Software Design Specification

탄소를 JAVA라

by

김민지, 김찬용, 안상현, 양승빈, 윤시형, 임동준, 최경식

Team1

Instructor: 이은석

Document Date: 2023.11.18

Faculty: Sungkyunkwan University

Contents

List of Image

List of Table

1. Preface

1.1 Objectives

본 디자인 명세서의 주요 독자와 문서의 목차, 각 장에서 다루는 내용에 대해 기술한다.

1.2 Readership

본 문서는 2023학년도 2학기 소프트웨어공학개론 (SWE3002-41) (이하 '교과목') 수업을 수강하는 대학생으로 이루어진 Team1이 고안하고 설계한 시스템 '탄소를 JAVA라'에 대한 디자인 명세서이다.

본 문서에서는 본 교과목 Team1의 구성원과 교수 및 조교를 주요 독자로 설정한다. 교육 및 문서작성 등 비상업적 목적으로 본 문서를 열람하고 사용하는 것을 허용한다.

1.3 Document Structure

1.3.1 Introduction
Introduction에서는 본 시스템의 필요성과 기능에 대한 설명을 포함하며, 시스템 디자인을 나타내기 위해 사용되는 diagram을 설명한다.

1.3.2 System Architecture - Overall
System Architecture - Overall에서는 시스템의 전반적인 구성을 설명하고 diagram을 활용하여 시각적으로 나타낸다. use-case diagram, sequence diagram이 사용된다.

1.3.3 System Architecture - Front end
System Architecture - Front end에서는 프론트엔드 시스템의 구조와 디자인에 대해 상세히 기술한다. Class diagram, sequence diagram을 사용하였다.

1.3.4 System Architecture - Back end
System Architecture - Back end에서는 백엔드 시스템의 구조와 디자인에 상세히 기술한다. Class diagram, sequence diagram을 사용하였다.

1.3.5 Protocol Design
Protocol Design에서는 프론트엔드 애플리케이션과 서버 사이의 통신 프로토콜에 대해 설명하며, request와 response 형식을 정의한다.

1.3.6 Testing Plan
Testing Plan에서는 시스템의 오류 및 결함 감지를 위한 테스트 계획을 기술한다.

1.3.7 Development Plan
Development Plan에서는 개발에 사용될 기술 스택과 시스템 환경에 대해 기술한다.

2. Introduction

2.1 Objectives

본 시스템의 기능과 동작 순서에 대해 소개한다. 본 문서에서 시스템 디자인 설명을 위해 사용되는 diagram의 특징을 기술한다.

2.2 System Overview

'탄소를 JAVA라' 서비스는 입력된 Java 코드를 실행할 경우 발생하는 탄소배출량을 계산하는 웹 서비스이다. 웹 페이지의 단일/다중 탭을 통해 입력된 Java 코드는 백엔드 서버에 전달되며, 서버에서는 탄소배출량을 계산한 후 계산된 탄소배출량과 분석 정보 및 서버 정보를 프론트엔드 시스템에 전달한다. 프론트엔드 시스템에서는 웹 페이지에 전달 받은 탄소배출량과 서버 정보를 표시하고, 탄소배출량 분석 정보를 시각화한다.

2.3 Applied Diagrams

2.3.1 Use Case Diagram
Use Case Diagram은 사용자와 시스템 간의 상호작용을 나타내는 다이어그램이다. 사용자와 시스템의 functional requirements를 연결하는 관계를 나타낸다. 사용자의 관점에서 시스템의 기능을 이해하기 쉬운 형태로 시각화한다.

2.3.2 Sequence Diagram
Sequence Diagram은 객체 간의 상호작용을 순서에 따라 표현하는 다이어그램이다. 객체들 사이의 메시지 교환과 시간 흐름을 나타내며, 수직선으로 시간의 흐름을 표현한다. 시스템의 특정 기능이나 시나리오를 이해하고 설계하는데 유용하다.

2.3.3 Class Diagram
시스템 내 클래스들과 그들 간의 관계를 도식화하는 정적 다이어그램이다. 클래스의 속성과 메서드, 클래스 간의 상속, 연관, 의존성과 같은 관계를 나타낸다. 시스템의 정적 구조를 나타내며, 시스템 내 클래스 구조를 파악하는데 용이하다.

2.3.4 Architecture Design Diagram (MVC)
애플리케이션을 모델, 뷰, 컨트롤러 세 부분으로 분리하여 소프트웨어의 구조를 나타낸다. 모델은 애플리케이션의 데이터와 비지니스 로직을 담당하며, 뷰는 사용자 인터페이스와 관련된 요소를 처리하고, 컨트롤러는 사용자의 입력을 받아 모델과 뷰를 조정한다. 개발 과정에서의 관심사를 명확하게 분리하여 복잡한 애플리케이션 구조를 단순하게 표현한다.

2.4 References

IEEE Std 1016-2009 IEEE Standard for Information Technology—Systems Design— Software Design Descriptions
https://standards.ieee.org/ieee/1016/4502/

2022 Spring 41 class team 1
https://github.com/skkuse/2022spring_41class_team1

2023 Spring 41 class team 4
https://github.com/skkuse/2023spring_41class_team4

2023 Spring 41 class team 10
https://github.com/skkuse/2023spring_41class_team10

3. System Architecture - Overall

3.1 Objectives

이 챕터에서는 프론트 엔드 설계에서 백 엔드 설계에 이르는 전 소프트웨어 시스템 구성에 대해 설명한다.

3.2 System Organization

이 서비스는 클라이언트 - 서버 모델을 적용하여 설계되었으며, 프론트엔드 애플리케이션은 사용자와의 모든 상호작용을 담당한다. 프론트 엔드 애플리케이션과 백 엔드 애플리케이션은 JSON 기반의 HTTP 통신을 통해 데이터를 주고받는다. 백엔드 애플리케이션은 프론트 엔드 애플리케이션으로부터 Java 코드를 전송받는다. 사용자가 Java 코드를 담은 요청을 보내면, 이를 처리하는 함수가 동작한다. 요청을 처리하는 함수는 먼저 사용자의 Java 코드를 실행하고, 그 결과를 획득한다. 실행 결과는 코드 출력물과 실행 시간 등으로 구성한다. 백엔드는 Java 코드를 실행한 결과를 바탕으로 탄소 배출량을 계산한다. 이 계산은 시스템 정보와 코드 실행 시간을 활용하여 이루어진다. 코드의 실행 시간은 시간(hour) 단위로 변환되어야 한다.

성공적으로 코드를 실행하면, 사용자에게 '성공' 상태와 함께 코드의 실행 결과, 실행 시간, 서버 정보가 JSON 형식으로 전달한다. 만약 실행 중 오류가 발생한다면 오류 정보가 담긴 결과를 사용자에게 반환한다. Java 코드 실행 중 발생하는 여러가지 예외는 서버 내부의 오류, 코드 실행 문제 등 다양한 원인에 의해 발생할 수 있다. 예외가 발생하면 서버는 HTTP 상태 코드와 함께 오류의 상세 정보를 문자열로 변환하여 프론트엔드에 전달한다. 이를 통해 서버의 안정성을 유지하며 사용자에게 유용한 오류 정보를 제공한다.

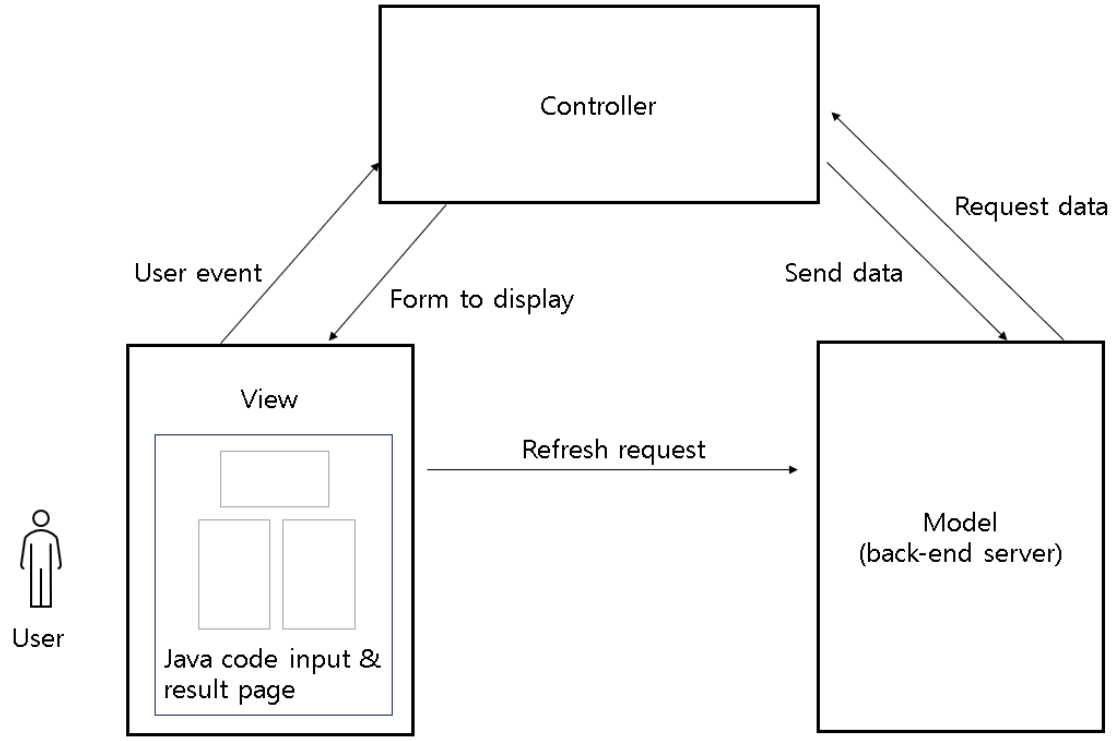

Image 3.1: Overall System Architecture

3.3 System Diagram

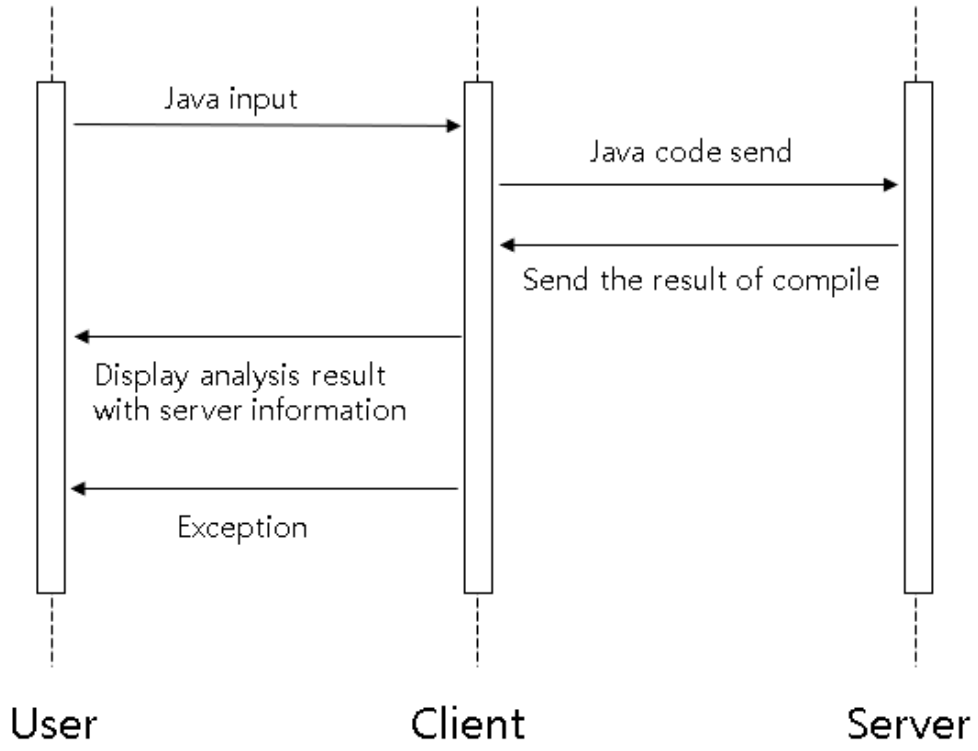

Image 3.2: Sequence Diagram - Overall

3.4 Use Case Diagram

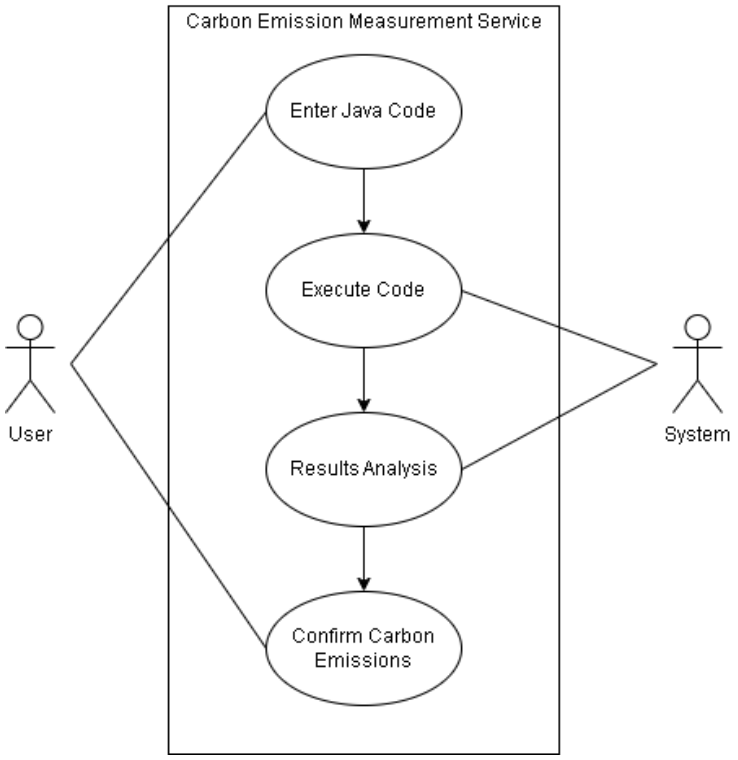

Image 3.3: Use Case - Overall

4. System Architecture - Front end

4.1 Objectives

이 장에서는 '탄소를 JAVA라' 프론트엔드 웹사이트의 구조, 속성 및 기능을 설명하고 탄소배출량 계산 시스템에 대한 각 구성 요소 사이의 관계를 설명한다.

4.2 Subcomponents

4.2.1 Code class
사용자의 코드 입력 정보를 포함하는 제출 클래스에 대한 세부 정보를 서술한다. 사용자는 탄소배출량을 계산하기 위해 주어진 코드 입력란에 Java 코드를 입력해야 한다. 사용자는 탭 추가 기능을 활용하여 여러 코드 입력 탭을 생성한 뒤 한 탭당 하나의 클래스를 입력하여 여러 클래스로 이루어진 프로그램 코드를 입력할 수 있다. 사용자는 코드 전송 버튼을 클릭하여 입력한 코드를 서버에 전송할 수 있다.

Objectives
- code: 사용자의 코드

Methods
- executeJavaCode(): Result

4.2.2 Result class
서버에서의 실행 결과를 포함하는 클래스에 대한 세부 정보를 서술한다. 웹사이트는 서버를 통해 Java 코드를 실행하고, 서버는 웹사이트의 요청에 따라 사용자에게 보여줄 실행 결과와 분석 데이터 및 서버 정보를 웹사이트에 전송한다.

Objectives
- execution_result: 실행 결과 (성공 또는 실패)
- elapsed_time: 실행 시간 (s)
- carbon_emission: 탄소배출량 (gCO2)
- cars: 동일한 양의 탄소(gCO2)를 자동차가 배출하며 이동할 수 있는 거리(km)로 환산한 값
- phones: 동일한 양의 탄소(gCO2)를 배출하며 충전할 수 있는 핸드폰 배터리 백분율(%)로 환산한 값
- air_conditioners: 동일한 양의 탄소(gCO2)를 배출하며 에어컨을 가동할 수 있는 시간(hour)으로 환산한 값
- trees: 동일한 양의 탄소(gCO2)를 나무 한 그루가 흡수하는 데 걸리는 시간(month)으로 환산한 값
- logical_cpu_cores: 서버의 논리 CPU 수
- physical_cpu_cores: 서버의 물리 CPU 수
- maximum_cpu_frequency: 최대 CPU 속도
- total_memory: 최대 메모리 크기
- city: 서버가 위치한 도시
- state: 서버가 위치한 주
- country: 서버가 위치한 국가 또는 지역
- server_message: Java 표준 출력

4.3 Class Diagram

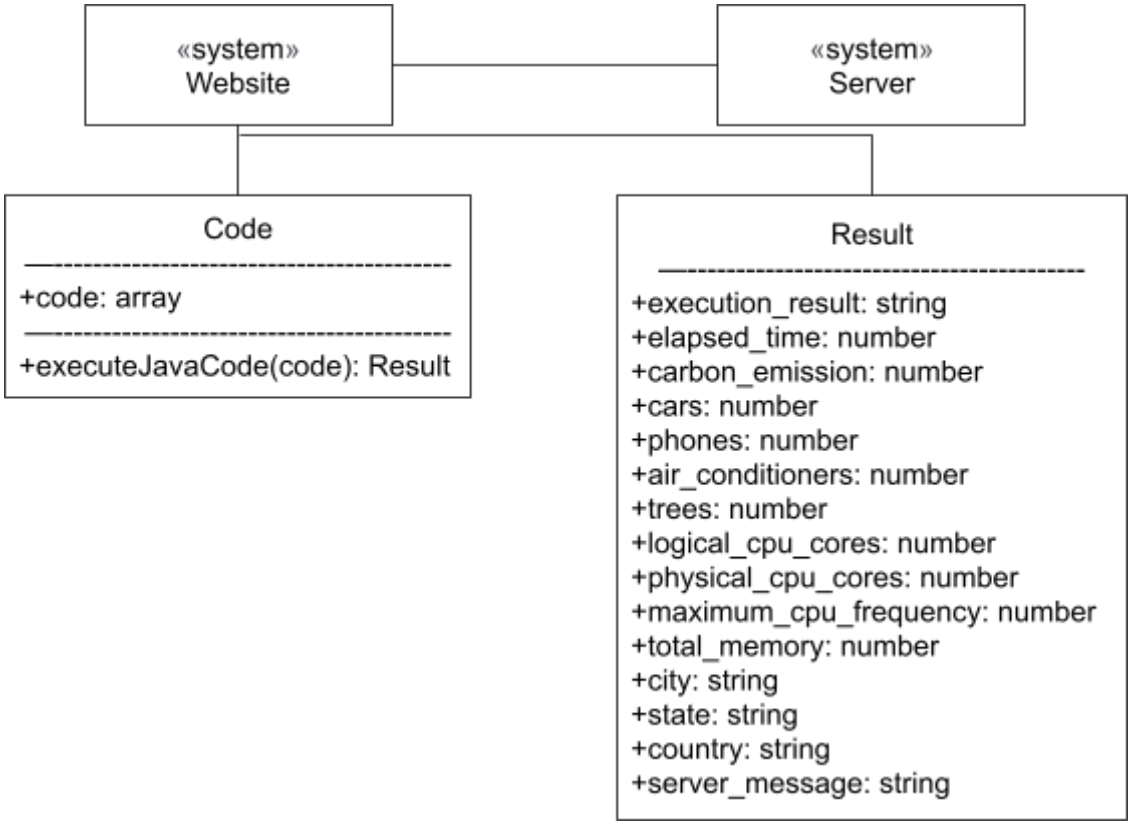

Image 4.1: Class Diagram - Front end

4.4 Sequence Diagram

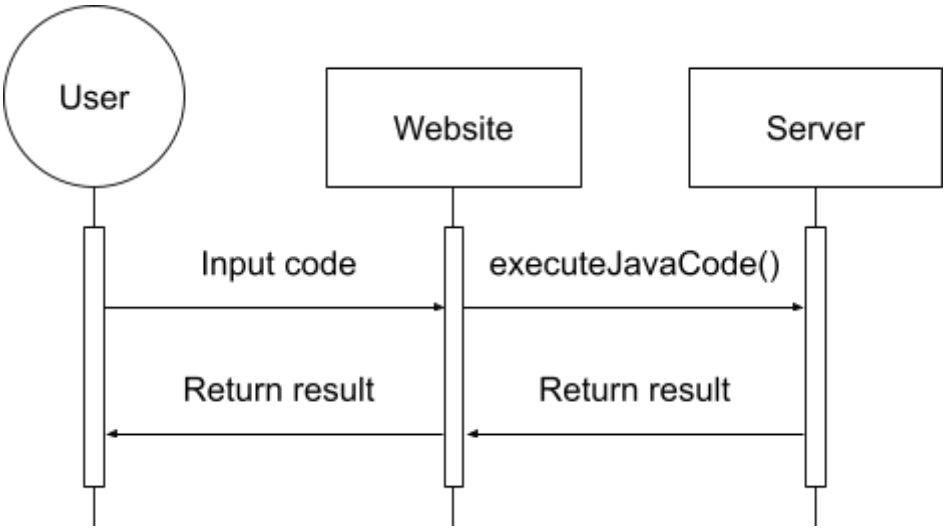

Image 4.2: Sequence Diagram - Front end

5. System Architecture - Back end

5.1 Objectives

이 챕터는 back end 시스템의 구조에 대해서 기술한다.

5.2 Architecture

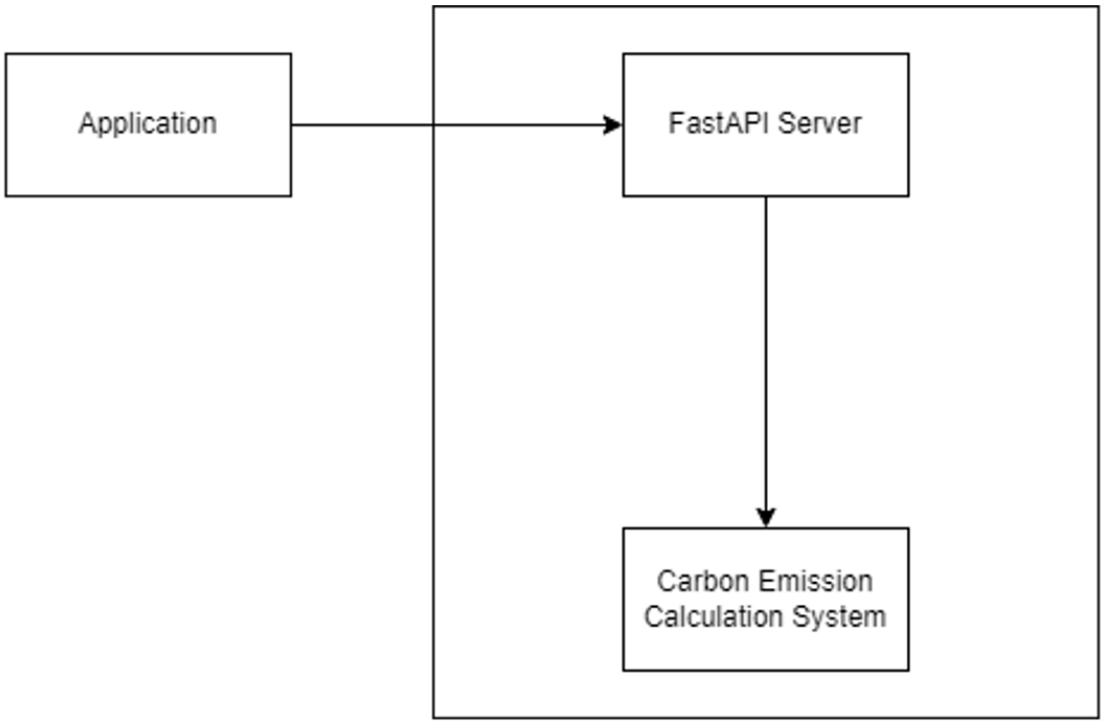

Image 5.1: Overall Back end Architecture

Java code를 컴파일해서 탄소 배출량을 계산해주는 본 시스템의 구조이다. Front end에서 요청이 들어오면 FastAPI 서버에서 해당 자바 코드를 컴파일한 후, 실행한다. 동시에 현재 서버의 환경을 분석한다. Java code의 실행 결과와 서버 환경 분석 결과를 바탕으로 탄소배출량을 계산한 후 다시 front end로 분석 결과를 보낸다.

5.3 Subcomponents

5.3.1 Java Code Executor Class
Java 코드를 컴파일한 후 실행하고, 실행 결과를 반환한다.

5.3.2 System Resource Analyzer Class
현재 시스템의 cpu와 메모리, 위치를 불러온다.

5.3.3 Carbon Emission Calculator Class
Java Code Executor로부터 얻은 결과와 System Resource Analyzer로부터 얻은 결과를 바탕으로 탄소 배출량을 계산한다.

5.4 Class Diagram

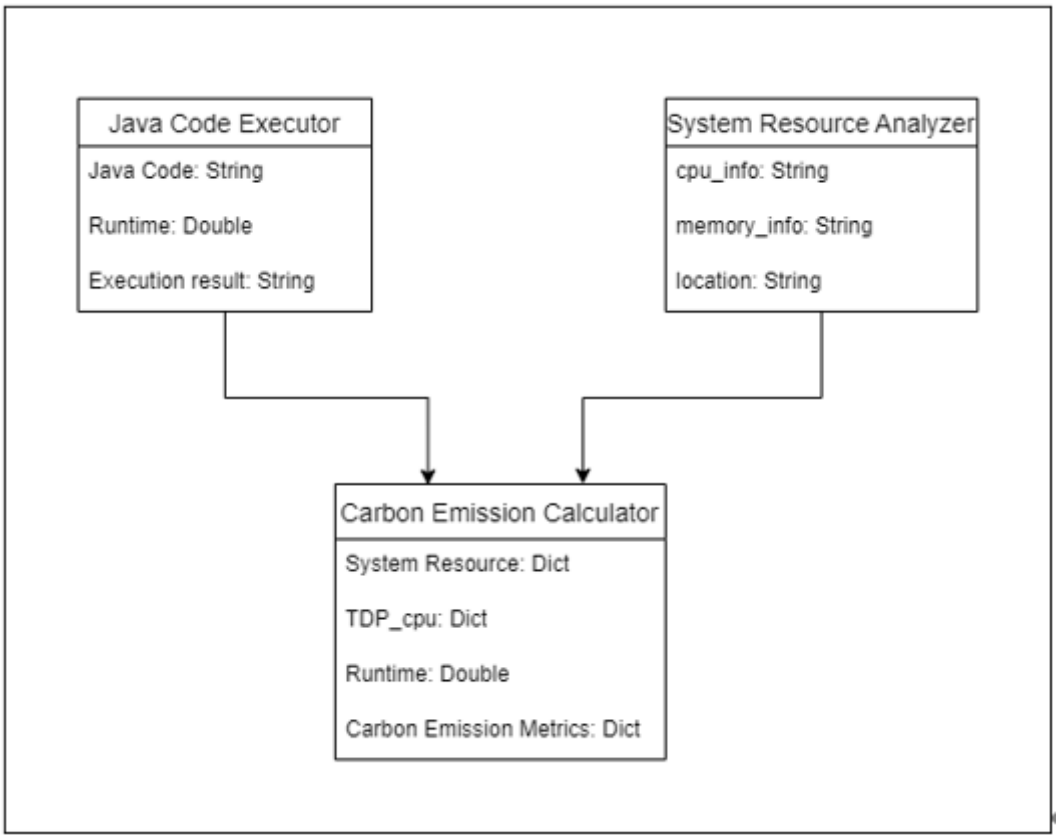

Image 5.2: Class Diagram - Carbon Emission Calculation System

5.5 Sequence Diagram

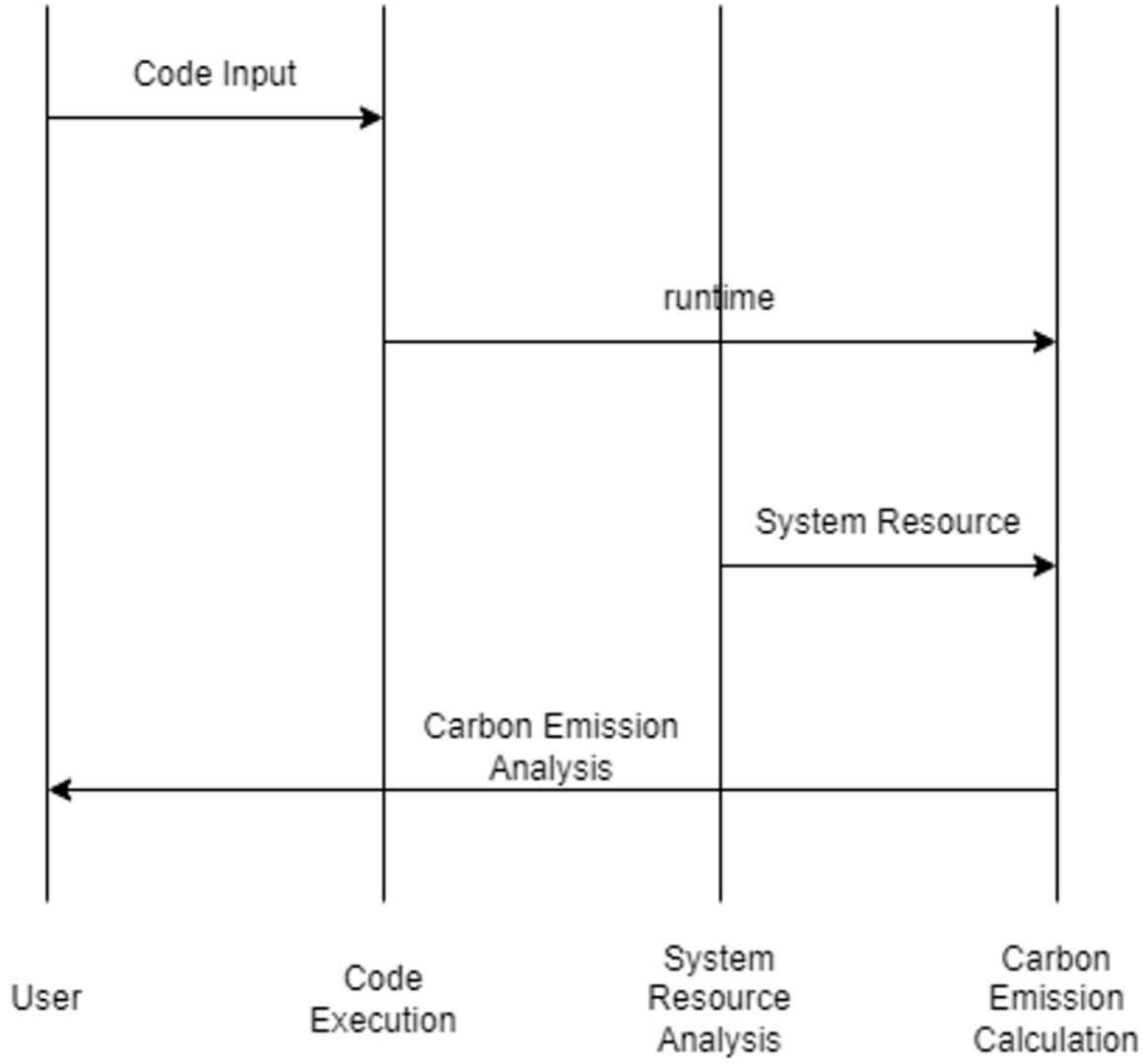

Image 5.3: Sequence Diagram - Carbon Emission Calculation System

6. Protocol Design

6.1 Objectives

이 챕터는 프론트 엔드 애플리케이션과 서버가 어떤 프로토콜로 서로를 연결하는지 기술한다. 또한 각 연결의 인터페이스를 어떤 형식으로 구성하고 있는지 서술한다.

6.2 HTTP

HTTP (HyperText Transfer Protocol)는 텍스트 기반의 통신 규약으로 인터넷에서 데이터를 주고 받을 수 있는 프로토콜이다. 클라이언트가 브라우저 혹은 URL을 통하여 서비스를 서버에 요청하면 서버에서는 해당 결과에 맞는 응답을 사용자에게 제공한다.

6.3 JSON

JSON (JavaScript Object Notation)라는 의미의 축약어로 데이터를 저장하거나 전송할때 많이 쓰이는 경량의 Data 형식이다. 서버 클라이언트 간의 데이터 교류에서 일반적으로 많이 사용된다. 본 프로젝트에서도 웹 서버와 웹페이지간의 데이터 교류에 사용된다.

6.4 HTTP Status Code

클라이언트가 서버에 요청을 보내면, 서버는 정보와 함께 3자리 숫자 상태 코드(Status Code)를 클라이언트에 전송한다. 정상적으로 통신을 완료하였을 경우 주로 200을 상태 코드로 전송한다. 클라이언트 혹은 서버의 문제로 인해 오류가 발생하였을 경우 백의 자리 숫자가 4 혹은 5인 상태 코드를 전송한다. 본 프로젝트의 모든 경우는 HTTP 상태 코드 표준을 따른다.

6.4 Access Test

<Request>

Attribute	**Detail**	
Protocol	HTTP	
Method	GET	
Request Body	None	None

Table 6.1: Table of Access Test Request

<Response>

Attribute	**Detail**	
Protocol	HTTP	
Success Code	200 OK	
Failure Code	HTTP error code = 400 (Bad Request, Overlap)	
Success Response Body	status	200
	Message	“hello from server”
Failure Response Body	Message	Failure message

Table 6.2: Table of Access Test Response

6.5 Execute Java code

<Request>

Attribute	**Detail**	
Protocol	HTTP	
Method	POST	
Request Body	Code	Java Code

Table 6.3: Table of Execute Java code Request

<Response>

Attribute	**Detail**	
Protocol	HTTP	
Success Code	200 OK	
Failure Code	HTTP error code = 500 (Internal Server Error)	
Success Response Body	status	“Success”
	output	java_execution_result["output"]
	runtime	java_execution_result['runtime']
	server_info	system_info
Failure Response Body	Message	Failure message

Table 6.4: Table of Execute Java code Response

7. Testing Plan

7.1 Objectives
이번 장에서는 3가지 주요 하위 그룹(development testing, release testing 및 user testing)을 포함하는 테스트에 대한 계획을 설명한다. 이러한 테스트는 애플리케이션의 잠재적인 오류와 결함을 감지하고 애플리케이션의 완성도를 높여 안정적인 시스템을 user에게 제공하기 위해 존재한다.

7.2 Testing Policy

7.2.1 Development Testing
Development Testing은 현재 시스템의 품질을 확인하고 코드의 오류를 초기에 감지하고 수정하기 위한 과정을 말하며 개발 초기단계부터 마지막 단계까지 계속해서 진행한다. 이 단계에서는 소프트웨어가 계속해서 개발 중인 단계에 있기 때문에 불안정할 수 있고, 프론트 엔드와 백 엔드(website와 server) 간 충돌이 일어날 수 있다. 따라서 이 단계에서는 data flow를 검토하고 개발 팀원간의 code review 과정 등을 활용하여 소프트웨어의 performance, reliability, safety에 초점을 맞추어 테스트를 진행한다.

Performance

본 시스템의 performance에 관한 지표는 다음과 같다.

- 사용자가 입력한 Java code의 탄소배출량 시각화까지의 지연시간이 12초 이내로 완료되어야 한다.
- 사용자의 다중 class입력으로 인한 Java code tab 변경 시 화면이 5초 이내로 표시되어야 한다.
- 1초 이내의 Runtime을 가지는 Java code 실행 시, 시스템 과부하 없이 작동해야 한다. 여러 사용자가 접속하더라도 5초 이내로 실행 완료되어야 한다.
- 1초 이상의 Runtime을 가지는 Java code 실행 시, 10초 이내에 실행을 완료해야 한다. 여러 사용자가 접속하더라도 10초 이내로 실행 완료되어야 한다.
- Java code 오류 발생 또는 10초 이상의 실행 시, 실행을 중지하고 11초 이내로 이를 사용자에게 알려주어야 한다.
- 시스템은 동시에 최소한 10명 사용자의 접속을 유지할 수 있어야 한다. 각 웹페이지는 동시에 최소한 10명의 사용자가 접속하여도 10초 이내 page load를 완료할 수 있어야 한다.

다음과 같은 지표를 만족할 수 있도록 실제 테스트를 다양한 환경에서 여러번 수행하여 지연시간이 위와 같은 지표를 만족하였는지 확인한다.

Reliability

시스템이 오류를 발생하지 않고 안전하게 작동하려면 시스템을 구성하는 각각의 하위 구성 시스템과 장치가 정상적으로 작동한 뒤, 전체 시스템으로 연결되어야 한다. 따라서 하위 시스템 개발 단계부터 개발 테스트를 순차적으로 거쳐 각 시스템을 전체 시스템에 통합하는 동안 오류를 반복적으로 확인하고 그에 따라 수정해야 한다.

Safety

시스템에 오류가 발생했을 때, 시스템에 치명적인 영향이 가지 않도록 빠르게 오류를 인식하고 이를 해결하기 위해서는 개발 초기 단계부터 오류가 발생했을 때의 대응 계획을 마련하고 코드를 지속적으로 refactoring하여 코드의 가독성과 유지보수성을 향상해야 한다. 본 시스템에서는 사용자가 입력한 Java code의 실행이 실패하거나, 긴 실행 시간을 포함한다면 컴파일을 종료하고 관련 오류 메시지를 사용자에게 제공함으로써 시스템 중단 없이 입력한 코드를 수정할 수 있도록 하였다.

7.2.2 Release Testing

Release Testing는 본 시스템이 사용자에게 출시되기 전에 소프트웨어/애플리케이션이 최신의 완전한 상태로 배포되어 안정적으로 작동하는지 테스트한다. 본 소프트웨어 배포는 일반적으로 모든 시스템의 기본 구현을 포함하는 알파 버전에서 시작하여 개발 테스트를 거친 뒤 사용자 및 Release Test를 포함한 추가 테스트를 위해 베타 버전을 Github에 release할 예정이다.

7.2.3 User Testing

User Testing은 본 시스템이 최종 사용자들에게 제공되기 전에 사용자 그룹에 의해서 테스트되는 단계이다. 따라서 개발 팀원을 포함한 금학기 소프트웨어공학개론(SWE3002-41) 수업과 관련된 사용자에게 베타버전을 배포하고 자체적인 사용 및 일어날 수 있는 여러 시나리오를 수행하며 각각의 진행과정에서 오류가 없는지 검토하는 방식으로 실행한다.

7.2.4 Testing Case

Test case는 앞서 살펴본 performance, reliability, safety에 초점을 맞추어 설계한다. 각각의 측면에서 최소 3개 이상의 test case와 scenario를 통해 본 소프트웨어에 대한 테스트를 진행한다.

8. Development Plan

8.1 Objectives

해당 챕터에서는 시스템의 개발 환경과 기술적 측면에 대해 서술한다.

8.2 Front end Environment

8.2.1 React

Image 8.1: React Logo

React는 사용자 인터페이스를 구축하기 위한 오픈 소스 JavaScript 라이브러리이다. 단일 페이지 응용 프로그램 개발에 적합하며, 웹뿐만 아니라 모바일 어플리케이션 개발에도 사용할 수 있다. '탄소를 JAVA라'의 웹사이트 사용자 인터페이스를 제작하는 데에 React를 사용한다. 사용자 인터페이스의 여러 컴포넌트를 디자인하고 제작하는 데에는 MUI를 사용하였으며, 코드 입력 기능을 구현하기 위해서 Monaco Editor를 React와 통합하여 사용하였다. 변수 이름은 Camel case를 따른다.

8.2.2 CSS

Image 8.2: CSS Logo

CSS는 웹 페이지의 디자인과 레이아웃을 정의하는 언어이다. 본 프로그램에서는 사용자가 접속하는 웹사이트에서 경험하는 전체적인 레이아웃과 그것들의 세부적 디자인을 구현하는 데에 CSS를 사용하였다. 변수 이름은 Kebab case를 따른다.

8.3 Back end Environment

8.3.1 Python

Image 8.3: Python Logo

Python은 명료성과 코드 가독성에 중점을 두고 설계된 고급 프로그래밍 언어이다. 간결하고 읽기 쉬운 문법을 가지고 있으며, 많은 라이브러리와 오픈소스가 뒷받침하고 있어 다양한 프로그래밍 요구사항에 유연하게 대처할 수 있다. 본 프로그램에서는 입력된 코드를 실행하고 런타임을 측정한 후, 계산 방식에 따라 코드의 탄소배출량을 계산하는 서버의 기능을 구현하기 위해 Python을 사용한다. 변수 이름은 Snake case를 따른다.

8.3.2 FastAPI

Image 8.4: FastAPI Logo

FastAPI는 Python 기반의 웹 프레임워크이다. 주로 API 개발에 초점을 맞추고 있으며, 높은 성능과 데이터 검증이 쉬운 특성을 가지고 있어 API 개발을 빠르고 효율적으로 만들어 개발자들에게 유연성과 확장성을 제공한다. 본 프로그램에서는 입력된 코드를 실행하고 런타임을 측정한 후, 클라이언트 웹사이트와 실행 결과를 송수신하기 위해 FastAPI을 사용한다. 변수 이름은 Snake case를 따른다.

8.4 Communication

8.4.1 Github

Image 8.5: Github Logo

Github는 개발자들이 소프트웨어를 개발하고 협업하는 데에 사용되는 플랫폼이다. 여러 명의 개발자가 하나의 소프트웨어를 함께 개발하고 관리하는 데에 용이하며, 각자 만든 요소들을 통합하기 쉽다. 본 프로젝트에서는 Github을 통해 코드를 공유하며 프로그램을 개발한다.

8.5 Constraints

본 시스템을 본 문서에 기재된 요구 사항과 제약사항을 바탕으로 개발하였다. 본 문서에서 명시하지 않은 요구사항과 제약사항은 개발자의 재량에 의해 조정할 수 있다. 아래의 사항들은 프로그램을 구현할 때 준수해야 할 사항이다.

- 사용자가 이해하기 쉬운 방향으로 시스템을 설계한다.
- 성능이 입증된 기술을 사용한다.
- 가급적 공개된 오픈소스를 사용한다.
- 사용자가 이용하기에 부족하지 않은 길이의 코드를 입력할 수 있도록 한다.
- 오류 발생의 경우 빠른 수정이 가능하도록 프로그램을 설계한다.
- 사용자가 효율적인 그린 코딩을 할 수 있도록 유도하는 방식으로 프로그램을 설계한다.
- 사용자가 필수 프로그램 이외에 다른 프로그램을 설치할 필요가 없도록 설계한다.

8.6 Assumptions and Dependencies

본 프로그램을 다음과 같은 환경을 바탕으로 탄소 배출량 계산을 진행한다는 가정을 바탕으로 개발하였으므로 이와 다른 환경에서 이 프로그램을 작동하면 원하는 결과가 도출되지 않을 수 있다.

- 1.0GHz 이상의 프로세서
- 4GB 이상의 RAM
- Windows, MacOS, Linux 환경
- 최신 버전의 Chrome, Firefox, Microsoft Edge 등 주요 웹 브라우저
- 인터넷 연결

9. Appendix

9.1. Software Design Specification Standard

본 요구사항 명세서는 IEEE 표준(IEEE Std 1016-2009 IEEE Standard for Information Technology — Systems Design — Software Design Descriptions)을 기반으로 작성하였다. 다만 디자인을 명세화 하는데 필요하지 않은 일부 구성요소는 활용하지 않았다.

9.2. Document History

Document History			
Date	Version	Description	Writer
2023-11-18	V 1.04	Preface, Introduction	양승빈
2023-11-18	V 1.05	System Architecture - Overall	김민지
2023-11-18	V 1.03	System Architecture - Front end	임동준
2023-11-18	V 1.04	System Architecture - Back end	김찬용
2023-11-18	V 1.04	Protocol Design	안상현
2023-11-18	V 1.05	Testing Plan	최경식
2023-11-18	V 1.03	Development Plan, Appendix	윤시형
2023-11-19	V 1.06	Formatting	전 팀원