Design Specification

Green Code

소프트웨어 공학개론 10 조

2020312627 권서영

2019312104 김동현

2019310333 남윤성

2018313451 송재현

2018314520 전윤혁

2019314159 조영길

Contents

1. Purpose

본 문서에서 예상되는 독자, 범위, 그리고 목적에 대해 설명한다. 또한 문서의 구조를 각 챕터별로 요약하여 설명한다.

1.1 Readership

본 문서의 주요 독자는 본 시스템의 개발자 및 소프트웨어 공학 개론 수업의 교수, 조교, 수강생이다.

1.2 Scope

본 문서는 Green Code 의 디자인과 구현 과정에 대한 상세한 설명을 제공한다. Green Code 는 입력된 프로그래밍 코드의 탄소 배출량을 계산하는 웹사이트이다.

1.3 Objective

본 문서의 목적은 프로젝트의 디자인 및 구현 과정을 체계적으로 설명하고, 프로젝트의 개요와 목표를 명확히 하는 것이다.

1.4 Document Structure

본 문서는 다음과 같이 구성된다.

1) Purpose
   해당 문서의 예상 독자, 범위, 목적, 구조에 대해 설명한다.

2) Introduction
   해당 문서를 작성하는데 사용된 도구, 다이어그램, 참고 자료에 대해 설명한다.

3) System Architecture – Overall
   전체 시스템 아키텍처와 그 구성 요소에 대해 설명한다.

4) System Architecture – Frontend

Frontend 아키텍처와 그 구성 요소에 대해 설명한다.

5) System Architecture - Backend
   Backend 아키텍처와 그 구성 요소에 대해 설명한다.

6) Protocol Design
   사용된 프로토콜과 그 설계에 대해 설명한다.

7) Testing Plan
   테스팅 정책을 설명한다.

8) Development Plan
   시스템 구현 계획 및 구현을 위한 개발 도구, 라이브러리 등의 개발 환경을 설명한다.

9) Supporting Information
   추가적인 정보와 문서 버전 정보를 제공한다.

2. Introduction

2.1 Objective

이 섹션에서는 해당 문서를 작성하는데 사용된 도구, 다이어그램, 프로젝트 범위에 대한 설명, 참고 자료에 대해 설명한다.

2.2 Applied Diagram

2.2.1 draw.io

본 문서의 다이어그램 작성에 사용되었다. 웹 기반의 다이어그램 작성 도구로, 사용이 간편하고 다양한 템플릿과 기능을 제공하여 복잡한 시스템의 구조를 명확하게 시각화 할 수 있게 한다.

2.2.2 Data Flow Diagram

시스템 내에서 데이터가 어떻게 이동하는지를 보여주는 다이어그램이다. 시스템의 이해도를 높이고, 데이터의 흐름과 변환 과정을 명확하게 설명한다.

2.2.3 Block Diagram

시스템의 구성 요소와 구성 요소 간의 관계를 블록과 화살표로 표현한 다이어그램이다. 복잡한 시스템의 전반적인 구조 및 작동 원리를 쉽게 설명한다.

2.2.4 Architecture Diagram

시스템의 전체적인 구조를 나타내는 다이어그램이다. 시스템의 구조와 데이터 흐름, 그리고 구성 요소 간의 관계를 명확하게 설명한다.

2.2.5 Class Diagram

객체 지향 프로그래밍에서 클래스와 그들 사이의 관계를 보여주는 다이어그램이다. 클래스와 클래스 간의 관계를 명확하게 나타냄으로써 시스템의 객체 지향 설계를 설명한다.

2.2.6 Sequence Diagram

시스템의 객체들이 어떻게 상호작용하는지를 시간 순서에 따라 나타내는 다이어그램이다. 시스템의 동적 행동을 설명한다.

2.2.7 Project Scope

본 문서에서 설계하는 Green Code 는 사용자들이 자신의 코드로 인한 탄소 배출량을 쉽게 확인할 수 있도록 하고, 이를 통해 코드 작성 방식을 개선하여 친환경적 코딩 습관을 갖출 수 있도록 하기 위하여 설계되었다. 복잡한 입력 대신 Java 코드만을 입력하면 해당하는 다양한 데이터를 시각적으로 확인할 수 있다.

2.3 References

"Documenting Software Architectures: Views and Beyond" - Paul Clements, Felix Bachmann, Len Bass, David Garlan, James Ivers, Reed Little, Paulo Merson, Robert Nord, Judith Stafford

IEEE Std 830-1998 - IEEE Recommended Practice for Software Requirements Specifications

https://github.com/skkuse/2022spring_41class_team1/blob/main/docs/Team%201%20Software%20Design%20Specification.pdf

3. System Architecture - Overall

3.1 System Organization

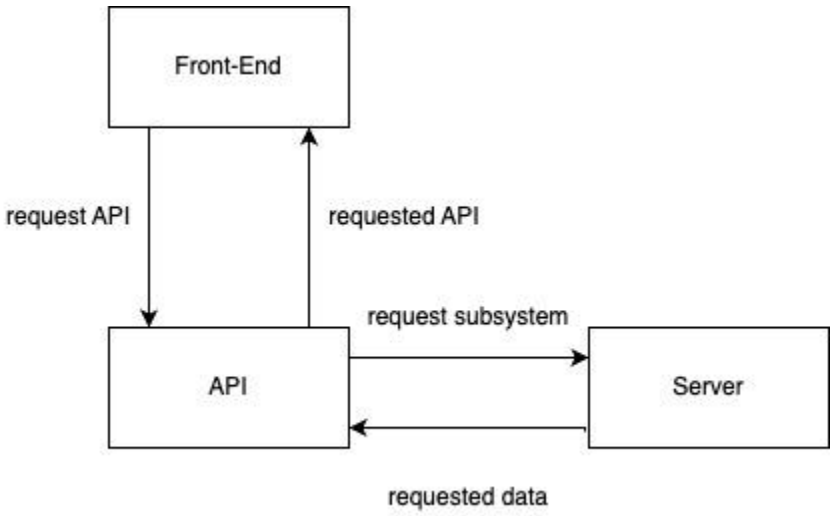

Figure 1. System Organization

Green Algorithm 의 시스템은 크게 frontend 와 backend 로 이루어진다. Frontend 는 사용자와 상호작용하여 데이터를 요청받고, 요청받은 데이터를 확인할 수 있도록 한다. Backend 는 Web server 로 구성된다. Server 는 여러 subsystem 들로 구성되어 있으며, frontend 는 fetch API 를 통해 subsystem 을 호출할 수 있다.

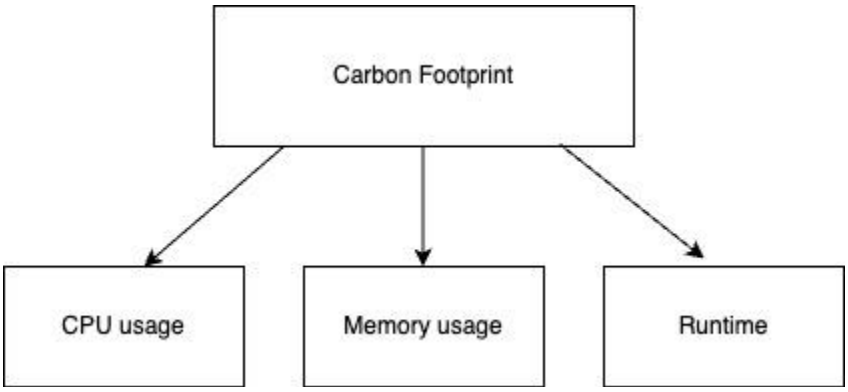

Figure 2. Carbon Footprint Calculation

Carbon Footprint 는 입력 받은 자바 소스코드를 실행할 때 사용된 주요 자원들을 계산해 도출한다.

3.2 System Architecture: Overall

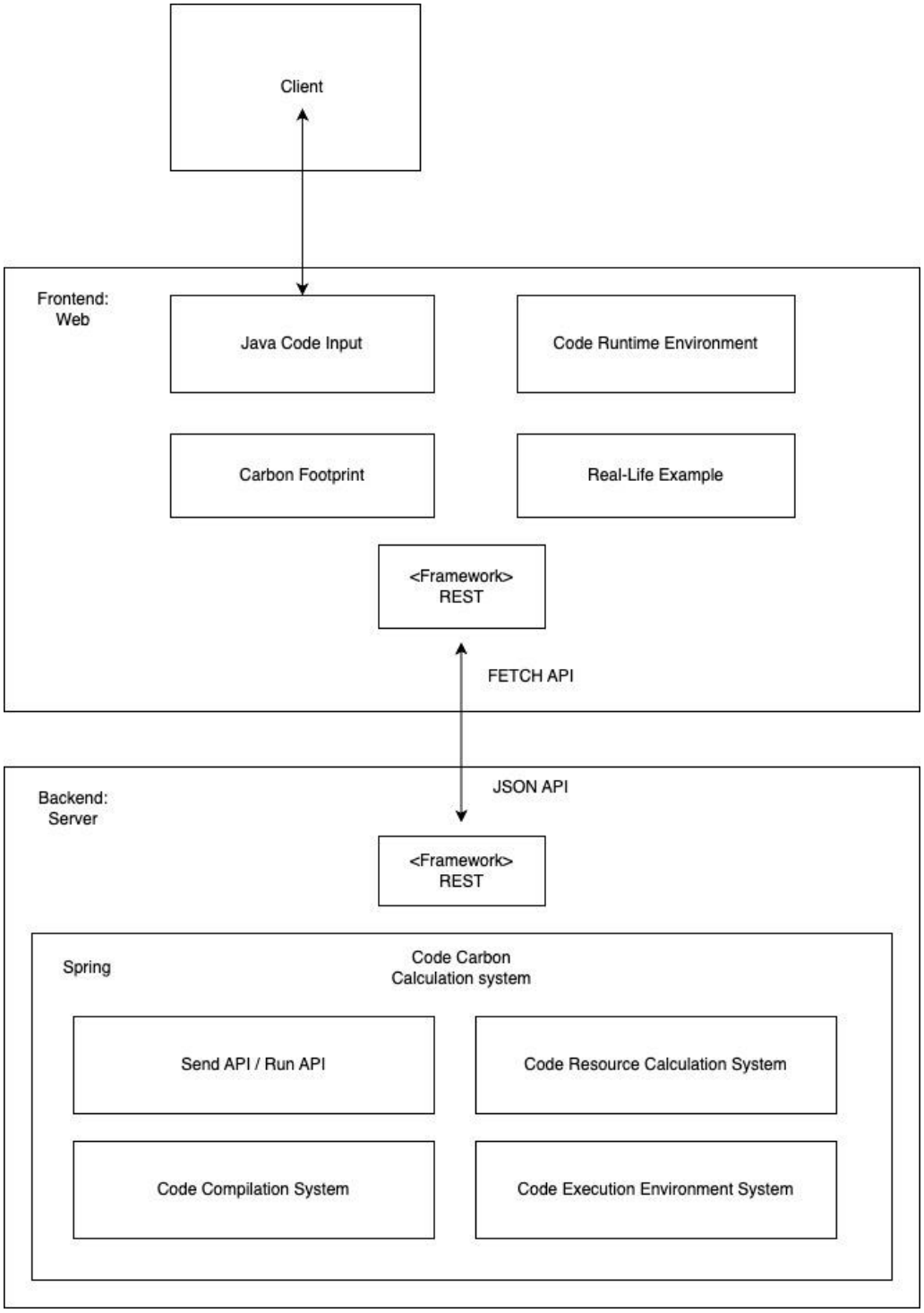

Figure 3. Overall System Architecture

Green Algorithm 의 세부적 구성 요소는 위 그래프에 명시된 바와 같다. 대략적으로 Frontend 는 user 에게 소스 코드를 입력받고 그리고 각 소스코드의 탄소 배출량을 표시해 주는 역할을 맡는다. Backend 는 프론트로부터 전달받은 소스코드를 컴파일하고, 실행하고, 실행환경을 통해 탄소 배출량을 계산하는 역할을 담당한다.

소스코드의 입력은 Java 로만 제한하고, 실행시간이 길어지면 backend 쪽에서 실행을 강제 종료한다.

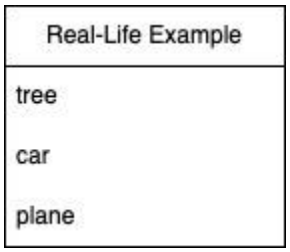

Figure 4. Real Life Example Displayed in Web Server

입력된 자바 소스코드의 탄소 배출량의 직관적 인식을 위해 위와 같은 실생활 자료들을 참고해 웹사이트에 표시해 준다.

전반적으로 복잡할 것이 없는 시스템 아키텍처이지만, 코드 보안과 사용자의 편의와 직관성을 고려한 알고리즘이다.

4. System Architecture – Frontend

4.1 Objectives

Frontend 에서의 목표는 사용자로부터 코드를 입력 받고, 그 코드를 backend 로 전달하여, backend 로부터 코드와 관련된 parameter (Runtime, Power draw for cores, usage, power draw for memory)을 전달받는 것이다.

또한 이 parameter 들을 이용하여 energy needed 와 carbon footprint 값을 계산할 수 있다. 추가적으로 계산된 carbon footprint 값을 이용해서 실생활에서 탄소배출량과 밀접한 관계에 있는 예시들을 시각적으로 표현해 탄소가 얼마나 배출되었는지 직관적으로 이해할 수 있게 해준다.

4.2 Components

• Client: 사용자로서 코드를 입력하여 입력한 코드를 바탕으로 한 정보들을 얻는다

• Web: 웹은 사용자로부터 코드를 입력 받고 backend server 에 코드를 넘긴다. Server 로부터 코드 관련 정보들을 받고 다시 웹에 표시한다.

• Server: 서버는 frontend 웹에서 전달받은 코드를 실행시키고 코드에 관련된 parameter 값들을 계산하여 다시 frontend 웹에 관련 정보들을 전달한다.

4.2.1 Class Diagram

• Carbon footprint = energy needed × carbon intensity

• energy needed

= runtime × power draw for cores × usage + power draw for memory × PUE ×PSF

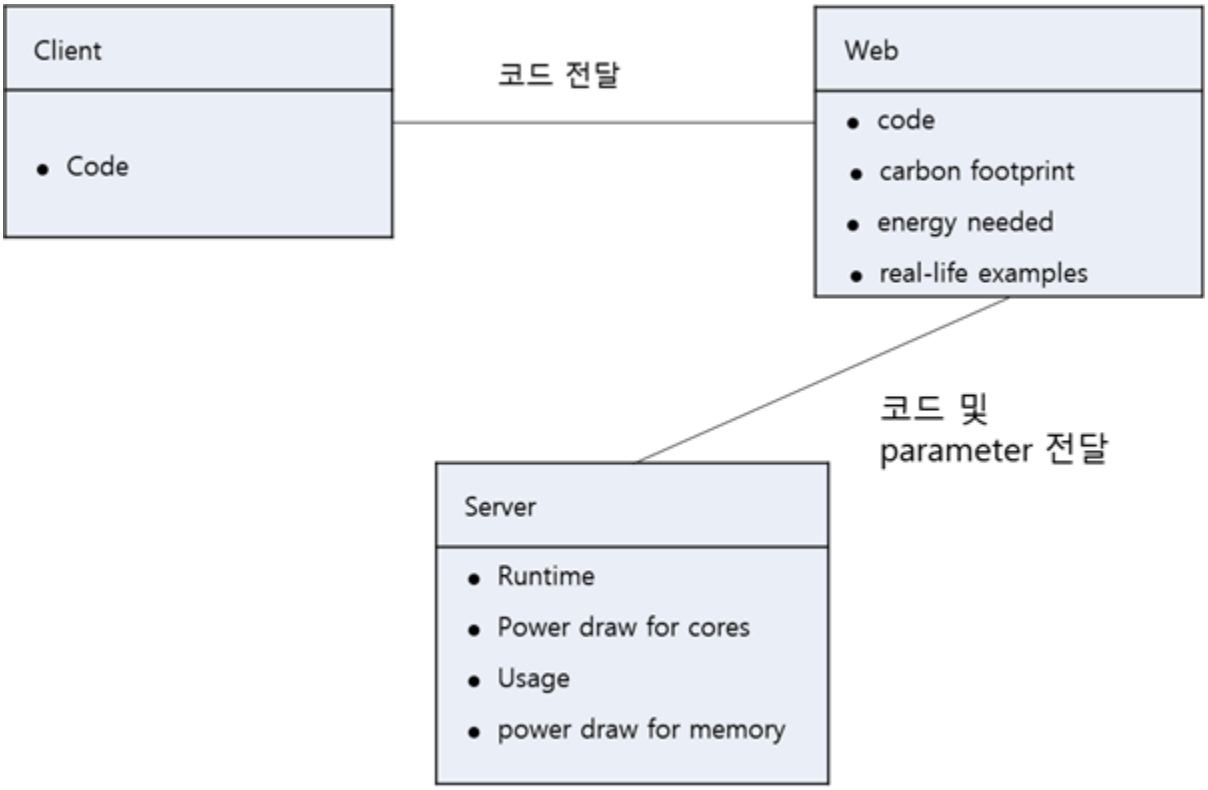

Figure 5. Frontend Class Diagram

4.2.2 Sequence Diagram

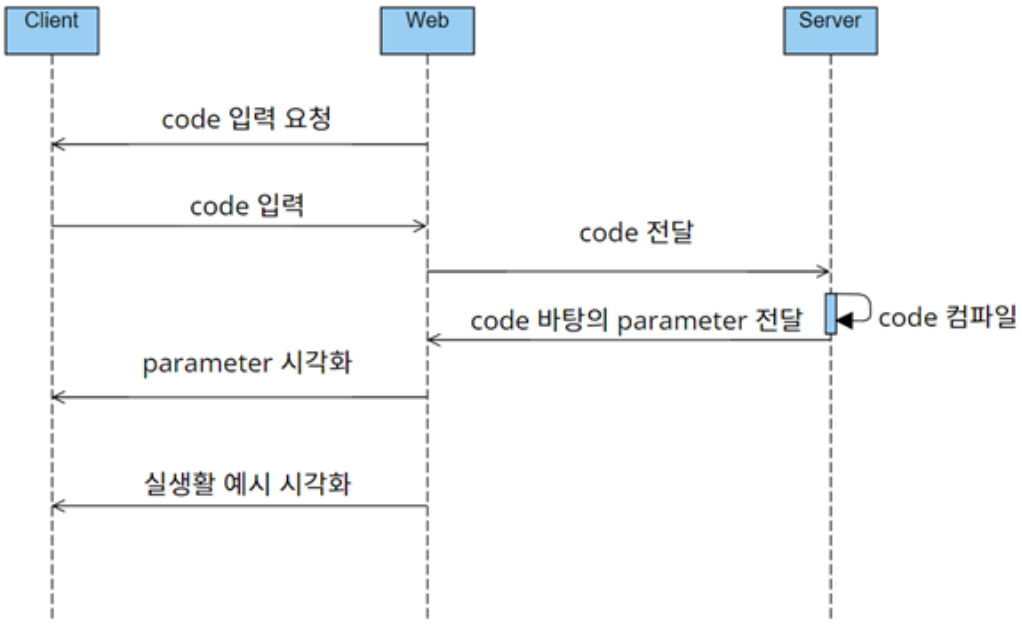

Figure 6. Frontend Sequence Diagram

5. System Architecture – Backend

5.1. Objectives

이 챕터는 이 섹션은 Backend 시스템의 구조를 상세히 이해하고, 각 Sub-System 이 객체 간의 관계를 나타내는 Class Diagram 과 데이터 흐름을 시각화 하는 Sequence Diagram 을 활용하여 이를 설명한다. 이를 통해 Backend 시스템의 주요 구성 요소를 식별하고 설명하여 시스템의 전체적인 이해도를 높일 수 있다. 구체적으로, 각 Sub-System 이 수행하는 주요 기능과 이를 지원하기 위한 객체 간의 상호 작용을 Class Diagram 을 통해 시각적으로 보여주며, Sequence Diagram 을 이용하여 데이터가 시스템 내에서 어떻게 흐르는지를 보여준다.

5.2. Overall Architecture

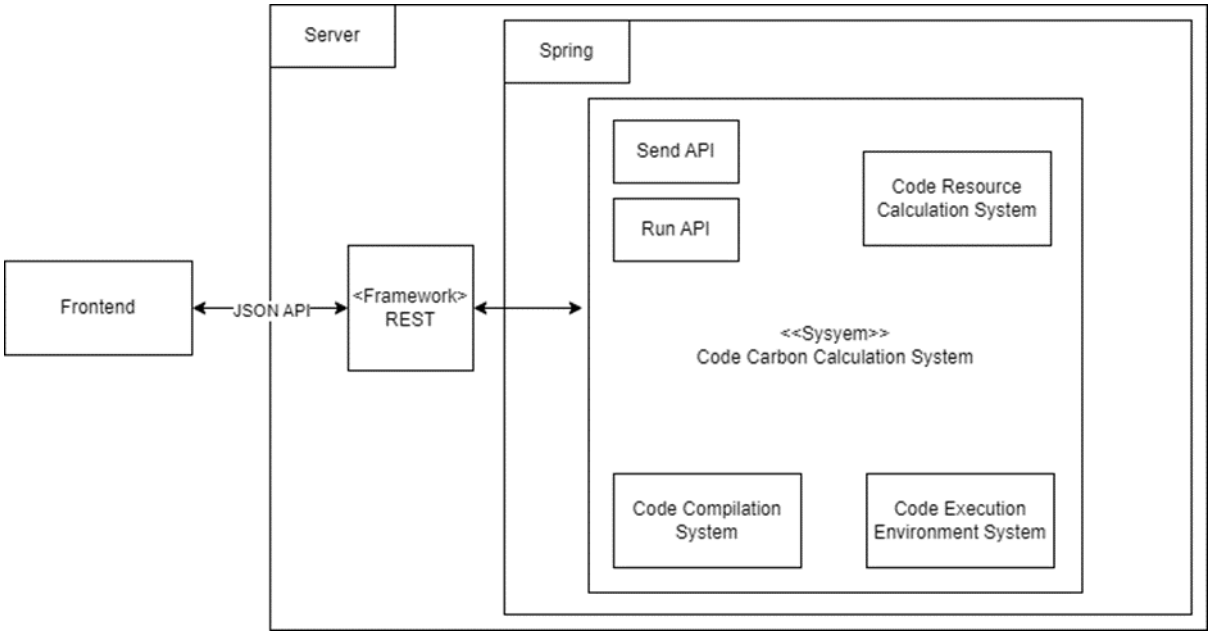

Figure 7. Backend Overall Architecture

사용자로 하여금 프로그래밍 코드를 입력하고, 해당 코드가 발생시키는 탄소 배출량을 계산해서 반환하는 본 시스템의 전반적인 구조는 위와 같다.

Front-end 로부터 요청이 들어오면, 입력 코드를 JSON API 형태로 받아 Spring Server 에서 탄소 배출량에 대한 계산을 진행하게 된다.

탄소 배출량 시스템은 크게 입력된 코드를 Compile 해주는 Code Compilation System, 코드의 실행 환경을 반환해주는 Code Execution Environment System, Runtime, Memory usage 등 코드의 자원을 계산하는 Code Resource Calculation System 으로 구성된다.

해당 시스템은 사용자 정보, 또는 입력 코드를 저장하는 작업이 불필요하므로, 데이터베이스를 사용하지 않는다.

Code Compilation System: 사용자 입력 코드를 Compile

Code Execution Environment System: 코드 실행 (서버) 환경 반환

Code Resource Calculation System: 코드가 사용하는 자원 반환

5.3. Subcomponents

5.3.1 Code Compilation System

Code Compilation System 은 사용자가 입력한 코드를 파일 형태로 저장하고, 해당 파일을 Java Compiler API 를 이용하여 컴파일하는 작업을 수행한다.

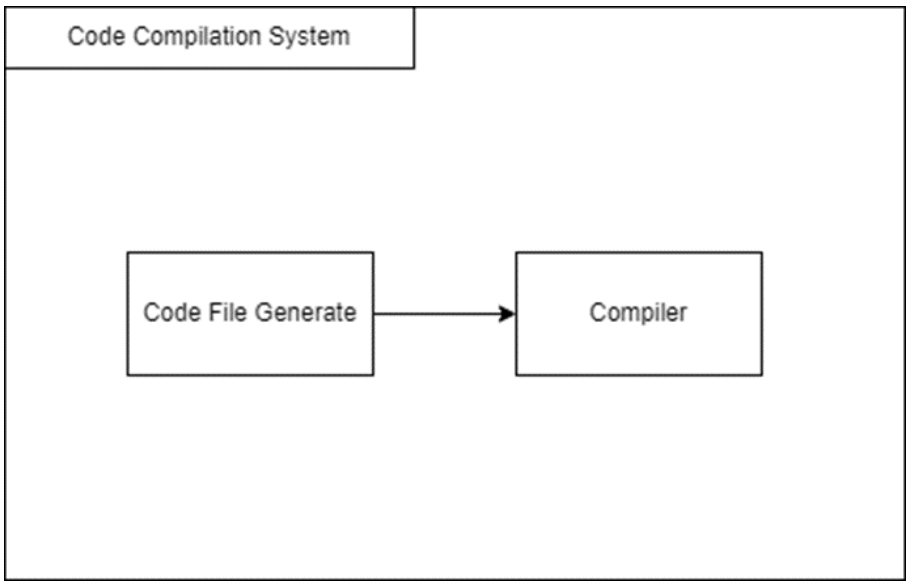

Figure 8. Backend Code Compilation Class Diagram

1) Code File Generate

사용자가 입력한 코드를 파일 형태로 저장한다. 사용자의 입력 코드는 기본적으로 JSON 형태로 전송된 후 문자열 형태로 추출되는데, 이 문자열을 파일 입력으로 넣게 된다. 사용자가 입력한 문자열 코드를 자바에서 제공하는 PrintWriter 를 이용해 Usercode.java 파일의 입력으로 넣게 되고, 결과적으로 컴파일 가능한 java 파일이 생성된다.

2) Code Compile

JavaCompiler 을 통해 자바 코드 파일을 컴파일한다. 이전에 생성된 Usercode.java 파일이 JavaCompiler 의 입력으로 들어간 후, 컴파일이 진행된다. 컴파일이 실패했을 경우, 코드의 몇 번째 line 에서 에러가 발생했는지 출력하고, 사용자에게는 컴파일이 실패했다는 메시지를 반환해준다. 컴파일이 성공했을 경우, 다음 단계로 진행되게 된다.

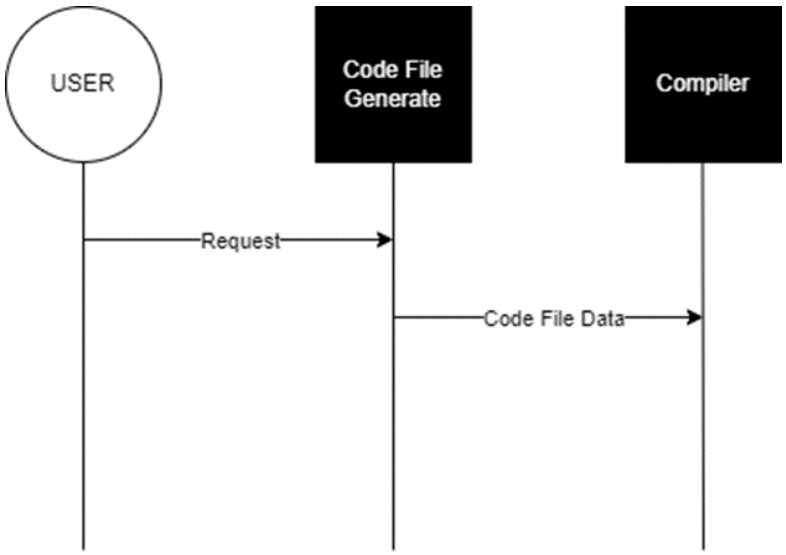

Figure 9. Backend Code Compilation Sequence Diagram

5.3.2 Code Execution Environment System

Code Execution Environment System 은, 입력된 코드가 어떤 환경에서 실행되었는지 사용자에게 정보를 전달하기 위한 System 이다. 이를 위해 코드를 실행하는 서버의 정보를 수집한다.

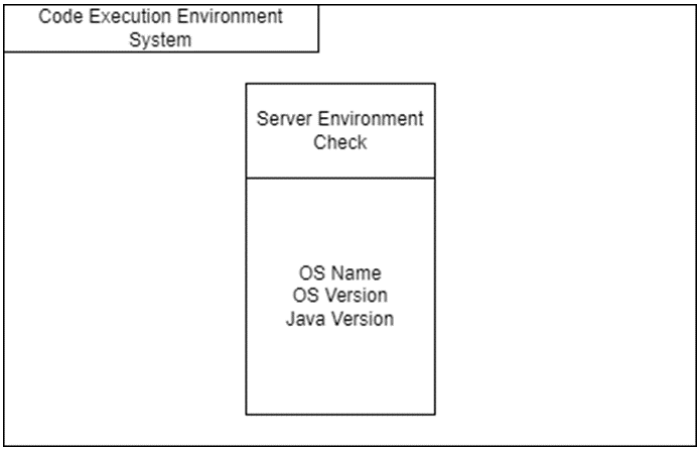

Figure 10. Backend Code Execution Environment Class Diagram

1) Server Environment Check

서버의 환경을 확인한다. 사용자 입력 코드의 실행과 더불어, System.getProperty() 함수를 통해 OS 의 이름과 버전, 그리고 Java 버전을 구한다. 이러한 정보는 코드의 탄소배출량이 사용자에게 반환될 때, 추가 정보로서 함께 전달되게 된다.

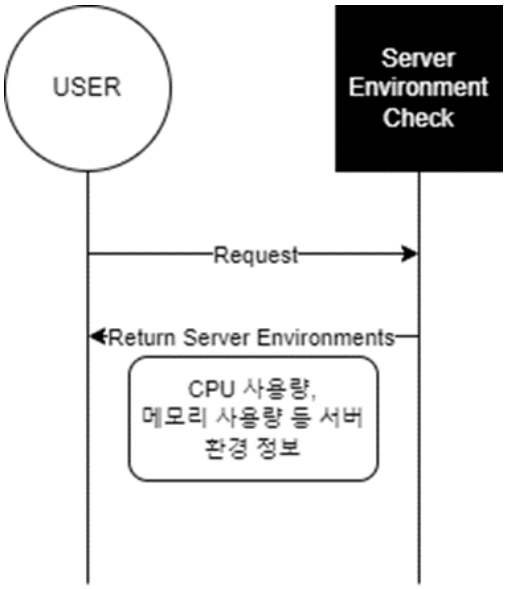

Figure 11. Backend Code Execution Environment Sequence Diagram

5.3.3 Code Resource Calculation System

Code Resource Calculation System 은 코드의 탄소배출량을 계산하는 주된 subsystem 이다. 해당 작업을 통해 탄소 배출량의 주된 요소인 코드의 Runtime, Memory Usage 등을 계산하게 된다.

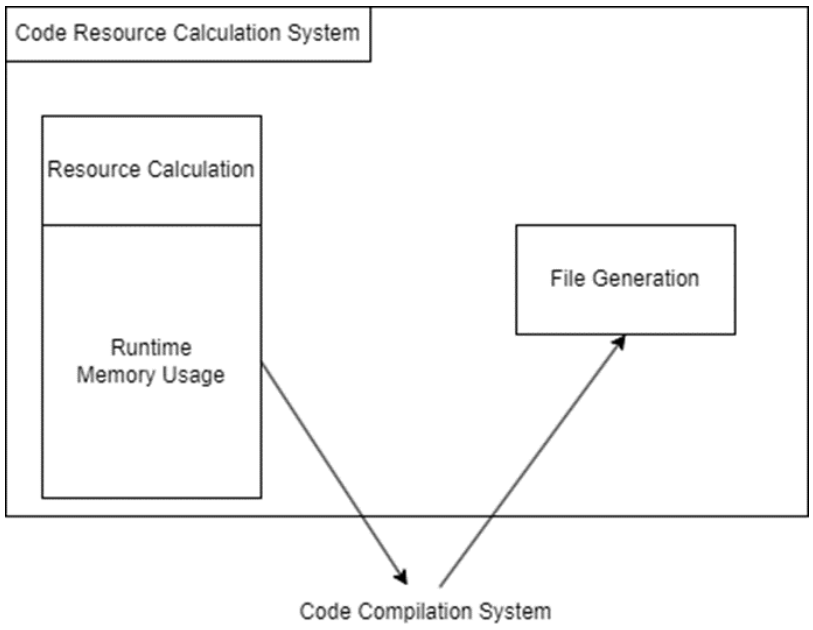

Figure 12. Backend Code Resource Calculation Class Diagram

1) Resource Calculation

기존에 사용자가 입력한 코드는 새로운 Process 에서 컴파일이 진행된다. 따라서 해당 Process 에서 코드의 실행 시간과 메모리 사용량을 측정해줘야 한다.

이러한 이유로, Code Compilation System 이 실행되기 전, 사용자 입력 코드의 main 함수의 바로 뒤에 필요한 자원 변수들의 계산을 시작하는 코드를 주입한다. 또한, main 함수가 끝나는 부분에 해당 변수들의 계산을 중지하고, 값을 파일에 저장하는 코드를 주입한다.

따라서 코드 탄소 배출량 계산의 전체 작업 흐름은 다음과 같다. 먼저 사용자가 코드 입력을 진행하면, 해당 코드에 자원 계산 부분을 더하여 java 파일로 저장한다. 이렇게 저장된 java 파일은 Code Compilation System 을 통해 새로운 Process 에서 실행되게 된다.

2) File Generation

새로운 Process 에서 자원 계산이 진행되기 때문에, 해당 계산 결과를 저장하는 파일의 생성이 필요하다. 이에 따라 File Generation Class 는 새로운 Process 내에서 생성되어, 해당 Process 가 종료 후 저장된 자원 사용량의 값을 확인할 수 있게 해준다.
결과적으로, 해당 파일의 값들을 읽어, 저장된 변수들을 탄소 배출량을 계산 식에 넣은 후 탄소 배출량 결과를 도출하게 된다.

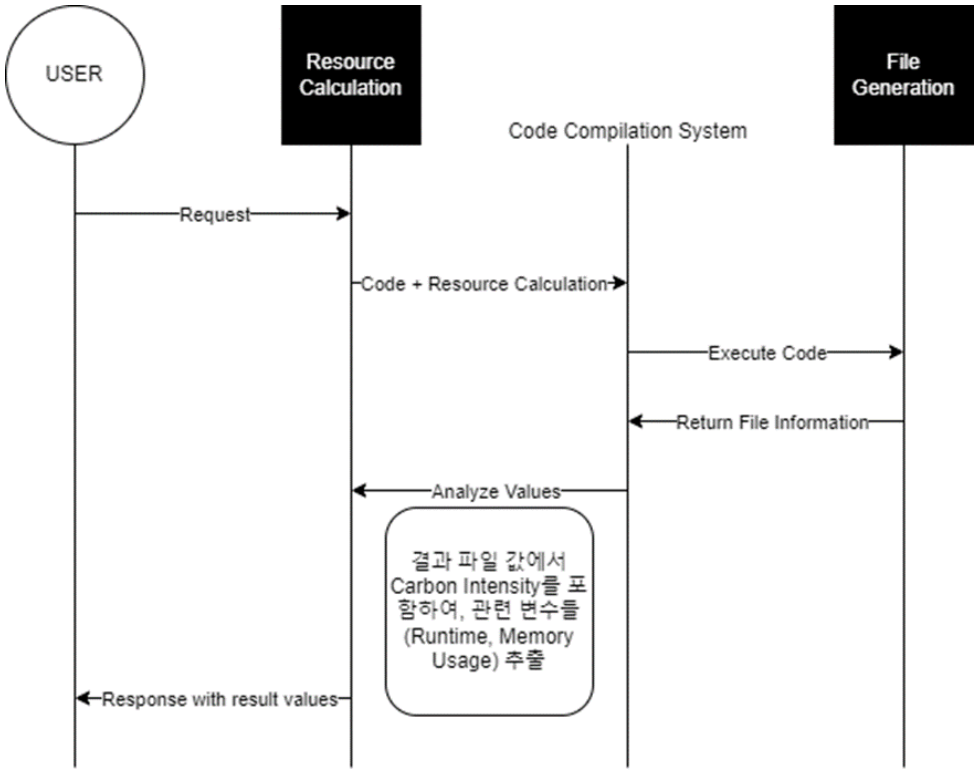

Figure 13. Backend Code Resource Calculation Sequence Diagram

6. Protocol Design

6.1 Objectives

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

6.2 Fetch API

Fetch API 는 웹 브라우저에서 제공되는 JavaScript API 로, 비동기적으로 서버와 데이터를 주고받는 데 사용된다. 이 API 를 활용하면 XMLHttpRequest 보다 더 간결하게 데이터를 다룰 수 있습니다. 주로 JSON 형식의 데이터를 주고받으며, 다양한 HTTP 메서드를 활용하여 서버와 효율적으로 통신할 수 있다.

6.3 HTTP

HTTP(HyperText Transfer Protocol)는 클라이언트와 서버 간에 데이터를 전송하는 데 사용되는 통신 프로토콜이다. 보통 웹 브라우저와 웹 서버 간의 통신에 쓰이며, HTTPS 를 통해 보안된 통신을 지원한다. HTTP 는 클라이언트가 서버에 요청을 보내고, 서버가 이에 응답하는 방식으로 동작한다.

6.4 Authentication

6.4.1 Java Code input

Request

Attribute	Detail	
Protocol	HTTP	
Method	POST	
Request Body	Code	User input of java code

Response

Attribute	Detail	
Protocol	HTTP	
Success Code	200 OK	
Failure Code	HTTP error code = 400	
Success Response Body	Message	Success message
Failure Response Body	Message	Failure message

6.4.2 Analysis-Data Receive

Request

Attribute	Detail
Protocol	HTTP
Method	GET

Response

Attribute	Detail	
Protocol	HTTP	
Success Code	200 OK	
Failure Code	HTTP error code = 400	
Success Response Body	Message	{ "data": "data-of-analysis" }
Failure Response Body	Message	Failure message

7. Testing plan

7.1. Test objectives

우리의 목적은 사용자로부터 코드를 받고 그 코드를 컴파일 할 때 걸린 시간 및 전력량과 그린 알고리즘을 통해 감소한 값을 받고 그로인한 감소 값을 웹페이지에 띄우는 것 이므로 그린 코드 알고리즘이 잘 작동하는지, 사용자의 코드를 오류 없이 받을 수 있는지, 사용자가 코드의 보안에 위배되는 악성 코드를 넣거나 코드의 런타임이 무한정 늘어나 사이트를 멈추게끔 유도할 때 그것을 잘 감지할 수 있는지, 사용자가 입력한 코드의 전력 사용량 및 탄소 배출량과 그린알고리즘을 통해 감소한 값을 웹화면에 잘 띄울 수 있는 지를 확인해야 한다.

7.2. Test Assumptions

•개인의 개발 능력을 고려하여 개발능력이 낮은 순으로 오류를 처리하다가 해결하지 못할 경우 다음사람이 이어서 오류를 해결한다.

•모든 결함과 그에 대한 해결은 문서화하여 작성한다.

•실제 코드 실행은 다양한 유저의 환경에서 실행되므로 이를 인지하여 특정 환경에 따라 코드의 길이 및 시간의 제한을 둔다.

•각 오류는 등급을 나누어 각 level 에 따른 priority 를 부여하여 높은 priority 순으로 오류 수정을 실행한다.

•최종 테스트는 실제 유저 n 명을 뽑아 사이트의 동작을 확인하여 기능적 오류나 불편한점 등을 받는다.

Level	영향
Critical	· 시스템을 손상시키거나 서버를 다운시키는 등 웹 사이트 실행자체에 영향을 주는 경우 · 웹 보안에 치명적인 영향을 주는 경우
High	· 서버의 속도를 더디게 만들거나 사이트가 원하는 기능을 제대로 하지 못하는 경우
Medium	· 유저가 사용하는 데에 불편함은 있으나 기능은 문제없이 작동하는 경우
Low	· 불분명한 오류 메시지가 있으나 사이트의 실행은 문제없이 되는 경우

7.3 Role Expectations

Backend	Level
그린 알고리즘을 통해 줄어든 값과 기존의 값을 Json 형식에 따라 문제없이 전달할 수 있는지 확인	High
그린 알고리즘의 이상점이 있는지 여러 testcase 를 통해 지속적인 확인	High
코드 실행 시 버전 오류 및 라이브러리 오류가 발생하는지 확인	High
코드의 실행 중 runtime 오류 및 process 의 수가 제한을 넘었는지 다양한 사용자가 접속 시 서버가 이상 없이 돌아가는지 확인	Critical

Frontend	Level
사용자의 코드를 Backend 로 Fetch API 를 통해 Json 형식에 따라 문제없이 전달할 수 있는지 확인	High
사용자의 코드를 암호화하여 보안적 문제를 해결 및 보안 취약점 확인	Critical

Backend 로부터 받은 값 및 그로인한 결과값들이 웹사이트 화면에 이상 없이 띄어지고 그래프 및 수치변환시 이상이 없는지 확인	Medium

추후에 오류 발생시 Level, Frontend, Backend 구분 후 추가할 수 있음

7.4 Test Algorithm

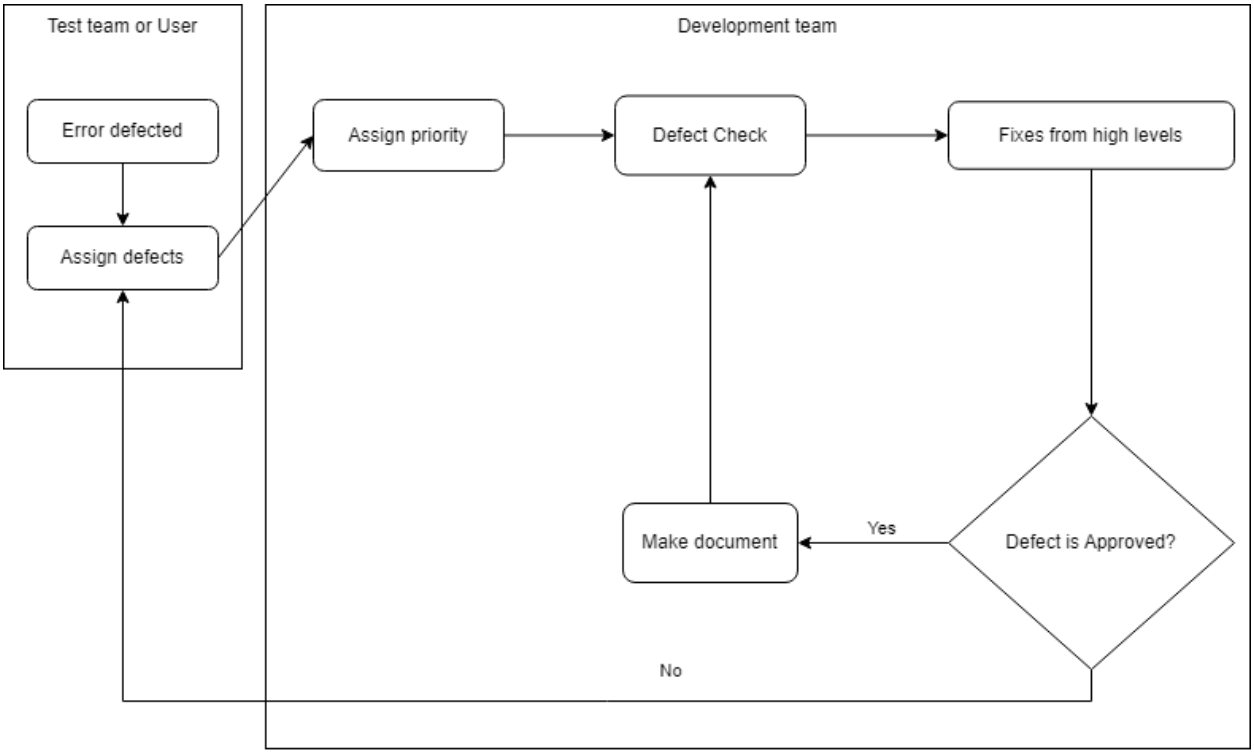

Figure 14. Test Algorithm

7.5 Test Environment

•HTML 5.0

•Java version “20.0.2”

•Spring-Boot “3.1.5”

8. Development Plan

8.1 Frontend Plan

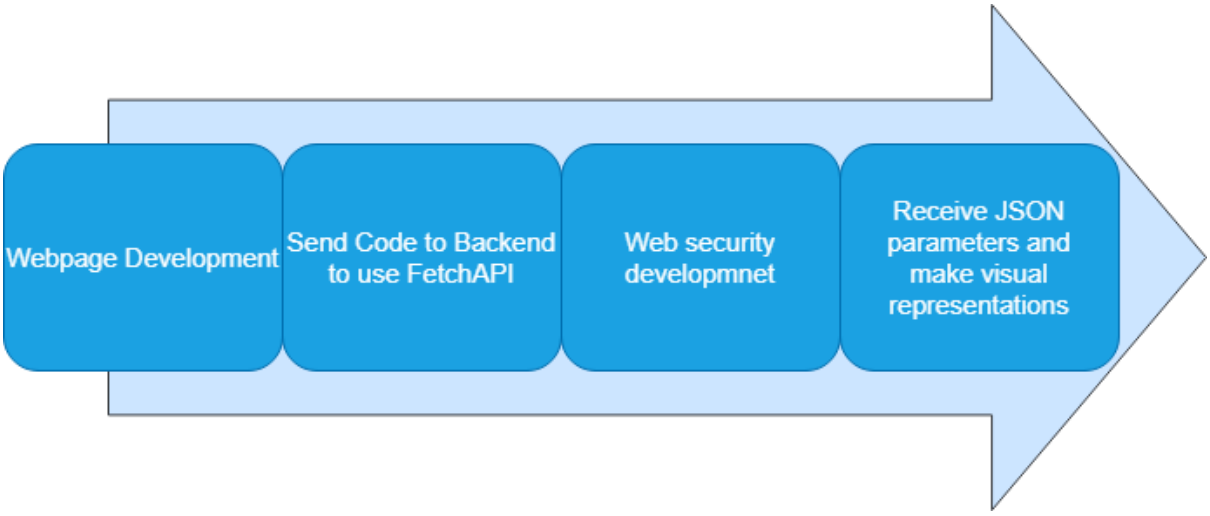

Figure 15. Frontend Plan

Frontend 의 전체적인 개발 흐름은 Figure 15 처럼 진행되고 Fetch API 를 통해 코드를 JSON 값으로 변환하여 보낼 때 Post_API function 을 활용하여 데이터를 보내는 과정에서 error 가 발생할 경우 Code 의 실행을 중단하고 PostAPI error 를 콘솔창에 호출하여 문제 발생 지점을 명확하게 보여주고 JSON parameter 를 Backend 로부터 받을 때 Receive_API function 을 활용하여 값이 정상적이면 0 값을 받을 때 오류가 생겼다면 -1 값이 비정상적이면 -2 을 return 하여 Backend 에 오류 상태를 보낸다.

8.2 Backend Plan

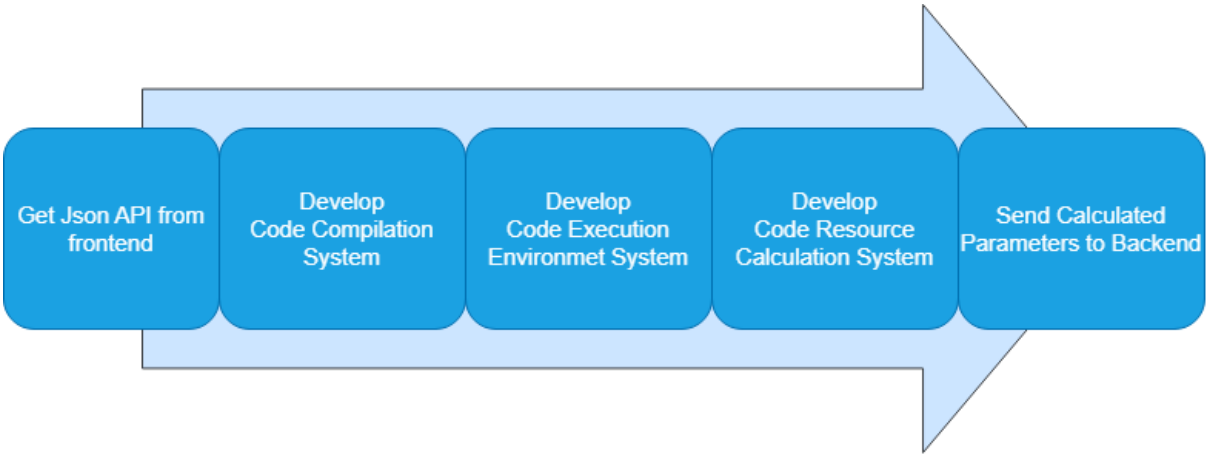

Figure 16. Backend Plan

Backend 의 전체적인 개발 흐름은 Figure 16 과 같이 진행되며 spring server 을 활용하여 진행되고 Java.io, java.util 라이브러리를 사용하여 Code Compilation System 에서 Frontend 로부터 받은 JSON 값에서 코드를 String 으로 바꾼 뒤 Code File 로 저장할 수 있게 하고 javax.tools 라이브러리를 사용하여 생성한 Code File 을 Compiler 에 넣은 후 새로운 Process 에서 Compile 한다.

9. Supporting Information

9.1 Software Requirements Specification

본 요구사항 명세서는 소프트웨어 요구사항 명세서 IEEE 권장사항(IEEE Std 830-1998 - IEEE Recommended Practice for Software Requirements Specifications)에 따라 작성되었다.

9.2 Document History

Document History			
Date	Version	Description	Writer
11/19	V1.00	System Architecture – Backend Integration	전윤혁
11/19	V1.00	System Architecture - Frontend	남윤성
11/19	V1.00	Protocol Design	김동현
11/19	V1.00	System Architecture - Overall	권서영
11/19	V1.00	Purpose Introduction Supporting Information	송재현
11/19	V1.00	Testing Plan Development Plan	조영길