



# CODEMETER

JAVA 코드 탄소 배출량 측정도구

4조

김동한 안낙균

유지훈 임소리

조민호 조유자

# 목차

01  
Goal

02  
Overview

- User Flow
- Tech Stacks
- System Architecture

03  
Implementation

- Frontend
- Backend

04  
Project  
Management

- Role
- Workflow

05  
Green Patterns

# 01

# Goal

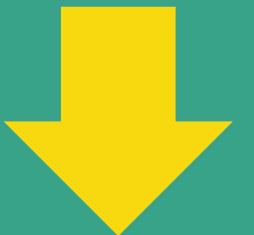
CODEMETER 프로젝트 목표



## ✓ Goal

개발자들이 본인의 코드가 환경에 미치는 영향을 파악

환경 친화적인 코드 작성 유도



지속가능한 발전에 기여

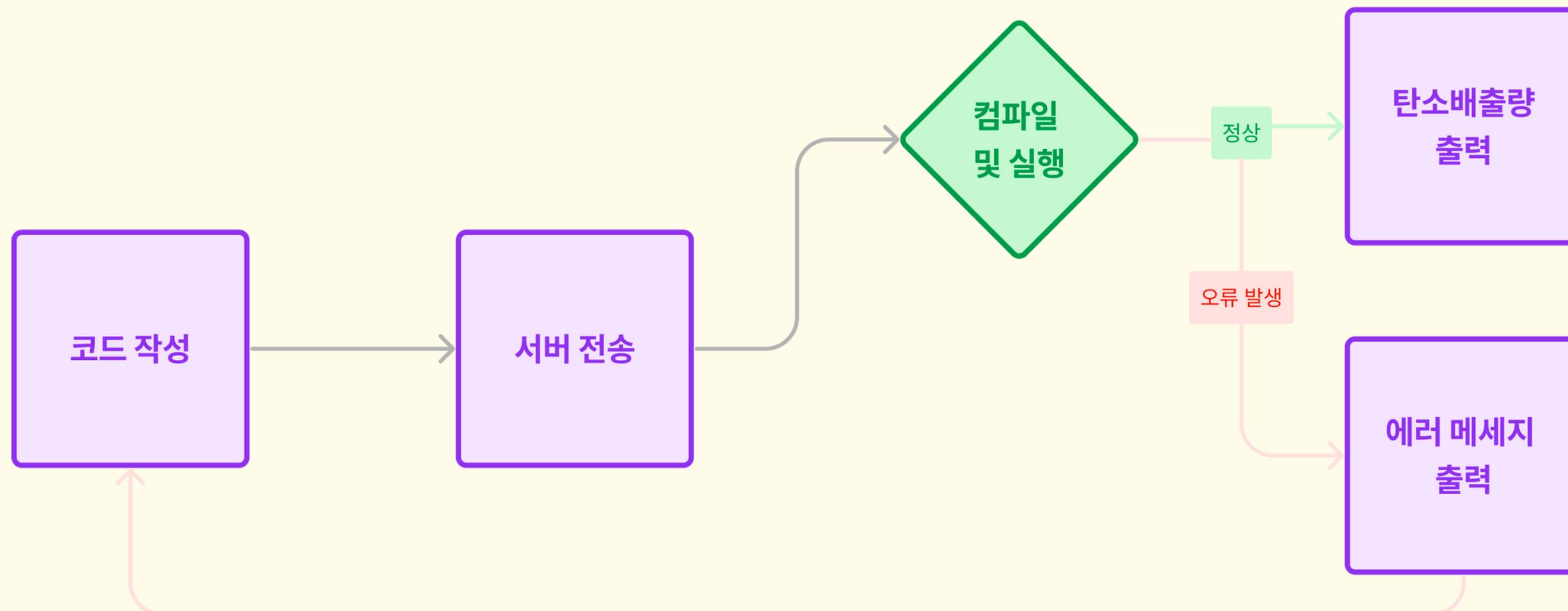
# 02

# Overview

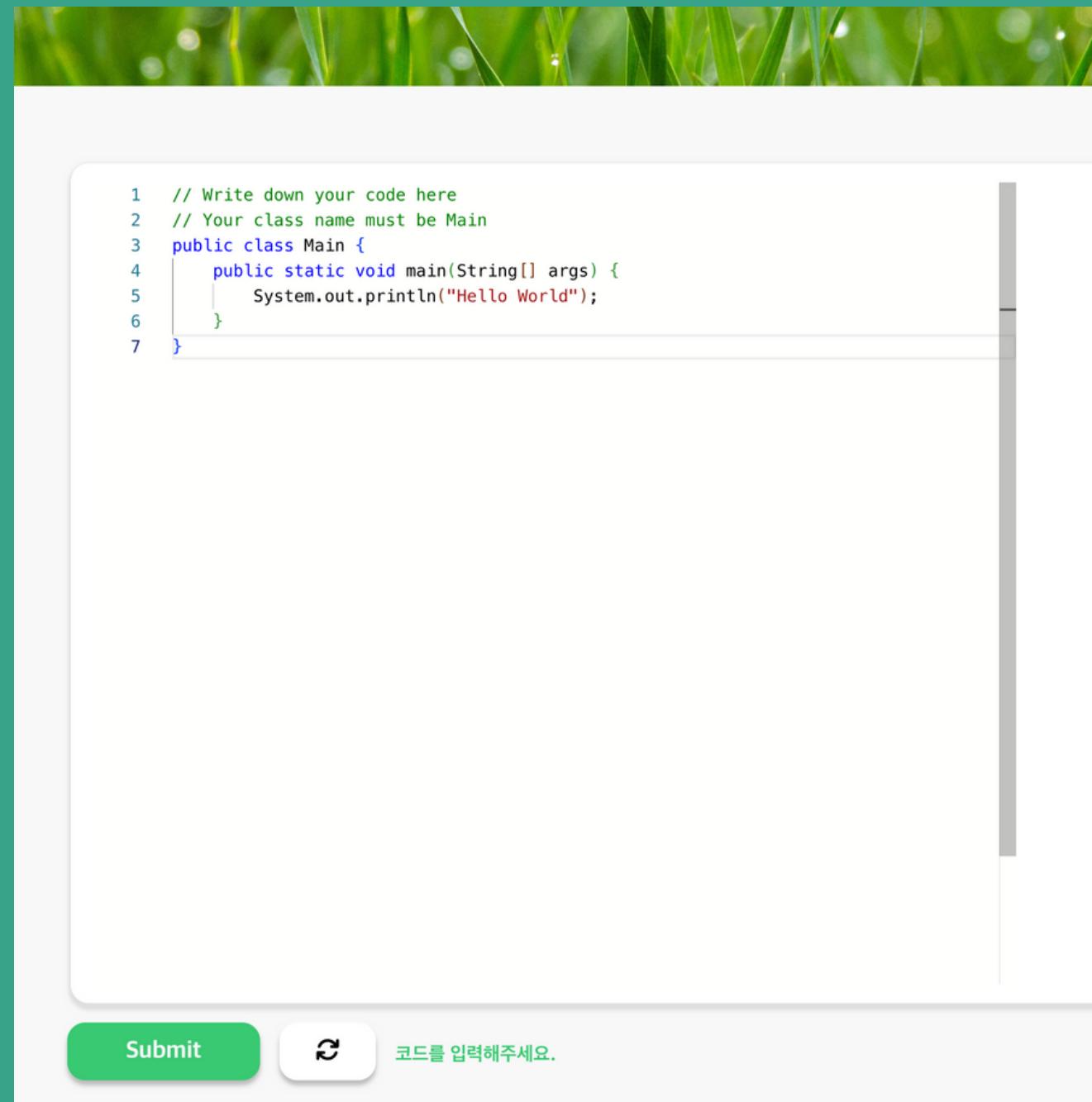
CODEMETER 프로젝트 개요



# ✓ User Flow



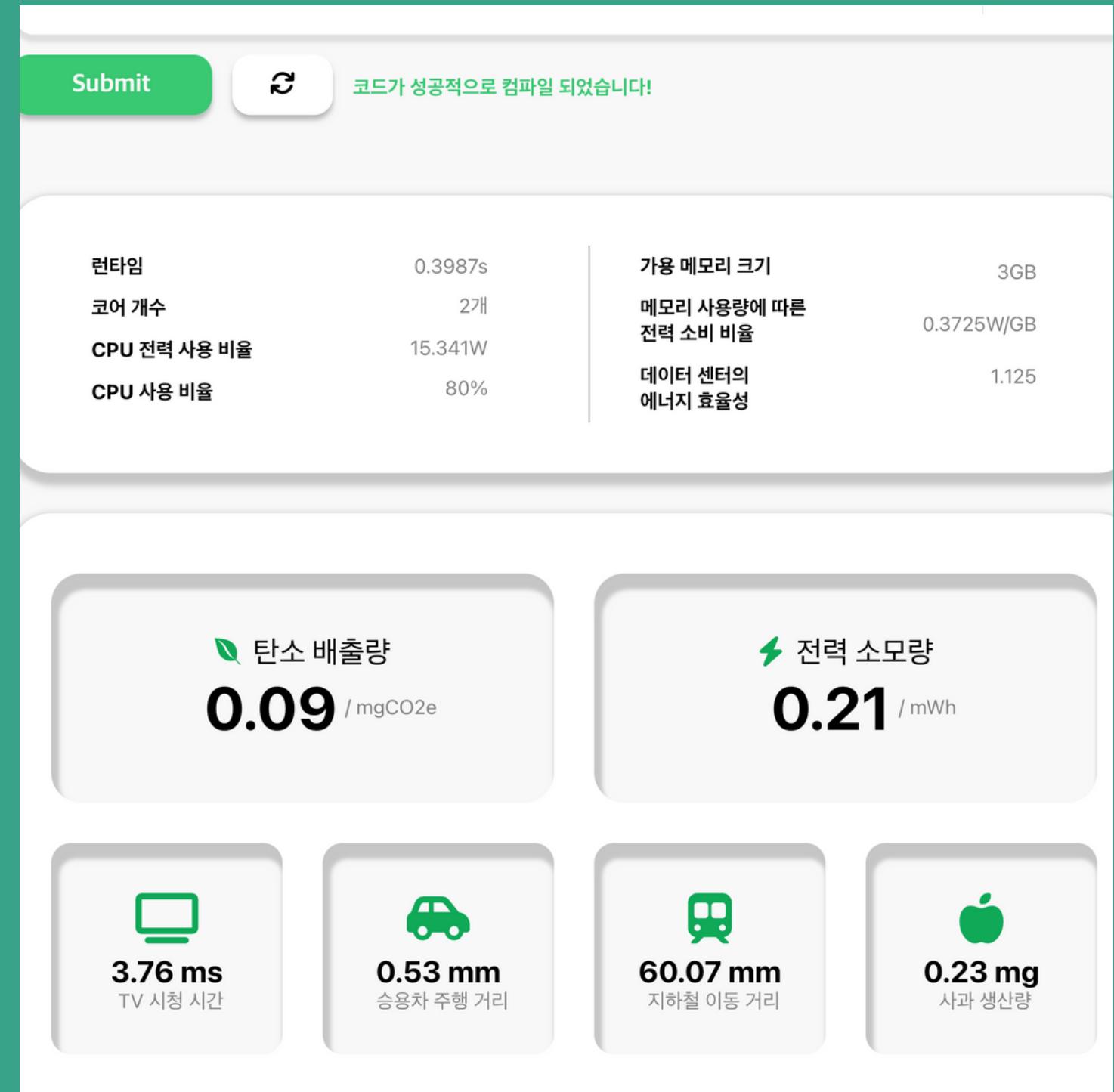
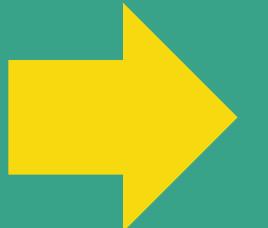
# ✓ Implementation Result



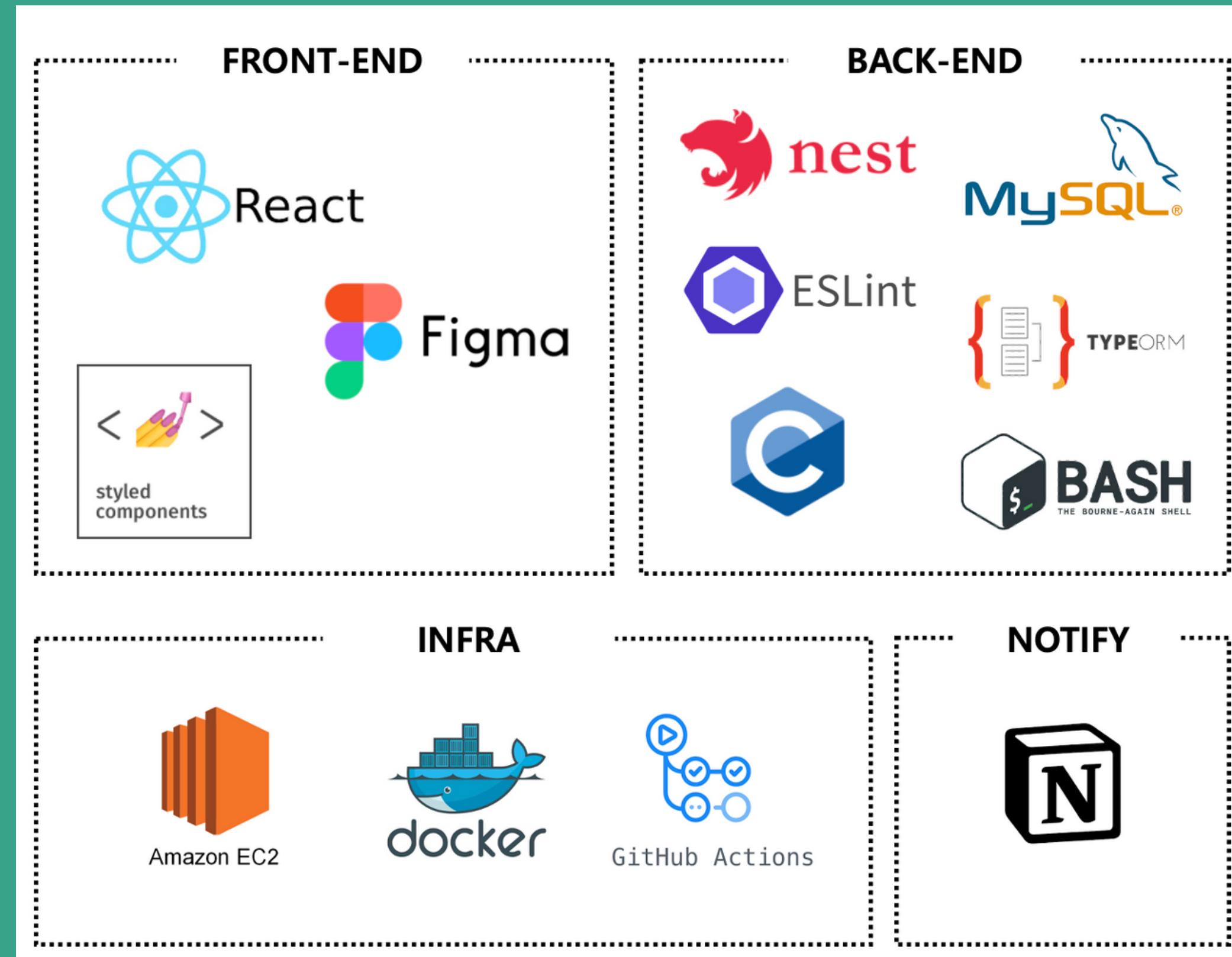
```
1 // Write down your code here
2 // Your class name must be Main
3 public class Main {
4     public static void main(String[] args) {
5         System.out.println("Hello World");
6     }
7 }
```

Submit  코드를 입력해주세요.

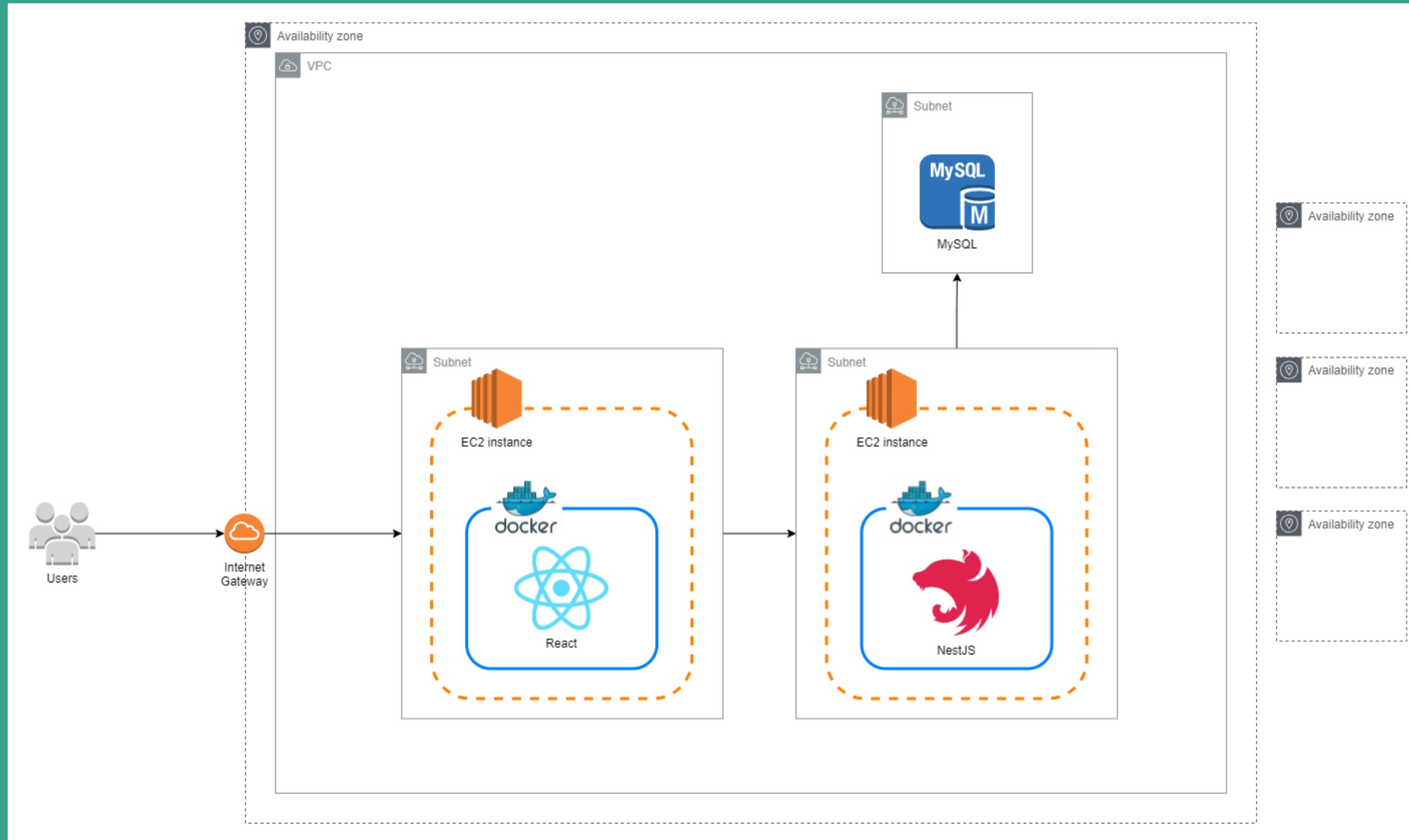
결과값 출력



# ✓ Tech Stacks



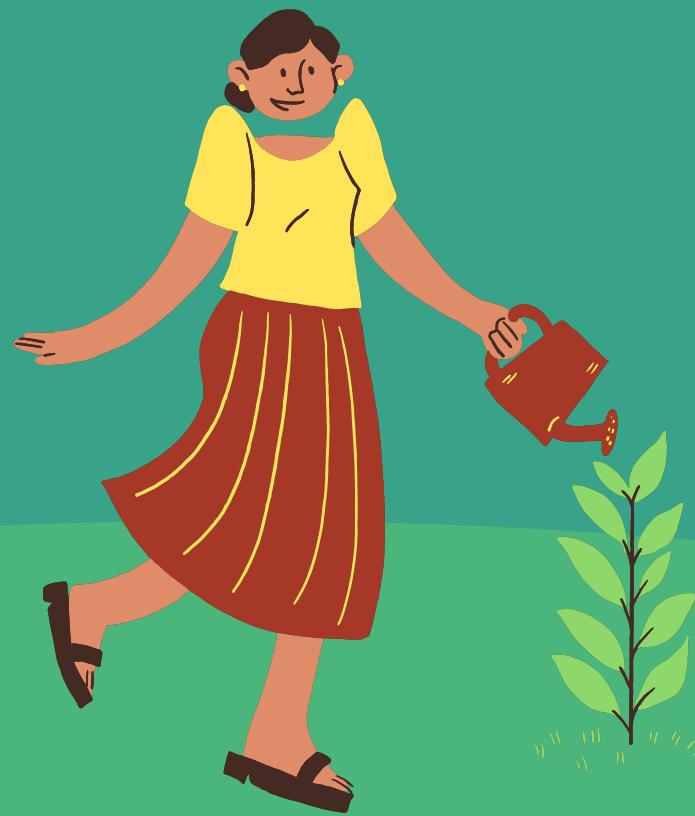
# ✓ System Architecture



03

# Implementation

CODEMETER 구현



# Frontend JAVA Editor

JAVA 코드 입력  
1000줄 제한

컴파일 진행 중

새로고침 버튼

Submit



코드를 실행중입니다.

Submit



코드를 입력해주세요.

제출 버튼

안내 문구

# Frontend JAVA Editor

입력 대기 상태

코드를 입력해주세요.

컴파일 및 실행 중

코드를 실행중입니다.

컴파일 성공

코드가 성공적으로 컴파일 되었습니다!

컴파일 에러

코드 컴파일 중 오류가 발생했습니다.

런타임 에러

코드 실행 중 오류가 발생했습니다.

```
1 // write down your
2 // code here
3
4 public class Main {
5     private static
6     final int LOOP
7     = 1000000;
8
9     public static
10    void main(String
11        [] args) {
12        int[] arr =
13        new int
14        [LOOP];
15        for (var i
16        = 0; i <
17        LOOP; i++) {
18            arr[i]
19            = i;
20        }
21    }
22}
```

Submit



코드가 성공적으로 컴파일 되었습니다!

# Frontend Guide

**CODEMETER**  
개요 설명

**사용법 안내**

**MathJax 수식**

**수식 설명**

## How It Works?

### CODEMETER란?

CODEMETER는 Java 코드 탄소배출량 측정 시스템입니다. 사용자로부터 Java 코드를 입력받고, 해당 코드의 실행시간과 메모리 사용량을 계산하여 특정 컴퓨팅 환경에서 코드가 배출하는 탄소배출량을 계산하고 사용자에게 출력해주는 웹 서비스입니다.

### 어떻게 사용하나요?

탄소배출량을 측정하고 싶은 JAVA 코드를 상단의 코드 입력란에 붙여넣습니다. RUN 버튼을 누르면 입력한 코드가 서버에 전송되며, 컴파일 성공 여부가 입력란 하단에 표시됩니다. 컴파일에 성공했다면, 현재의 설명란에 서버의 스펙 정보와 탄소배출량이 표시됩니다. 코드를 실행하기 전, 에러나 무한 루프 등의 요소가 없는지 확인해주세요.

### 무엇을 보여주나요?

탄소배출량을 kg단위로, 전력 소모량을 kWh 단위로 계산하여 표시합니다. 이 때, 에너지는

$$E = t * (n_c * P_c * u_c + n_m * P_m) * PUE * 0.001$$

의 식을 통해 계산됩니다. t는 런타임, n\_c는 코어 개수, P\_c는 CPU 전력 사용 비율, u\_c는 CPU 사용 비율, n\_m은 가용 메모리 크기, P\_m은 메모리 사용량에 따른 전력 소비 비율이며 PUE는 데이터 센터의 에너지 효율성으로, 모두 하드웨어 스펙 표시란에 기재되어 있습니다. 이외에도, 탄소 배출량의 실감을 위해 탄소 배출량을 TV 시청 시간, A4 용지 사용량 등 일상에서 친숙하게 접할 수 있는 값으로 환산하여 나타냅니다.

## How It Works?

### CODEMETER란?

CODEMETER는 Java 코드 탄소배출량 측정 시스템입니다. 사용자로부터 Java 코드를 입력받고, 해당 코드의 실행시간과 메모리 사용량을 계산하여 특정 컴퓨팅 환경에서 코드가 배출하는 탄소배출량을 계산하고 사용자에게 출력해주는 웹 서비스입니다.

### 어떻게 사용하나요?

탄소배출량을 측정하고 싶은 JAVA 코드를 상단의 코드 입력란에 붙여 넣습니다. RUN 버튼을 누르면 입력한 코드가 서버에 전송되며, 컴파일 성공 여부가 입력란 하단에 표시됩니다. 컴파일에 성공했다면, 현재의 설명란에 서버의 스펙 정보와 탄소배출량이 표시됩니다. 코드를 실행하기 전, 에러나 무한 루프 등의 요소가 없는지 확인해주세요.

### 무엇을 보여주나요?

탄소배출량을 kg단위로, 전력 소모량을 kWh 단위로 계산하여 표시합니다. 이때, 에너지는

$$E = t * (n_c * P_c * u_c + n_m * P_m) * PUE * 0.001$$

의 식을 통해 계산됩니다. t는 런타임, n\_c는 코어 개수, P\_c는 CPU 전력 사용 비율, u\_c는 CPU 사용 비율, n\_m은 가용 메모리 크기, P\_m은 메모리 사용량에 따른 전력 소비 비율이며 PUE는 데이터 센터의 에너지 효율성으로, 모두 하드웨어 스펙 표시란에 기재되어 있습니다. 이외에도, 탄소 배출량의 실감을 위해 탄소 배출량을 TV 시청 시간, A4 용지 사용량 등 일상에서 친숙하게 접할 수 있는 값으로 환산하여 나타냅니다.

# Frontend

# Hardware Spec

|              |          |                      |            |
|--------------|----------|----------------------|------------|
| 런타임          | 72.98 ms | 가용 메모리 크기            | 2GB        |
| 코어 개수        | 1개       | 메모리 사용량에 따른 전력 소비 비율 | 0.3725W/GB |
| CPU 전력 사용 비율 | 8W       | 데이터 센터의 에너지 효율성      | 1.2        |
| 메모리 사용량      | 38.25 MB |                      |            |

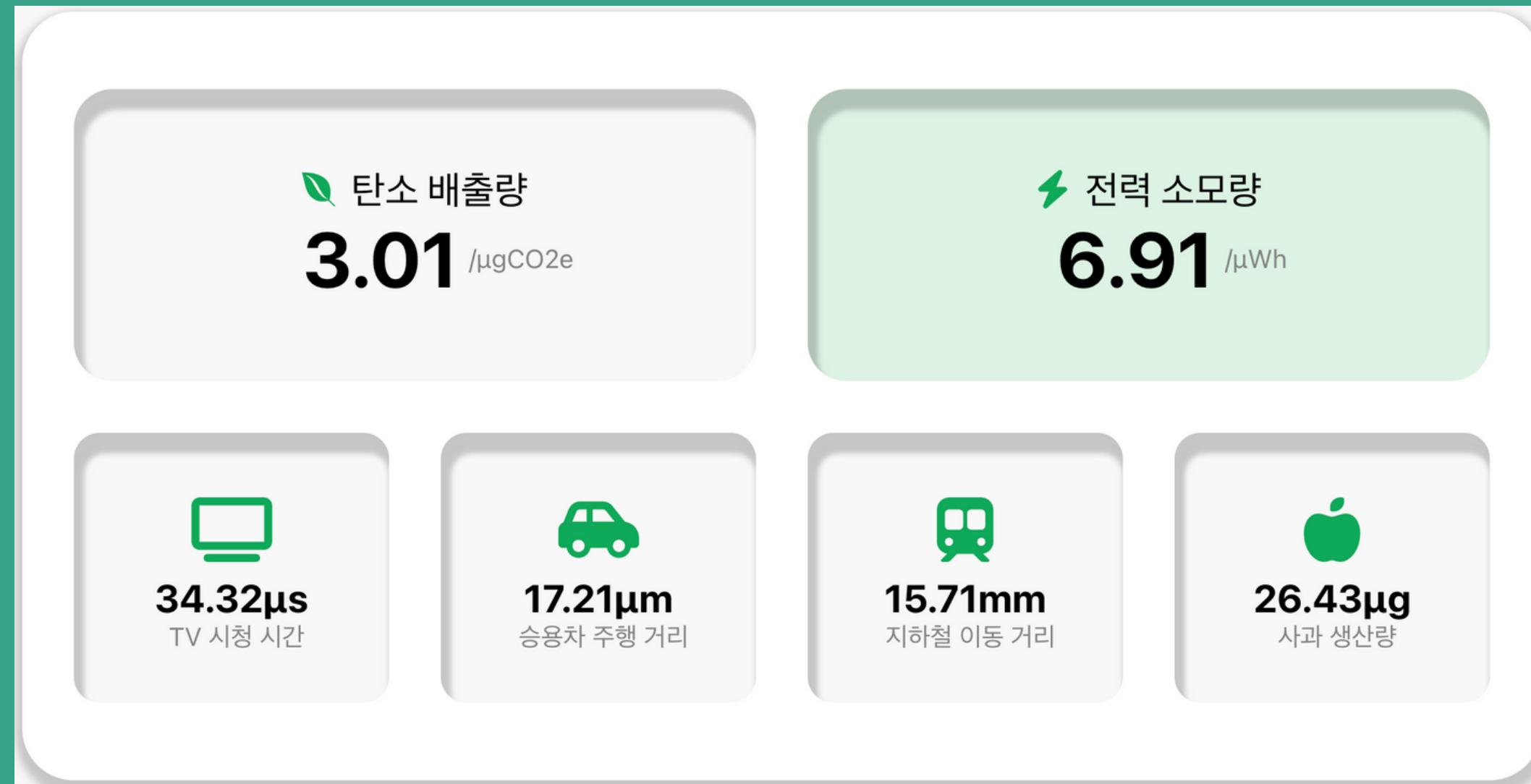
태블릿

|                      |            |
|----------------------|------------|
| 런타임                  | 72.98 ms   |
| 코어 개수                | 1개         |
| CPU 전력 사용 비율         | 8W         |
| 메모리 사용량              | 38.25 MB   |
| 가용 메모리 크기            | 2GB        |
| 메모리 사용량에 따른 전력 소비 비율 | 0.3725W/GB |
| 데이터 센터의 에너지 효율성      | 1.2        |

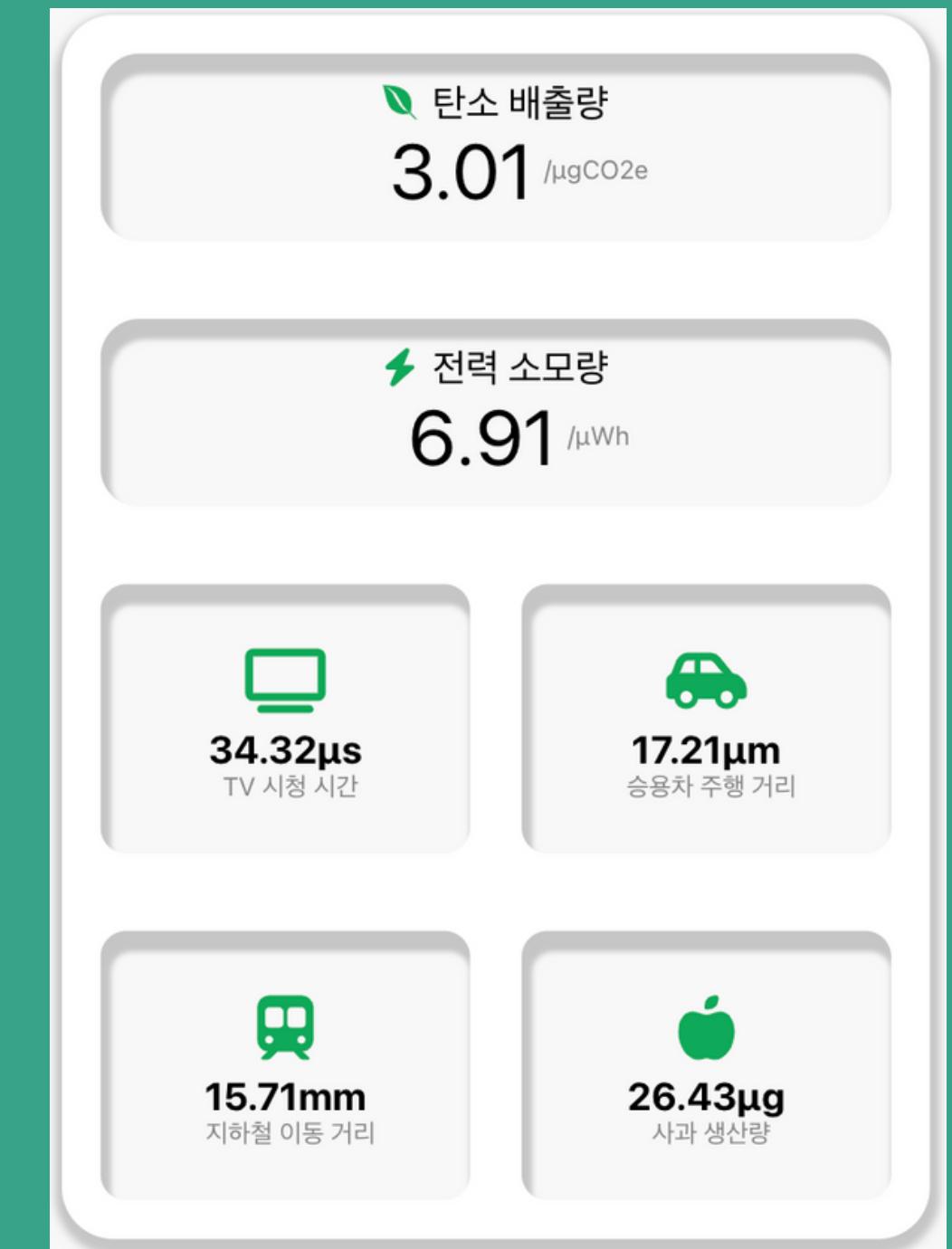
|              |          |
|--------------|----------|
| 런타임          | 72.98 ms |
| 코어 개수        | 1개       |
| CPU 전력 사용 비율 | 8W       |
| 메모리 사용량      | 38.25 MB |

모바일

# Frontend Carbon Card

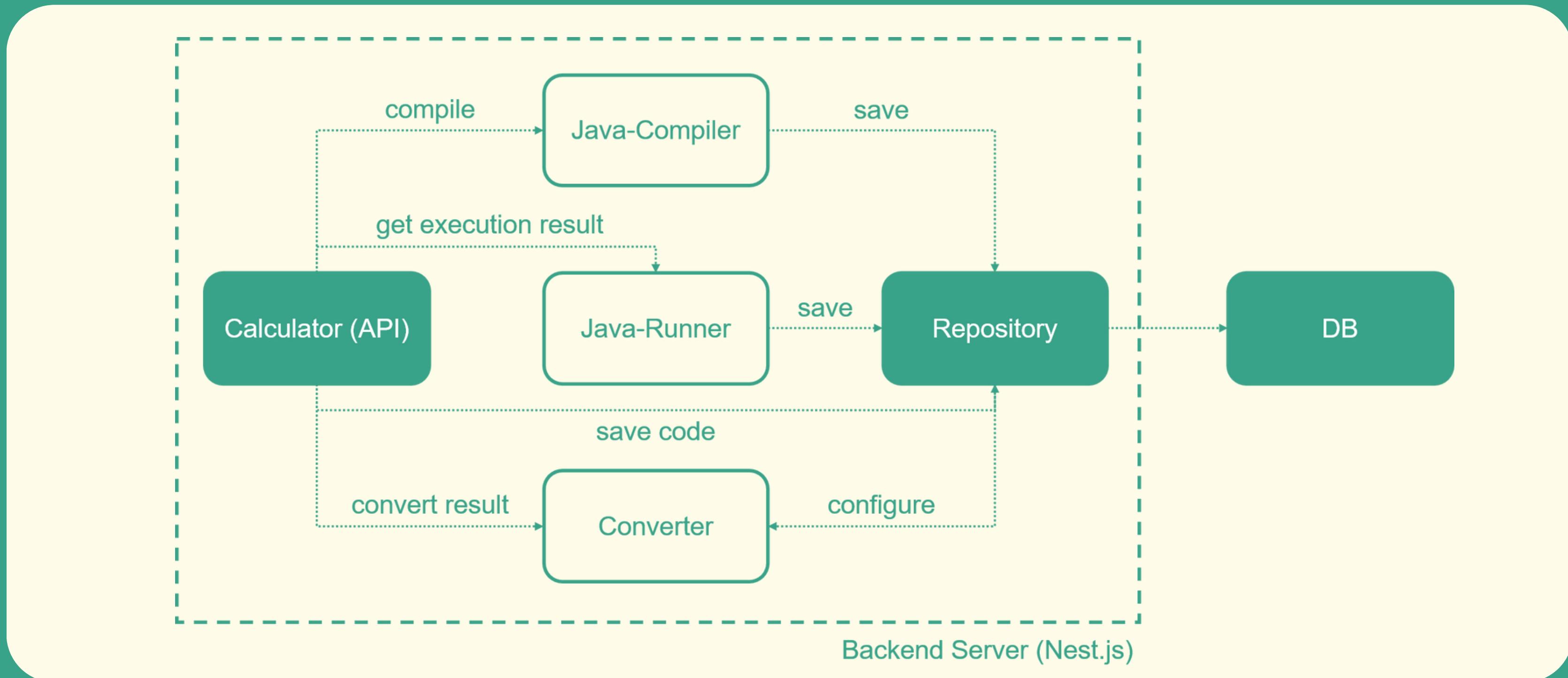


데스크탑



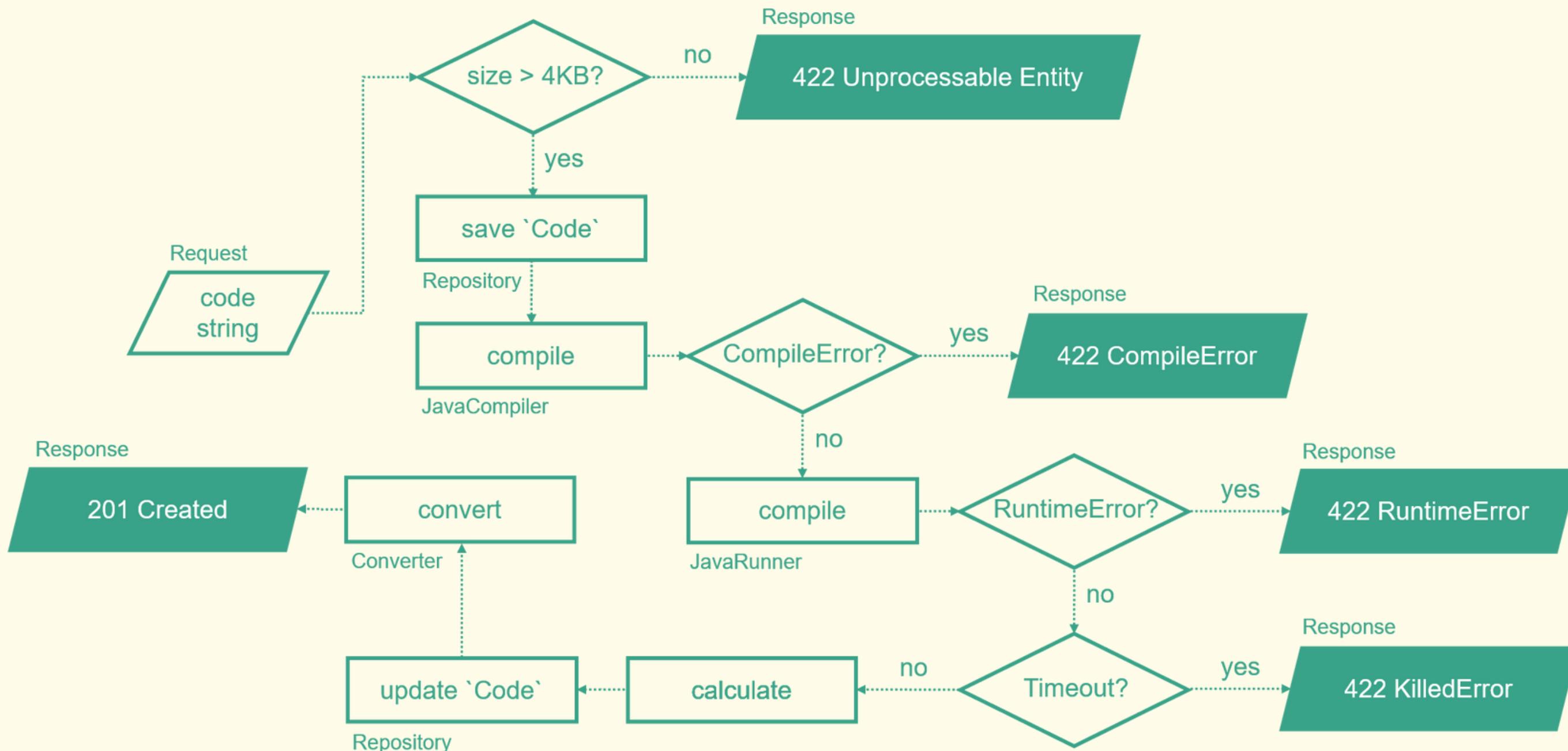
태블릿

# Backend Overall Structure



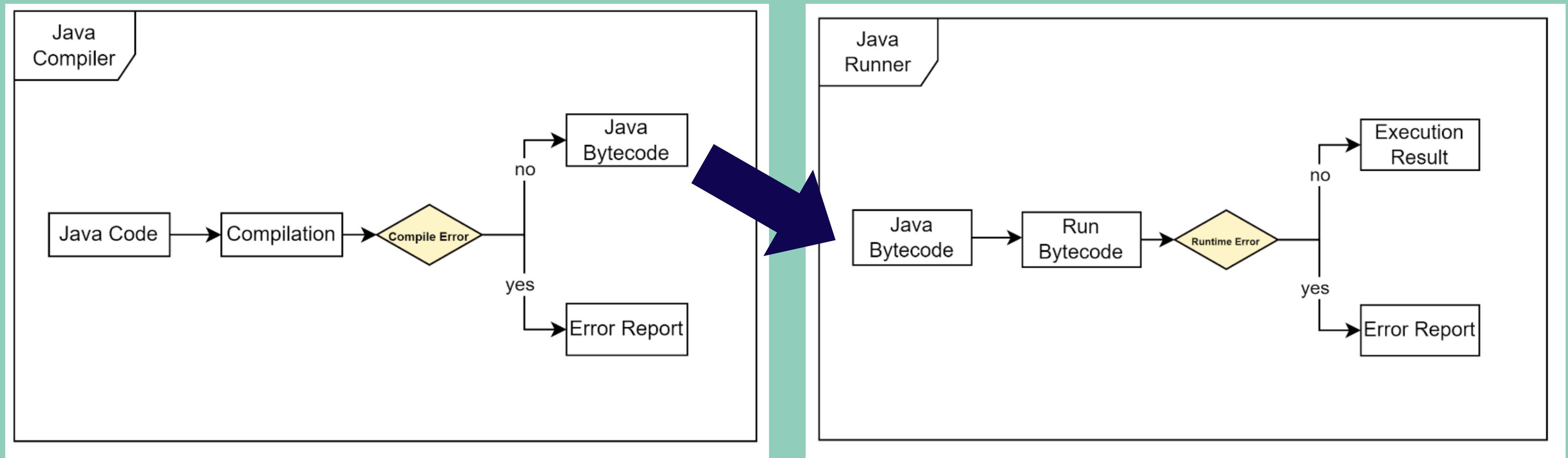
# Backend

# Carbon Emission Calculator (API)



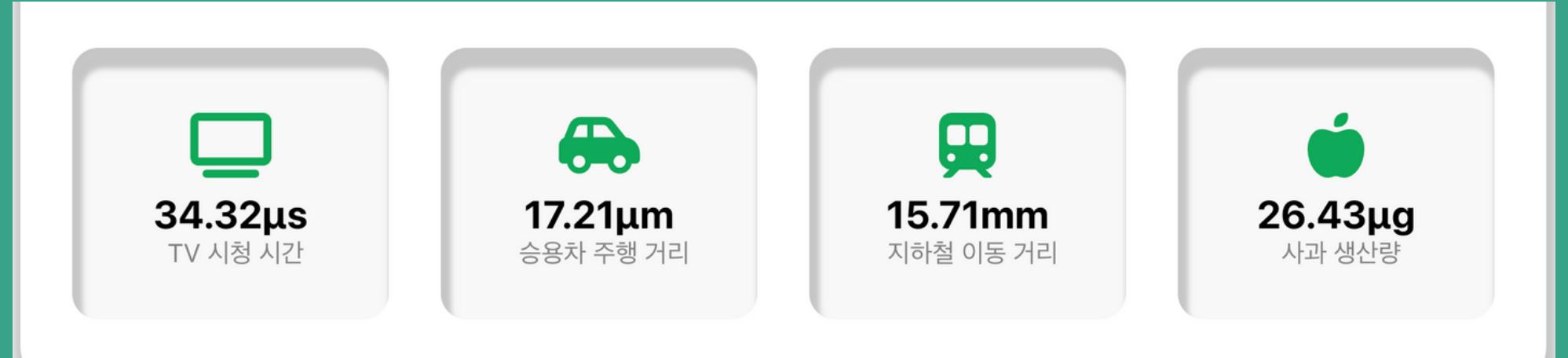
# Backend

# Java Compiler & Runner



# Backend

# Real usage Converter



|   | id | name  | emission |
|---|----|---|----------|
| 1 | 1  | TV_CARBON_EMISSION_PER_HOUR                     | 88       |
| 2 | 2  | CARBON_EMISSIONS_OF_AVERAGE_PASSENGER_IN EUROPE | 175      |
| 3 | 3  | SUBWAY_TRAVEL_DISTANCE_KILOMETER_PER_GCO2E      | 0.653595 |
| 4 | 4  | APPLE_PRODUCTION_PER_GCO2E                      | 2.5      |

```
export enum Reference {  
    TV_WATCH_TIME = 'TV_CARBON_EMISSION_PER_HOUR',  
    DRIVING_DISTANCE = 'CARBON_EMISSIONS_OF_AVERAGE_PASSENGER_IN_EUROPE',  
    SUBWAY_DISTANCE = 'SUBWAY_TRAVEL_DISTANCE_KILOMETER_PER_GCO2E',  
    APPLE_PRODUCTION = 'APPLE_PRODUCTION_PER_GCO2E',  
}
```

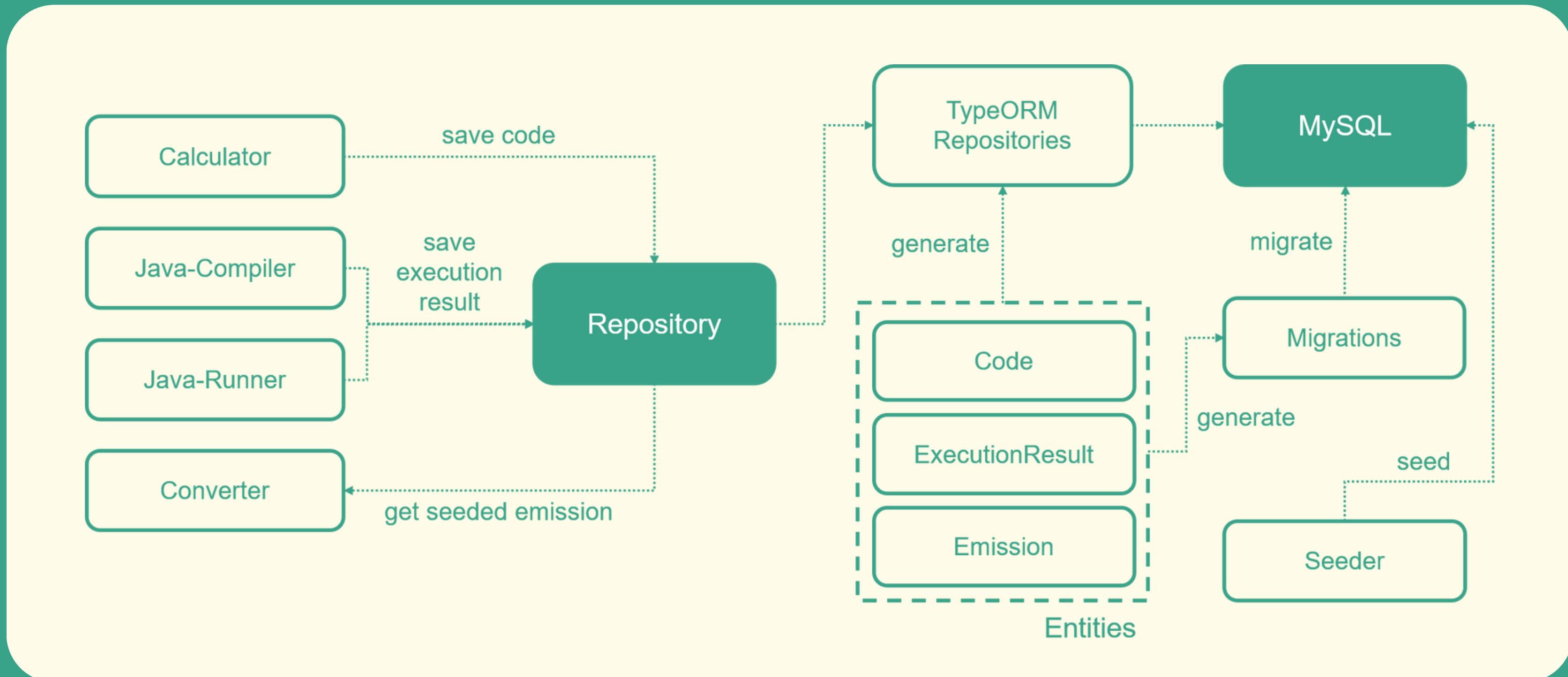
RDB, Enum 형식 이용하여 관리되는  
Real usage 변환식 계수

```
27  /**  
28  * 탄소 배출량을 입력으로 받아, 실생활 사용량으로 변환하는 메서드입니다.  
29  * @param carbonEmission 탄소 배출량 (gCo2e 단위)  
30  */  
31  2 usages ▲ HOYA  
32  convertCarbonEmission(  
33  carbonEmission: number,  
34  ): CarbonEmissionConvertedResultDto {  
35  // 1. 탄소 배출량 계산  
36  // μgCO2e 단위  
37  const convertedCarbonEmission: number = carbonEmission * MICRO;  
38  // 2. 전력 소모량 계산  
39  // μWh 단위  
40  const energy: number = (carbonEmission / this.config.CI) * KILO_TO_MICRO;  
41  // 3. TV 시청 시간 계산  
42  // μs 단위  
43  const tvWatchingTime: number =  
44  (carbonEmission / this.config[Reference.TV_WATCH_TIME]) *  
45  HOUR_TO_MICROSECOND;  
46  // 4. 승용차 주행 거리 계산  
47  // μm 단위  
48  const passageCar: number =  
49  (carbonEmission / this.config[Reference.DRIVING_DISTANCE]) *  
50  KILO_TO_MICRO;  
51  // 5. 지하철 이동 거리 계산  
52  // mm 단위  
53  const subwayTravelDistance: number =  
54  carbonEmission * this.config[Reference.SUBWAY_DISTANCE] * KILO_TO_MILLI;  
55  // 6. 사과 생산량 계산  
56  // μg 단위  
57  const appleProduction: number =  
58  carbonEmission * this.config[Reference.APPLE_PRODUCTION] * MICRO;  
59  // return new CarbonEmissionConvertedResultDto(  
60  //     convertedCarbonEmission,  
61  //     energy,  
62  //     tvWatchingTime,  
63  //     passageCar,  
64  //     subwayTravelDistance,  
65  //     appleProduction,  
66  // );  
67  };
```

실생활 사용량 계산 로직

# Backend

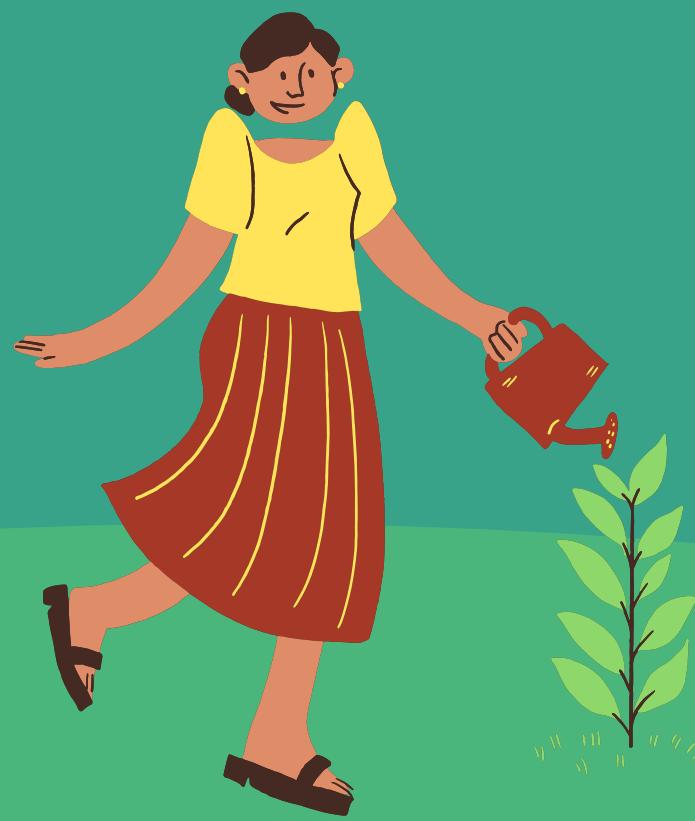
## DB (MySQL + TypeORM)



04

# Project Management

프로젝트 관리



# Role



**Front-end**

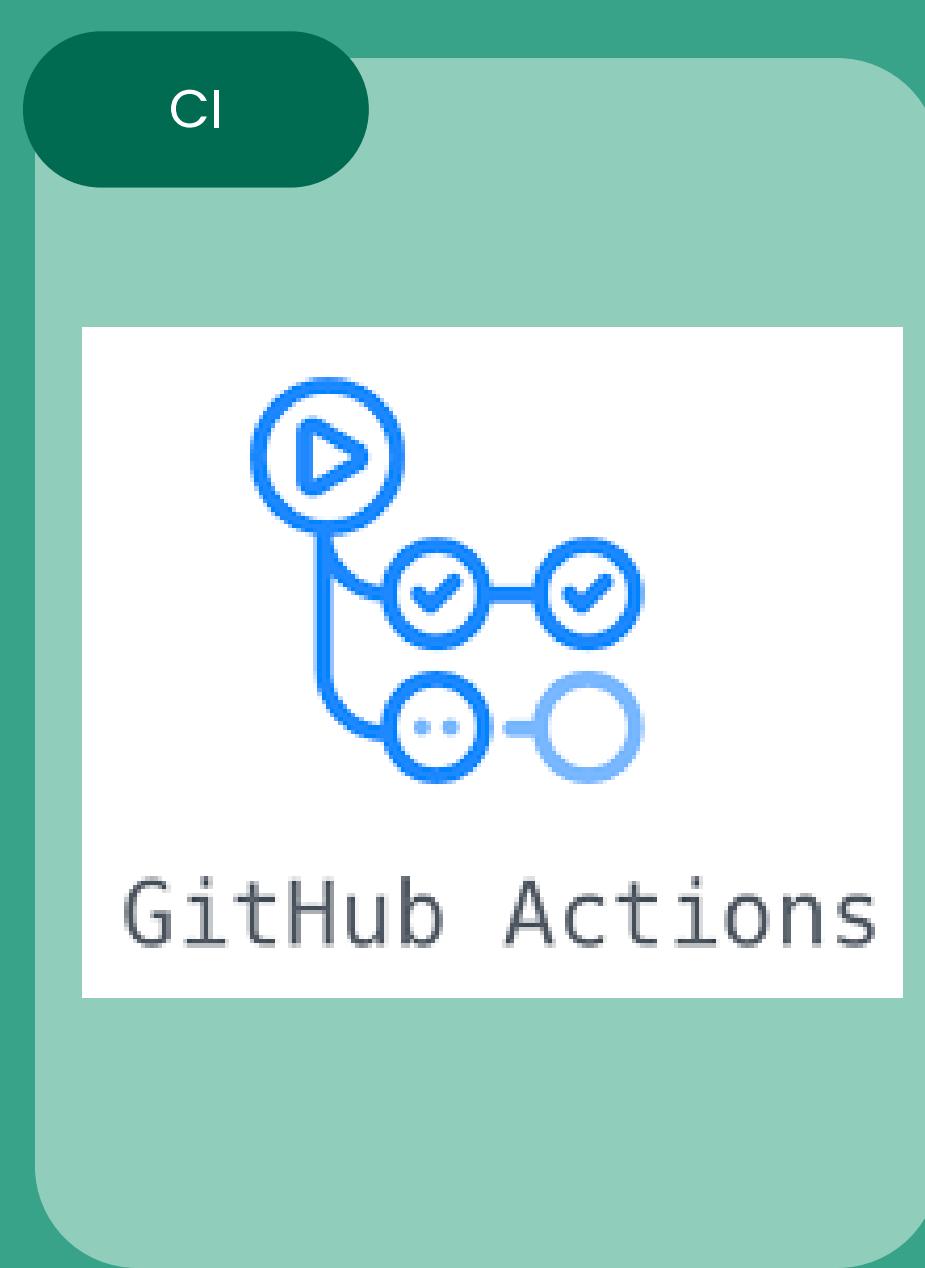
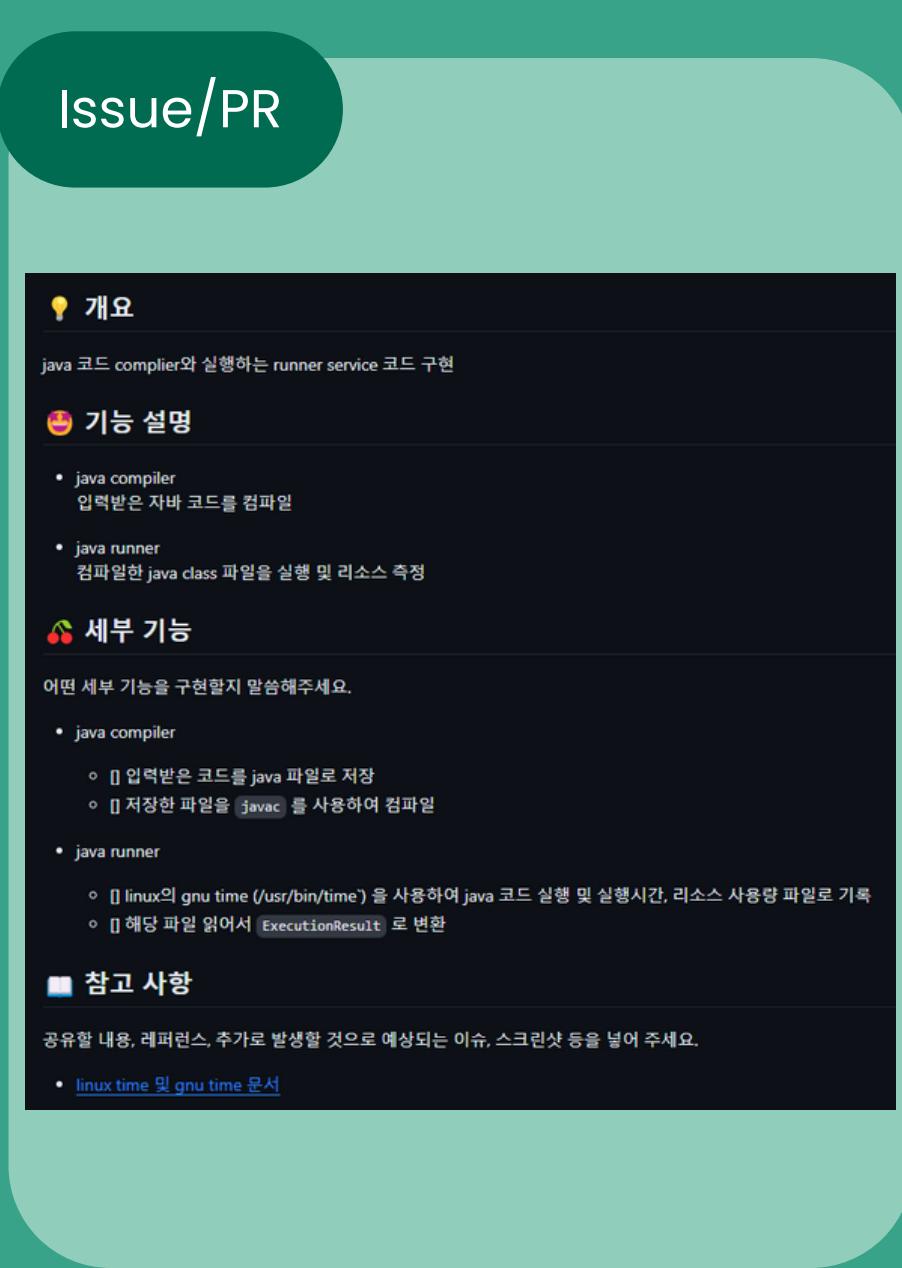
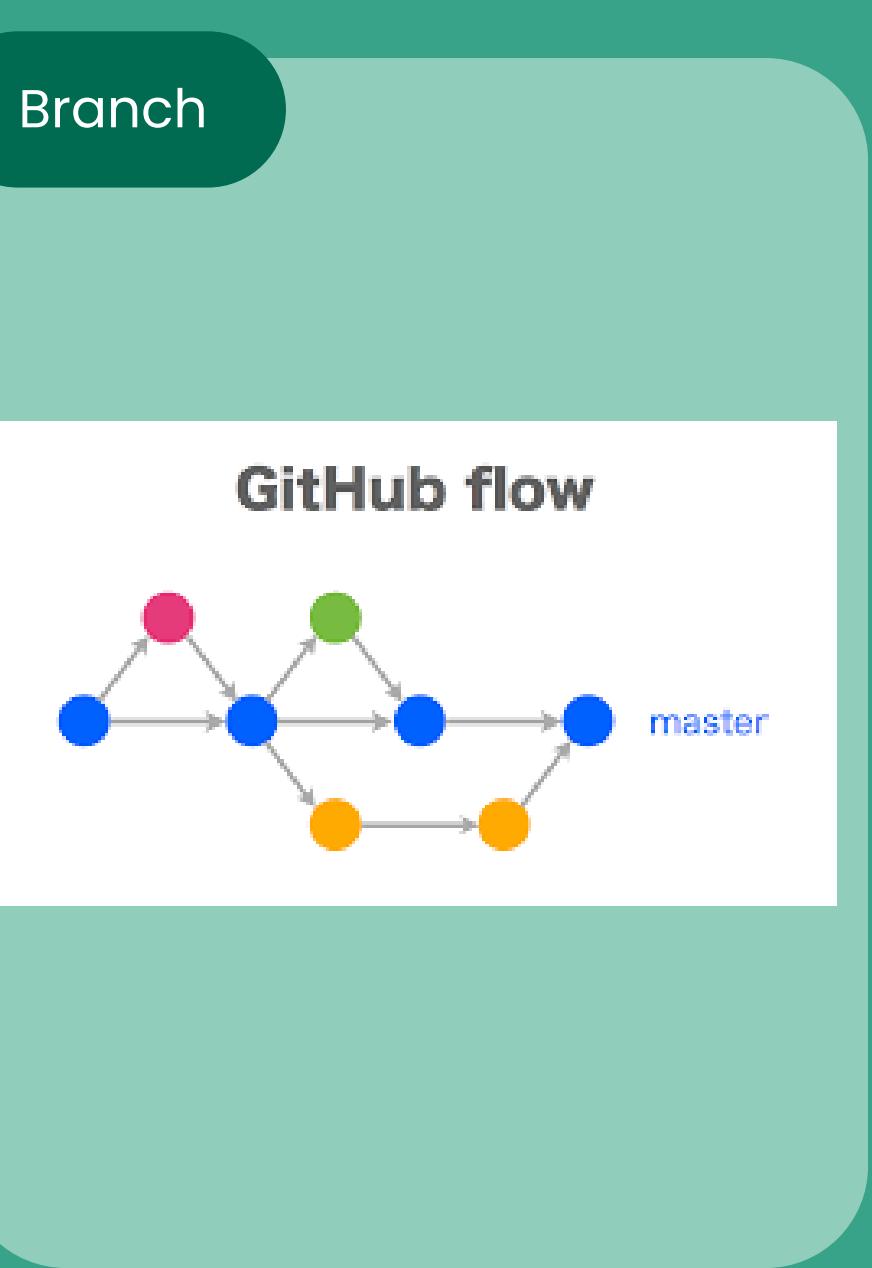
김동한, 조유지

**Back-end**

조민호, 임소리, 유지훈, 안낙균

# Workflow

- Github



# Workflow

- Code Quality

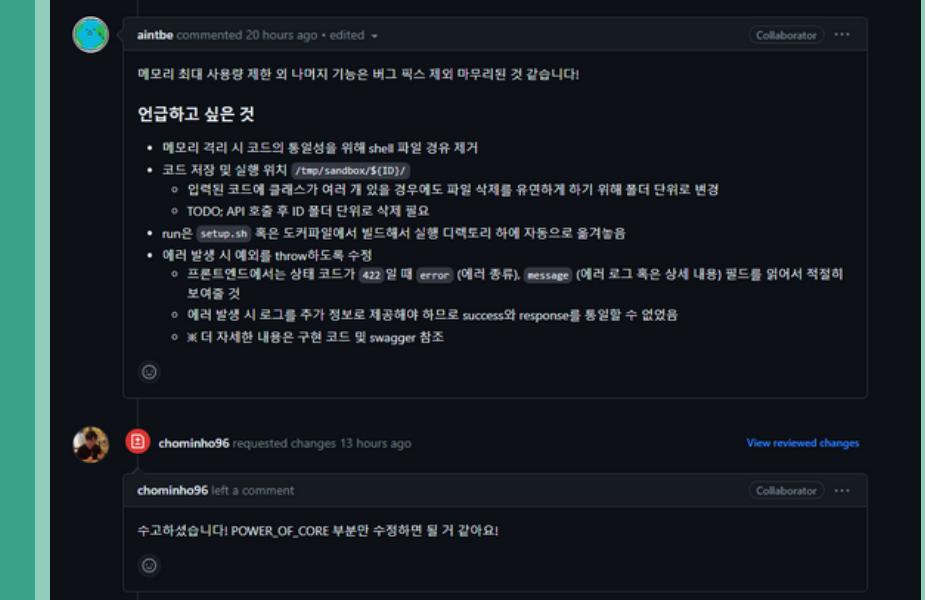
Code Convention



Commit Convention

**Angular git**  
Commit Message  
Convention

PR



# Workflow

- Documentation

## Notion

### Energy consumption

$$E = t \times (n_c \times P_c \times u_c + n_m \times P_m) \times PUE \times 0.001 \quad (1)$$

where  $t$  is the running time (hours),  $n_c$  the number of cores and  $n_m$  the size of memory available (gigabytes).  $u_c$  is the core usage factor (between 0 and 1).  $P_c$  is the power draw of a computing core and  $P_m$  the power draw of the memory (Watt).  $PUE$  is the efficiency coefficient of the data centre.

1. E : 전체 에너지 소비량 (kWh)
2. t : 런타임 (h)
3. n\_c : 사용하는 코어 수
4. p\_c : TDP : thermal design power (Watt), CPU 전력 사용 비율
  - CPU 모델마다 제조 회사에서 상수값으로 제공함
  - path : data/v2.1/TDP\_cpu.csv

| index         | in Watt |         |              |   |
|---------------|---------|---------|--------------|---|
| model         | TDP     | n_cores | TDP_per_core | source  |
| A8-7680       | 45      | 4       | 11.3         | <a href="https://www.techpowerup.com/cpu-specs/">https://www.techpowerup.com/cpu-specs/</a>   |
| A9-9425 SoC   | 15      | 2       | 7.5          | <a href="https://www.techpowerup.com/cpu-specs/">https://www.techpowerup.com/cpu-specs/</a>   |
| AMD 7552      | 200     | 48      | 4.2          | <a href="https://www.amd.com/system/files/documents/AMD-EPYC-7552.pdf">https://www.amd.com/system/files/documents/AMD-EPYC-7552.pdf</a> |
| AMD EPYC 7251 | 120     | 8       | 15.0         | <a href="https://www.amd.com/en/products/cpu/amd-epyc-7251">https://www.amd.com/en/products/cpu/amd-epyc-7251</a>                       |

## Swagger

CODEMETER 1.0 OAS 3.0

Java Code 턴소 배출량 계산기

탄소 배출량 API

POST /carbon-emission 탄소 배출량 생성 API

Parameters

No parameters

Request body required

application

Example Value | Schema

```
{ "code": "\nimport java.util.ArrayList;\n\npublic class Main {\n    public static void main(String[] args) {\n        System.out.println(\"Hello World\");\n    }\n}\n" }
```

Responses

Code Description

201 Media type application/json

Content Accept header

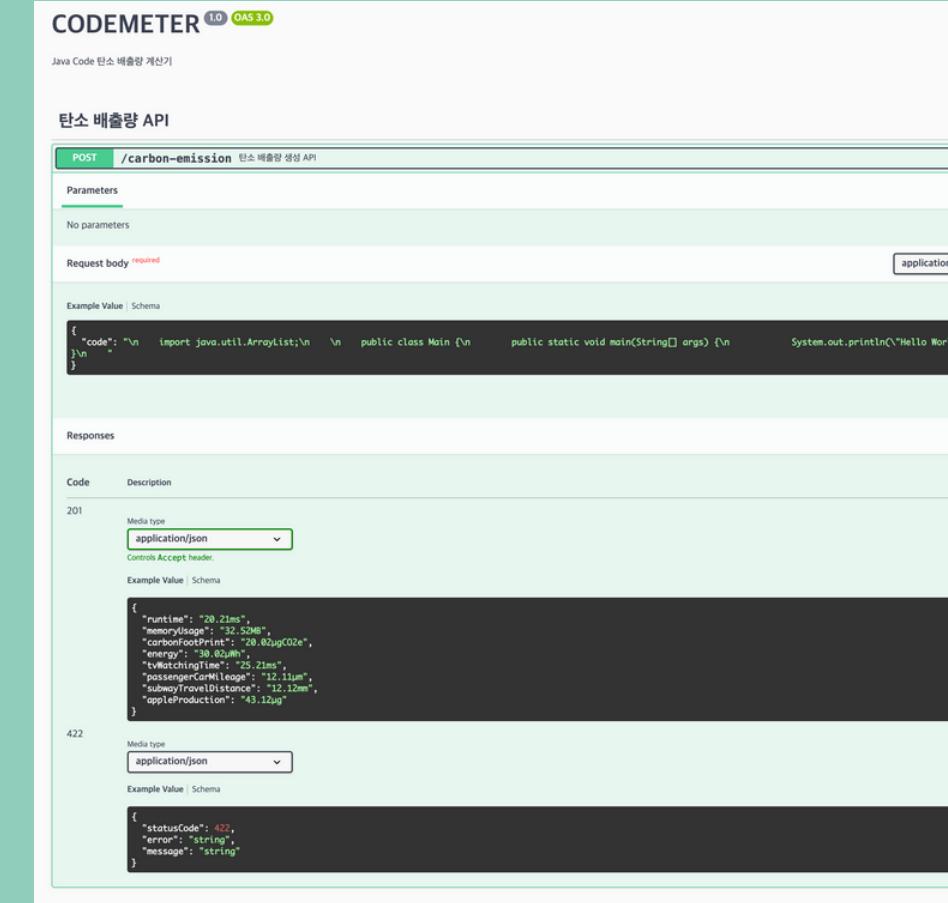
Example Value | Schema

```
{ "runtime": "20.21ms", "memoryUsage": "32.52MB", "carbonFootprint": "28.02kgCO2e", "tvWatchingTime": "25.21ms", "passengerCarMileage": "12.11km", "subwayTravelDistance": "12.12mm", "appleProduction": "43.12kg" }
```

422 Media type application/json

Example Value | Schema

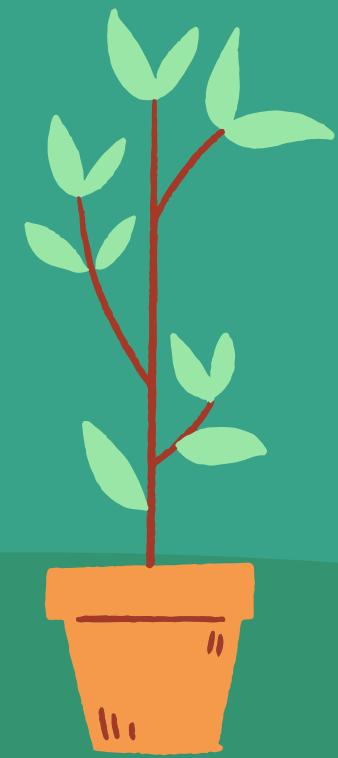
```
{ "statusCode": 422, "error": "string", "message": "string" }
```



05

# Green Patterns

그린화 패턴



# 그린화 알고리즘

- Green Algorithms 논문에 제시된 수식 사용

$$C = \boxed{t} \times (n_c \times P_c \times u_c + \boxed{n_m} \times P_m) \times \text{PUE} \times \text{CI} \times 0.001$$

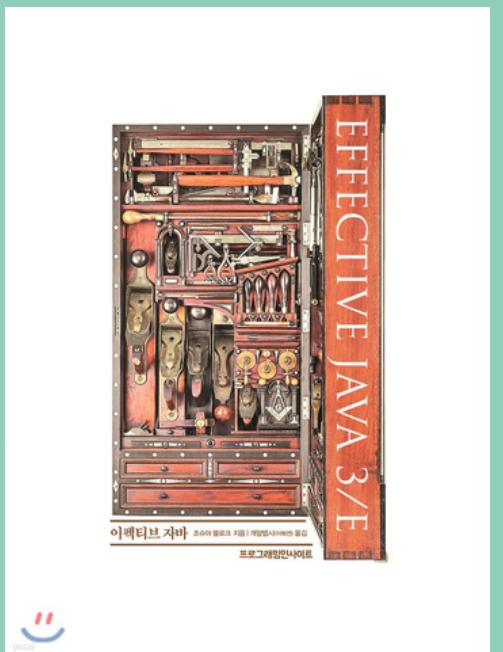
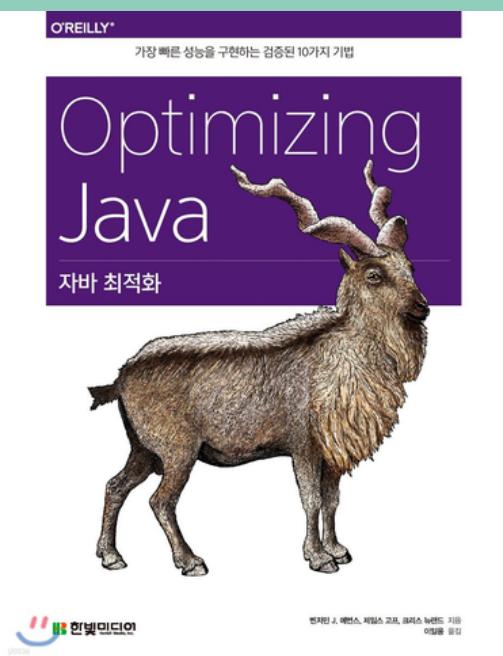
- C : Carbon emission (gCO<sub>2</sub>)
  - t : Runtime (h)
  - n\_m : Peak Memory usage (GB)  
simple approximation!
- Key factors

→ 런타임 or 메모리 사용량을 줄이는 패턴 조사

# 그린화 패턴 수집 방법

- JAVA 성능 최적화 스터디 진행

스터디 참고 서적



Notion을 통한 관리

2023.11.08 스터디

▼ 김동한

ArrayList vs Linked List

Reference

Java의 LinkedList와 ArrayList에 대한 비교  
언녕 프로그래밍  
<https://www.holaxprogramming.com/2014/02/12/java-list-interface/>

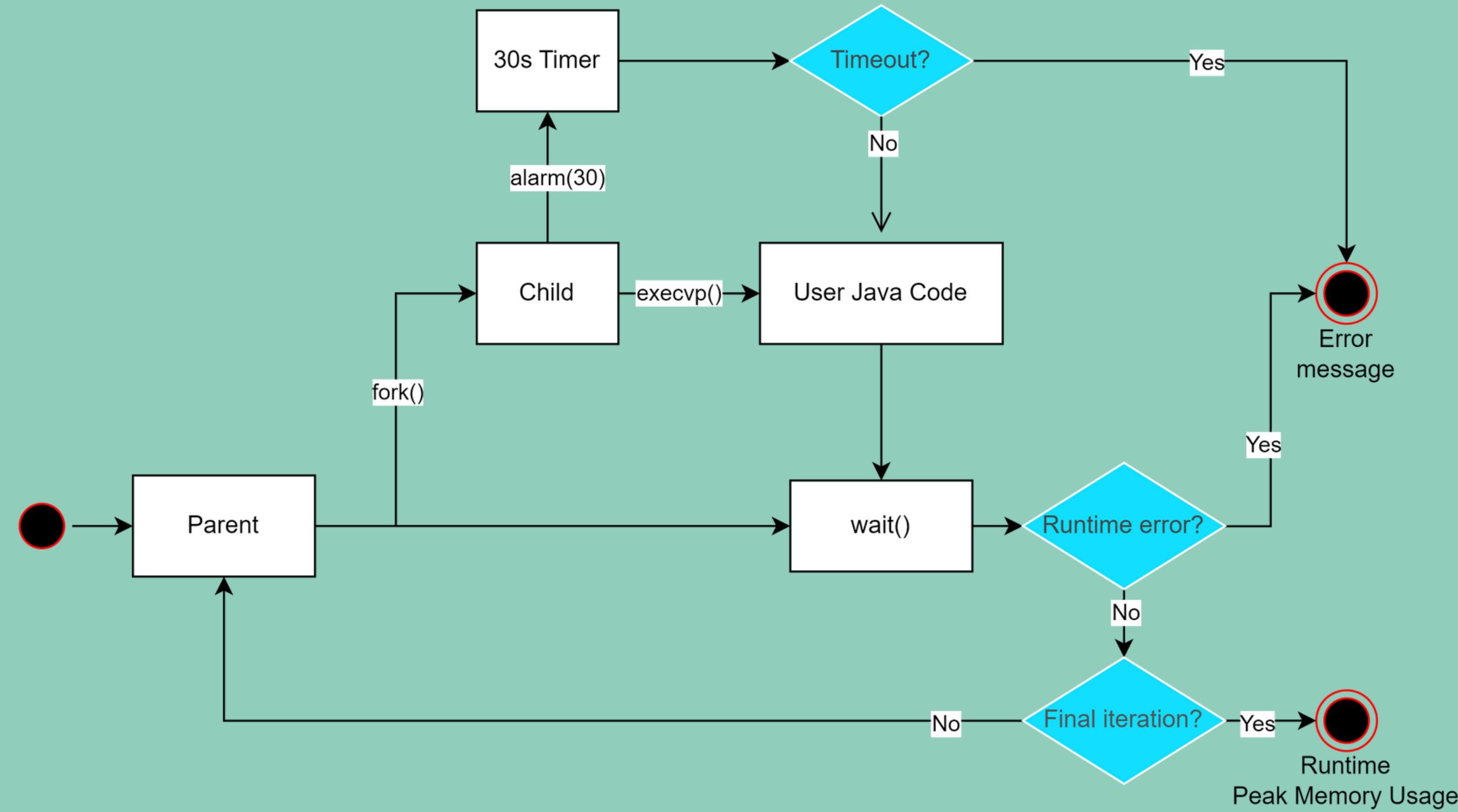
ArrayList

+ :: ArrayList는 내부적으로 데이터를 배열로 관리하여 데이터의 추가 삭제를 할 때 임시 배열을 생성해 데이터를 복사하는 특징을 가지고 있다.

따라서 데이터의 추가/삭제가 빈번한 경우 ArrayList의 성능 저하를 일으킨다.

```
public class Before {  
    private static final int INSERT_COUNT = 100000;  
    private static final List<Integer> numbers = new ArrayList<>();  
  
    public static void main(String[] args) {  
        for (int i = 1; i <= INSERT_COUNT; i++) {  
            numbers.add(0, i);  
        }  
    }  
}
```

# 그린화 패턴 Java Runner



# 그린화 패턴 테스트

- Cloud 환경 기반 테스트 수행

AWS EC2



Amazon EC2

인스턴스 (1/1) 정보

Instance를 속성 또는 (case-sensitive) 태그로 찾기

인스턴스 상태 = running X      필터 지우기

| Name              | 인스턴스 ID             | 인스턴스 상태 | 인스턴스 유형  | 상태 검사      |
|-------------------|---------------------|---------|----------|------------|
| skku-se-java-r... | i-004620f49c2336680 | 실행 중    | t2.micro | 2/2개 검사 통과 |

코드 반복 테스트

반복 실행 - iteration : 10

```
./run_process.sh short-circuit 10
```

Result

```
☰ result_mean_10.txt
Before: 1 0.158931 39152
After: 1 0.102867 39091
```

[status] [runtime(s)] [mem\_usage(KB)]

→ 탄소 배출량 측정

# 그린화 패턴 테스트 결과

- 총 24개 패턴

## 1. Use Early Return

a. return문을 함수 끝에 모아 정리하는 경우가 많은데, 이 경우 함수의 모든 로직을 실행한 뒤 결과값을 반환하게 된다. 따라서 함수가 종료할 수 있는 조건일 경우 그 즉시 return을 하는 early return 방법으로 런타임과 메모리 모두 이점을 가져갈 수 있다.

```
public class Before {
    public static int function(int a) {
        int result = 0;
        for (int i = 0; i < 100000; i++) {
            result += i;
        }

        if (a < 9999) return a;
        return result;
    }

    public static void main(String[] args) {
        for(int i = 0; i < 10000; i++) {
            function(i);
        }
    }
}
```

Before

```
public class After {
    public static int function(int a) {
        if (a < 9999) return a;

        int result = 0;
        for (int i = 0; i < 100000; i++) {
            result += i;
        }
        return result;
    }

    public static void main(String[] args) {
        for(int i = 0; i < 10000; i++) {
            function(i);
        }
    }
}
```

After

b. 실행 시간/메모리 사용량 측정값 및 탄소 배출량

(실행 시간, 메모리 사용량은 각 코드를 5번씩 실행하여 도출한 평균값이다.)

|                              | Before       | After        |
|------------------------------|--------------|--------------|
| Runtime (s)                  | 0.470562     | 0.068820     |
| Memory Usage (MB)            | 35.708       | 35.606       |
| Carbon Emission ( $gCO_2e$ ) | 4.568106e-04 | 0.668084e-04 |
| 감소량 (%)                      |              | 85.38%       |

Result Example



Thank you  
감사합니다

06

# Appendix

부록



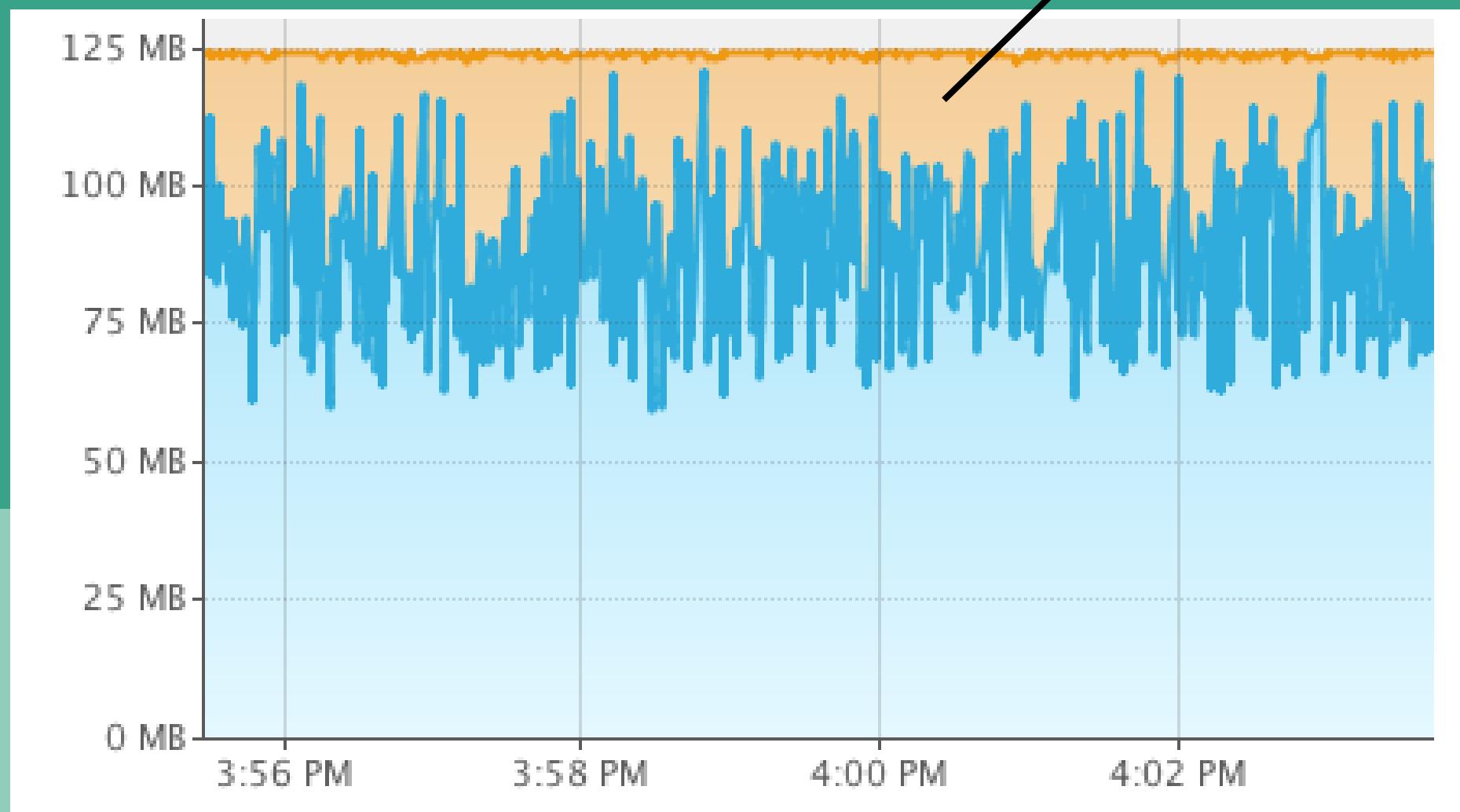
# 그린화 알고리즘 세부 사항

$$C = t \times (n_c \times P_c \times u_c + n_m \times P_m) \times \text{PUE} \times \text{CI} \times 0.001$$

| 기호             | 설명   | 단위                 | 비고   |
|----------------|--|--------------------|--|
| C              | Carbon Impact<br>(탄소 배출량)                    | gCO <sub>2</sub> e | $C = E \times CI$  |
| E              | 전체 에너지 소비량                                   | kWh                | -  |
| t              | Runtime<br>(사용자 코드 실행 시간)                    | h                  | 실제로 사용자 코드를 실행할 때에는 런타임이 ms 단위로 굉장히 짧은 경우가 대부분이지만, 계산 시 h로 변환하여 사용해야 함 |
| n <sub>c</sub> | 사용하는 코어 수                                    | -                  | 본 시스템에서는 사용자 코드를 실행하기 위해 하나의 코어를 isolation하여 사용함                       |
| P <sub>c</sub> | Thermal Design Power<br>(TDP - CPU 전력 사용 비율) | Watt               | CPU 모델마다 제조 회사에서 상수값으로 제공함   |

|                |   |                        |   |
|----------------|---|------------------------|---|
| u <sub>c</sub> | CPU 사용 비율                                   | %                      | 사용한 전체 코어에 대한 평균값으로 사용하나, 실시간으로 추적하기 힘든 관계로 1로 설정                                   |
| n <sub>m</sub> | 사용 가능한 메모리 사이즈                              | GB                     | 사용자 코드를 실행할 때의 메모리 Peak 사용량을 n <sub>m</sub> 으로 취급하며, 이를 위해 프로세스 status의 MAXRSS 값 사용 |
| P <sub>m</sub> | 메모리 사용량에 따른 전력 소비 비율                        | Watt/GB                | 다양한 요인에 영향을 받으나, 이전 실험들을 기반으로 간단하게 0.3725 W/GB로 설정                                  |
| PUE            | Power Usage Effectiveness (데이터 센터의 에너지 효율성) | -                      | 각 데이터 센터의 PUE에 대한 값이 제공됨  |
| CI             | Carbon Intensity<br>(탄소 집약도)                | gCO <sub>2</sub> e/kWh | 지역에 따라 다른 값을 가짐   |

# 그린화 알고리즘



- Peak 메모리 사용량으로 근사하기 때문에 계산 오차 발생 가능한 점 인지

# 그린화 알고리즘

- CPU 사용량은 상수항으로 취급

$$C = t \times (n_c \times P_c \times u_c + n_m \times P_m) \times \text{PUE} \times \text{CI} \times 0.001$$

- **n\_c**: 사용한 CPU core 수 | **u\_c**: core usage
- **( n\_c x u\_c ) set to 1**

# 그린화 패턴 Java Runner

- struct rusage

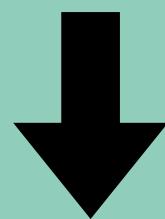
```
/* needed by wait4() */
struct rusage {
    struct timeval ru_utime;      // userspace time
    struct timeval ru_stime;      // system time (kernel time)
    long    ru_maxrss;           // 최대 physical memory 상주 size
```

- **Runtime = ru\_utime + ru\_stime**
- **Peak Memory usage = ru\_maxrss**

Backend

# Carbon Emission Calculator (Algorithm)

$$C = t \times \frac{(n_c \times P_c \times u_c + n_m \times P_m) \times \text{PUE} \times \text{CI} \times 0.001}{}$$

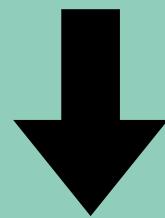


.env에 저장된  
상수값 대입

$$C = \underline{t} \times (8 + \underline{n_m} \times 0.3725) \times 0.5232$$

Runtime

Memory Usage



Carbon Emission