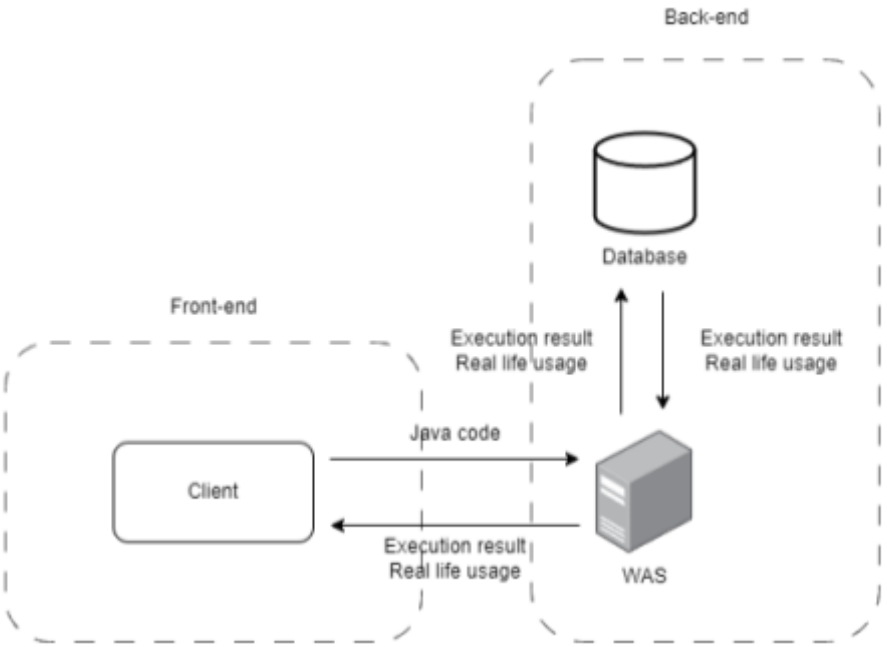

Figure 3.1: Overall System Architecture

3.2.1. System Diagram

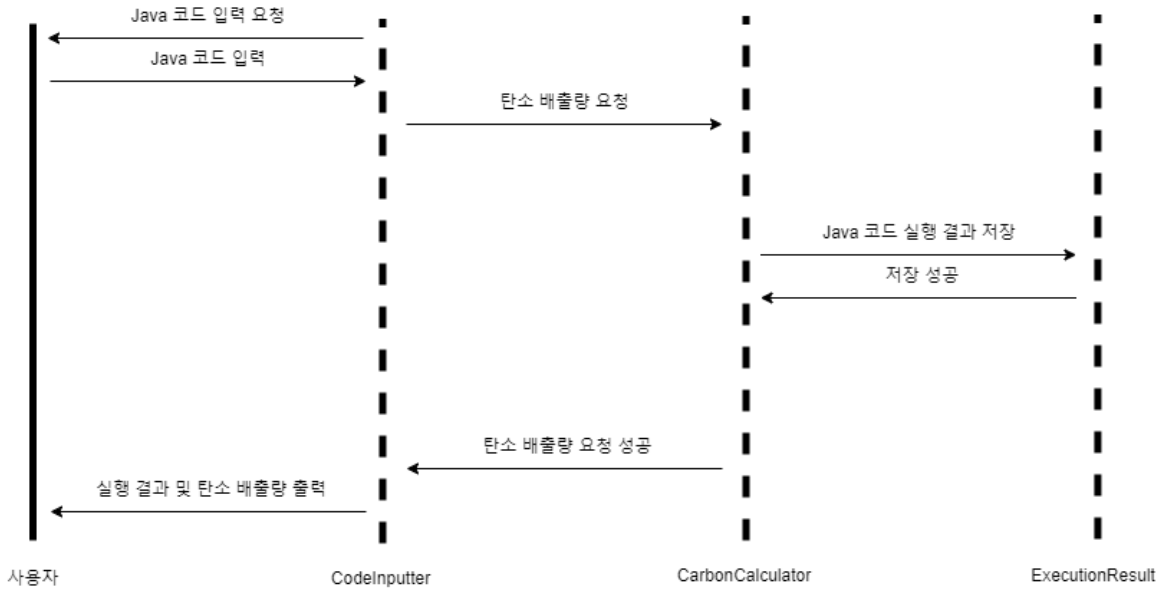

Figure 3.2: Sequence Diagram

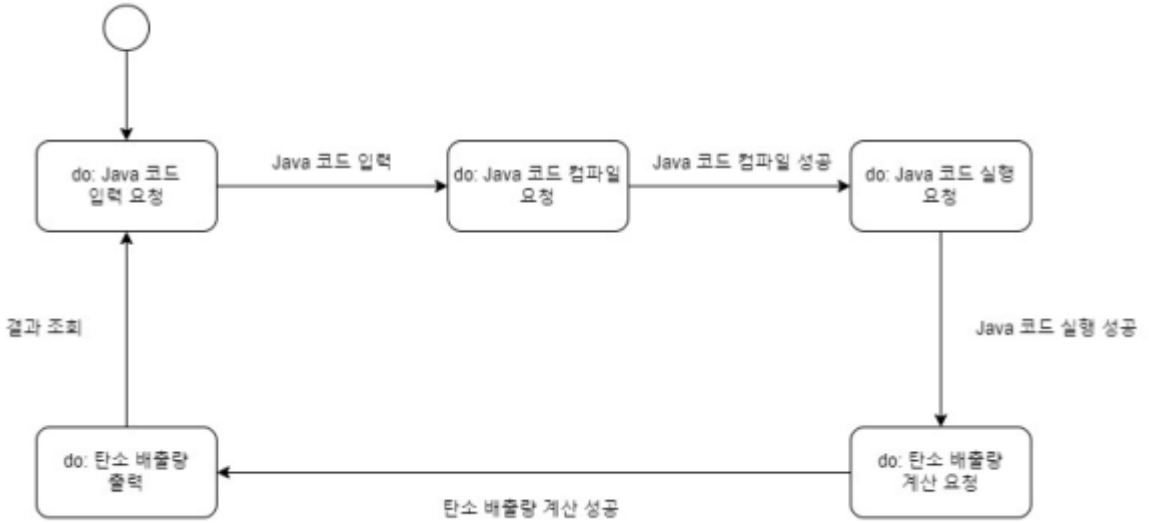

Figure 3.3: State Diagram

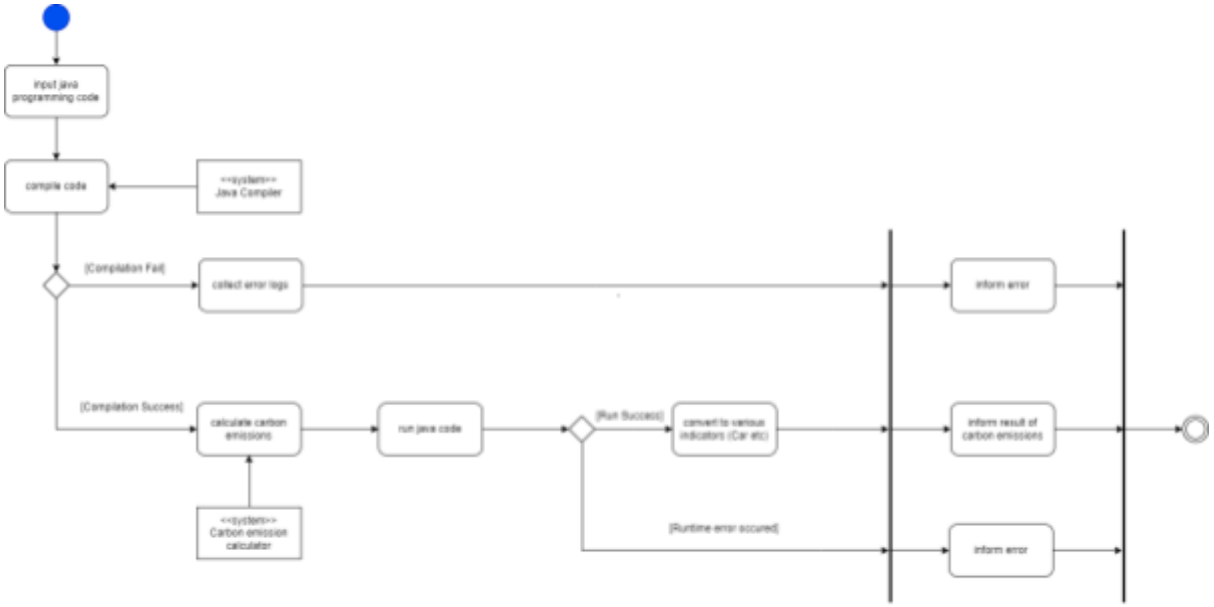

Figure 3.4: Process Diagram

3.3. Use Case Diagram

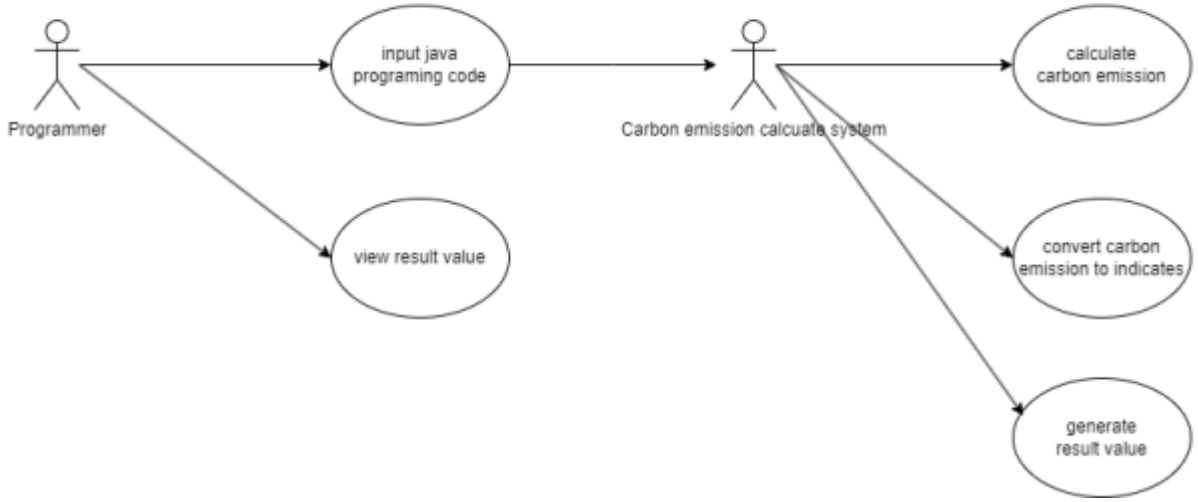

Figure 3.5: Use case Diagram

4. 시스템 아키텍처 - Frontend

4.1. Objectives

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

4.1.1. Overall Architecture

본 시스템에서는 사용자로부터 JAVA 코드를 입력받기 위해서 웹 어플리케이션을 프론트엔드로 구성한다.

4.1.2. Carbon Calculation

4.1.2.1. Attributes

- Code: 사용자가 입력하는 Java code
- CodeInputter: 사용자가 코드를 입력하는 인터페이스
- CarbonCalculator: 하드웨어 스펙을 이용해 코드의 탄소배출량을 계산하는 계산기
- ExecutionResult: 코드의 실행 이후 도출되는 계산값

4.1.2.2. Class Diagram

Objects identification: Code Input(코드 입력창)에서 Java code를 입력받고, 코드의 탄소배출량을 계산함과 더불어 런타임 등의 실행 결과 값을 생성해 DB에 저장한다. 또한 계산된 탄소배출량을 일상에서의 탄소배출량으로 환산하여, 그 계산값들을 클라이언트에게 넘겨 화면에 표시하도록 한다.

4.1.2.3. Attributes of Classes

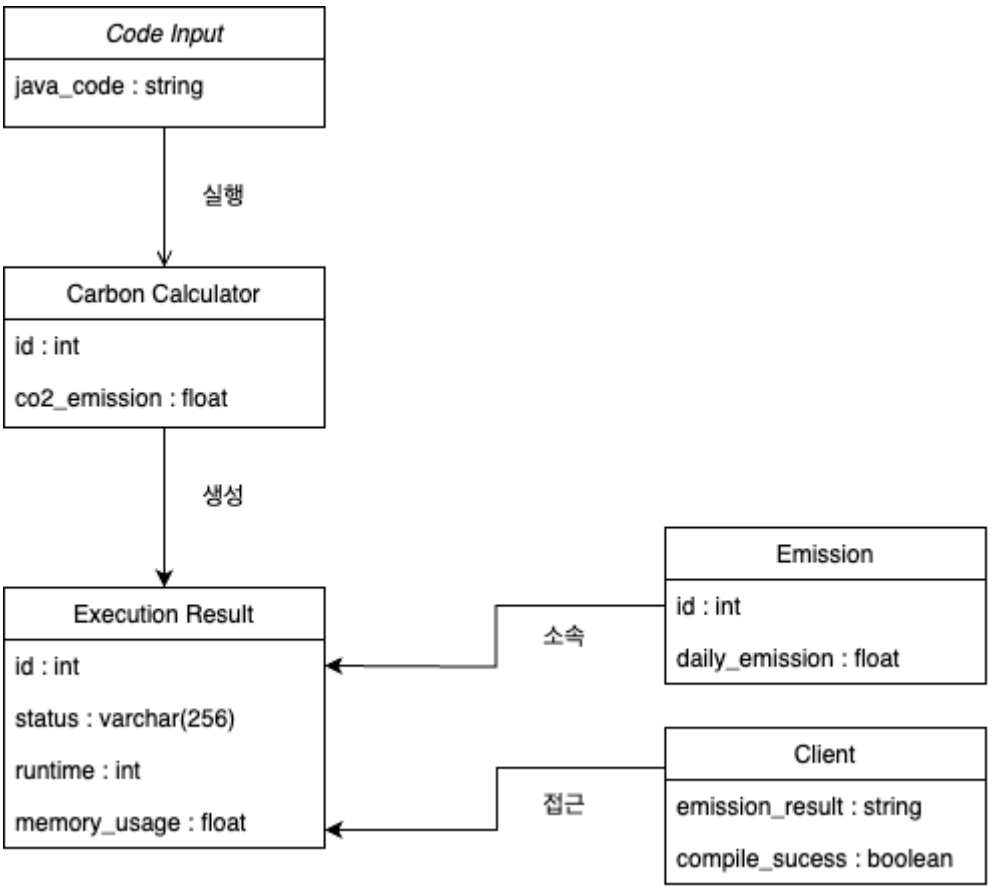

Figure 4.1: Class Diagram

4.1.2.4. Sequence Diagram

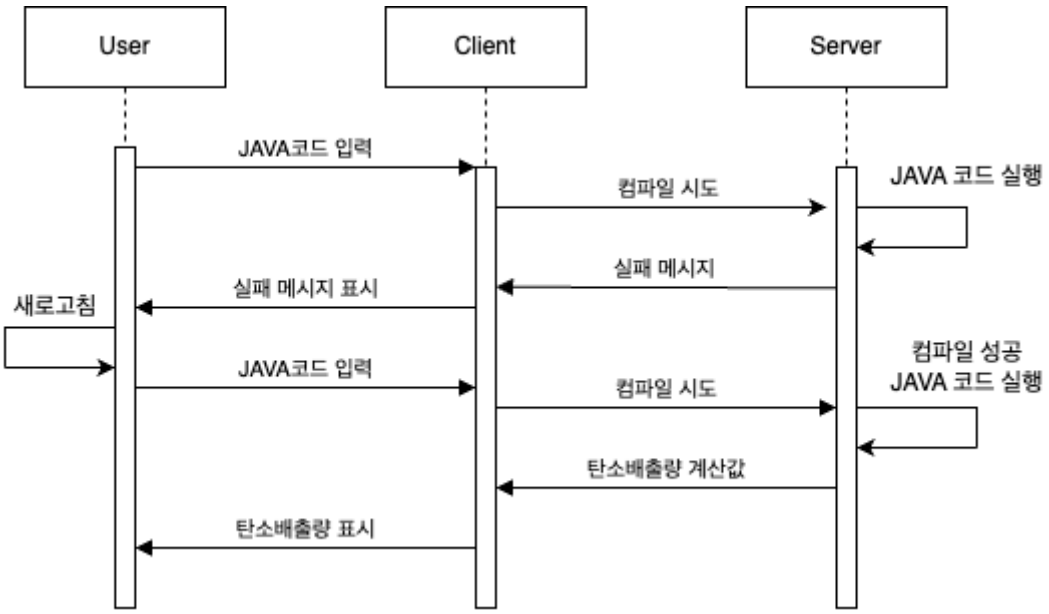

Figure 4.2: Sequence Diagram

5. 시스템 아키텍처 - Backend

5.1. Objectives

이 챕터에서는 본 시스템의 백엔드 서버 구조와 이를 구성하는 각 컴포넌트의 동작 방식, 그리고 컴포넌트 간에 주고받는 데이터의 흐름을 기술한다.

5.2. Overall Backend Architecture

전체적인 백엔드 서버 구조는 다음과 같다.

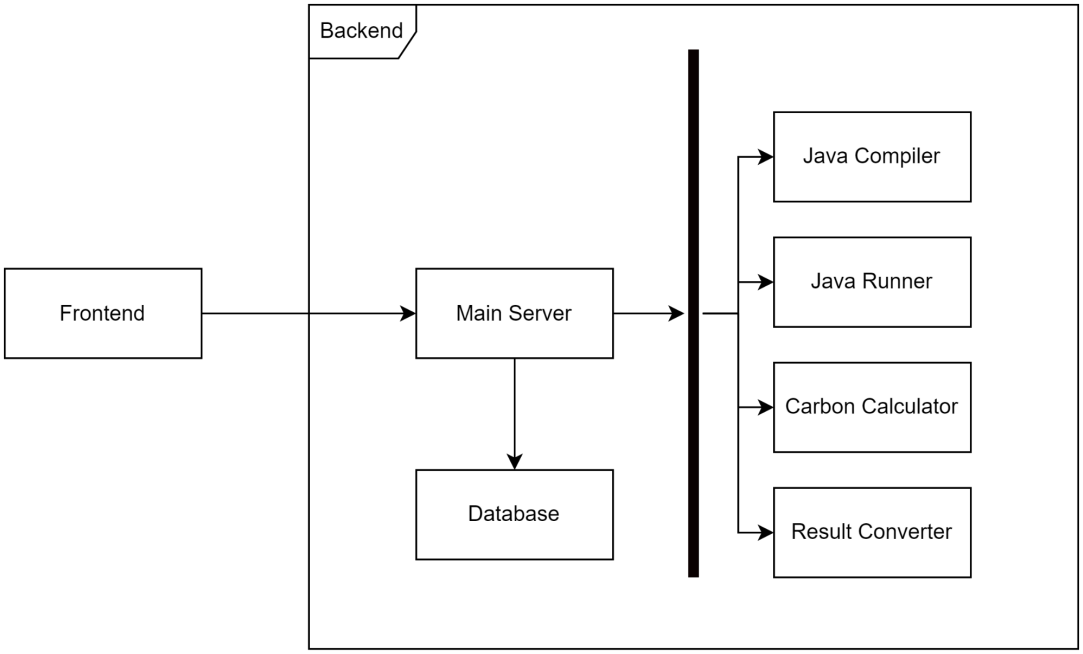

Figure 5.1: Overall Backend Architecture

백엔드 서버는 위와 같이 Main Server, Java Compiler, Java Runner, Carbon Calculator, Result Converter, Database의 6가지 Subcomponents로 구성되어 있다. Main Server가 백엔드에서 수행해야 하는 기능을 총괄하며, 프론트엔드에서 Java 코드를 전달받아 각 Subcomponents를 통해 필요한 데이터를 순차적으로 생성하고 최종적으로 코드의 탄소 배출량 및 실생활 사용량과의 비교 결과를 프론트엔드에 전달한다.

백엔드에서의 데이터 생성 flow는 다음과 같다.

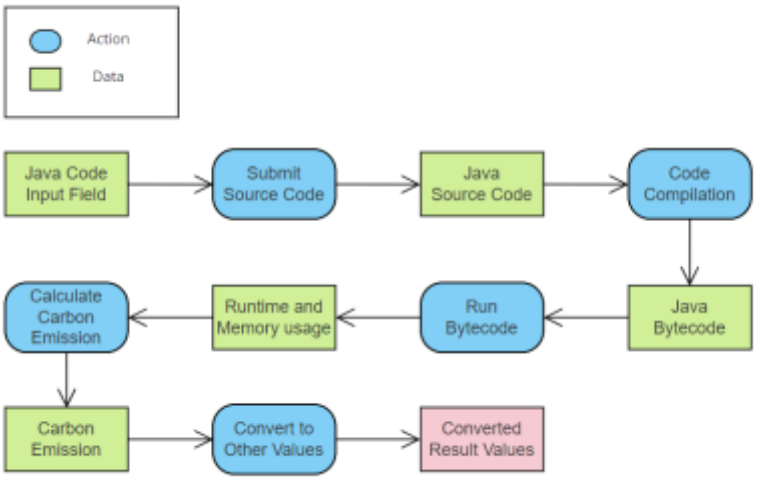

Figure 5.2: Backend Activity Diagram

5.3. Subcomponents

5.3.1. Main Server (WAS)

Main Server는 프론트엔드에서 전송된 Java 코드를 받아와 Java Compiler 및 Java Runner를 통해 실행하고, 코드의 실행 시간, 메모리 사용량을 측정한다. 이러한 정보를 Carbon Calculator에 전달하여 탄소 배출량을 계산하고, Result Converter를 통해 계산 결과를 필요한 형식으로 변환하여 프론트엔드에 반환한다. 모든 과정이 성공적으로 수행되었다면 해당 히스토리를 Database에 저장하고 관리한다.

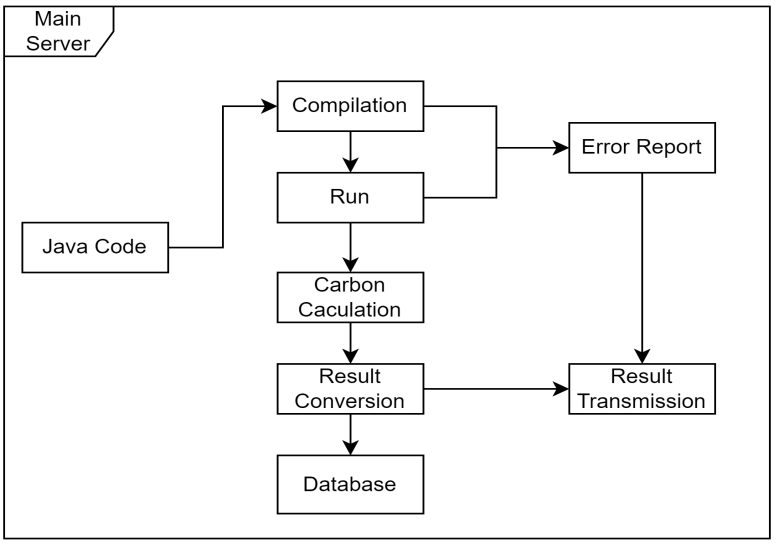

Figure 5.3: Backend - Main Server

5.3.2. Java Compiler

Java Compiler는 Main Server가 프론트에서 전송받은 Java 코드를 컴파일한다. 만약 컴파일에 성공했다면 Java Bytecode를 생성하며, 실패했다면 에러 코드와 메시지를 반환한다.

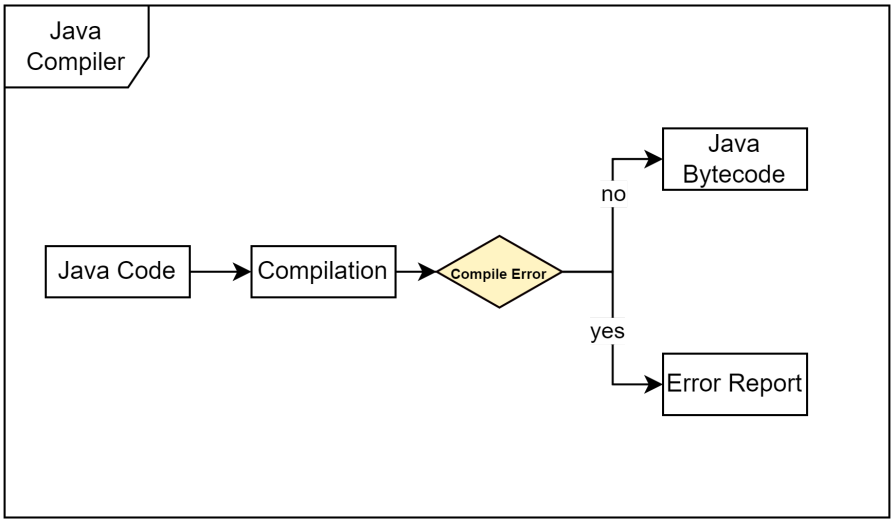

Figure 5.4: Backend - Java Compiler

5.3.3. Java Runner

Java Runner는 Java Compiler를 통해 Java bytecode를 생성한 Main Server가 실제로 해당 코드를 실행하는 작업을 수행한다. 이를 통해 Java 코드의 탄소 배출량 측정에 필요한 실행 시간 및 메모리 사용량을 얻을 수 있다. 만약 런타임 에러가 발생하면 에러 코드 및 메시지를 반환한다.

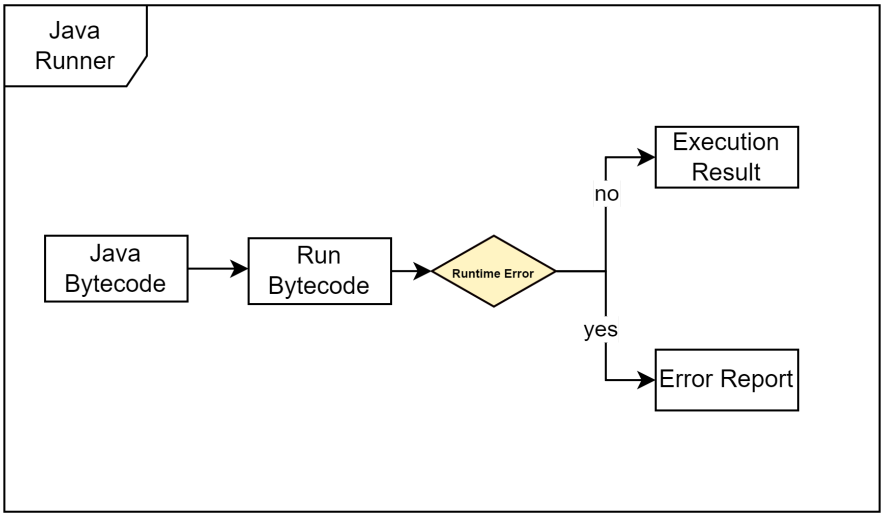

Figure 5.5: Backend - Java Runner

5.3.4. Carbon Calculator

Carbon Calculator는 Java Runner를 통해 얻은 Java 코드의 실행 시간 및 메모리 사용량 정보를 바탕으로 탄소 배출량을 측정한다. 탄소 배출량 계산식은 본 시스템의 요구사항 명세서에 작성된 [4.1 탄소 배출량 계산식 (Green Algorithm)]을 따른다.

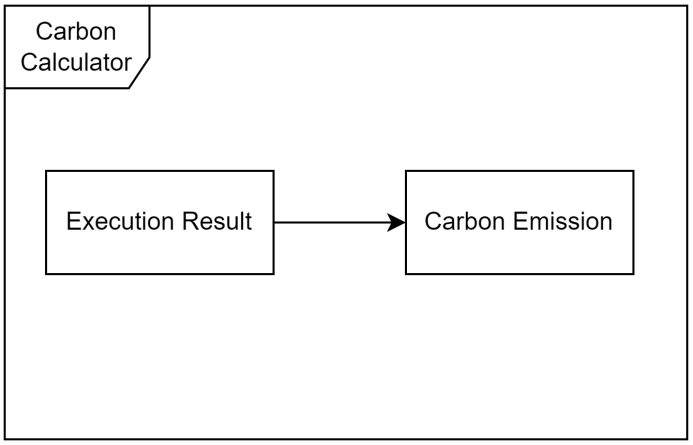

Figure 5.6: Backend - Carbon Calculator

5.3.5. Result Converter

Result Converter는 Carbon Calculator를 통해 측정한 Java 코드의 탄소 배출량 결과를 다양한 실생활 사용량 형식으로 변환하여 프론트엔드에 반환한다.

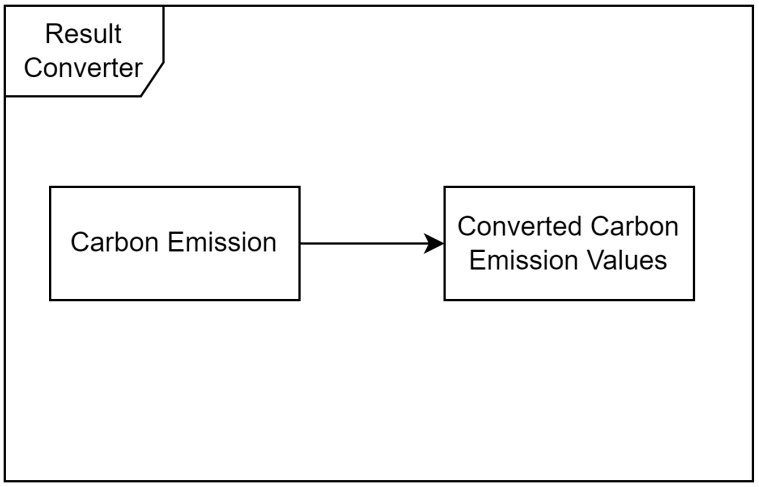

Figure 5.7: Backend - Result Converter

5.3.6. Database

Database는 프론트엔드에서 전송된 Java 코드와 이에 대한 실행 정보 및 탄소 배출량 측정 결과를 저장하고 관리한다. 7챕터의 데이터베이스 디자인에 Database의 데이터 구조를 포함한 자세한 정보가 제공되어 있다.

6. 프로토콜 디자인

6.1. Objectives

이 챕터는 애플리케이션과 백엔드 서버가 상호작용하기 위해 설정한 규약들에 대해 설명한다.

6.2. JSON

JSON(JavaScript Object Notation)이란 Javascript 객체 문법으로 구조화된 데이터를 표현하기 위한 문자 기반의 표준 포맷이다. 애플리케이션과 백엔드가 데이터를 주고받기 위한 포맷으로 JSON을 채택하였고, http header의 content-type을 사용하여 포맷을 명시하였다.

6.3. HTTP

HTTP(Hyper Text Transfer Protocol)는 W3 상에서 정보를 주고받을 수 있는 프로토콜이다. 애플리케이션은 웹 위에서 동작하므로 HTTP 프로토콜을 따르도록 하였고, 브라우저와 프론트 서버 혹은 클라이언트와 백엔드 서버가 통신할 때 사용한다.

6.4. RESTful API

Restful API란 REST 아키텍처 스타일의 제약 조건을 준수하고 RESTful 웹 서비스와 상호작용할 수 있도록 하는 API이다. API의 명확한 이해를 위해 채택하였으며, REST 규칙을 지키도록 규약을 설정하였다.

6.5. Application Programming Interface

6.5.1. Calculate Carbon Emission

6.5.1.1. Request

Attribute	Detail	
Protocol	HTTP	
Content-Type	JSON	
Endpoint	/carbon-emissions	
Method	POST	
Request Body	javaCode	측정할 자바 코드

6.5.1.2. Response

Attribute		Detail
Protocol	HTTP	
Content-Type	JSON	
Status Code	2XX - 성공 4XX - 사용자 오류 5XX - 서버 오류	
Response Body	emission	탄소 배출량
	powerConsumption	전력 소모량
	tvEmission	TV 탄소배출량과의 비교값
	carEmission	승용차 탄소배출량과의 비교값
	elevatorEmission	엘레베이터

		탄소배출량과의 비교값
	paperEmission	A4 용지 탄소배출량과의 비교값

7. 데이터베이스 디자인

7.1. Objectives

이 챕터에서는 시스템의 데이터 구조에 대해 설명한다. ER Diagram을 활용하여 데이터 구조를 시각화한다. 그 다음 Entity에 대해 자세히 설명한다.

7.2. ER Diagram

본 시스템은 3가지 entity로 이루어져있다.

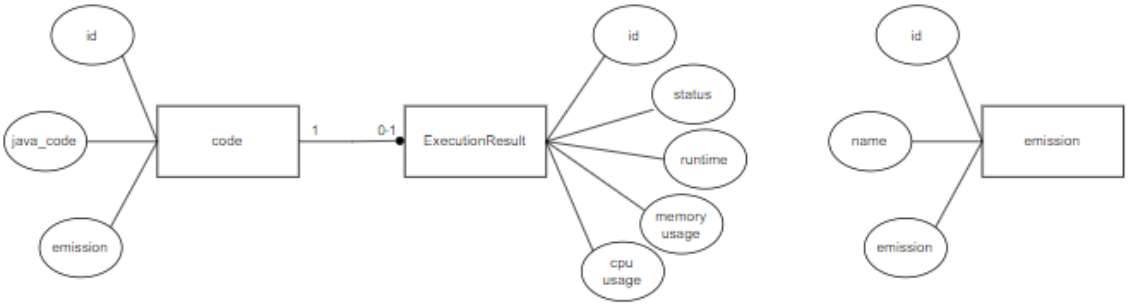

Figure 7.1: Code, ExecutionResult, Emission ER Diagram

7.2.1. Code

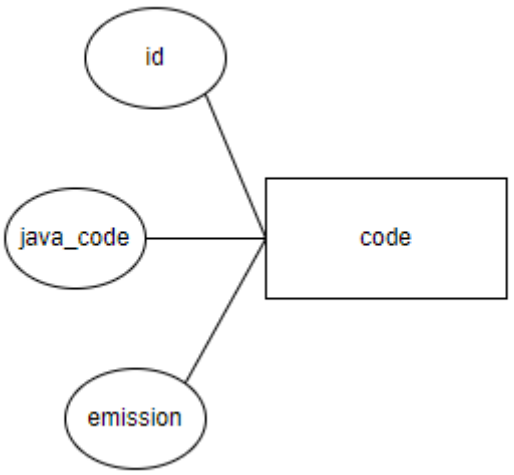

Figure 7.2: Code ER Diagram

Code Entity는 입력한 java code의 정보이다. id, java_code, emission을 attribute로 가지며, id가 primary key이다. java Code는 사용자가 입력한 코드이고, 이를 통해 계산한 탄소배출량이 emission이다.

7.2.2. ExecutionResult

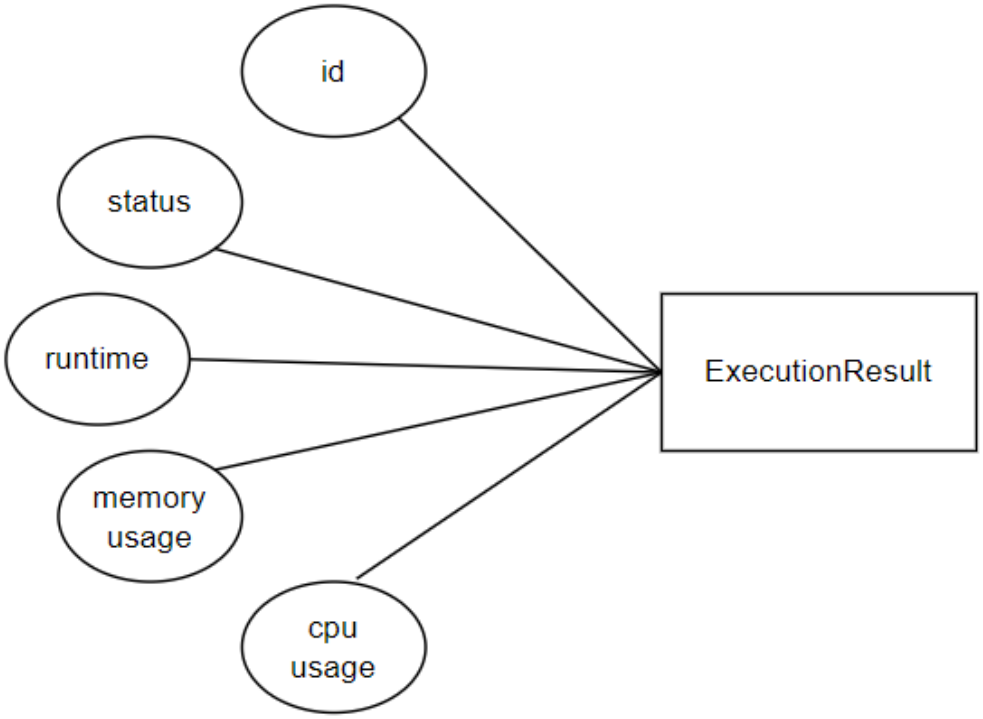

Figure 7.3: ExecutionResult ER Diagram

Execution Result Entity는 입력한 java code를 실행한 결과 정보이다. id, status, runtime, memory_usage, cpu_usage를 attribute로 가지며, primary key는 id이다. status는 코드 실행 결과 정보이고, runtime은 코드 실행 시간, memory usage는 코드 실행 시 메모리 사용량, cpu usage는 코드 실행 시 cpu 사용량에 대한 정보이다.

7.2.3. Emission

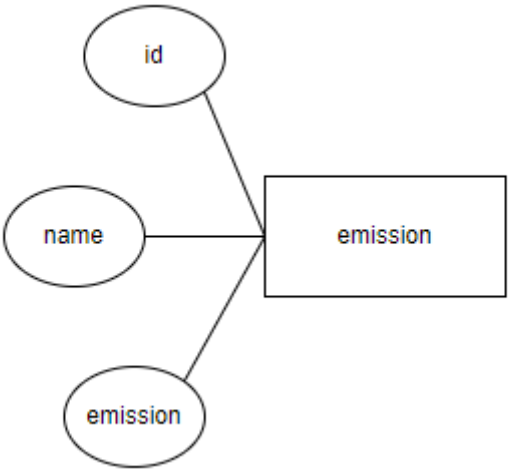

Figure 7.4: Emission ER Diagram

Emission Entity는 실 생활에서의 탄소 배출량 정보이다. id, name, emission을 attribute로 가지며, primary key는 id이다. name은 탄소 배출량의 이름으로 “승용차 1KM 주행”, “A4 용지 1장” 등 탄소 배출량을 구분할 수 있는 이름정보이다. emission은 탄소 배출량 정보이다.

7.3. Relational Schema

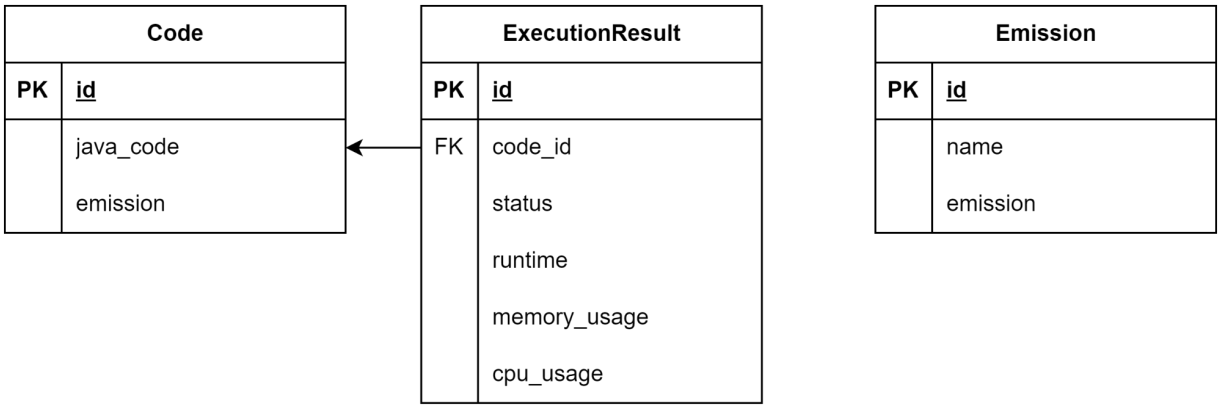

Figure 7.5: Relational Schema

7.4. SQL DDL

7.4.1. Code

```
CREATE TABLE code(
  id          int     NOT NULL AUTO_INCREMENT,
  java_code   text    NOT NULL,
  emission    float,
  PRIMARY KEY(id)
) ENGINE=InnoDB;
```

7.4.2. ExecutionResult

```
CREATE TABLE execution_result(
  id            int           NOT NULL  AUTO_INCREMENT,
  code_id       int           NOT NULL,
  status        varchar(255)  NOT NULL,
  runtime       bigint,
  memory_usage  float,
  cpu_usage     float,
  PRIMARY KEY(id),
  FOREIGN KEY (code_id) REFERENCES code(id)
) ENGINE=InnoDB;
```

7.4.3. Emission

```
CREATE TABLE emission(
  id        int           NOT NULL AUTO_INCREMENT,
  name      varchar(255)  NOT NULL,
  emission  float         NOT NULL,
  PRIMARY KEY(id)
) ENGINE=InnoDB;
```

8. 테스팅 계획

8.1. Objectives

이 챕터에서는 시스템의 테스팅과 관련한 계획을 설명한다. 테스팅에는 각 컴포넌트들에 대한 단위 테스트 및 통합 테스트, 실제 환경에서 애플리케이션을 실행하는 유저 테스트를 포함한다.

8.2. Testing Policy

8.2.1. 단위 테스트(Unit Test)

단위 테스트는 소프트웨어의 개별 구성 요소, 각 컴포넌트의 동작에 대한 테스트이다. 본 애플리케이션에서는 Code(탄소배출량 계산), executionResult(코드 실행 및 실행 시간, 메모리 사용량 측정), resultConverter(실생활 탄소배출량 비교)에 대하여 각각 단위 테스트를 진행한다.

8.2.2. 통합 테스트(Integration Test)

통합 테스트는 소프트웨어의 개별 컴포넌트를 결합하여 전체 시스템을 테스트하는 과정을 의미한다. 통합 테스트는 단위 테스트를 통과한 구성 요소를 결합하여 수행되며, 시스템의 기능을 테스트하고, 구성 요소 간의 상호 작용을 확인한다. 본 시스템에서는 Java compiler로부터 런타임 시간, cpu 및 메모리 사용량을 받아 Carbon Emission Calculator로부터 탄소 배출량을 측정하고, 이를 Result Converter로 보내 실생활 탄소배출량 값으로 환산하는 과정을 테스트한다.

8.2.3. 유저 테스트(User Test)

실제 배포 환경에서 사용자가 웹 애플리케이션을 통해서 Java 코드를 입력하고 이에 대한 결과값으로 탄소배출량 및 전력소모량, 실생활 환산값을 전달받아 UI에 적절한 값으로 출력하는 전체 과정을 테스트한다.

9. Development Plan

9.1. Objectives

해당 챕터에서는 CODEMETER 시스템의 개발 환경 및 기술을 프론트엔드와 백엔드로 나누어 설명한다.

9.2. Frontend Environment

9.2.1. JavaScript

Figure 9.1: JavaScript Logo

JavaScript는 객체 기반의 스크립트 프로그래밍 언어이다. 주로 웹 개발에서 동적이고 상호작용 가능한 웹 페이지를 만들 때 사용된다. CODEMETER에서는 사용자가 상호작용 할 수 있는 기본적인 동적 웹 페이지를 구현하는 용도로 사용할 것이다.

9.2.2. ReactJS

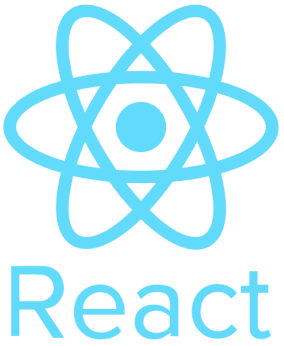

Figure 9.2: ReactJS Logo

React는 사용자 인터페이스를 구축하기 위해 Facebook에서 개발한 JavaScript 라이브러리이다. 재사용 가능한 UI 구성 요소를 만들 수 있어 복잡하고 반응적인 웹 애플리케이션을 효율적으로 구축할 수 있다. CODEMETER에서는 React를 이용해 소프트웨어를 구조화하여 코드 입력 및 결과 표시를 위한 컴포넌트들을 재사용 가능한 형태로 생성할 것이다.

9.2.3. Styled-components

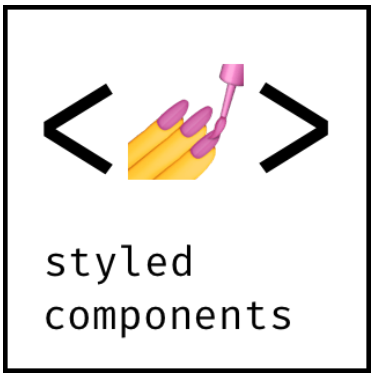

Figure 9.3: Styled-Components Logo

Styled-components는 React에서 사용되는 JavaScript 라이브러리로, JavaScript로 작성된 CSS를 이용해 컴포넌트별 스타일을 만든다. 모듈성이 높고 유지보수가 쉽다는 장점이 있기에, CODEMETER에서는 기본 CSS 대신 Styled-components를 이용해 컴포넌트들의 스타일을 만들고자 한다.

9.3. Backend Environment

9.3.1. GitHub

Figure 9.4: Github Logo

Github는 소프트웨어를 개발하고 Git을 통해 버전을 관리하기 위해 사용되는 툴이다. 이를 통해 여러 명의 개발자가 하나의 프로젝트를 동시에 관리하며 개발할 수 있으며, 각각의 컴포넌트들을 통합하기 쉽다. 따라서 CODEMETER의 개발 및 버전관리를 위해 이를 사용하고자 한다.

9.3.2. NestJS

Figure 9.5: NestJS Logo

NestJS는 확장 가능하고 유지보수 가능한 서버 측 응용 프로그램을 개발하기 위한 진보적인 Node.js 프레임워크이다. TypeScript와 모듈 아키텍처를 활용하여 조직적이고 효율적인 코딩을 촉진한다. CODEMETER에서는 NestJs 프레임워크를 활용하여 객체 지향적 설계 원칙을 준수하며 서버를 구성하고, 모듈 시스템을 적극적으로 활용하여 코드를 모듈화하고 의존성을 관리할 것이다.

9.3.3. MySQL

Figure 9.6: MySQL Logo

MySQL은 널리 사용되는 오픈 소스 관계형 데이터베이스 관리 시스템으로, 구조화된 데이터를 효과적으로 저장하고 관리하는 강력하고 확장 가능한 솔루션으로 웹 개발에서 많이 사용된다. CODEMETER에서는 MySQL을 사용하여 Java 프로그래밍 코드 및 해당 코드의 컴파일, 실행 성공 여부, 런타임, 탄소 배출량 정보등을 데이터베이스에 저장할 것이다.

9.4. Constraints

본 시스템은 이 문서에서 언급된 내용들에 기반하여 디자인되고 구현될 것이다. 이를 위한 세부적인 제약사항은 다음과 같다.

- 기존에 널리 쓰이고 있는 기술 및 언어들을 사용한다.
- 사용자의 입력 및 코드를 실행하고 결과를 저장하는 시간은 30초를 넘기면 안된다.
- Java code의 line 수는 1000자를 넘길 수 없고, 코드의 사이즈는 4KB를 넘길수 없다.
- Java code의 main 함수가 포함된 클래스의 이름은 Main이어야한다.
- 로열티를 지불하거나 separate license를 요구하는 기술 및 software의 사용을 지양한다.
- 전반적인 시스템 성능을 향상시킬 수 있도록 개발한다.
- 사용자가 편리하게 이용할 수 있도록 개발한다.
- 가능한 오픈소스 소프트웨어를 사용한다.
- 시스템 비용과 유지보수 비용을 고려하여 개발을 진행한다.
- 시스템 자원의 낭비를 막을 수 있도록 소스 코드를 최적화한다.
- 개발 및 유지보수 과정에서 ESLint 및 Prettier를 적극 활용하여 코드의 일관성과 품질을 유지하며, 컴파일 에러 및 코드 스타일 관련 문제를 사전에 예방한다.
- 협업 과정에서는 Git 커밋 메시지의 일관성을 유지하기 위해 AngularJS Git 커밋 컨벤션을 따른다.
- 자바스크립트 스타일 가이드와 HTML 표준을 따른다.
- 변수 이름과 함수 이름은 camel case를 사용한다.

9.5. Assumptions and Dependencies

본 문서의 모든 시스템은 데스크탑 환경에 기반하여 디자인 및 구현되었다고 가정하며 작성되었다. CODEMETER는 Window 10 이상, Google Chrome 버전 96 이상을 타겟으로 하고 있다. 이보다 낮은 버전으로 실행하거나 V8 Engine이 아닌 브라우저를 사용할 시 정상 동작하지 않을 수 있다.