SOFTWARE ENGINEERING SWE3002_41

Software Design Specification

of

ECO2

2023.11.19

Team 7

차승일 김민수 임소현 정단호 이시혁 장영우 유규환

Contents

1. Preface 7
1.1 Readership 7
1.2 Scope / Objective 7
1.3 Document Structure 7
2. Introduction 8
2.1 Objectives 8
2.2 Applied Diagrams 8
2.2.1 Used Tools 8
2.2.2 Diagram Types 8
2.2.2.1 Use case Diagram 8
2.2.2.2 Context Diagram 8
2.2.2.3 Sequence Diagram 9
2.2.2.4 Class Diagram 9
2.2.3 Project Scope 9
2.2.4 References 10
3. System Architecture - Overall 10
3.1 Objectives 10
3.2 System Organization 10
3.2.1 Context Diagram 10
3.2.2 Sequence Diagram 11
3.2.3 Use case Diagram 11
4. System Architecture - Frontend 12
4.1 Objectives 12
4.2 Subcomponents 12
4.2.1 Code 12
4.2.1.1 Attribute 12
4.2.1.2 Methods 12
4.2.1.3 Context Diagram 13
4.2.1.4 Sequence Diagram 13

4.2.2 Statistics 14
4.2.2.1 Attribute 14
4.2.2.2 Methods 14
4.2.2.3 Context Diagram 15
4.2.2.4 Sequence Diagram 15
4.2.3 Server Setting 16
4.2.3.1 Attribute 16
4.2.3.2 Methods 16
4.2.3.3 Context Diagram 18
4.2.3.4 Sequence Diagram 19
4.2.4 Information 19
4.2.4.1 Attribute 19
4.2.4.2 Methods 20
4.2.4.3 Context Diagram 21
4.2.4.4 Sequence Diagram 21
5. System Architecture-Backend 21
5.1 Objectives 21
5.2 Overall architecture 22
5.3 Sub Systems 23
5.3.1 Class Diagram for Controller and Service 23
5.3.2 Sequence Diagram 24
6. Protocoal Design 24
6.1 Objectives 24
6.2 Protocol Details 25
6.2.1 HTTP 25
6.2.2 RESTful API 25
6.2.3 React Context API 25
6.3 API 26
6.3.1 탄소 배출량 조회 26
6.3.1.1 Request 26
6.3.1.2 Response 26
6.3.2 CPU 모델명 리스트 조회 27

6.3.2.1 Request 27
6.3.2.2 Response 27
6.3.3 GPU 모델명 리스트 조회 28
6.3.3.1 Request 28
6.3.3.2 Response 28
6.3.4 대륙 리스트 조회 28
6.3.4.1 Request 28
6.3.4.2 Response 29
6.3.5 국가 리스트 조회 29
6.3.5.1 Request 29
6.3.5.2 Response 29
6.3.6 지역 리스트 조회 30
6.3.6.1 Request 30
6.3.6.2 Response 30
7. Testing Plan 31
7.1 Objectives 31
7.2 Testing Policy 31
7.2.1 Development Testing 31
7.2.2 Release Testing 31
7.2.3 User Testing 32
7.3 Test Case 32
7.3.1 Code 32
7.3.2 Statistics 32
7.3.3 Server setting 33
8. Development Plan 33
8.1 Objectives 33
8.2 Environment and Tools 34
8.3 Constraints 36
8.4 Assumptions and Dependencies 37
9. Supporting Information 37
9.1 Software design specification 37
9.2 Document history 37

List of Figures

[Figure 1] Context Diagram
[Figure 2] Sequence Diagram
[Figure 3] Use case Diagram
[Figure 4] Code Context Diagram
[Figure 5] Code Sequence Diagram
[Figure 6] Statistics Context Diagram
[Figure 7] Statistics Sequence Diagram
[Figure 8] Server Context Diagram
[Figure 9] Server Sequence Diagram
[Figure 10] Info Context Diagram
[Figure 11] Info Sequence Diagram
[Figure 12] Backend Overall Architecture
[Figure 13] Backend Class Diagram
[Figure 14] Backend Sequence Diagram
[Figure 15] React Logo
[Figure 16] HTML Logo
[Figure 17] Tailwind Logo
[Figure 18] Spring boot Logo
[Figure 19] Docker Logo

List of Tables

[Table 1] 탄소배출량 조회 Request
[Table 2] 탄소배출량 조회 Response
[Table 3] CPU 모델명 리스트 조회 Request
[Table 4] CPU 모델명 리스트 조회 Response
[Table 5] GPU 모델명 리스트 조회 Request
[Table 6] GPU 모델명 리스트 조회 Response
[Table 7] 대륙 리스트 조회 Request
[Table 8] 대륙 리스트 조회 Response
[Table 9] 국가 리스트 조회 Request
[Table 10] 국가 리스트 조회 Response
[Table 11] 지역 리스트 조회 Request
[Table 12] 지역 리스트 조회 Response
[Table 13] Code test case
[Table 14] Statistics test case
[Table 15] Server test case

1. Preface

1.1 Readership

본 문서는 시스템 제작에 관여하는 독자의 이해를 돕기 위해 작성되었다. 본 문서의 주요 독자는 7명으로 이루어진 Team 7과 소프트웨어공학개론 강의 조교님, 이은석 교수님이며 2023 2학기 소프트웨어공학개론 수강자가 있다. 본 문서를 상업적 용도로 활용할 시 Team 7의 허가를 얻어야 하며 학습 및 교육 용도, 정보 취득 목적으로 사용할 시 재배포 및 수정에 제약은 없다.

1.2 Scope / objective

본 문서는 Java 코드의 탄소배출량 및 유의미한 지표들을 직관적으로 확인하기 원하는 ECO2 서비스의 개요와 design pattern들을 소개하기 위한 문서이다. 이 서비스는 성균관대학교 이은석 교수님의 2023년 2학기 소프트웨어공학개론 수업을 수강한 Team 7 member들에 의해 제작되었다. 현재 전 세계적으로 탄소배출량의 증가 추이가 exponential하게 증가하는 추세이며, 앞으로 탄소배출량은 더욱 늘어날 것으로 예상되는 바이다. 특히 최근 AI의 수요와 발전이 가속화됨에 따라 소프트웨어 개발자의 코드로 인한 탄소배출량의 비율이 전체적으로 늘어나는 중이므로, 개발자 스스로 본인 코드의 탄소배출량을 직관적으로 확인하여 친환경적인 코드 개발에 힘쓸 필요가 있겠다. 이에 본 개발진은 사용자가 PC 스펙을 입력한 후 Java 코드의 탄소배출량을 Code 부분에 삽입한 후 submit할 시 객관적인 지표를 확인할 수 있도록 하였고, 개발자 스스로 Java 코드를 수정하며 점차 탄소배출량을 줄여가도록 하는 것을 지향하였다.

추가로 본 문서, 디자인명세서는 개발하기 위해 클라이언트와 서버, API 등을 디자인하는 과정에서 도출한 요구 사항 명세서에 근거하여 작성되었다.

1.3 Document Structure

1. Preface : 본 문서의 Readership, Scope/Objective에 관해 간결히 소개한다.
2. Introduction : 본 문서를 작성하는데 사용된 다양한 다이어그램, 도구, Reference 들의 전반을 소개한다.
3. Overall System Architecture : 시스템의 전체적 구조를 Context diagram, Use-case diagram, Sequence diagram을 통해 설명한다.
4. System Architecture (Frontend) : Frontend 부문에 대한 시스템 구조를 기능 별로 나누고, 각 부분을 다양한 Diagram을 통해 설명한다.
5. System Architecture (Backend) : Backend 부문에 대한 시스템 구조를 기능 별로 나누고, 각 부분을 다양한 Diagram을 통해 설명한다.
6. Protocol Design : 클라이언트와 서버 간의 통신을 위한 API 등의 프로토콜 디

자인을 설명한다.

7. Testing Plan : 본 시스템의 이후 진행할 테스팅 계획에 대해 설명한다.

8. Development Plan : 본 시스템을 개할 하는 데 있어 사용된 개발 환경 및 도구에 관해 설명한다.

9. Supporting Information : 본 문서의 수정 및 변경을 날짜 별로 기록한다.

2. Introduction

2.1 Objectives

본 챕터에서는 서비스를 설계하는 데 사용된 다이어그램, 해당 다이어그램 작성에 사용된 도구, 그리고 개발 범위에 대해 설명한다.

2.2 Applied Diagrams

2.2.1 Used Tools

Microsoft사의 Powerpoint 프로그램을 통해 기본적으로 제공하는 도형과 아이콘을 사용하여 본 문서에서 사용된 다이어그램을 제작하였다.

2.2.2 Diagram Types

2.2.2.1 Use case diagram

Use case diagram은 시스템과 사용자의 상호작용을 다이어그램으로 표현한 것이다.

시스템에서 제공해야 하는 기능이나 서비스를 명세화한 것이며, 보통 System, Actor, Usecase, Relation으로 구성되어 있다.

System은 만들고자 하는 서비스를 나타낸다. Actor는 시스템과 상호작용하는 외부의 존재로, 시스템을 사용하는 사람일 수도 있으며, 사람이 시스템일 수도 있다.

Usecase는 시스템이 Actor에게 제공해야 하는 사용자 입장에서 바라본 서비스 또는 기능이다.

마지막으로 Relation은 Actor와 Usecase 사이의 관계, 또는 두 개의 Usecase 사이에 나타나는 관계로 4가지 종류(Association, Include, Generalization, extend)로 구성된다.

2.2.2.2 Context diagram

Context diagram은 높은 수준의 data flow 다이어그램이다. 시스템, 그리고 그 시스템과 상호작용하는 외부 구성 요소들을 나타내며, 그들 사이의 세부적인 정보의 흐름을 나타낸다. 많은 비즈니스에서 프로젝트의 위험한 상황

을 미리 대비하기 위해 사용하고 있으며, 앞서 말한 것처럼 세부적인 정보의 흐름을 파악할 수 있으므로, 시스템을 보다 세부적으로 이해하기 위해 사용된다. 개발자보다는 프로젝트의 이해 관계자가 주로 사용하므로 이해관계자가 관련 내용을 쉽게 이해할 수 있도록 작성해야 한다.

2.2.2.3 Sequence diagram

Sequence Diagram은 어떠한 순서로 어떤 객체들과 어떻게 상호작용하는지를 표현하는 다이어그램이다. 문제 해결에 필요한 객체를 정의하고, 객체 사이에 송/수신되는 메시지들을 시간의 흐름에 따라 표시해준다.

구성요소로는 생명선(Lifeline), 활성 박스(Activation Box), 메시지(Message), 객체(object)가 있다. 생명선(Lifeline)은 객체가 생성, 소멸, 또는 활성화될 때를 나타내는 선이다. 활성 박스(Activation Box)는 객체가 다른 객체와 상호작용하며 활성화 상태인 것을 나타내는 박스다. 메시지(Message)는 객체 간에 주고받는 데이터로, 일반적으로 요청(request)과 응답(response)으로 구성된다. 객체(object)는 시나리오의 기능을 수행하는데 필요한 클래스의 객체이다.

2.2.2.4 Class diagram

class diagram은 시스템을 구성하는 클래스, 클래스의 속성 및 기능, 동작 방식, 그리고 객체 간의 관계를 표현하여 시스템의 구조를 보여주는 정적 구조 다이어그램이다.

전체 시스템의 구조와 클래스의 의존성을 파악하는데 사용되며, 클래스 간의 정적인 관계를 정의함으로써 시스템을 이해하는데 용이하다.

일관된 형식으로 시스템을 분석, 설계하는 방식을 제공하며, 객체지향에 가장 가까운 방식이다.

2.2.3 Project Scope

본 문서에서 설계하는 탄소 배출량 계산 서비스는 사용자들이 본인의 자바 소스 코드의 탄소 배출량을 확인하고, 결과를 통해 탄소 배출량을 줄이는 방향으로 코드 최적화를 진행하는 것을 돕기 위해 만들어졌다.

개발자를 비롯해 본 서비스를 사용하는 사용자들이 본인의 소스 코드의 탄소 배출 수준을 확인하고, 프로젝트 개발 시 지속적으로 코드 최적화를 진행하며 탄소 배출 수준을 고려할 수 있도록 하고자 한다.

2.2.4 References

- https://github.com/skkuse/2022spring41classteam1
- https://github.com/skkuse/2023spring41classteam9

3. System Architecture – Overall

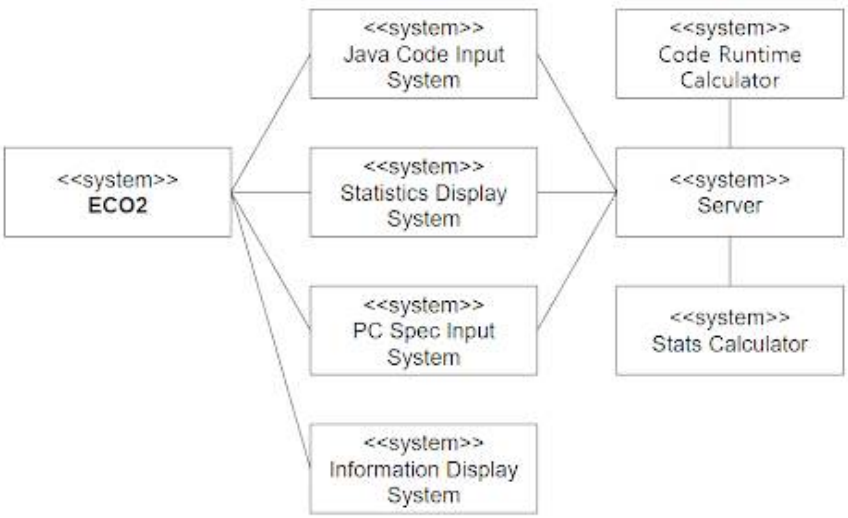
3.1 Objectives

본 System Architecture 부분에서는 frontend에서 backend에 이르는 프로젝트 전반의 system architecture에 대해 기술한다.

3.2 System Organization

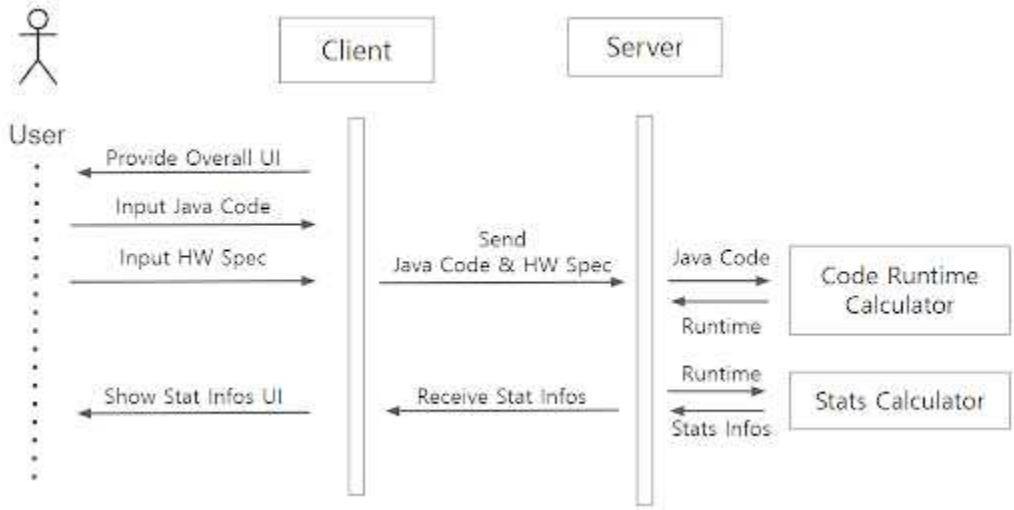
이 서비스는 client - 서버 모델을 적용하여 설계되었으며, frontend 어플리케이션은 사용자와의 모든 상호작용을 담당한다. 프론트 엔드 어플리케이션과 백 엔드 어플리케이션은 JSON 기반의 HTTP 통신을 통해 데이터를 주고받는다. 백엔드 어플리케이션은 자바코드 및 하드웨어 스펙 요청을 프론트 엔드로부터
백엔드 어플리케이션은 설계 사양 요청을 프론트 엔드로부터 컨트 롤러로 배포하고, 처리한 후 JSON 형식으로 전달한다. 백 엔드 어플리케이션은 사용자가 작성한 머신러닝 코드를 전달받고 이를 실행하여 얻은 결과를 다시 내보낸다. 결과를 내보냄과 동시에 데이터베이스에 문제 및 코드의 정오, accuracy를 저장하고 사용자가 결과에 관한 정보를 요청하면 업데이트된 정보가 전달된다.

3.2.1 Context Diagram



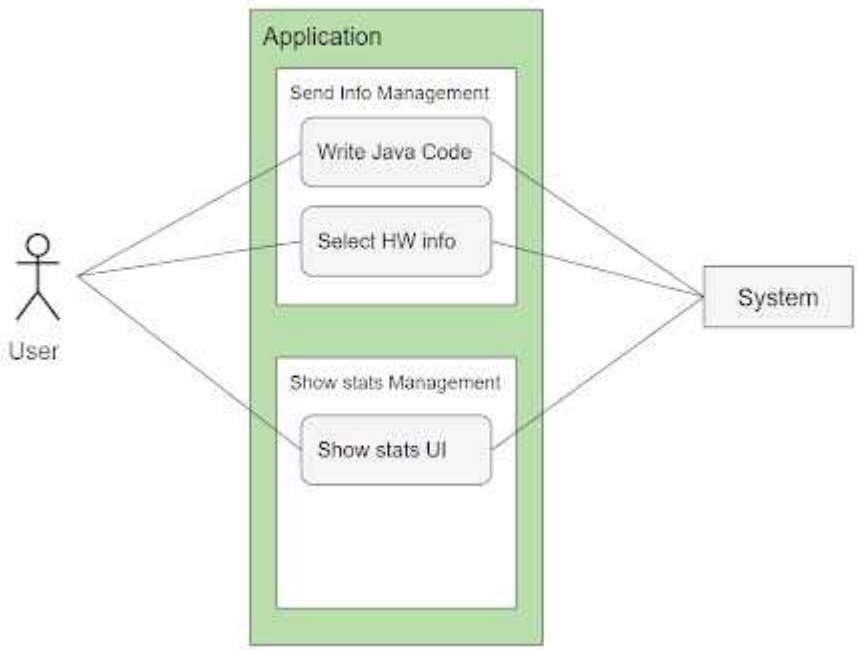
[Figure 1] Context Diagram

3.2.2 Sequence Diagram



[Figure 2] Sequence Diagram

3.2.3 Use case Diagram



[Figure 3] Use case Diagram

4. System Architecture – Frontend

4.1 Objectives

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

4.2 Subcomponents

4.2.1 Code

이 페이지는 주요 페이지에서 사용자가 Java 코드를 입력하는 영역을 담당한다. 사용자는 여기에서 Java 언어로 작성된 코드를 자유롭게 입력할 수 있으며, 작성이 완료되면 제출 버튼을 통해 Json 형태의 데이터를 전송하여 백엔드와의 상호 작용이 가능하다.

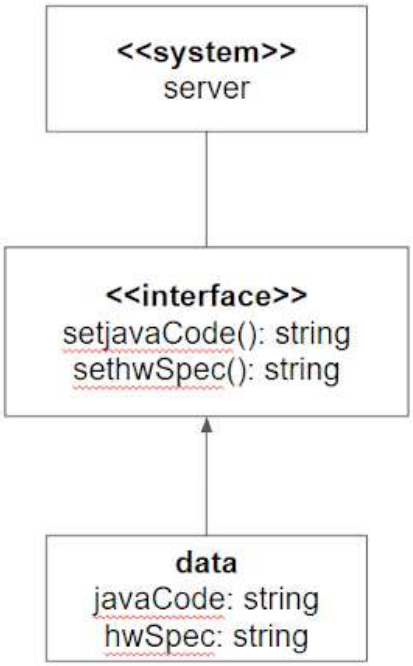
4.2.1.1 Attribute

- Code: 자바 코드를 입력하는 부분에 해당된다.
- submit 버튼: 코드 및 하드웨어 스펙을 Json 형태의 데이터로 저장하여 백엔드와 상호 작용하는 역할을 한다.
- clear 버튼: 자바 코드를 초기화하는 역할을 한다.

4.2.1.2 Methods

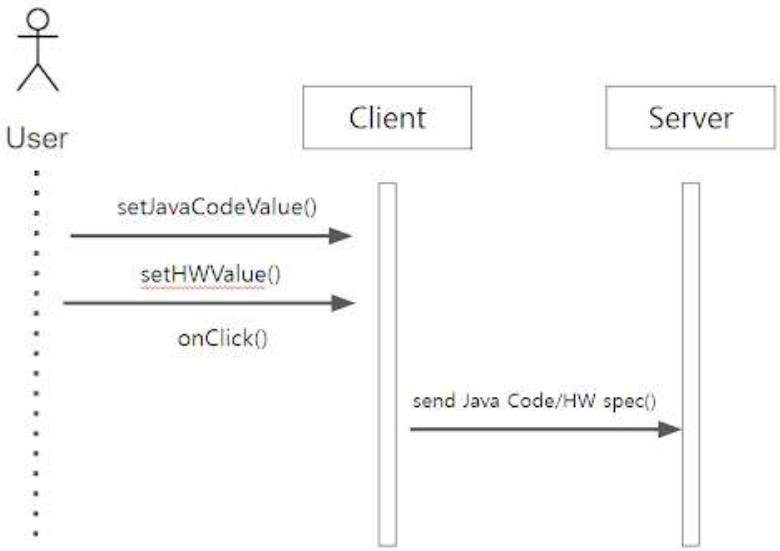
- setJavaCodeValue(): string으로 구성된 자바 코드를 json 형태로 파싱한다.
- setHWValue(): string으로 구성된 하드웨어 스펙을 json 형태로 파싱한다.
- submitButton(): submit 버튼을 클릭했을 시 작동하는 함수를 의미한다.
- clearButton(): clear 버튼을 클릭했을 시 작동하는 함수를 의미한다.

4.2.1.3 Context Diagram



[Figure 4] Code Context Diagram

4.2.1.4 Sequence Diagram



[Figure 5] Code Sequence Diagram

4.2.2 Statistics

본 Stat.js 파일은 server에서 계산된 Carbon footprint를 제공받아 이를 시각적인 지표로 바꾸는 자바스크립트 파일이다. 먼저 setStats() 함수를 통해 서버로부터 json형식을 반환 받고, parsing 과정을 거쳐 stat_components, hw_components, region_components 객체를 setting한다. 이 후 setting된 값에 의거하여 html 코드부문을 수정하여 사용자에게 보여준다.

4.2.2.1 Attribute

- stat_components : key, value 객체를 저장한다.
- carbon : 탄소배출량 [g/CO2e]
- car : 탄소배출량을 passanger car로 변환한 수치 [km]
- plane : 탄소배출량을 flight from Incheon to London으로 변환한 수치 [%]
- energy_needed : 필요한 에너지의 양 [kwh]
- tree : 현재 수준으로 탄소배출량이 한달 동안 발생했을 때, 소모되는 나무 개수 [tree-months]
- hw_components : 각 하드웨어로부터 기인된 탄소배출량의 비율을 저장한다. 각 부분에서 배출된 탄소가 없을 시 null을 저장한다.
- cpu_rate : CPU에서 배출된 탄소의 양
- gpu_rate : GPU에서 배출된 탄소의 양
- memory_rate : memory에서 배출된 탄소의 양
- region_components : 지역을 타지역으로 변경했을 때 예상되는 탄소배출량 수치이다.
- 현재 위치, 호주, 인도, 중국, 미국, 영국, 캐나다, 프랑스, 스위스, 스웨덴에서의 수치를 각 변수에 저장.

4.2.2.2 Methods

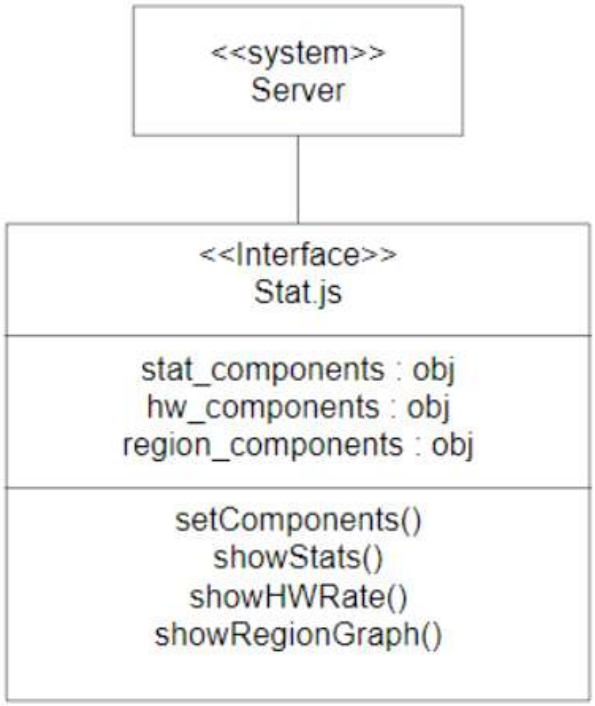
-setComponents() : Server로부터 API로 반환 받은 json 파일을 parsing하여 각 Attributes에 저장한다.

- showStats() : UI에 탄소배출량 외 stats_component 객체의 요소의 수치들을 표시한다.

- showHWRate() : UI에 hw_component 객체에 저장된 요소들의 값을 원형 그래프로 표시한다.

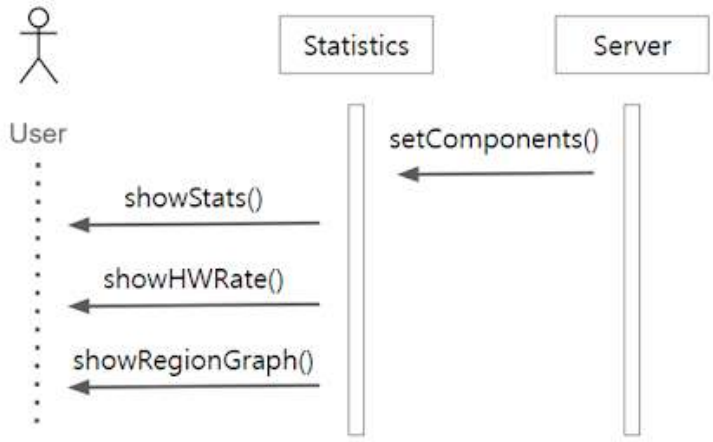
- showRegionGraph() : 각 나라별 탄소배출량의 추이를 막대그래프로 표시한다.

4.2.2.3 Context Diagram



[Figure 6] Statistics Context Diagram

4.2.2.4 Sequence Diagram



[Figure 7] Statistics Sequence Diagram

4.2.3 Server Setting

탄소 배출량을 계산하는 데에 필요한 모든 컴퓨터 하드웨어 관련 정보를 관리하는 클래스이다.

4.2.3.1 Attribute

- hw_dict : frontend에서 관리하는 전역변수로 관리되는 딕셔너리로, API에 전송할 하드웨어 데이터를 딕셔너리 형태로 관리한다.
- memory : 사용 가능한 메모리를 나타내는 값으로, 단위는 GB이다.
- location : 탄소 강도(Carbon intensity)를 계산하는 데에 필요한 값으로, 해당 국가의 전기가 얼마나 많은 탄소를 배출하는지 나타내는지를 의미한다.
- psf : 전력 스케일링 계수로, 시스템의 전력 사용량을 조정하기 위한 값이다. 기본 값은 1로, 사용자가 필요시 값을 조정할 수 있다.
- platform (pue) : 데이터 센터의 전력 사용 효율성을 나타내는 값으로, IT 장비에 공급된 에너지 대비 총 에너지 소비량의 비율이다. 기본 값 1을 사용한다.
- type of core : 코어 유형에 관한 값으로, cpu, gpu, both 중에 선택 가능하다.
- number of cores : 코어의 개수에 관한 값이다.
- model : 계산하는 데에 사용되는 model 종류로, 이를 통해 한 core 당 TDP (Thermal Design Power)을 계산한다.
- usage : 프로세서가 최대 성능으로 작동하는 비율이다. 기본 값은 1이며, 사용자는 필요시 0~1 사이의 소수값으로 값을 설정할 수 있다.

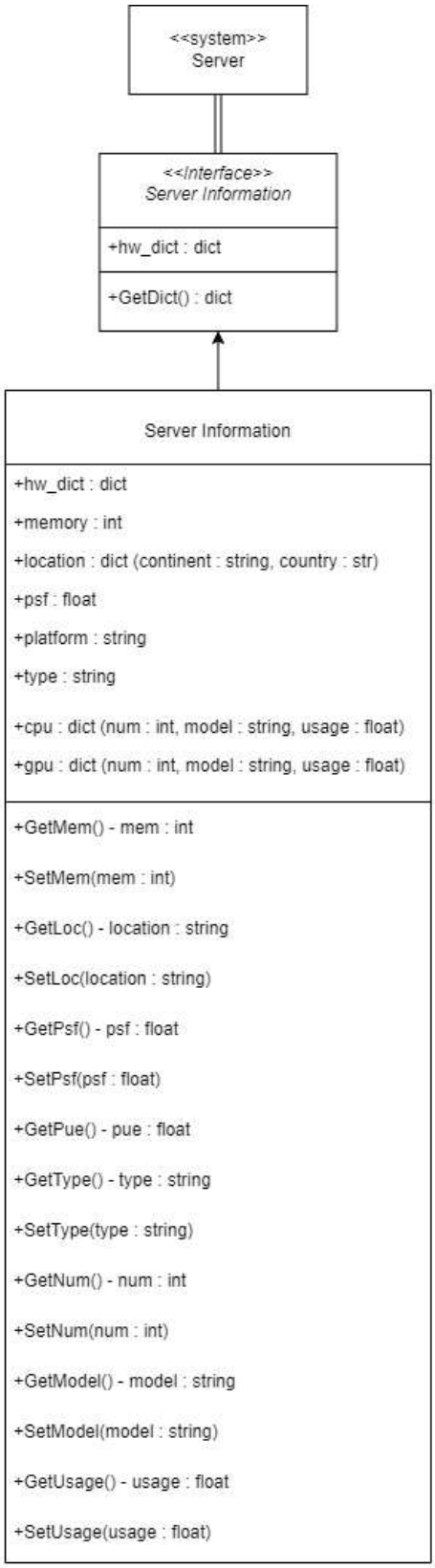
4.2.3.2 Methods

- GetDict() : server class에서 관리하는 hw_dict를 반환한다.
- GetMem() : hw_dict의 memory의 값을 받아온다.
- SetMem(new_mem) : hw_dict의 memory 값을 사용자가 입력한 새로운 memory 값으로 변경한다.
- GetLoc() : hw_dict의 location 값을 받아온다.
- SetLoc(new_loc) : hw_dict의 location 값을 사용자가 입력한 새로운 location 값으로 변경한다.
- GetPsf() : hw_dict의 psf 값을 받아온다.
- SetPsf(new_psf) : hw_dict의 psf 값을 사용자가 입력한 새로운 psf 값으로 변경한다.
- GetPue() : hw_dict의 pue 값을 받아온다. 사용자는 pue 값을 따로 설

정할 수 없다.

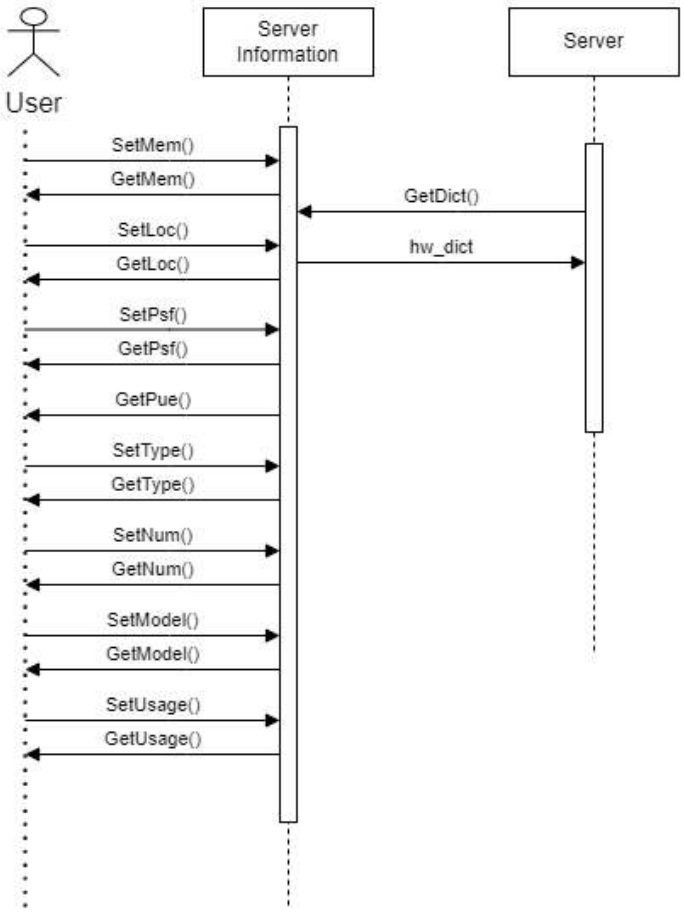
- GetType() : hw_dict의 core의 type을 받아온다.
- SetType(new_type) : hw_dict의 core type을 사용자가 입력한 새로운 core type으로 변경한다.
- GetNum() : hw_dict의 num of cores를 받아온다.
- SetNum(new_num) : hw_dict의 num of cores를 사용자가 입력한 새로운 num of cores로 변경한다.
- GetModel() : hw_dict의 model을 받아온다.
- SetModel(new_model) : hw_dict의 model을 사용자가 입력한 새로운 model로 변경한다.
- GetUsage() : hw_dict의 usage 값을 받아온다.
- SetUsage(new_usage) : hw_dict의 usage를 사용자가 입력한 새로운 usage로 변경한다.

4.2.3.3 Context Diagram



[Figure 8] Server Context Diagram

4.2.3.4 Sequence Diagram



[Figure 9] Server Sequence Diagram

4.2.4 Information

본 서비스의 의의, 사용법 등을 포함한 여러가지 정보가 담긴 페이지로, 사이드 바에서 Info 텍스트 버튼을 클릭하면 나오는 페이지다.
사용자는 해당 페이지를 통해 본 서비스에서 사용된 탄소 배출량 계산법, 서비스를 통해 제공받는 것, 서비스와 관련된 문서 및 코드가 저장된 링크 등을 확인할 수 있다.

4.2.4.1 Attribute

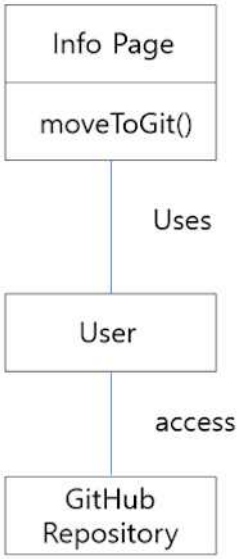
- Info page는 내용에 따라 각각의 박스로 분류되어, 여러 개의 박스로 하나의 페이지를 구성하고 있다.

- "Problem : CO2" box: 본 서비스의 동기가 되는 현실 문제인 탄소 배출 문제에 관련한 내용을 담고 있다.
- "What is Green Software?" box: 탄소 배출량을 줄이기 위해 소프트웨어 분야에서 어떤 방식의 개발을 해야 하는지, 어떤 중요한 요소가 있는지에 대한 내용을 담고 있다.
- "Services Provided" box: 본 서비스를 통해 사용자가 제공받는 것에 대한 내용을 담고 있다.
- "Carbon Emissions Formula" box: 본 서비스에 사용된 탄소 배출량 계산법에 대한 내용을 담고 있다.
- "Input Method" box: 본 서비스를 사용하는 방법(코드 입력 방식)에 대한 내용을 담고 있다.
- "Code / More Information" box: 본 서비스와 관련된 문서 및 코드가 저장된 github링크를 담고 있다.

4.2.4.2 Methods

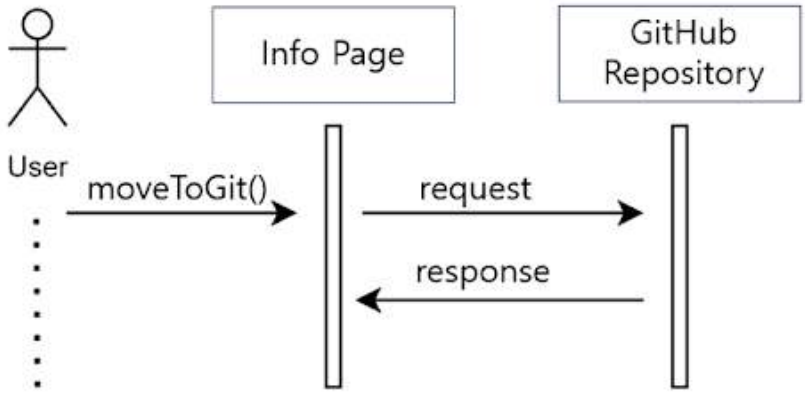
- "Code / More Information" box를 제외한 모든 박스들은 정적으로 각 주제에 해당하는 내용을 보여주고 있으므로 따로 Methods가 존재하지 않는다.
- moveToGit(): 본 서비스에 사용된 코드와 관련 문서들이 포함된 github repository로 이동한다.

4.2.4.3 Context Diagram



[Figure 10] Info Context Diagram

4.2.4.4 Sequence Diagram



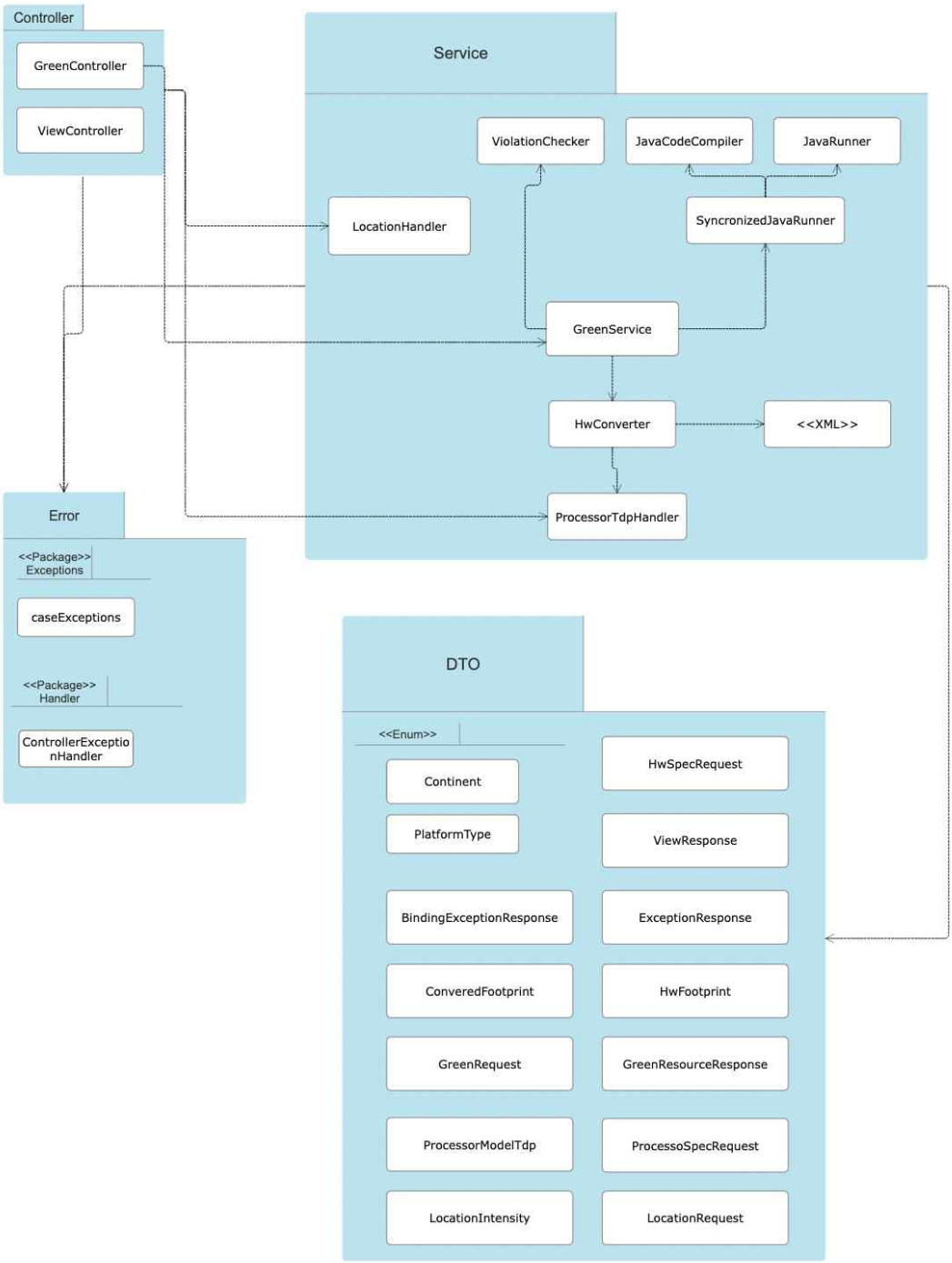
[Figure 11] Info Sequence Diagram

5. System Architecture – Backend

5.1 Objectives

이 장에서는 백엔드 시스템의 구조, 속성 및 기능을 설명하고 이를 전체 시스템의 동작 원리로 설명한다.

5.2 Overall Architecture

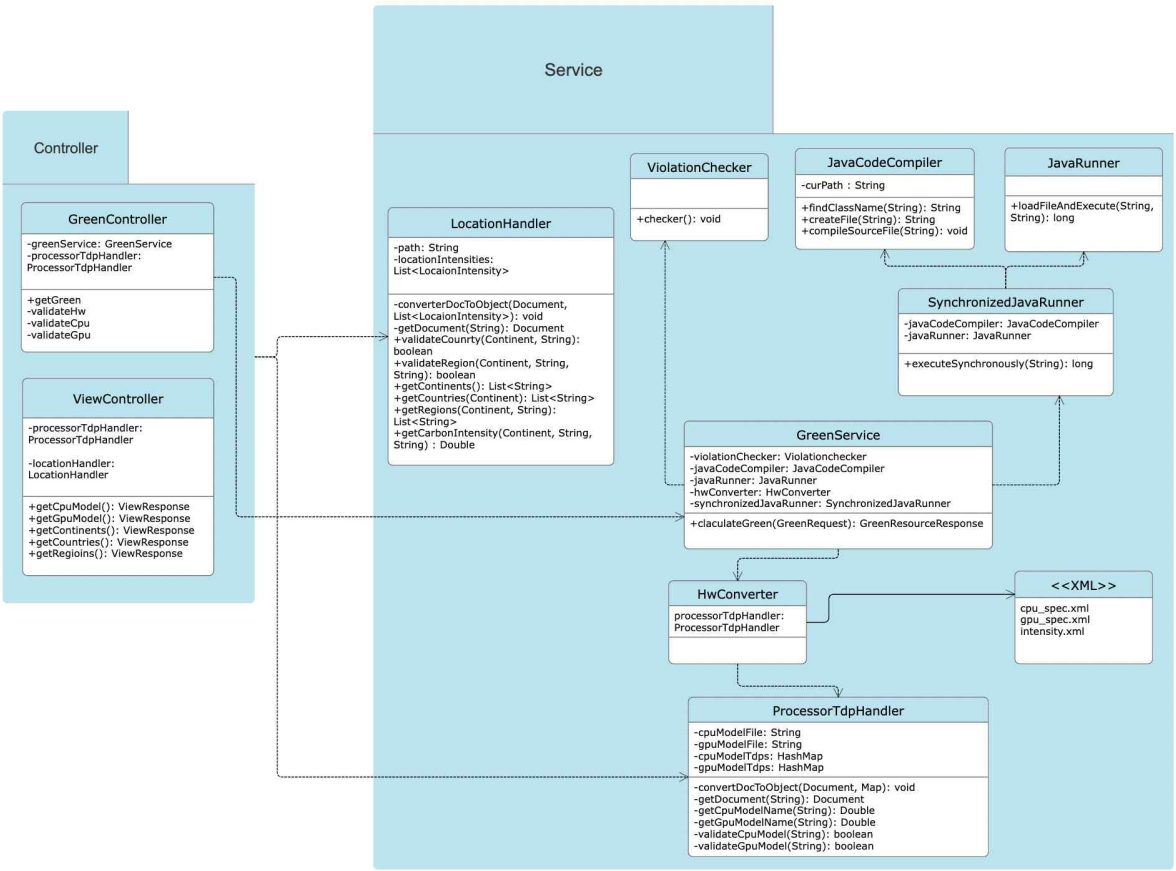


[Figure 12] Backend Overall Architecture

Controller에서 사용자의 요청을 받아서 Service와 Error를 통해 처리하고 결과를 다시 클라이언트로 반환하는 역할을 한다. Service에서 Controller로부터 입력 받은 Java Code를 검사 및 컴파일, 실행하여 실행 시간과 탄소 배출량을 계산한다.

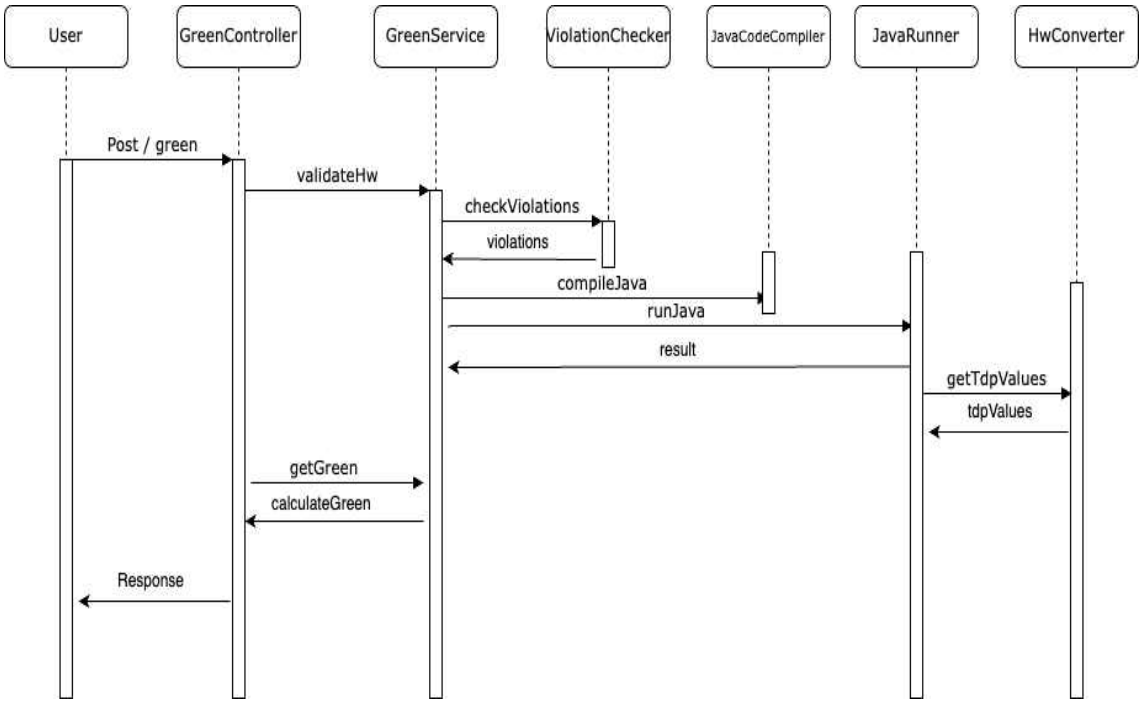
5.3 Sub Systems

5.3.1 Class Diagram for Controller and Service



[Figure 13] Backend Class Diagram

5.3.2 Sequence Diagram



[Figure 14] Backend Sequence Diagram

User가 Http Post 메서드를 Java Code를 서버에 전송하면 GreenController에서 GreenService를 통해 코드의 유효성을 검사하고, .class파일 생성, 컴파일 .java파일 생성, classLoader를 통해 JVM에서 입력받은 코드의 main함수 실행, 시간 체크, .class, .java파일 삭제 라는 일련의 과정을 거치게 된다.
이때, ViolationChecker를 통해 코드의 유효성 검사를 하고, JavaCodeCompiler를 통해 파일 생성 및 컴파일, JavaRunner를 통해 실행 및 시간을 체크합니다.

6. Protocal Design

6.1 Objectives

이 장에서는 앞서 소개한 프론트엔드와 백엔드의 component 간의 통신 절차를 설명한다. 이를 위해 사용되는 환경은 Spring boot 프레임 워크이며, 통신할 때 사용되는 형태는 JSON을 사용한다.

6.2 Protocol Details

6.2.1 HTTP

HTTP는 HyperText Transfer Protocol의 줄임말로, 직역하면 하이퍼텍스트 전달 프로토콜이다. 하이퍼텍스트(HyperText)는 인터넷 사용자가 필요한 정보의 자유로운 검색을 가능하도록 해주는 텍스트의 전개 방식이다. HTTP는 이러한 하이퍼텍스트 방식의 정보를 교환하기 위한 하나의 규칙이다. 즉, HTML과 같은 문서를 전송하기 위해 사용되며 OSI 7 계층에서 응용 계층에 있는 프로토콜이다. HTTP는 웹 브라우저와 웹 서버의 소통을 위해 디자인되었으며, 전통적인 클라이언트-서버 아키텍처 모델에서 클라이언트가 HTTP 메시지 양식에 맞춰 요청을 보내면, 이에 서버는 HTTP 메시지 양식에 맞춰 응답을 한다. HTTP는 특정 상태를 유지하지 않는 무상태성(Stateless)이 특징이다. HTTP 메시지는 클라이언트와 서버 사이에서 데이터가 교환되는 방식이며, Request와 Response 두 가지 유형이 있다. 클라이언트가 서버에 Request를 보내면, 서버는 Response로 클라이언트의 요청에 응답하는 방식이다.

6.2.2 RESTful API

REST(Representational State Transfer)는 HTTP URI(Uniform Resource Identifier)을 통해 Resource를 명시하고, HTTP Method(POST, GET, PUT, DELETE, PATCH)를 통해 해당 URI에 대한 CRUD Operation을 적용하는 것을 의미한다. REST 역시 서버-클라이언트 구조로 작동하며, HTTP 프로토콜의 인프라를 그대로 사용하므로 REST API 사용을 위한 별도의 인프라를 구축할 필요가 없다. HTTP 표중 프로토콜에 따르는 모든 플랫폼에서 사용이 가능하다. 본 ECO2 서비스에서는 사용자가 JAVA 코드를 입력 후, 계산 결과 처리를 위해 RESTful API를 사용하였다.

6.2.3 React Context API

React Context API는 React 애플리케이션에서 전역적으로 데이터를 관리하고 공유하기 위한 방법이다. 이는 React 컴포넌트 트리를 통해 데이터를 'props'로 전달하는 전통적인 방법의 복잡성과 제한을 극복하기 위해 도입되었다. Context API를 사용하면, 데이터가 여러 레벨의 컴포넌트를 중첩하여 거쳐야 하는 복잡성을 피할 수 있다. Context를 사용하면 특정 데이터를 전역적으로 설정할 수 있으며, 필요한 모든 컴포넌트에서 직접 접근이 가능하다. 이를 통해 데이터를 필요로 하는 모든 컴포넌트에 쉽게 데이터를 제공할 수 있다. 본 ECO2 서비스에서는 사용자가 코드 입력시 서버로 전송할 여러 컴퓨터 하드웨어 사양에 관련된 데이터를 전역 변수로 관리하기 위해 Context API를 사용하였다.

6.3 API

6.3.1 탄소배출량 조회

사용자가 JAVA 코드를 입력 후 서버에서 탄소배출량 계산을 위해 사용되는 API 이다.

6.3.1.1 Request

요소	설명
요청 타입	POST
경로	/green
HTTP 버전	HTTP/1.1
헤더	Content-Type: application/json;charset=UTF-8
	Content-Length: 752
	Host: localhost:8080
본문 (Body)	hwSpecRequest: 하드웨어 사양 요청
	cpuSpecRequest: CPU 사양
	modelName: "testCpuModel"
	usageFactor: 0.8
	coreNumber: 8
	tdp: null
	gpuSpecRequest: null
	memoryGigaByte: 8
	psf: 2.7
	locationRequest: 위치 요청
	continent: "NORTH_AMERICA"
	country: "Canada"
	region: "Nunavut"
	javaCode: Java 코드 예시

[Table 1] 탄소배출량 조회 Request

6.3.1.2 Response

요소	설명
상태 코드	200
상태 메세지	ok

HTTP 버전	HTTP/1.1
헤더	Content-Type: application/json
	Content-Length: 252
본문 (Body)	"totalCarbonFootprint":126.78
	"hwFootprint":{"cpuCarbonFootprint":32.01,"gpuCarbonFootprint":null,"memoryCarbonFootprint":52.26}
	convertedFootprint":{"energyNeeded":11.111,"treeMonths":22.222,"passengerCar":33.333,"flightFromIncheonToLondon":44.44}

[Table 2] 탄소배출량 조회 Response

6.3.2 CPU 모델명 리스트 조회

6.3.2.1 Request

요소	설명
요청 타입	GET
경로	/model/cpu
HTTP 버전	HTTP/1.1
헤더	Host: localhost:8080

[Table 3] CPU 모델명 리스트 조회 Request

6.3.2.2 Response

요소	설명
상태 코드	200
상태 메세지	ok
HTTP 버전	HTTP/1.1
헤더	Content-Type: application/json
	Content-Length: 4171
본문 (Body)	"count":186
	"data":["AMD Athlon 64 (Desktop) FX-62", "Intel Core i5-8600K",..., "AMD Ryzen 9 7950X"]

[Table 4] CPU 모델명 리스트 조회 Response

6.3.3 GPU 모델명 리스트 조회

6.3.3.1 Request

요소	설명
요청 타입	GET
경로	/model/gpu
HTTP 버전	HTTP/1.1
헤더	Host: localhost:8080

[Table 5] GPU 모델명 리스트 조회 Request

6.3.3.2 Response

요소	설명
상태 코드	200
상태 메세지	ok
HTTP 버전	HTTP/1.1
헤더	Content-Type: application/json
	Content-Length: 5667
본문 (Body)	"count":207
	"data":["AWGeForce RTX 2080 Super Max-Q (Laptop)","WGeForce RTX 3080 Ti (Laptop)",... ,"Intel Core i5-8600K"]

[Table 6] GPU 모델명 리스트 조회 Response

6.3.4 대륙 리스트 조회

6.3.4.1 Request

요소	설명
요청 타입	GET
경로	/location
HTTP 버전	HTTP/1.1
헤더	Host: localhost:8080

[Table 7] 대륙 리스트 조회 Request

6.3.4.2 Response

요소	설명
상태 코드	200
상태 메세지	ok
HTTP 버전	HTTP/1.1
헤더	Content-Type: application/json
	Content-Length: 95
본문 (Body)	"count":7
	"data"["EUROPE","NORTH_AMERICA","AFRICA","ASIA","OCEANIA","WORLD","SOUTH_AMERICA"]

[Table 8] 대륙 리스트 조회 Response

6.3.5 국가 리스트 조회

6.3.5.1 Request

요소	설명
요청 타입	GET
경로	/location/{continent}
HTTP 버전	HTTP/1.1
헤더	Host: localhost:8080

[Table 9] 국가 리스트 조회 Request

6.3.5.2 Response

요소	설명
상태 코드	200
상태 메세지	ok
HTTP 버전	HTTP/1.1
헤더	Content-Type: application/json
	Content-Length: 65
본문 (Body)	"count":3
	"data": ["Canada","United States of America", "Mexico"]

[Table 10] 국가 리스트 조회 Response

6.3.6 지역 리스트 조회

6.3.6.1 Request

요소	설명
요청 타입	GET
경로	/location/{continent}/{country}
HTTP 버전	HTTP/1.1
헤더	Host: localhost:8080

[Table 11] 지역 리스트 조회 Request

6.3.6.2 Response

요소	설명
상태 코드	200
상태 메세지	ok
HTTP 버전	HTTP/1.1
헤더	Content-Type: application/json
	Content-Length: 234
본문 (Body)	"count":14
	"data": ["New Brunswick", "Northwest Territories", ...,"Newfoundland and Labrador"]

[Table 12] 지역 리스트 조회 Response

7. Testing Plan

7.1 Objectives

본 장에서는 요구사항 명세서에서 기술한 시나리오를 바탕으로 통합 시스템의 동작 여부를 확인한다. Unit testing, Integration testing, Functional testing, User Acceptance Testing을 거쳐서 서비스 출하 전에 대부분의 오류, 결함을 발견하고자 한다. Development testing, Release testing, User testing 의 관점으로 테스트 과정을 설계, 세부 테스트 케이스를 설명하겠다.

7.2 Testing Policy

7.2.1 Development Testing

개발 과정 중 발생 가능한 다양한 위험과 비용을 최소화하는 것이 이 프로젝트의 주요 목표다. 이를 위해 소프트웨어가 기대한 기능을 수행하는지 철저히 확인하는 절차를 거친다. 초기 단계에서는 시스템의 각 부분, 즉 서브 시스템이나 컴포넌트별로 분리된 유닛 테스트를 실시한다. 이러한 유닛 테스트는 각각의 모듈이나 기능이 계획대로 작동하는지 평가하고, 특히 각 함수가 입력과 출력을 정확히 처리하는지에 초점을 맞춘다. 그리고 구현된 컴포넌트의 지속 가능한 유지 관리와 숨겨진 오류의 탐지를 위해 코드 리뷰를 진행한다. 유닛 테스트를 완료한 후에는 컴포넌트들을 차례로 결합해가며 통합 테스트를 수행한다. 이 통합 테스트는 여러 구성 요소들이 통합되었을 때 이들이 서로 제대로 작동하는지를 검증한다. 이 단계는 각 시스템 간의 인터페이스가 효과적으로 작동하는지를 평가하는 것을 포함한다. 뿐만 아니라, 시스템의 기능적 측면뿐 아니라 성능, 보안, 안정성과 같은 비기능적 요소도 심사해야 한다. 이를 위해, 정적 코드 검사나 데이터 흐름 분석과 같은 방법론을 적용할 수 있다.

7.2.2 Release Testing

이 시스템이 사용자의 환경에서 원활하게 작동하고, 모든 요구 조건을 만족시키며 기대하는 품질 기준에 부합하는지 평가하는 것이 중요하다. 새 버전의 출시와 함께 이루어지는 배포 과정이 제대로 이루어지는지도 확인한다. 시스템은 최신 버전으로 완벽하게 배포되어야 하며, 작동 중 문제가 발생해서는 안 된다. 보통은 원하는 기능을 포함한 초기 단계의 알파 테스트를 실시하고, 그 결과로 나온 피드백과 발견된 결함을 수정하여 이어서 베타 테스트를 진행한다. 알파 테스트는 개발팀 내부에서 실시하고, 베타 테스트는 제한된 범위의 사용자를 대상으로 공개적으로 진행한다.

7.2.3 User Testing

이 프로젝트는 사용자의 기대치와 요구사항을 만족시키는지 평가하는 과정을 포함한다. 예상 사용자가 시스템을 사용하는 과정을 시뮬레이션하여, 시스템이 요구사항 사양을 만족하는지 여부를 확인한다. 이를 위해, 동시에 접속하는 사용자 수 등을 고려한 시스템 사용 시나리오를 개발하고, 이 시나리오를 통해 시스템이 모든 Use Case를 만족하는지 검증한다. 이러한 검증을 통해 시스템의 베타 버전이 얼마나 유용한지 평가하고, 최종적으로 실제 환경에 배포될 것인지를 결정한다.

7.3 Test Case

7.3.1 Code

Element	Description
Summary	ECO2 시스템의 주 기능으로, 소스 코드를 입력하여 탄소 배출량 결과를 확인한다.
Pre-Conditions	사용자는 IDE 공간에 소스 코드를 입력한다.
Test Steps	1. 사용자는 로그인 필요없이 소스 코드를 입력하고 제출 버튼을 누른다. 2. 서버는 소스코드와 함께 전송된 HW 정보를 토대로 탄소 배출량을 계산하여 값을 반환한다.
Expected Results	소스 코드에 대한 탄소 배출량 결과가 표시된다.

[Table 13] Code test case

7.3.2 Statistics

Element	Description
Summary	ECO2 시스템의 부가 기능으로, 앞서 계산된 탄소 배출량의 각종 통계 자료를 확인한다.
Pre-Conditions	사용자는 코드를 통해 이미 탄소 배출량을 계산을 완료한 상태이다.
Test Steps	1. 사용자는 탄소배출량과 관련된 통계 자료를 다양한 시각화 기법이 적용된 자료로 확인한다.

Expected Results	탄소배출량에 관한 통계 자료를 확인한다.

[Table 14] Statistics test case

7.3.3 Server Setting

Element	Description
Summary	탄소 배출량 계산에 필요한 컴퓨터 하드웨어 정보를 관리할 수 있는 기능으로, 사용자는 필요에 따라 값을 변경할 수 있다.
Pre-Conditions	사용자가 값을 입력하지 않아도 이미 사용되는 기본 값이 각각 존재한다.
Test Steps	1. 사용자는 server 페이지에서 현재 설정된 하드웨어 값을 확인한다. 2. 변경을 원할 경우에는 다른 값을 입력하여 저장한다. 3. 이후 소스 코드를 입력하고, 탄소 배출량 계산 시 변경된 하드웨어 정보가 적용되어 계산된다.
Expected Results	탄소 배출량 계산에 사용되는 하드웨어 정보에 관한 값을 변경한다.

[Table 15] Server test case

8. Development Plan

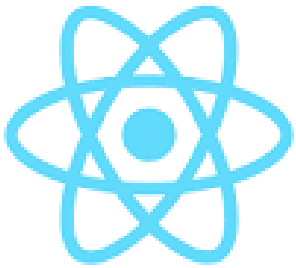
8.1 Objectives

이 장에서는 시스템을 구축하는 데 필요한 기술적 요소와 개발 환경, 그리고 개발 과정 중 준수해야 할 제한 사항들에 대해 설명한다.

8.2 Environment and Tools

8.2.1 Frontend Environment

8.2.1.1 React



[Figure 15] React Logo

React는 사용자 인터페이스를 구축하기 위한 JavaScript 라이브러리로, 주로 싱글 페이지 애플리케이션(SPA)의 개발에 사용된다. Facebook에 의해 개발되었으며, 컴포넌트 기반의 개발을 가능하게 하여 코드의 재사용성을 높이고 관리를 용이하게 한다. React는 선언적 프로그래밍을 지원하여 애플리케이션의 상태에 따라 UI를 효율적으로 업데이트한다. 또한, 가상 DOM을 사용하여 실제 DOM의 변경을 최소화함으로써 애플리케이션의 성능을 향상시킨다. React Native를 통해 모바일 애플리케이션 개발도 지원한다.

8.2.1.2 HyperText Markup Language

[Figure 16] HTML Logo

HTML은 웹페이지 구축에 사용되는 주요 마크업 언어로, W3C가 HTML 및 CSS 표준을 관리한다. 이 언어를 사용하여 제목, 단락, 목록 등의 본문 구조를 정의할 뿐만 아니라, 링크나 인용문 등을 포함하는 구조화된 문서를 생성할 수 있다. HTML은 태그를 사용하여 요소를 정의하며, 웹 브라우저 등의

HTML 처리 기기에서 동작하는 자바스크립트를 포함하거나 참조할 수 있다.

8.2.1.3 Tailwindcss

[Figure 17] Tailwind Logo

Tailwind CSS는 유틸리티 중심의 CSS 프레임워크로, 웹 개발자들이 빠르고 효율적으로 반응형 웹 인터페이스를 구축할 수 있도록 설계되었다. 이 프레임워크는 개발자가 HTML 내에서 직접 다양한 사전 정의된 유틸리티 클래스를 사용하여 스타일을 적용할 수 있게 한다.
또한, Tailwind는 반응형 디자인을 자연스럽게 지원하는데, 모바일 우선 접근 방식을 통해 다양한 화면 크기에 맞게 스타일을 조정할 수 있으며, 이는 미디어 쿼리 접두사를 사용하여 쉽게 구현할 수 있다. Tailwind는 사용하지 않는 CSS를 자동으로 제거하는 PurgeCSS와 통합되어 있어, 최종 배포 파일의 크기를 최소화하기 때문에, 웹사이트의 로딩 시간을 단축하고 전반적인 성능을 개선하는 데 도움이 되는 CSS 프레임워크이다.

8.2.2 Backend Environment

8.2.2.1 Spring boot

[Figure 18] Spring boot Logo

Spring Boot는 Java 기반의 오픈 소스 프레임워크로, Spring 프레임워크 위에 구축되어 있으며, 간결하고 빠른 웹 애플리케이션 개발을 가능하게 한다. Spring Boot는 "convention over configuration" 원칙을 따르며, 개발자가 최소한의 설정으로 즉시 실행 가능한 스탠드얼론 애플리케이션을 쉽

게 만들 수 있도록 지원한다. 이는 내장된 서버, 보안 설정, 메모리 관리 등을 포함해 많은 기본 설정을 자동화함으로써 개발 시간을 크게 단축시킨다.

Spring Boot는 Spring의 강력한 주입(IoC) 기능과 AOP, 데이터 접근 기술 등을 통합하며, 마이크로서비스 아키텍처를 구현하는 데 이상적인 환경을 제공한다. 또한, Spring Boot는 다양한 'Starters'를 제공하여, 데이터베이스, 보안, 데이터 처리와 같은 일반적인 애플리케이션 요구사항을 쉽게 추가할 수 있도록 한다. 개발자는 이러한 스타터를 이용하여 필요한 의존성을 프로젝트에 자동으로 포함시키고, 복잡한 빌드 구성을 간소화할 수 있다.

8.2.2.2 Docker

[Figure 19] Docker Logo

Docker는 애플리케이션을 컨테이너라는 격리된 환경에서 실행할 수 있게 해주는 오픈 소스 플랫폼이다. 이 컨테이너는 애플리케이션과 필요한 모든 의존성을 포함하여, 어떤 환경에서도 동일하게 작동할 수 있다. 개발, 테스트, 프로덕션 등 다양한 환경에서 애플리케이션의 일관된 동작을 보장한다.

8.3 Constraints

본 ECO2 시스템은 이 문서에서 언급된 내용들에 기반하여 디자인되고 구현될 것이다. 이를 위한 세부적인 제약사항은 다음과 같다.

- 본 ECO2 시스템은 별도의 데이터베이스 시스템을 사용하지 않았다. 이는 사용자의 개인 정보를 저장할 필요성이 없다고 판단하여, 사용자의 편리성 및 시스템의 성능 (속도, 반응성)을 높이고자 하였다.
- 사용자의 이용성을 최우선으로 하여 이 시스템을 이용하기 위해 복잡한 입력을 받는 일은 지양한다.
- 본 ECO2 시스템의 목적이 소스 코드의 탄소배출량을 계산하여 개발자 역시 환경친화적 목적을 달성할수 있도록 하기 위함이므로, 시스템 개발에 있어서도 환경 낭비가 없게끔 코드 최적화를 지속적으로 진행한다.
- 소스 코드는 개발을 진행한 당사자가 아니더라도 쉽게 이해하고 재사용이 가능하게끔 누구나 사용하는 보편적인 구조로 작성한다.

- frontend와 backend의 호환성을 고려하여 주고받는 API 간의 데이터 유형, 내용, 변수명 등을 통일한다.

8.4 Assumptions and Dependencies

본 문서의 모든 시스템은 데스크탑 환경에 기반하여 디자인 및 구현되었다고 가정하며 작성되었다. 또한 윈도우 10, 11 기반의 OX 환경을 기반으로 하여 작성되었으며 따라서 다른 OS나 조건을 만족하지 않는 환경에서 시스템의 지원은 보장할 수 없다.

9. Supporting Information

9.1 Software design specification

본 소프트웨어 디자인 명세서는 IEEE 권장 사항에 맞추어 작성되었다 (IEEE Standard for Information Technology Systems Design Software Design Descriptions, IEEE-Std-830). 다만 본 시스템의 설계 사항을 용이하게 확인할 수 있도록 본래의 양식을 일부 수정하여 사용하였다.

9.2 Document history

Date	Description	Version
2023/11/19	1. Preface	1.0
2023/11/19	2. Introduction	1.0
2023/11/19	3. System Architecture-Overall	1.0
2023/11/19	4. System Architecture-Frontend	1.0
2023/11/19	5. System Architecture-Backend	1.0
2023/11/19	6. Protocol Design	1.0
2023/11/19	7. Testing Plan	1.0
2023/11/19	8. Development Plan	1.0
2023/11/19	9. Supporting Information	1.0