

[지구 구조대] Software Requirements Specification

by TEAM 9

김영석 안윤지 한승호 한용준 홍서윤 황수영

Instructor: 이은석

TA: 김진영 최동욱 허진석 김영경

Document Date: 17 April, 2024

Faculty: Sungkyunkwan University

** Authors' name in alphabetical order*

1. Introduction.....	5
1.1. Purpose.....	5
1.2. Scope	5
1.3. Definitions, Acronyms and Abbreviations.....	5
1.4. References	6
1.5. Overview.....	6
2. Overall Description	6
2.1. Product Perspective	6
2.1.1. System Interfaces.....	7
2.1.2. User Interfaces	7
2.2. Product functions.....	8
2.2.1. JAVA Code input 및 전송.....	9
2.2.2. 탄소 배출량 측정 및 제공.....	9
2.2.3. 그린패턴 리팩토링 및 제공.....	10
2.3. User classes and characteristics	10
2.3.1. Java 코드 개발자.....	10
2.4. Design and implementation constraints	11
2.5. Assumption and dependencies.....	11
2.5.1. 그린패턴 리팩토링 코드 제공 시 가정하는 사항.....	11
2.5.2. 프로젝트 단위의 코드 입력에 대한 가정 사항.....	11
2.5.3. 하드웨어 사양 및 코드 실행 환경.....	12
3. Specific Requirements.....	12
3.1. External interfaces.....	12
3.1.1. User Interface	12
3.2. Function requirements.....	13
3.2.1. Use Case.....	13
3.2.2. Use Case Diagram	17
3.2.3. Data Dictionary.....	17
3.2.4. Data Flow Diagram.....	18
3.3. Performance Requirements	18
3.3.1. Static Numerical Requirement.....	18
3.3.2. Dynamic Numerical Requirement.....	19

3.4. Logical Database Requirements.....	19
3.5. Design Constraints	19
3.5.1. Standard Development Tools	19
3.5.2. Standard Compliance.....	19
3.5.3. Physical Constraints	19
3.6. Software System Attributes	20
3.6.1. Reliability.....	20
3.6.2. Availability	20
3.6.3. Security	20
3.6.4. Maintainability	20
3.6.5. Portability.....	21
3.7. System Architecture.....	21
3.8. System Evolution.....	22
3.8.1. Expected Change of User Requirements.....	22
4. Appendixes	22
4.1. Green pattern	22
4.2. Software Requirements Specification	22
4.3. Formula of Carbon Emission	22
4.4. Document history	23

1. Introduction

1.1. Purpose

“지구 구조대”는 Java 코드 탄소 배출량 측정 및 그린패턴 리팩토링의 두가지 기능을 Java 개발자를 포함하는 여러 사용자층에게 제공하는 것에 목적을 둔다. 이 문서는 지구 구조대의 requirements를 명확히 하여 시스템이 제공해야 할 기능, 제한사항을 정의한다. 이를 통한 추후 requirements 변경시, 유지 보수 및 관리시 개발자, 관리자 등의 관련자들의 참고를 목적으로 한다.

1.2. Scope

지구 구조대는 Web app을 통해서는 사용자로부터 단일 Java 코드를 입력받고, CLI app을 통해서는 사용자로부터 Java 프로젝트를 입력받아 해당 코드에 대한 코드 탄소 배출량 측정과 그린패턴 리팩토링 두가지 기능을 공통의 백엔드 시스템에서 수행하게 된다.

해당 시스템은 평균적인 HTTP 요청으로 다루기에 시간이 오래 걸릴 수 있는 컴파일 아하 “compile job”)과 탄소 배출량 측정 아하 “measure job”)을 포함하는데, consumer-producer pattern을 사용함으로써 오래 걸리는 job을 HTTP 컨텍스트와 분리시켜 사용자에게 더 나은 사용성을 제공하는 것을 포함한다. 또한 정확한 measure job의 수행을 위해 docker 등으로 별도의 격리된 logical machine을 사용하고, 가장 보편적인 사양으로 구성하려고 노력되 정확한 측정은 애초에 불가능함을 인지한다.

1.3. Definitions, Acronyms and Abbreviations

탄소배출량	개인 조직 또는 활동이 대기 중에 방출되는 온실 가스인 이산화탄소(CO2)와 다른 온실 가스들을 생산하는 양을 측정하는 지표 [1]
그린패턴	환경 친화적인 소프트웨어 개발 및 운영을 위한 패턴이나 관행을 일컫는 말. 그린패턴은 소프트웨어 시스템이 에너지를 효율적으로 사용하고, 자원 소비를 최소화하며, 환경 영향을 최대한 줄이는 것을 목표로 함
그린패턴 리팩토링	시스템의 주요 기능 중 하나로, 기존 탄소배출량이 많았던 코드를 리팩토링하여 효율적으로 만들어 탄소를 덜 배출하는 코드를 제안한다.
탄소 배출량 측정	시스템의 주요 기능 중 하나로, 소스코드가 동작할 때 발생하는 탄소배출량을 [1] 을 따라 측정한다.
CLI	Command-Line Interface의 약자로, 사용자가 컴퓨터와 상호 작용하기 위해 명령어를 입력하는 인터페이스. 해당 시스템은 Linux, MacOS의 bash, zsh 등의 환경에서 동작하는 CLI app을 제공한다.
Consumer-Producer pattern	Message Queue 형태의 작업 목록을 가운데 두고 작업을 생산해내는 주체와 작업을 처리하는 주체를 분리시키는 설계 방법. 해당 시스템에서는 Compile, Measure job을 처리하는 worker의 scale-out, fault tolerance 등의 관리 용이성 때문에 이 패턴을 채택했다.

Compile Job	시스템의 주요 과정 중 하나로, 자바로 작성된 .java파일 혹은 java 프로젝트를 실행 가능한 binary로 컴파일하는 작업의 단위
Measure Job	시스템의 주요 과정 중 하나로, Compile job에서 생성된 binary의 탄소 배출량을 측정하는 작업

1.4. References

- [1] Lannelongue, J. Grealey, M. Inouye, Green Algorithms: Quantifying the Carbon Footprint of Computation. Adv. Sci. 2021, 8, 2100707, pp8. <https://doi.org/10.1002/advs.202100707>
- [2] <https://calculator.green-algorithms.org/>
- [3] <https://github.com/GreenAlgorithms/green-algorithms-tool>

1.5. Overview

본 요구사항 명세서는 지구구조대 시스템의 목적, 요구사항과 제약사항을 포함한다. 시스템의 목적, 요구사항을 먼저 high-level에서 다루기 위해 사용자 입장에서 제공하는 기능과 프로토타이핑된 UI에 대해 설명한다. 이때 지구구조대의 사용자 층이나 시스템이 가지는 제약사항에 대한 사용자의 특성을 파악하고 시스템의 제약사항을 정의한다.

이후 세부적인 external interface, 기능적 요구사항을 설명을 use case를 바탕으로 설명하거나, data flow를 diagram으로 설명한다. 이때 사용자의 input으로부터 output이 도출되는 전 과정을 Web App과 CLI App 각각에 대해 설명한다.

2. Overall Description

2.1. Product Perspective

기후 변화의 영향이 날로 심각해지면서, 디지털 경제에서도 탄소배출량을 줄이는 일이 고려되고 있다. 디지털 서비스 수요가 증가하면서 데이터 센터부터 통신 네트워크, 기기까지 모든 부분에서 에너지 소비가 더 커지며 중요성은 더 커지고 있다. 이러한 상황에서 지구구조대는 탄소 배출량 측정 및 탄소배출 절감을 위한 코드개선을 목표로 한다. 지구구조대를 통해 코딩을 하고자 하는 개인, 단체는 본인이 제작한 코드가 어느 정도의 에너지를 소비하는지, 또한 이에 대한 피드백 즉 에너지를 더 적게 소비하기 위하여 이 소프트웨어에 의해 제안된 코드를 제공받는다. 특히 대규모시스템을 개발, 운영하는 기업이나 단체는 비효율적인 코드가 에너지 사용에 큰 영향을 미칠 수 있으므로, 보다 최적화된 솔루션을 적용할 경우, 에너지 절약 효과와 탄소배출 감소가 클 것으로 예상된다.

2.1.1. System Interfaces

고객이 자신의 Java 코드의 탄소 배출량을 측정하고자 할때, 지구구조대 웹사이트를 통해 해당 코드를 제출한다. 서버는 이를 받아 컴파일 단계, 탄소 배출량 측정 단계, 그리고 그런때론 인식 및 수정 단계로 순차적으로 처리한다. 탄소 배출 측정은 정확성을 보장하기 위해 독립된 환경에서 수행된다. 컴파일 단계에서 서버는 오류발생 유무, 악성코드의 포함여부를 확인한다. 서버에 의해 앞의 두가지 사항중 한 가지라도 감지가 될 시, 서버는 고객의 코드 분석을 중단한다. 이후 고객은 각 상황에 대한 설명이 담긴 여러 화면을 보게 되고 코드를 다시 제출하여야 한다. 여러화면에는 어떠한 이유로 서버측에서 중단이라는 판단을 했는지에 대한 설명이 제시되어 있다.

2.1.2. User Interfaces

고객은 분석하고자 하는 코드의 규모에 따라 Web App과 CLI App 두가지 형태로 시스템을 이용할 수 있다. 단일 Java 코드에 대하여 측정을 하고자 할때는 Web app 형태로 접근을 하고 Java 프로젝트 단위로 시스템에 입력을 하고자 한다면 CLI App 형태로 접근을 한다

- Web app 형태로 접근을 하는 과정은 고객이 지구구조대 웹사이트에 접속한다. 웹사이트 코드 입력란에 코드를 입력하면 2.1.1에 해당하는 현재 진행중인 단계와 진행 상황을 실시간으로 제공받는다. 서버의 모든 작업이 끝난 후 고객은 제출한 코드의 탄소 배출량 측정 결과와 배출량 절감을 위해 수정된 코드를 제공받는다.

(예시)

Prototype 0.0.0

지구구조대

Enter your Java code below

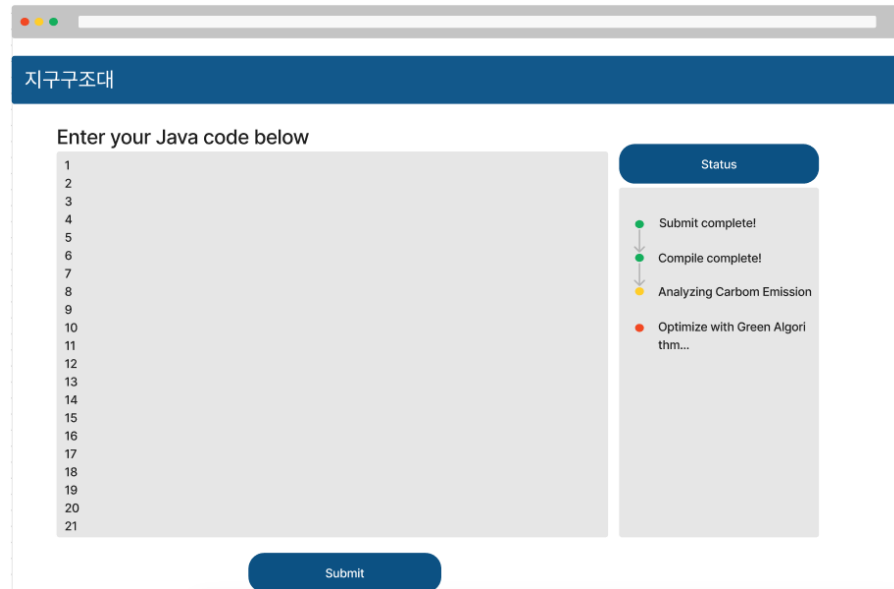
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21

upload your java code

Submit

Error!
SyntaxErrorExample.java:7:
error: ';' expected

Prototype 0.0.1



코드를 입력하고 'submit' 버튼을 누르면 제출이 된다. status 창은 진행단계 상태 혹은 에러 약성코드 탐지시 서버측의 사용자 코드 분석 중단에 대한 내용을 보여주기 위해 존재한다. Prototype 0.0.0 에서 0.0.1로의 변경사항은 Java 코드 업로드기능 삭제다. 0.0.0에서 status 창 위의 'upload your java code' 버튼으로 사용자의 로컬에 있는 Java 코드를 입력받는 기능을 제시하였으나 웹사이트에서의 Java 코드는 단순한 코드를 대상으로 하기 때문에 간단한 copy/paste로 가능하다고 판단하여 0.0.1에서 기능을 삭제하였다. 또한 0.0.0에서 제시되지 않았던 진행단계를 status창에 명시하도록 하였다.

- 개발자들은 주로 프로젝트 단위로 개발을 한다. 프로젝트 단위 내에서 다양한 서브시스템간 상호작용에 의해 단일 코드만으로는 전체 프로젝트 측면에서의 탄소 배출량을 정확하게 측정하기는 쉽지 않다. 즉 emergent properties를 고려하여 프로젝트 단위로 측정을 하고 최적화된 솔루션을 받을 수 있도록 CLI App 형태로 입력을 받는 것이 필요하다. CLI를 통해 고객은 명령어를 통해 directory내의 JAVA 파일을 모두 입력하면 2.1.1 의 과정을 통해 전체 프로젝트 단위에서 측정된 탄소 배출량, 최적화 된 솔루션을 제공받는다.

2.2. Product functions

본 시스템이 사용자에게 제공하는 기능을 간략하게 설명한다. 사용자는 Web App과 CLI App 두가지 형태로 시스템에 접근할 수 있다. 각 기능은 선형적인 구조를 하고있어 Java code input 및 전송 - 탄소배출량 측정 및 그로테넨 리팩토링 시행 - 결과물 제공 의 순서를 하고있다. 각 기능들은 3.2 Function Requirements에서 더 자세히 설명한다.

2.2.1. JAVA Code input 및 전송

Web App	<ul style="list-style-type: none"> ● 기본적인 text editor가 제공되어 이를 통해 소스 코드를 작성할 수 있다. 클립보드를 통한 copy/paste가 가능해야 하며 단일 JAVA 코드를 입력받는다. ● Submit 버튼을 통해 서버로 코드를 전송할 수 있으며 탄소 배출량 측정과 그로테넨 리팩토링을 동시에 진행한다. ● 코드가 전송 및 시행되는 중 대략적인 진행단계를 status 창을 통해
---------	---

	<p>확인할 수 있다 (컴파일 진행중, 컴파일 성공, 탄소배출량 측정중, 분석 완료)</p> <ul style="list-style-type: none"> ● 실행이 거부되거나 에러로 완결되지 않았을 경우 status창을 통해 현황을 확인할 수 있다
CLI App	<ul style="list-style-type: none"> ● CLI 명령어를 통해 Directory 경로를 입력받는다. 해당 경로 Directory가 Java 프로젝트의 루트라고 가정 프로젝트 전체가 업로드된다 ● 탄소 배출량 측정과 그로테이션 리팩토링은 명령어가 구분되어 원하는 기능을 선택하여 사용할 수 있다. 사용자는 CLI 명령어에 대해 -h flag로 사용법을 찾아보거나 man으로 자세한 정보를 확인할 수 있다 ● 코드가 전송 및 실행되는 중 대략적인 진행단계를 로딩 바를 통해 확인할 수 있다 ● 실행이 거부되거나 에러로 완결되지 않았을 경우 원인을 마지막 line에서 확인할 수 있다

2.2.2. 탄소배출량 측정 및 제공

[App에 따른 탄소배출량 측정값 제공 방법]

Web App & CLI App	Measure job 실행 환경에서 컴파일된 바이너리를 실행한 뒤 탄소 배출량 계산 알고리즘에 따라 탄소 배출량을 측정한다.
Web App	기존 코드와 개선된 코드의 탄소 배출량을 비교할 수 있도록 수치와 이미지로서 제공된다. 동시에 일반인에게 친숙한 다양한 지표로 환산되어 그래프 이미지와 함께 제공된다.
CLI App	측정 결과는 간략한 텍스트로서 제공된다. 일반인에게 친숙한 몇가지 지표로 환산되어 텍스트로 함께 제공된다.

2.2.3. 그로테이션 리팩토링 및 제공

[App에 따른 그로테이션 리팩토링 결과 제공 방법]

Web App & CLI App	<ul style="list-style-type: none"> ● 입력된 코드 및 프로젝트에서 자원을 낭비하는 코드에서 시스템에서 자원하는 그로테이션이 존재하는지 찾고 리팩토링을 제안한다. ● 사용자는 그로테이션 리팩토링 제안의 코드 diff를 line by line으로 색상 등 시각적 표시로 확인할 수 있으며 수정된 이유와 로직에 대한 간단한 설명을 확인할 수 있다.
-------------------	---

Web App	그린패턴 리팩토링 제안 결과물은 웹페이지에 코드 에디터 화면을 좌우로 구분하여 diff를 표시한다. 좌우 화면이 함께 scroll되어 라인별로 비교하기 쉽게 한다.
CLI App	리팩토링 제안이 존재하는 파일에 대해 리팩토링을 적용하기에 앞서 사용자가 각 파일별로 검토할 수 있도록 코드 diff를 CLI상에서 제공한다. 제출한 모든 파일에 대해 각각 한 화면씩 display되며 사용자는 화면을 scroll하여 코드를 검토할 수 있고, y/n를 통해 수정여부를 파일별로 채택(y) 혹은 기각(n)할 수 있다. 채택된 파일은 수정된다.

2.3. User classes and characteristics

본 프로젝트가 타겟으로 하는 사용자는 어떤 유형이 있으며 어떤 특성을 가지는지 기술한다.

2.3.1. Java 코드 개발자

- 소스코드에 대한 탄소 배출량 측정 도구를 필요로 한다. 탄소배출로 인한 환경 문제에 관심이 있으며 이 문제를 해결하기 위한 노력의 일환으로 소스코드 탄소 배출량 절감을 시도한다. 잘못 개발된 소스코드는 코드가 제공하는 기능에 비해 과하게 에너지를 소모하고, 이에 따라 탄소 배출량을 증가시킬 수 있음을 아하고 있다. 소스 코드, 또는 프로젝트를 사용함에 있어서 탄소 배출량을 측정하고, 탄소 배출과 관련하여 더 개선할 점이 무엇인지 알고자 한다.
- 코드 작성에 Java 언어를 사용한다. Java 언어를 사용한 소스코드나 프로젝트를 작성한다.
- 환경문제와 관련된 전문적 지식이 부족하다. 환경에는 관심이 있으나 탄소 배출량 측정법이나 측정에 사용되는 단위 등에 대한 전문적인 도메인 지식을 갖고 있지는 않다.

2.4. Design and implementation constraints

설계와 구현에 있어 고려해야 하는 여러 non-functional requirements에 대해 설명한다.

Usability requirement	<ul style="list-style-type: none"> ● 측정된 탄소 배출량 예상치를 사용자에게 이해하기 쉽게 제시해야 한다. ● 일반적으로 프로그램 개발에 있어 단일 소스코드 파일만 사용되는 경우는 드물기 때문에 프로젝트 단위의 입력도 받을 수 있어야 한다.
Development requirement	<ul style="list-style-type: none"> ● 추후에 그린패턴을 새롭게 추가하기 쉽도록 확장성을 고려하여 개발해야 한다. ● 소스코드 탄소 배출량 절감의 의도를 가진 시스템인 만큼, 본 시스템도 마찬가지로 탄소 배출량에 대해 최적화하여 개발되어야 한다.
Operational requirement	동시 사용자를 고려하여 서버에서 동시에 여러 요청을 처리할 수 있도록 해야 한다.

Security requirement	사용자가 악의적인 코드를 입력할 수 있다는 가능성을 고려하여 이에 대응할 수 있도록 해야 한다
----------------------	--

2.5. Assumption and dependencies

기본 가정과 함께 요구 사항 제언에 영향을 미치는 요소에 대해 설명한다

2.5.1. 그라운드 룩업 코드 제공 시 가정하는 사항

- 본 프로젝트는 Java 언어를 사용하여 코드를 작성할 수 있는 사용자를 타겟으로 하기 때문에 수정된 코드를 제안할 때에 사용자가 이에 대한 이해를 바탕으로 적용 여부를 결정할 수 있다고 가정하고, 수정된 코드에 대한 별도의 설명을 제공하지 않는다
- 그라운드 룩업 코드는 기존의 코드와 비교했을 때 다르게 동작할 수 있으며, 이것은 탄소배출 절감을 위한 일종의 제언이므로, 코드의 기능과 동작을 점검하고 제언사항을 수용할지 말지 결정하는 책임은 사용자에게 있다

2.5.2. 프로젝트 단위의 코드 입력에 대한 가정 사항

- gradle로 관리/빌드하는 프로젝트라고 가정한다
- 입력된 디렉토리 하위의 모든 파일을 가져왔을 때 gradle build 명령어로 빌드될 수 있는 프로젝트를 대상으로 한다

2.5.3. 하드웨어 사양 및 코드 실행 환경

탄소 배출량은 코드 실행 환경에 따라 다르게 측정될 수 있는데, 사용자가 원하는 실행 환경에 맞추서 매번 Measure job worker를 provisioning하는 데에는 어려움이 있다. 따라서 탄소 배출량의 측정을 위해 사용되는 코드 실행 환경은 서버의 하드웨어 사양에 의존하게 됨을 명시하고, 이에 대한 정보를 사용자에게 정확히 제공한다. 이 접근 방식은 연구에서의 일관성을 유지하고 사용자가 정확한 환경 정보를 인지할 수 있도록 보장한다.

3. Specific Requirements

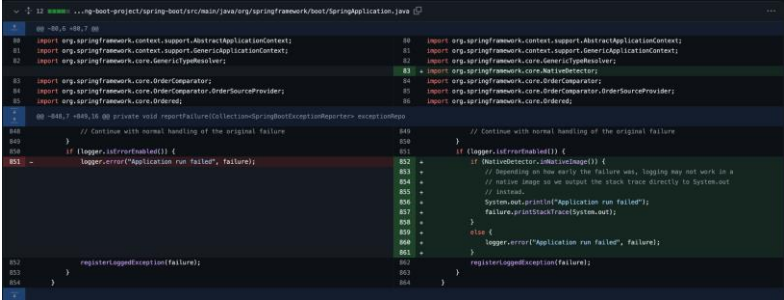
3.1. External interfaces

전체 시스템을 하나로 묶어서 보면, 외부에 노출하는 인터페이스는 전부 end-user와 상호작용하기 위한 User Interface뿐이다. 주로 사용자로부터 Java 코드를 입력받고, 처리 결과를 보여준다.

3.1.1. User Interface

[User Interface 1: Web app을 통한 Java 코드 제출]

목적	<ul style="list-style-type: none"> ● 단일 Java 코드의 탄소 배출량 측정 ● 단일 Java 코드에 대한 그라운드 룩업
입력	단일 Java 코드를 web app상의 코드 에디터에 입력한 뒤 제출버튼을 클릭하여 입력한다.
출력	<ul style="list-style-type: none"> ● 탄소 배출량 예상치

	<ul style="list-style-type: none"> ● 그론폰틴 리팩토링
탄소 배출량 측정 단위 및 허용 오차	<ul style="list-style-type: none"> ● 측정 단위 kg C를 사용하며 이해를 돕기 위해 실생활의 다양한 예시로도 표현한다 (자동차/비행기 이동거리 나무 몇그루 등) ● 허용 오차 탄소 배출량을 예상하는 것뿐 이론적으로 정확할 수 없음을 고려하면 허용 오차를 정하는 것이 큰 의미가 없다 사용자에게 이러한 점을 반드시 인지시킨다
그론폰틴 리팩토링 시각화 방법	<ul style="list-style-type: none"> ● github에서 git diff를 보여주듯이 좌우 화면을 분할하여 변경 제안을 시각화한다 (예시) 

[User Interface 2: CLI app을 통한 Java 코드 제출]

목적	<ul style="list-style-type: none"> ● Java 프로젝트 단위의 탄소 배출량 측정 ● Java 프로젝트 단위의 그론폰틴 리팩토링
입력	Java 프로젝트 루트 디렉토리에서 CLI app의 명령어를 사용하여 하위 디렉토리에 존재하는 모든 Java 코드 파일을 입력한다
출력	상동 하지만 시각화 방법이 CLI에 맞추어 최적화된다
탄소 배출량 측정 단위 및 허용 오차	상동
그론폰틴 리팩토링 시각화 방법	Lint를 도와주는 CLI app 등에서 diff를 보여주기 위해 사용하는 방법과 유사하게 각 파일별로 변경 제안을 시각화한다

3.2. Function requirements

3.2.1. Use Case

[Use case 1: Web app을 통한 탄소 배출량 측정 & 그론폰틴 리팩토링]

Actor	<ul style="list-style-type: none"> ● Java 개발자 ● 정부 기관 ● 환경 단체
-------	--

Description	Web app을 통해서 사용자가 Java 단일 코드를 제출하면 해당 코드 실행 시 배출되는 탄소 배출량과 그 때의 리팩토링 결과를 제시한다
Normal Course	<ol style="list-style-type: none"> 1. Java 개발자가 Web app 에 접속한다 2. Web app 내 code editor에 작성하고자 하는 코드를 입력한다 3. 코드가 HTTP 서버로 전송된다 4. HTTP 서버가 compile job을 produce한다 <ol style="list-style-type: none"> a. 임의의 job id를 발행하고 DB에 해당 id로 row를 추가한다. 코드를 storage에 업로드하고 그 경로를 DB에 저장한다 b. Compile Job Message Queue에 job id를 포함하는 message를 enqueue 한다(=produce). 5. Compile job produce가 성공했다면 <ol style="list-style-type: none"> a. 그 때의 리팩토링을 수행한다 b. HTTP 서버는 그 때의 리팩토링 결과를 포함하여 200 응답을 내려준다 c. Web app은 그 때의 리팩토링 결과를 먼저 디스플레이한다 d. Web app은 탄소 배출량 측정 job status를 확인하기 위해 DB에 주기적으로 polling한다. Compile, measure job이 전부 완료되기 전까지 web app에서는 progress를 보여준다 6. Compile worker가 compile job을 consume하고 처리한다 <ol style="list-style-type: none"> a. Compile Job Message Queue를 바라보면서 대기하던 worker는 message(job)가 들어오면 꺼낸다 b. 꺼낸 job의 코드 경로를 DB에서 확인한 뒤 storage로부터 다운로드한다 c. Java 컴파일러를 실행하고 결과 바이너리를 storage에 업로드하고 그 경로를 DB에 저장한다 7. Compile worker가 measure job을 produce한다 <ol style="list-style-type: none"> a. Measure Job Message Queue에 같은 job id를 포함하는 message를 enqueue 한다(=produce). b. 일련의 과정이 성공 시 Compile job을 완료했음을 Compile Job MQ에 notify한다 (중복 처리 방지) 8. Measure worker가 measure job을 consume하고 처리한다 <ol style="list-style-type: none"> a. Measure Job Message Queue를 바라보면서 대기하던 worker는 message(job)가 들어오면 꺼낸다 b. 꺼낸 job의 바이너리 경로를 DB에서 확인한 뒤 storage로부터 다운로드한다 c. 해당 바이너리를 N회 실행한다. measure worker는 각 마다 논리적 마신이므로 subprocess로 바로 실행해도 다른 영향을 적게 받음 d. 실행 결과 CPU/메모리 사용량을 바탕으로 탄소 배출량 측정하고, 그 결과를 DB에 업데이트한다 e. 일련의 과정이 성공 시 Measure job을 완료했음을 Measure Job MQ에 notify한다 9. DB를 Polling하던 web app은 job status가 완료된 걸 확인하면 DB에 있는 탄소

	배출량 측정 결과를 디스플레이한다.
Precondition	- 사용자가 문제없이 컴파일 가능한 Java 코드 형식을 갖춘 텍스트를 제출한다.
Post Condition	- 사용자가 Web app을 통해 탄소 배출량과 그때그때 리팩토링 결과를 확인할 수 있다.
Assumptions	- 사용자가 Chrome 웹 브라우저를 사용한다.

[Use case 2: CLI app을 통한 탄소배출량 측정]

Actor	<ul style="list-style-type: none"> ● Java 개발자 ● 정부기관 ● 환경 단체
Description	사용자가 CLI app을 통해 Java 프로젝트 단위의 코드를 제출하면 해당 코드 실행 시 배출되는 탄소 배출량을 측정하여 제시한다.
Normal Course	<ol style="list-style-type: none"> 1. Java 개발자가 CLI 환경에서 프로젝트 루트 디렉토리에 접근한다 2. 탄소 배출량 측정 명령어를 입력한다 (예시) <code>`earthsaver --measure .`</code> 3. 코드가 HTTP 서버로 전송된다 4. HTTP 서버가 compile job을 produce한다 <ol style="list-style-type: none"> a. 임의의 job id를 발행하고 DB에 해당 id로 row를 추가한다. 코드를 storage에 업로드하고 그 경로를 DB에 저장한다. b. Compile Job Message Queue에 job id를 포함하는 message를 enqueue 한다(=produce). 5. compile job produce가 성공했다면 <ol style="list-style-type: none"> a. HTTP 서버는 200 응답을 내려준다. b. CLI app은 탄소 배출량 측정 job status를 확인하기 위해 DB에 주기적으로 polling한다. Compile, measure job이 전부 완료되기 전까지 CLI app에서는 progress를 보여준다. 6. 상동 7. 상동 8. 상동 9. DB를 Polling하던 CLI app은 job status가 완료된 걸 확인하면 DB에 있는 탄소 배출량 측정 결과를 디스플레이한다.
Precondition	- 사용자가 문제없이 컴파일 가능한 Java 코드 형식을 갖춘 텍스트를 제출한다.
Post Condition	- 사용자가 CLI app을 통해 탄소 배출량 측정 결과를 확인할 수 있다.
Assumptions	- 사용자가 linux, macOS 환경에서 zsh, bash 등의 셸을 사용한다.

	<ul style="list-style-type: none"> - Java 프로젝트 루트 디렉토리 하위에 있는 모든 .java 확장자 파일만으로 정상적으로 컴파일 가능한 프로젝트이다.
--	---

[Use case 3: CLI app을 통한 그라운드 리팩토링]

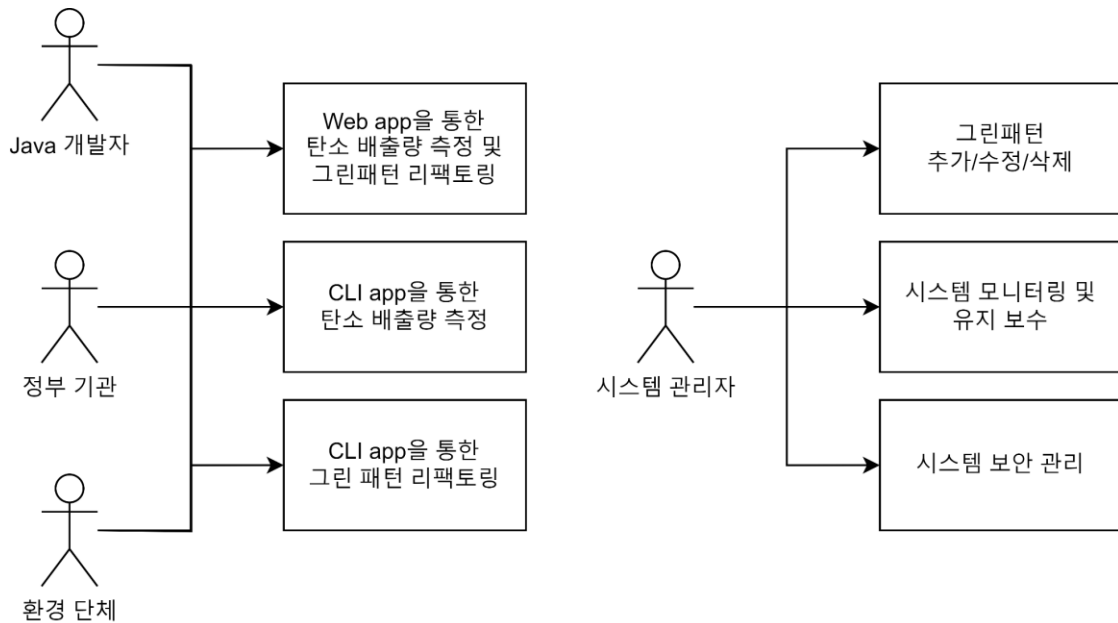
Actor	<ul style="list-style-type: none"> ● Java 개발자 ● 정부 기관 ● 환경 단체
Description	사용자가 CLI app을 통해 Java 프로젝트 단위의 코드를 제출하면 각 .java 파일에 대한 그라운드 리팩토링 결과를 제시한다. 파일마다의 리팩토링 결과를 사용자가 검토한 후 실제 적용 여부를 결정할 수 있다.
Normal Course	<ol style="list-style-type: none"> 1. Java 개발자가 CLI 환경에서 프로젝트 루트 디렉토리에 접근한다 2. 그라운드 리팩토링 명령어를 입력한다 (예시) <code>`earthsaver --green .`</code> 3. 코드가 HTTP 서버로 전송된다 4. HTTP 서버가 그라운드 리팩토링을 수행하고, 그라운드 리팩토링 결과를 포함하여 200 응답을 내려준다. 5. CLI app은 응답을 바탕으로 각 파일별 리팩토링 결과를 한번에 한 파일씩 보여준다. (CLI app "less" 같은 방식으로) 6. 사용자가 파일의 리팩토링 결과를 검토하고 적용 여부를 [y/n]로 입력한다.
Precondition	<ul style="list-style-type: none"> - 사용자가 문제없이 컴파일 가능한 Java 코드 형식을 갖춘 텍스트를 제출한다.
Post Condition	<ul style="list-style-type: none"> - 사용자가 CLI app을 통해 그라운드 리팩토링 결과를 확인할 수 있다.
Assumptions	<ul style="list-style-type: none"> - 사용자가 linux, macOS 환경에서 zsh, bash 등의 셸을 사용한다.

[Use case 4: 시스템 관리자의 그라운드 추가/수정/삭제]

Actor	<ul style="list-style-type: none"> ● 시스템 관리자
Description	시스템 관리자가 그라운드의 유지보수 혹은 개선을 위해 추가/수정/삭제 등의 action을 할 수 있어야 한다.
Normal Course	<ol style="list-style-type: none"> 1. 관리자가 추가/수정/삭제하고자 하는 그라운드에 대해 구현체의 변경이 필요한 경우 구현체를 추가/수정/삭제한다. 2. 지구구조대 시스템의 그라운드 configuration 파일에서 필요한 변경을 반영한다. 3. 유닛테스트로 검증 후 시스템을 배포한다.
Precondition	<ul style="list-style-type: none"> - 그라운드 리팩토링 시스템이 손쉽게 추가/수정/삭제할 수 있는 configurable한 구조로 개발되어야 한다.

Post Condition	- 사용자가 업데이트된 그린패턴으로 리팩토링을 제안받을 수 있게 된다
Assumptions	- configuration 파일은 Github을 통해 code repository를 통해 관리된다. 향후 필요하다면 Central dogma같은 service configuration repository와 연동될 수 있다

3.2.2. Use Case Diagram



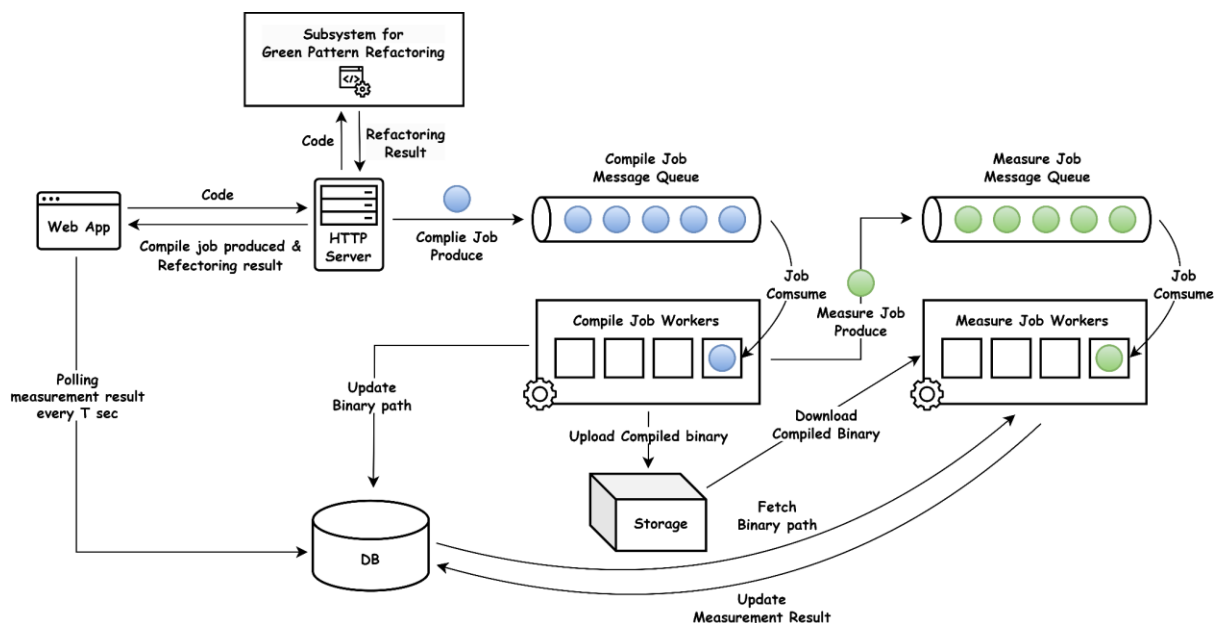
3.2.3. Data Dictionary

[Firestore collection: job]

Field name	Type	Description
id	UUID	job을 구분하기 위한 UUID로서 message queue에는 해당 job id만을 포함하는 message가 있고 그걸 바탕으로 DB를 조회한다
code_path	string	storage상에 저장된 Java code의 path. Compile worker에서 코드를 다운받기 위해 사용된다.
binary_path	string	storage상에 저장된 compiled Java binary의 path. Measure w

		orker에서 실행하려는 binary를 다운받기 위해 사용된다
status	Enum	COMPILE_ENQUEUED COMPILING MEASURE_ENQUEUED MEASURING DONE의 5가지 state를 표현한다. front-end에서 DB를 polling하며 해당 값을 확인하여 status에 맞는 적절한 display를 한다
carbon_emission	float	Measure job의 결과인 탄소 배출량 측정 결과가 여기에 저장된다. C K g 단위

3.2.4. Data Flow Diagram



3.3. Performance Requirements

3.3.1. Static Numerical Requirement

- Web app 접근 자체는 동시에 최대 100명의 사용자를 지원한다.
- Web app과 CLI app을 통해 코드를 제출함으로써 트리거되는 '탄소 배출량 측정' 및 '그린패턴 리팩토링' 기능은 동시에 최대 10명의 사용자를 지원한다.

3.3.2. Dynamic Numerical Requirement

- Web app과 CLI app에서 HTTP 요청을 통해 코드를 제출하고 job이 생성되어 HTTP 응답을 돌려받는 데 걸리는 시간은 3초 이내여야 한다.
- Compile job 실행에 걸리는 시간은 프로그램 크기에 따라 다르겠지만 최대 30초 이내여야 한다.
- Measure job 실행에 걸리는 시간은 프로그램 실행 시간에 따라 다르겠지만 최대 1분 이내여야 한다.
- Web app과 CLI app에서 job status를 확인하기 위해 polling하는 주기는 5초이다.

3.4. Logical Database Requirements

- 시스템은 job 정보를 저장하기 위해 Firestore NoSQL key value database를 사용한다.
- front-end에서 polling을 통해 job의 처리 상태와 최종 결과를 확인하기 위해 Firebase에서 제공하는 SDK를 이용한다.

3.5. Design Constraints

3.5.1. Standard Development Tools

- Web app은 React, Figma를 CLI app은 python을 사용해 개발된다.
- 백엔드 시스템은 각 서브시스템 별로 가장 적절하다고 생각되는 dev tool을 사용하여 개발된다. (3.7 참고)
- DB는 단순한 데이터 저장을 위하여 Firestore (혹은 이와 비슷한 managed key-value DB)를 사용한다.

3.5.2. Standard Compliance

- 시스템은 HTML, CSS, js standard style 을 따른다.
- UpperCamelCase를 따르는 식별자는 다음과 같다.
클래스, 구조체, 프로토콜, 익스텐션, 열거형
- LowerCamelCase를 따르는 식별자는 다음과 같다.
변수형, 상수, 함수, 속성, 메소드, 파라미터
- JS의 경우 Prettier를 이용한 코드 포맷, ESLint를 이용한 코드 품질 개선을 개발 및 유지 보수에 적극 활용한다.

3.5.3. Physical Constraints

- 본 시스템은 탄소 배출량 측정 결과를 저장하기 위해 Firestore를 사용하여 필요한 데이터를 데이터베이스에 저장할 수 있어야 한다.
- 웹 사이트를 통한 코드 분석은 간단한 자바코드를 대상으로 하므로 pc, laptop에서의 구동을 요구하고 CLI를 통한 코드 분석은 MacOS, Linux를 필요로 한다.

3.6. Software System Attributes

3.6.1. Reliability

- 사용자의 코드를 분석하고 리팩토링하는 과정에서 발생할 수 있는 오류나 예외 상황에 대비하여 적절한 예외 처리 기능을 구현한다. 이를 통해 시스템이 안정적으로 동작할 수 있도록 한다.
- 사용자의 악의적인 코드 제출로 인해 시스템이 망가지지 않도록 악의적인 코드 탐지 및 시스템 복구 기능을 구현한다.

3.6.2. Availability

- Web app으로 접근하는 사용자는 동시에 최대 100명까지 수용할 수 있어야 한다.
- 사용자가 코드를 제출함으로써 실행되는 탄소 배출량 측정 및 그에 따른 리팩토링 기능은 동시에 최대 10명까지 지원할 수 있어야 한다.

- 사용자는 CLI app을 통해서 프로젝트 단위로 코드를 제출하여 탄소 배출량 측정 및 그린패턴 리팩토링 제안을 받을 수 있어야 한다.

3.6.3. Security

- 사용자의 코드를 저장하지 않고, 그린패턴 리팩토링된 코드 역시 1회성으로만 제공하여 사용자 코드가 외부에 유출되는 것을 방지한다.
- 사용자가 제출한 Java 코드에 악의적인 코드가 포함될 수 있으므로, 사용자의 input 코드를 실행하는 환경을 독립된 컨테이너로 한정하여 리소스를 격리한다. 이를 통해 악의적인 코드가 시스템에 영향을 미치는 것을 방지한다.

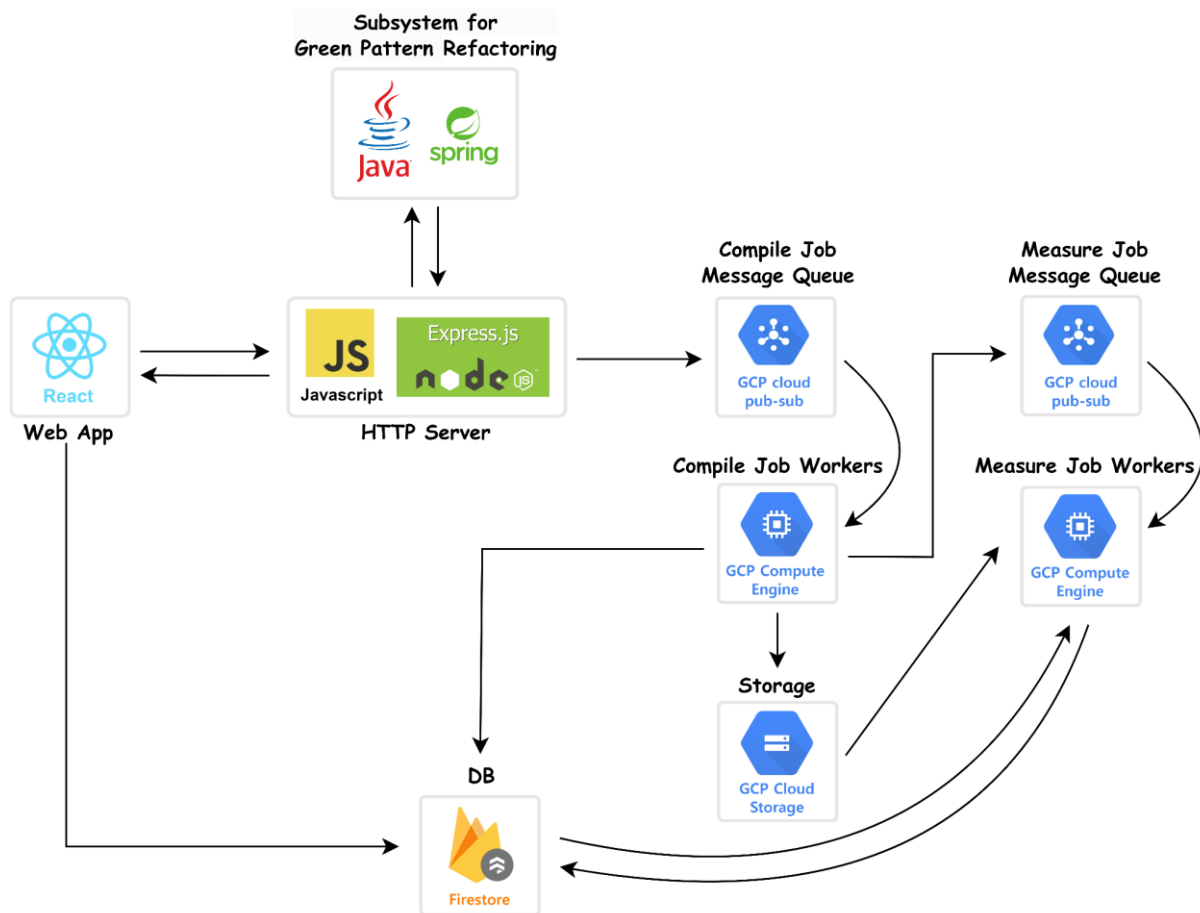
3.6.4. Maintainability

- 먼저 Front-end는 CLI app, Web app으로, Back-end 시스템은 각 서버 시스템별로 관심사를 분리하여 각 영역을 독립적으로 배포하고 업데이트하여 안정적인 유지보수를 돕는다.
- Front-end (Web app, CLI app)
 - Web app의 경우 React의 컴포넌트 기반 아키텍처를 채택하여 모듈화된 코드를 작성한다.
 - 컴포넌트의 재사용성을 고려하여 작은 단위로 나누고, 컴포넌트 간의 의존성을 최소화하여 유연하고 관리하기 쉬운 코드를 작성한다.
- Back-end
 - HTTP 서버의 경우 Node.js의 Express 프레임워크를 활용하여 node.js의 api를 단순화하고 코드를 간결하고 직관적으로 만든다. Express의 모듈화된 미들웨어와 라우팅을 통해 코드를 구성한다.
 - 그린패턴 리팩토링 서비스 향후 추가적인 그린패턴을 반영할 수 있는 확장성을 고려한다.
- Worker failure에 대한 tolerance
 - Compile/measure job worker가 동작 중 예상치 못한 이유로 실패하는 job failure가 발생하더라도, 다른 worker가 손쉽게 message queue에서 다시 job을 consume할 수 있으므로 failure에 tolerant한 구조로서 관리 비용을 줄일 수 있다.

3.6.5. Portability

- Web app의 경우 Javascript가 활성화된 인터넷 브라우저(Chrome, Edge, Safari등)에서 동작할 수 있어야 한다.
- Linux, MacOS의 bash, zsh등의 보편적인 CLI 환경에서 동작할 수 있어야 한다.

3.7. System Architecture



3.8. System Evolution

3.8.1. Expected Change of User Requirements

- 그린패턴 리팩토링 로직은 추가될 수 있으며, 치명적 결함이 발견될 경우 쉽게 제거할 수 있어야 한다
- 프로그래밍 언어(Java)나 빌드도구(Gradle)에 대한 확장성이 요구될 수 있으므로 이를 고려하여 개발해야 한다
- 사용자가 도구를 직접 사용하는 것 외에도, Github 마켓플레이스를 통해 제공한다면 CI로 자동화하는 것도 기대할 수 있다

4. Appendixes

4.1. Green pattern

그린 패턴의 목적을 달성하기 위한 탄소 배출량을 줄이는데 가장 효과적인 방법은 알고리즘의 효율성을 높이는 것이다. 일반적으로 실행 속도의 향상이 효율성의 향상을 의미하지만, 알고리즘 최적화의 일부로는 메모리 최소화도 포함된다. 메모리로부터의 전력 소비는 주로 사용 중인 실제 메모리가 아닌 사용 가능한 메모리에 따라 달라지며, 사용 가능한 메모리는 일반적으로 알고리즘의 한 단계에 필요한 최대 메모리이다. 특히 전력을 많이 소비하는 알고리즘이 실행되는

횡수를 제한하는 방법도 고려할 수 있는데 파라미터 미세 조정을 필요한 최소한으로 제한하고 다버깅을 위해 소규모 여제를 구축하는 것은 하나의 실용적인 방안이 될 수 있다.

4.2. Software Requirements Specification

본 요구사항 명세서는 IEEE Recommend Practice for Software Requirements Specifications, IEEE-Std-830)의 형식에 따라 작성되었다.

4.3. Formula of Carbon Emission

탄소배출량은 아래 식에서 C를 통해 구해진다[1].

$$\boxtimes = \boxtimes \times \boxtimes \boxtimes \quad (1)$$

$$\begin{aligned} \Delta &= \Delta \times (\Delta_{\Delta} \times \Delta_{\Delta} \times \Delta_{\Delta} + \Delta_{\Delta} \times \Delta_{\Delta}) \times \Delta \Delta \Delta \times \Delta \Delta \times 0.001 \\ (2) \end{aligned}$$

[illegible]

4.4. Document history

Date	Version	Description	Writer
2024-05-05	v1.0.0	Product functions, System evolution, Prototyping	김영석
2024-05-05	v1.0.0	All diagrams, Document formatting, User classes and characteristics, Design and implementation constraints, Assumption and dependencies	안윤지
2024-05-05	v1.0.0	External Interfaces, Function requirements, Performance requirements, Logical DB requirements, System architecture	한승호
2024-05-05	v1.0.0	Purpose, Product perspective, Prototyping	한용준
2024-05-05	v1.0.0	Scope	홍서윤
2024-05-05	v1.0.0	Definitions, Acronyms and Abbreviations, References, Overview, Software system attributes	황수영

