[지구 구조대]

Design Specification

by TEAM 9

김영석 안윤지 한승호 한용준 홍서윤 황수영

Instructor: 이은석

TA: 김진영 최동욱 허진석 김영경

Document Date: 8 May, 2024

Faculty: Sungkyunkwan University

** Authors' name in alphabetical order*

1. 서론

1.1. Readership

본 문서는 Java Code의 탄소배출량을 측정하고 그린패턴 리팩토링을 제안하는 도구인 지구구조대 개발에 참여한 개발자, 운영자, 사용자 등의 stakeholder를 대상으로 그들이 이 문서를 활용하여 시스템을 이해하는 데에 도움을 주기 위한 목적으로 작성되었다.

1.2. Scope

본 문서는 지구구조대 시스템의 전반적인 아키텍처 및 작동 방식에서부터 각 서브시스템, 컴포넌트, 오브젝트 단위의 구체적인 설명까지를 포함한다. 특히 탄소 배출량 측정과 그린 패턴 리팩토링이라는 두가지 기능을 web app과 cli app에서 제공하기 위한 시스템의 객체지향적 설계를 그 범위로 한다.

1.3. Objective

시스템을 구현하는 개발자로 하여금 의사결정을 필요로 하는 사항을 이 문서 작성 과정에서 최대한 해소함으로써 구현에 소요되는 시간과 비용을 예측 가능하게 될 것을 기대한다. 또한 본 문서가 추후의 논의, 유지보수, 운영 과정에서의 기준으로 활용할 예정이다.

1.4. Document Structure

1) 서론: 본 문서의 목적, 예상 독자 및 문서의 구조에 대해 설명한다
2) 소개: 본 문서를 작성하는데 사용된 도구들과 다이어그램들에 대해 설명한다
3) 시스템 아키텍처 - Overall: 시스템의 전체적인 구조를 System Diagram, Use-case Diagram, Sequence Diagram을 이용하여 서술한다.
4) 시스템 아키텍처 - Frontend: Frontend 시스템의 구조를 서술한다.
5) 시스템 아키텍처 - Backend: Backend 시스템의 구조를 서술한다.
6) 프로토콜 디자인: 여러 서브시스템간 커뮤니케이션에 필요한 프로토콜에 대한 약속을 서술한다.
7) 데이터베이스 디자인: 시스템에서 사용될 key-value database의 collection schema를 서술한다.
8) 테스팅 계획: 시스템을 테스트하기 위한 계획을 서술한다.
9) 개발 계획: 시스템 개발 과정에서의 협업 방법이나 기술 스택 등에 관련된 계획을 서술한다.

2. 소개

2.1. Objectives

이번 챕터에서는 독자의 본 문서에 대한 이해를 돕기 위해 본 문서에 사용된 diagram에 대해 설명하고, 프로젝트의 범위와 reference를 다룬다.

2.2. Applied Diagrams

2.2.1. Used Tools

온라인 기반의 다이어그램 작성 도구인 Draw.io를 활용하여 간단하고 직관적인 사용자 인터페이스를 그릴 수 있다.

2.2.2. Use Case Diagram

Use case Diagram은 시스템에서 제공해야 하는 기능이나 서비스를 명세화한 다이어그램이다.

2.2.3. Class Diagram

Class Diagram은 시스템을 구성하는 클래스, 그 속성, 기능 및 객체들 간의 관계를 표현하여 시스템의 정적인 부분을 보여준다.

2.2.4. Context Diagram

Context Diagram은 Data Flow Diagram에서 가장 상위에 있으며, 시스템과 외부 요소 간의 상호 작용을 개략적으로 보여준다.

2.2.5. Entity Relationship Diagram

ER Diagram은 구조화된 데이터와 그들 간의 관계를 직관적으로 표현하는 다이어그램으로, 현실 세계의 요구사항을 반영하여 데이터베이스 설계에 활용된다.

2.3. Project Scope

본 문서에서 설계하는 java 코드 탄소 배출량 측정 및 그린패턴 리팩토링 웹 개발은 사용자들이 문서화를 진행하고 개발을 해봄으로써 실제 산업에서 진행되는 개발 process를 경험하도록 한다.

2.4. 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

[4] https://github.com/skkuse/2023fall_41class_team2

3. 시스템 아키텍처 - Overall

3.1. Objectives

이 챕터에서는 두가지 프론트엔드(Web/CLI app) 시스템과 백엔드 시스템 전체를 아우르는 하나의 시스템의 아키텍처에 대해 이야기한다.

3.2. System Organization

시스템은 크게 프론트엔드와 백엔드로 나뉘어있다. 프론트엔드는 유저로부터 코드를 다양한 형태로 입력받기 위한 목적을 가진 시스템이다. 백엔드는 입력받은 코드에 대해 탄소배출량 측정과 그린패턴 리팩토링이라는 두가지 기능을 수행하기 위한 목적을 가진 시스템이다. 각 시스템 내의 세부적인 동작에 대해서는 챕터 4,5에서 상세히 다룬다. 여기에서는 거시적인 관점으로 프론트엔드와 백엔드 시스템을 바라보려고 한다.

먼저 CLI app과 Web app 모두 동일하게 프론트엔드라면 백엔드와 상호작용하게 되는데, 이때 크게 두가지 방법이 사용된다.

첫째로 HTTP를 사용하여 유저가 제출한 코드 파일을 byte 형태로 HTTP 요청의 body에 포함하여 백엔드로 보낼 수 있다. 백엔드는 상대적으로 빠르게 1초 내외로 처리 가능한 결과(그린패턴 리팩토링, compile job produce) 들을 담아 HTTP 응답에 포함시켜 프론트엔드로 보내게 된다.

둘째로 compile job이나 measure job의 경우 처리에 수초에서 수십초까지 소요되는 것이 예상되어, job은 message queue를 사용해 관리하고, job처리는 message queue를 바라보고 있는 worker들에게 위임하는 producer-consumer pattern을 채택했다. Worker의 job 처리 결과는 db에서 관리하고, 프론트엔드는 db를 주기적으로 polling하여 탄소배출량 측정 결과를 받아갈 수 있게 했다. 이렇게 함으로써 HTTP 요청에 대한 응답은 작업과 무관하게 최대한 빠르게 돌려줌으로써 탄소배출량 측정 작업이 진행되는 동안 프론트엔드에서 더 유연한 대응을 가능하게 했다.

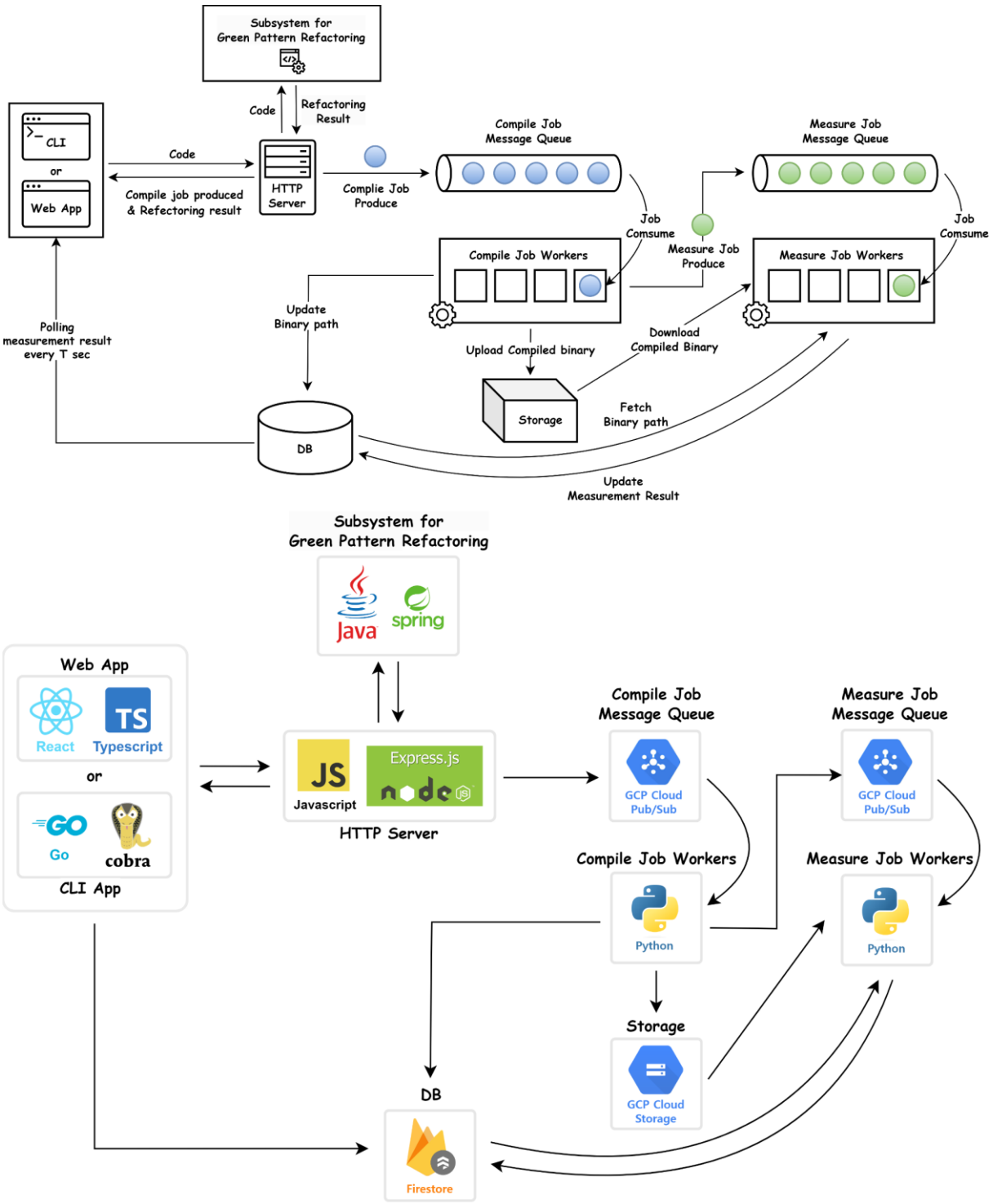


Figure 3.1: Overall System Architecture

3.2.1. System Diagram

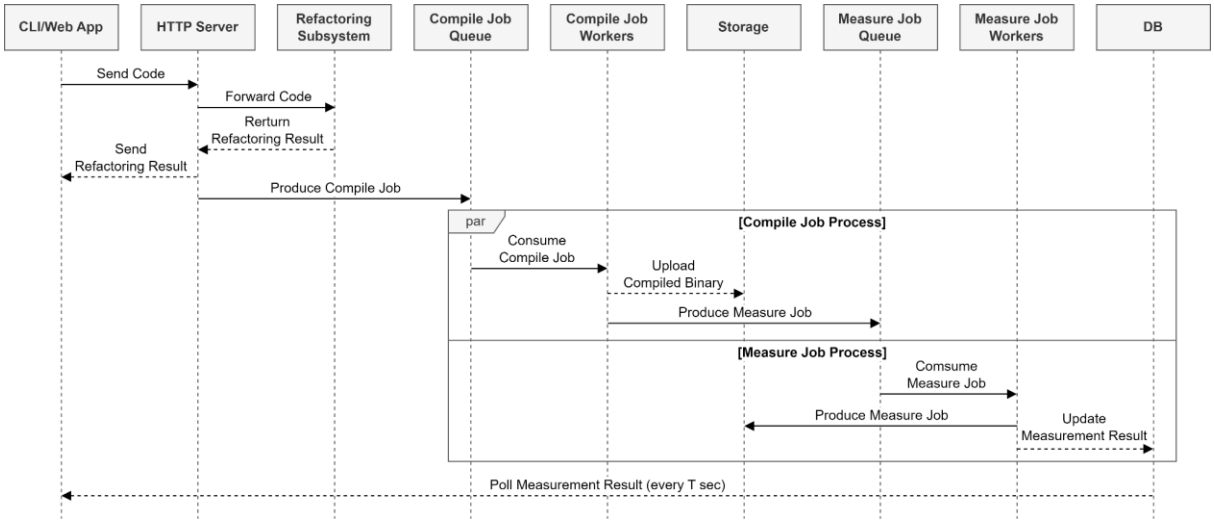


Figure 3.2 : Sequence diagram

3.3. Use case diagram

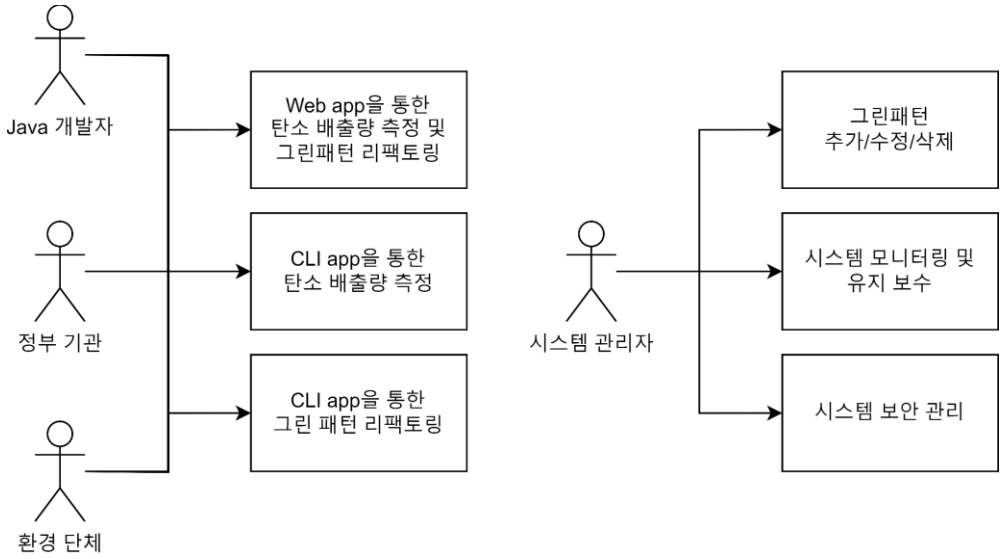


Figure 3.3 : Use case diagram

4. 시스템 아키텍처 - Frontend

4.1. Objectives

이 파트는 front-end 시스템 구조에 대해 기술한다.

4.1.1. Overall Architecture

본 시스템에서 Frontend는 Web browser을 통해 접근할 수 있는 web app과 CLI app 두가지로 구성되어 사용자는 자신의 환경과 목적에 맞게 선택하여 사용할 수 있다. UI에 직접적인 영향을 주는 기능은 각자 구현되며, 나머지 중복되는 기능은 재사용 가능하게 구현된다.

4.2. Components

4.2.1. Classes

4.2.1.1. Carbon Class

4.2.1.1.1. Attributes

- carbonEmission : 측정된 탄소배출량이다.
- carbonCar : 자동차가 측정된 탄소배출량을 배출하며 이동할 수 있는 거리
- carbonPlane : 비행기가 측정된 탄소배출량을 배출하며 이동할 수 있는 거리
- carbonTree : 측정된 탄소배출량에 대응되는 나무의 수

4.2.1.1.2. Methods

- setCarbonEmission : 코드를 입력받아 탄소배출량을 반환
- emissionConvert : 탄소배출량을 입력받아 다른 단위로 환산

4.2.1.2. Code Class

4.2.1.2.1. Attributes

- decodedRefactoredCode : 리팩토링된 그린코드
- refactoredCode : base64로 인코딩 된 그린코드

4.2.1.2.2. Methods

- codeRefactoring : 유저코드를 백엔드로 보내 그린코드를 받아오는 함수
- decodeCode : base64로 인코딩된 그린코드를 입력받아 디코딩하는 함수
- getDiff : 기존 코드와 리팩토링 이후의 코드를 비교하는 함수

4.2.1.3. MainPage Class

4.2.1.3.1. Attributes

- javaCode : 유저에게 입력받은 자바 코드
- encodedCode : base64로 인코딩된 유저 코드

4.2.1.3.2. Methods

- checkStatus : 현재 status를 Field에 display하는 함수
- displayAll : 작업이 끝난 후 모든 결과 값을 보여주는 함수
- encodeJavaCode : 유저 코드를 base64로 인코딩하는 함수
- submitCode : setCarbonEmission과 codeRefactoring을 포함하는 함수

4.2.2. Diagrams

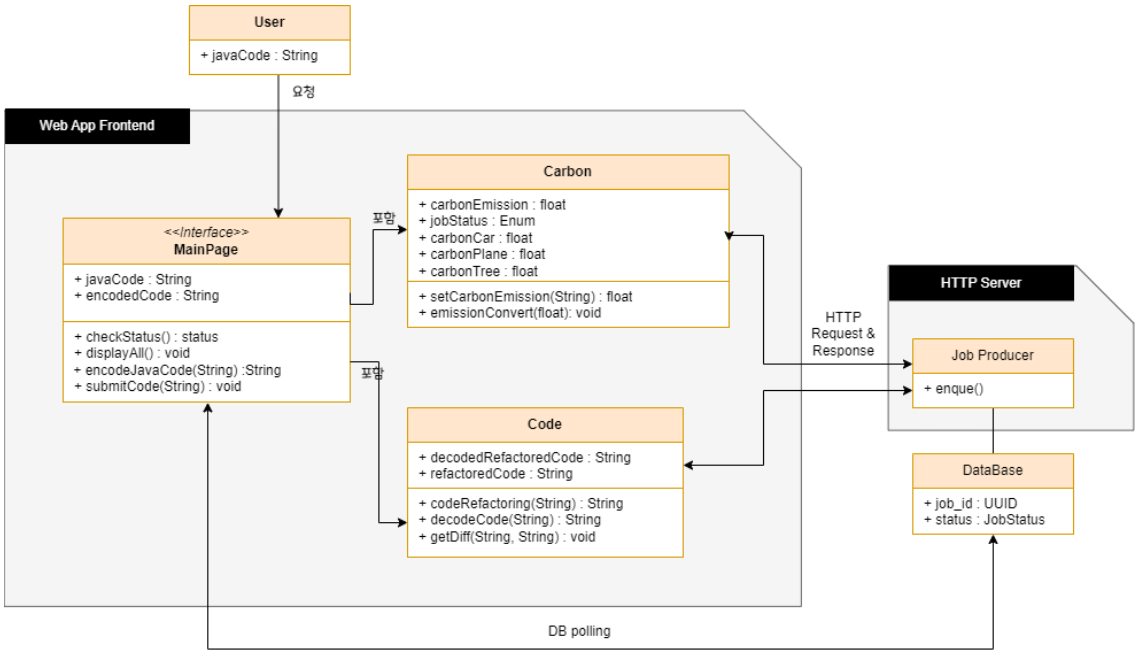


Figure 4.1 : Class Diagram for Web App

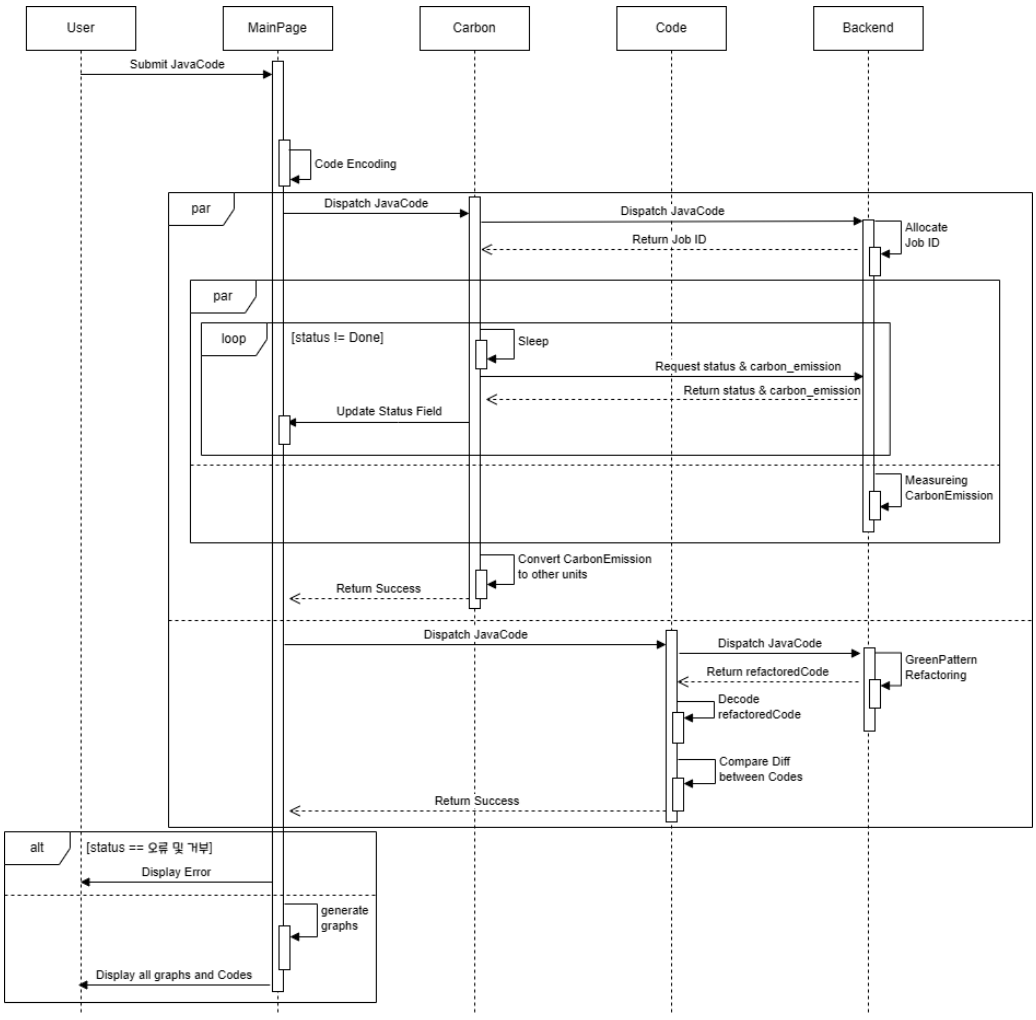


Figure 4.2 : Sequence Diagram for Web App

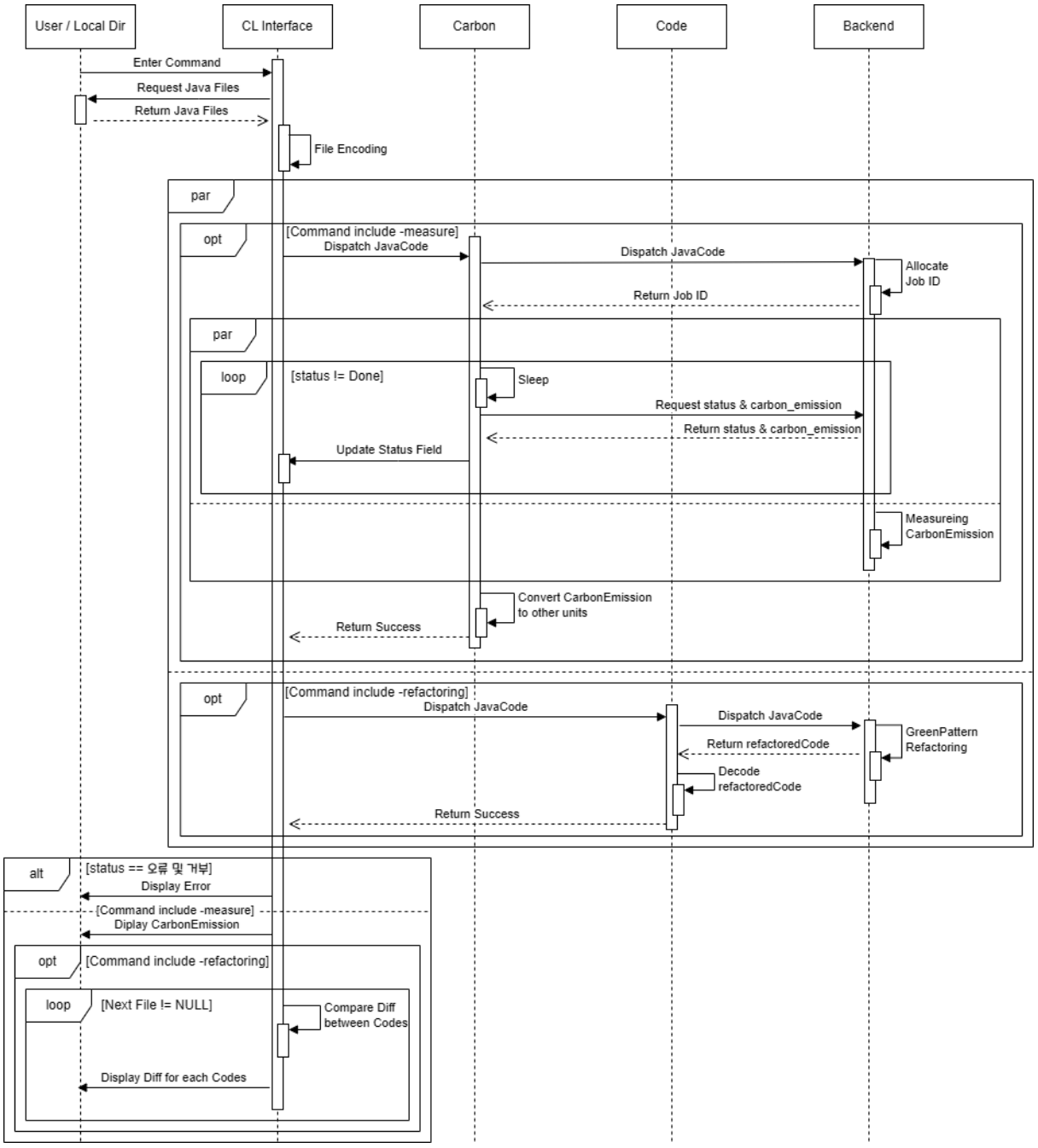


Figure 4.3 : Sequence Diagram for CLI App

5. 시스템 아키텍처 - Backend

5.1. Objectives

이 파트는 back-end 시스템의 구조에 대하여 기술한다.

5.2. Overall Architecture

전체적인 백엔드 서버 구조는 다음과 같다.

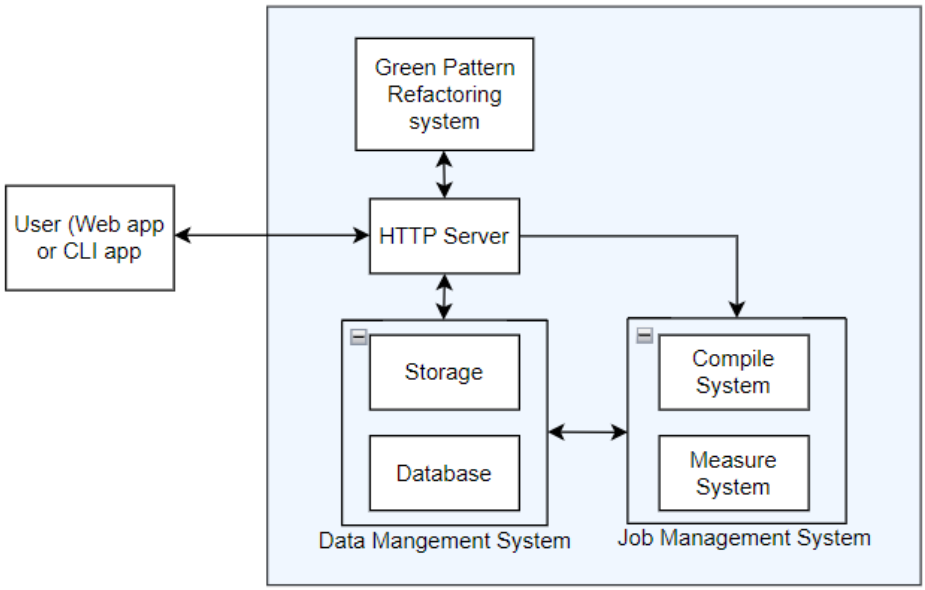


Figure 5.1 Overall architecture

User(Web app 혹은 CLI app)는 HTTP Server로 Java코드를 보내고 이에 대한 그린패턴 리팩토링 결과, job의 상태를 HTTP Server에게 요청하여 받을 수 있다. HTTP Server는 user를 통해 받은 Java코드에 대한 처리를 위해 Green Pattern Refactoring system 과 상호작용, 그리고 job을 생성하며 job에 대한 측정을 위해 Job management system과 상호작용한다. job들은 Java code에 대한 작업을 의미한다. Job Management System에서는 job에 대해 즉 user가 전송한 Java 코드 컴파일 측정을 실행한다. 그 후 결과를 Data Management System에 전달하여 저장한다.

5.3. Subcomponents

5.3.1. HTTP Server

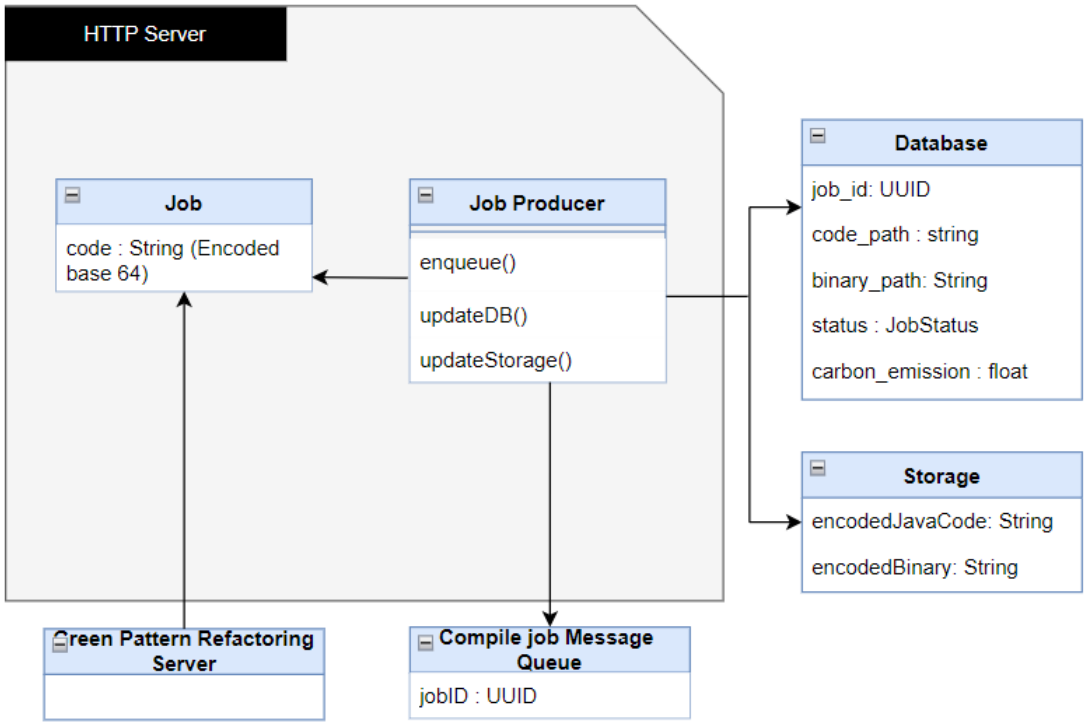


Figure 5.2 Class Diagram - HTTP Server

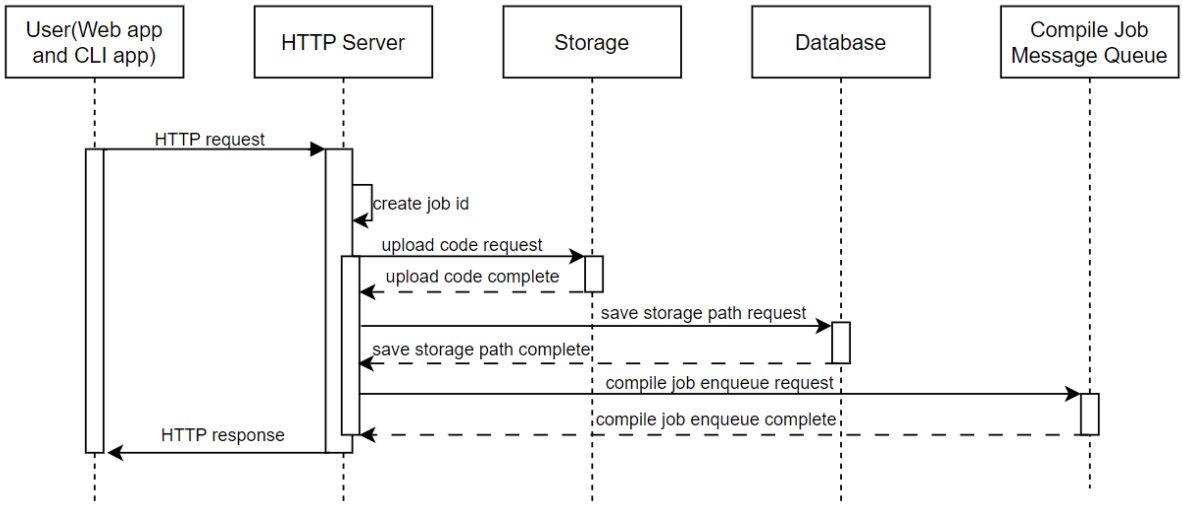


Figure 5.3.1 Sequence Diagram - HTTP Server(1)

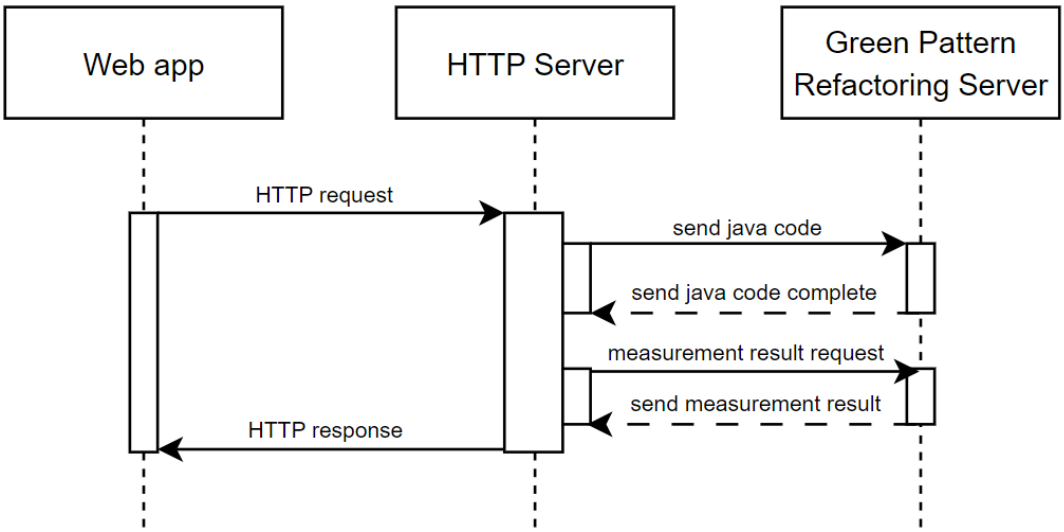


Figure 5.3.2 Sequence Diagram - HTTP Server(2)

5.3.2. Green Pattern Refactoring Server

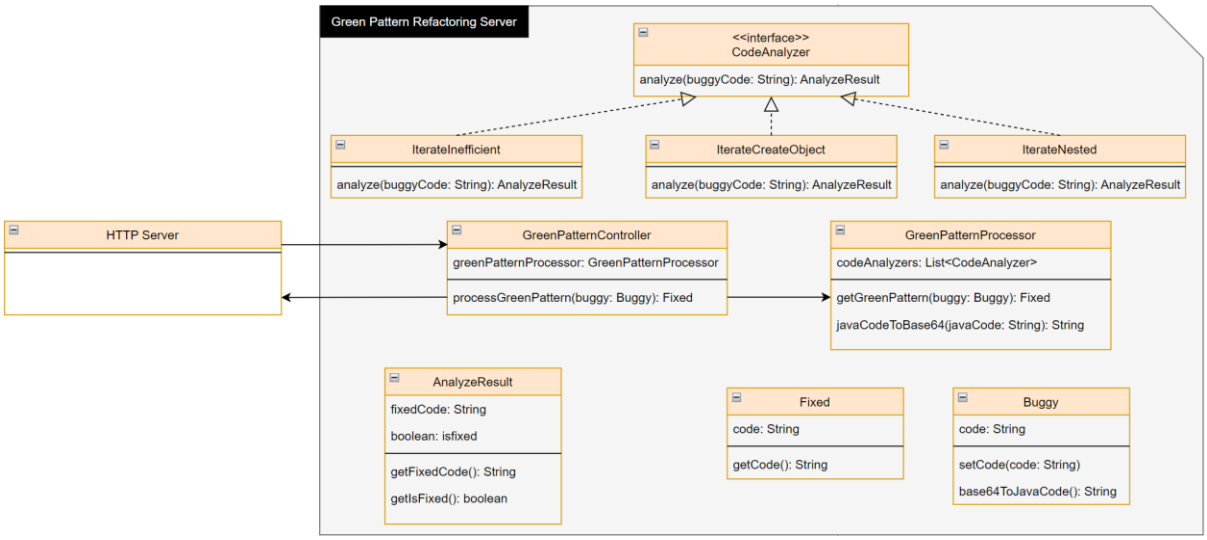


Figure 5.4 Class Diagram - Green Pattern Refactoring Server

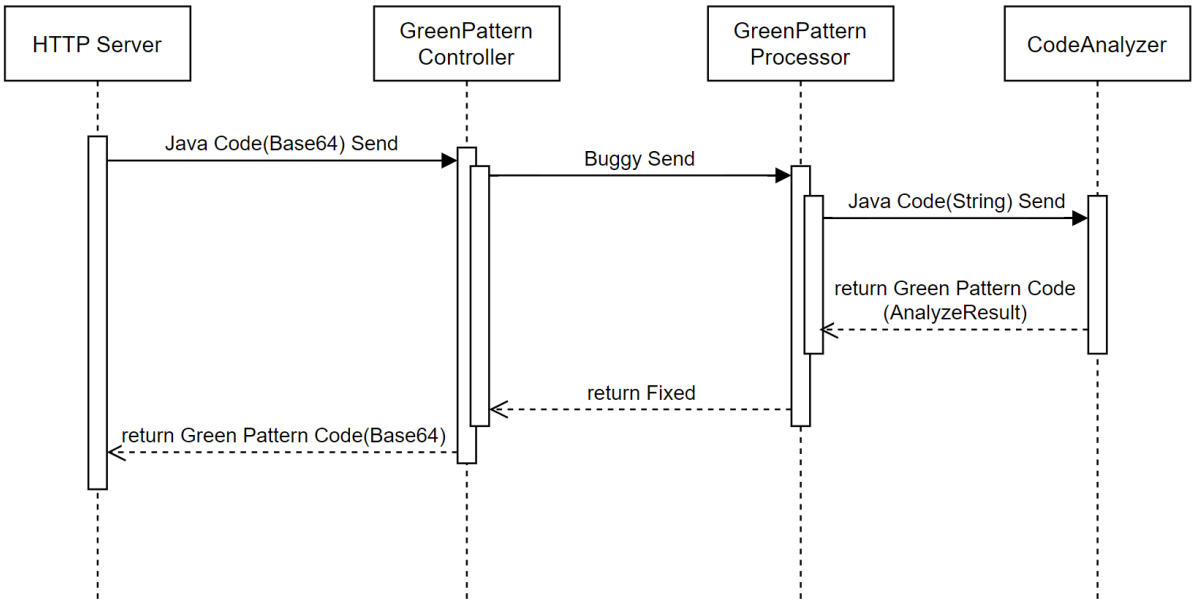


Figure 5.5 Sequence Diagram - Green Pattern Refactoring Server

5.3.3. Job Management System

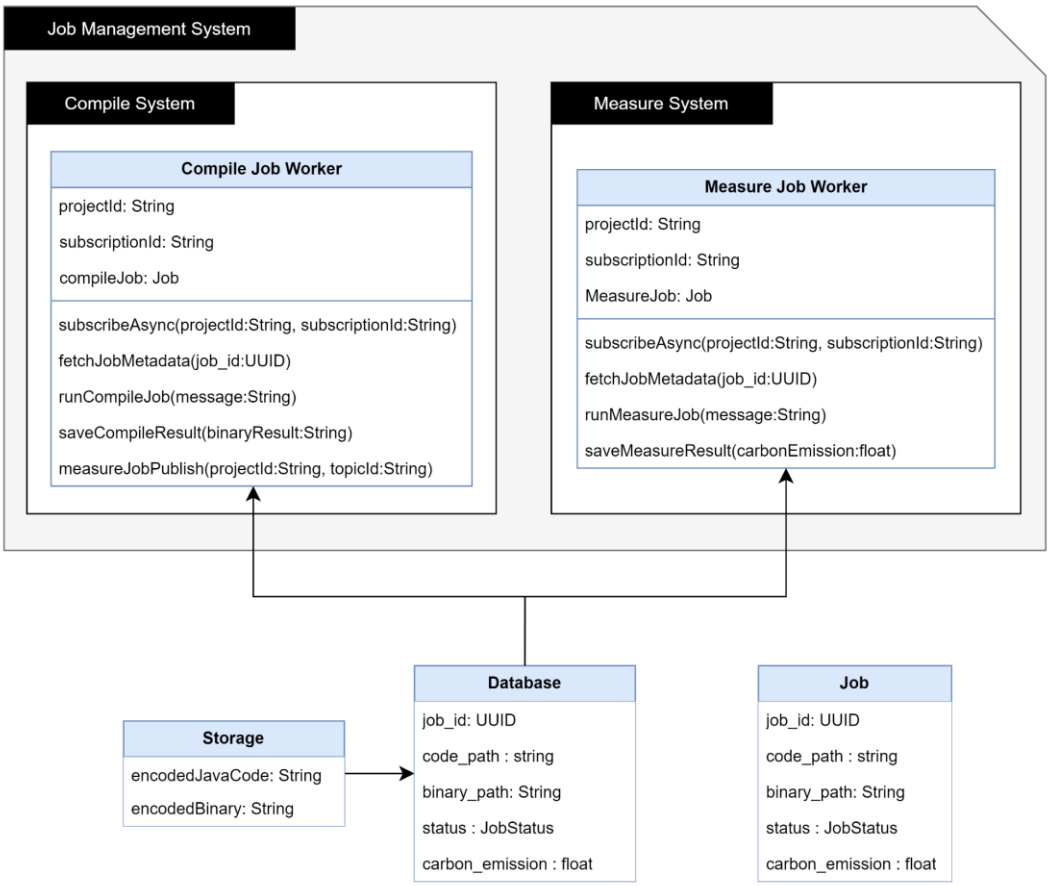


Figure 5.6 Class Diagram - Job Management System

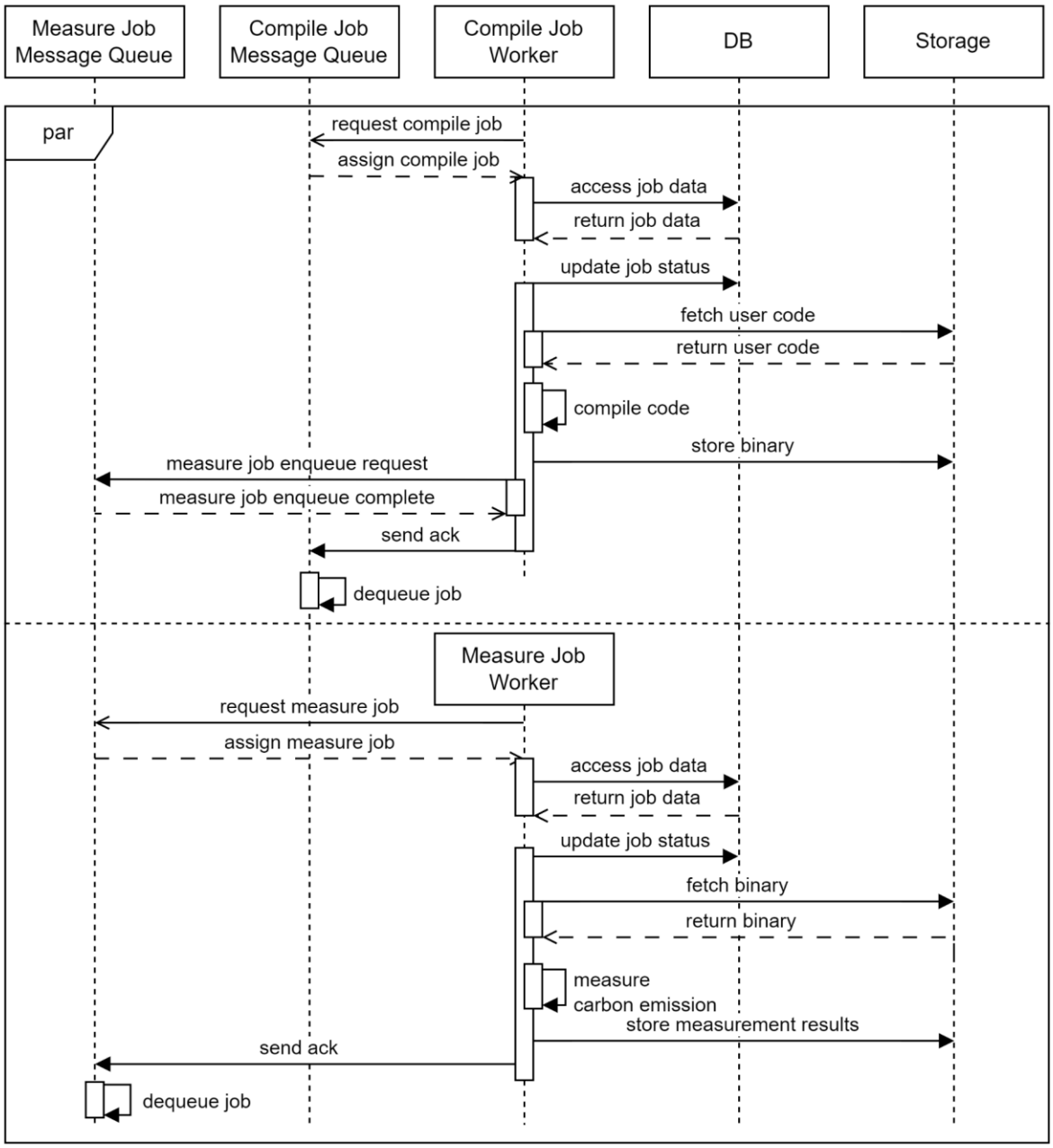


Figure 5.7 Sequence Diagram - Job Management System

6. 프로토콜 디자인

6.1. Objectives

이 챕터에서는 front-end 애플리케이션과 back-end 서버가 어떤 프로토콜로 상호작용하는 지를 기술하고 각 인터페이스를 정의한다.

6.2. JSON

JSON(JavaScript Object Notation)이란 Javascript 객체 문법으로 구조화된 데이터를 표현하기 위한 문자 기반의 표준 포맷이다. 백엔드와 프론트엔드는 데이터를 JSON 포맷으로 주고받는다.

6.3. HTTP

본 애플리케이션은 웹위에서 동작한다. HTTP는 W3상에서 정보를 주고 받을 수 있는 요청/응답 프로토콜로, 본 앱은 해당 프로토콜을 따른다.

6.4. File representation

CLI Application은 Java코드의 character set을 UTF-8로 인코딩 후 base64로 인코딩하여 전송한다. Web application에서 back-end로 Java 코드를 전송할 때, CLI Application과의 인터페이스 통합을 위해, 동일한 방식으로 코드를 전송한다. back-end 단에서는, 받은 문자열을 바이트 배열로 변환후 바이트스트림으로 파일을 저장한다.

6.5. Interface

6.5.1. Between Web/CLI App and HTTP Server

6.5.1.1. Send Java code for green pattern refactoring

6.5.1.1.1. request

Attribute	Detail
Protocol	HTTP
Content-type	JSON
Endpoint	/user/refactoring_code
Method	POST

Request body	type RequestBody = {
	files: File[];
	}
	type File = {
	fileRelativePath: string;
	fileB64Encoded: string; }

6.5.1.1.2. response

Attribute	Detail
Protocol	HTTP
Content-type	JSON

Status Code	200 ok

Request body	type RequestBody = {
	files: File[];
	}
	type File = {
	fileRelativePath: string;
	fileGreenB64Encoded: string;
	}

6.5.1.2. Send Java code for measure

6.5.1.2.1. request

Attribute	Detail
Protocol	HTTP
Content-type	JSON
Endpoint	/user/measure_carbonEmission
Method	POST

Request body	type RequestBody = {
	files : File[];
	}
	type File = {
	fileRelativePath: string;
	fileB64Encoded: string;
	}

6.5.1.2.2. response

Attribute	Detail
Protocol	HTTP
Content-type	JSON
Status Code	200 ok
Response body	type ResponseBody = {

	job_id: string; }

6.5.2. Between HTTP Server and Green Pattern Refactoring Server

6.5.2.1. Send Java code

6.5.2.1.1. request

Attribute	Detail
Protocol	HTTP
Content-type	application/json
Endpoint	/green_pattern
Method	POST

Request body	type RequestBody = {
	files: File[];
	}
	type File = {
	fileRelativePath: string;
	fileB64Encoded: string; }

6.5.2.1.2. response

Attribute	Detail
Protocol	HTTP
Content-type	application/json
Status Code	200 ok

Request body	type RequestBody = {
	files: File[];
	}
	type File = {
	fileRelativePath: string;
	fileB64Encoded: string; }

6.5.3. Between Web/CLI App and DB

6.5.3.1. DB polling for get Job Status & Carbon emission

6.5.3.1.1. request

Attribute	Detail
Protocol	HTTP
Content-type	JSON
Endpoint	/status_request
Method	POST

Request body	type ResponseBody = {
	job_id: string;
	}

6.5.3.1.2. response

Attribute	Detail
Protocol	HTTP
Content-type	JSON
Status Code	200 ok

Response body	type ResponseBody = {
	job_status: Status;
	carbon_emission: number;
	}
	
	type Status = “COMPILE_ENQUEUED” | “COMPILING” | “MEASURE_ENQUEUED” | “MEASURING” | “DONE” | “ERROR”

7. 데이터베이스 디자인

7.1. Objectives

이 챕터에서는 job의 구체적인 정보를 저장하기 위한 Firstore collection의 스키마에 대해 설명한다.

7.2. Collection schema

[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. code_path 하위에 main.java 단일 파일이 있거나, gradle로 빌드 가능한 형태의 프로젝트가 존재해야 한다. ex. gs://earth-saver/jobs/[JOB_ID]/code
binary_path	string	storage상에 저장된 compiled Java binary의 path. Measure worker에서 실행하려는 binary를 다운받기 위해 사용된다.
status	Enum	COMPILE_ENQUEUED | COMPILING | MEASURE_ENQUEUED | MEASURING | DONE | ERROR의 6가지 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. CKg 단위.

8. 테스팅 계획

8.1. Objectives

이 챕터에서는 시스템을 테스트하기 위한 계획에 대해 설명한다. 시스템 테스트는 단위 테스트, 통합 테스트, 유저 테스트로 나누어 단계적으로 진행한다.

8.2. Testing Policy

8.2.1. 단위 테스트 (Unit test)

단위 테스트를 통해 시스템의 각 부분(임의의 테스트 가능한 규모)이 의도대로 동작하는지 확인한다. 본 시스템의 경우 여러 서브시스템으로 나뉘어 있는데, 각 서브시스템 안에서도 핵심적인 로직을 담당하는 컴포넌트 위주로 단위 테스트를 진행할 계획이다.

단위 테스트 실행을 위한 자동화된 테스트 코드를 작성하고, 핵심 로직 외의 네트워크 call과 같은 나머지 로직은 같은 interface를 따르는 mock 객체로 대체하여 핵심 로직 테스트에만 집중할 수 있게 한다.

8.2.2. 통합 테스트 (Integration test)

단위 테스트에서 검증된 시스템의 각 부분이 통합되어 동작할 때는, 단위 테스트에서 검증하지 못한 부분에서 문제가 생길 수 있다. 통합 테스트는 이런 부분을 검증하는 데에 초점을 둔다. 특히 단위 테스트에서 mock으로 대체된 로직들이 실 환경의 로직으로도 잘 동작하는지 확인하는 것이 중요하겠다. 대표적으로 GCP의 Cloud PubSub을 message queue로서 사용하는 job worker 시스템의 경우 통합 테스트 단계에서 실 환경의 Cloud Pub Sub과 통합하여 테스트하게 된다.

통합 테스트 실행을 위해 테스트 시나리오를 사전에 정하고 자동화 가능한 부분을 최대한 스크립트화하여 테스트 실행 비용을 낮출 수 있다.

8.2.3. 유저 테스트 (User test)

실제 배포 환경에서 유저가 서비스에 처음 접근할때부터 시스템이 제공하는 기능을 성공적으로 이용하기까지 문제가 없는지 확인하기 위한 테스트이다. 기대하는 유저 테스트 시나리오를 작성하고 그것을 바탕으로 예상되는 동작이 나오는지 확인할 수 있다.

9. 개발 계획

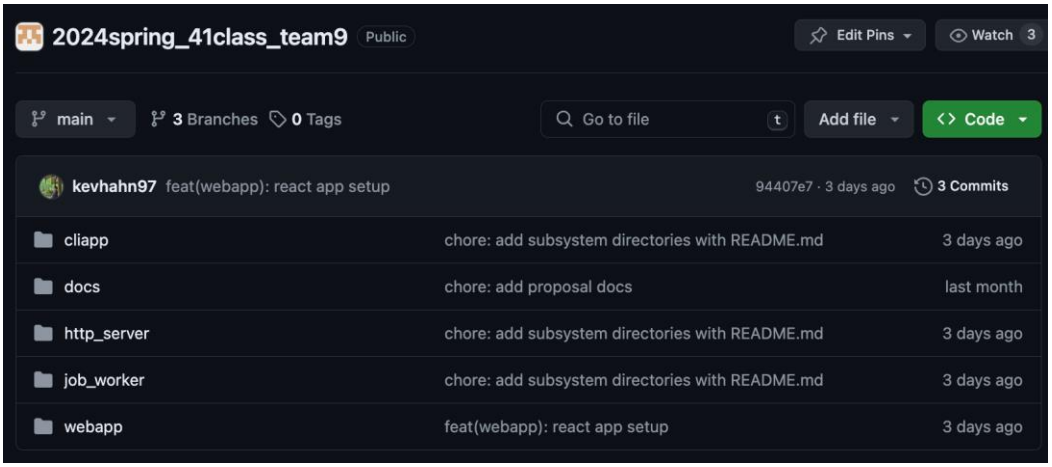
9.1. Objectives

이 챕터에서는 지구구조대 개발을 위한 여러 방면의 계획을 안내한다. 대표적으로 팀의 원활한 협업을 위한 계획, 각 서브시스템별 기술 스택, 개발 계획상의 제약사항, 가정, 의존성을 다룬다.

9.2. Collaboration Plan

9.2.1. Mono repository

하나의 github repository 안에서 여러 서브시스템의 개발/테스트/통합/배포를 관리한다. 각 서브시스템을 디렉토리로 구분하여 아래와 같은 형태를 갖는다.



9.2.2. Code Version Control: Git flow

여러 개발자가 동시에 작업하며, 서로의 작업간 원활한 리뷰와 conflict 해소를 위해 git flow 방식으로 코드 버전을 관리한다. 작업자가 자신의 작업 브랜치를 생성, 작업을 마친 뒤 작업 브랜치를 main에 머지할 수 있는 상태로 만든 뒤 Pull Request를 열게 된다. 최소 1명 이상의 리뷰어가 approve하면 main에 머지할 수 있는 정책을 두어 협업을 효율화하려고 한다.

또한 main에 머지될 때는 항상 squash and merge만 허용하는 정책을 두어, main 브랜치의 commit history가 한눈에 이해하기 쉬운 형태로 유지하려고 한다.

9.3. Subsystems Plan

9.3.1. Web app

- Tech stack: React + Typescript
- UI library: React MUI
- Deployment: Single page app으로build하여 google cloud storage를 통해static file servin g

9.3.2. CLI app

- Tech stack: Go + Cobra
- Deployment: go build binary를Web app을 통해 다운로드 받을 수 있는 형태

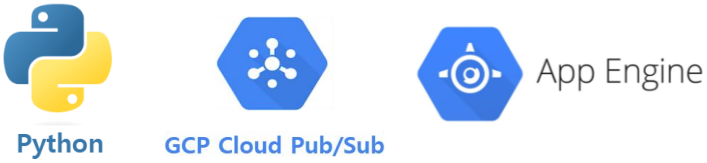
9.3.3. HTTP server

- Tech stack: node.js
- Deployment: Google App Engine (managed kubernetes cluster)

9.3.4. Green pattern refactoring server

- Tech stack: Java + Spring
- Deployment: Google App Engine (managed kubernetes cluster)

9.3.5. Job workers



- Workers: Python
- Message queues: Google Cloud Pub-Sub
- Deployment: Google App Engine

10. Appendix

10.1. Document history

Date	Version	Description	Writer
2024-05-25	v1.0.0	Frontend overall & Diagrams, Interface of Web/CLI App	김영석
2024-05-25	v1.0.0	Job Management System architecture & diagram	안윤지
2024-05-25	v1.0.0	Preface, Introduction, DB design, Testing plan, Development plan, General managing	한승호
2024-05-25	v1.0.0	Overall architecture of Backend, HTTP Server architecture & diagram, Interface of HTTP Server.	한용준
2024-05-25	v1.0.0	Preface, Introduction	홍서윤
2024-05-25	v1.0.0	Green Pattern Refactoring Server architecture & diagram	황수영