



JAVA 코드 탄소배출량 측정 &  
그린화 패턴 제시 플랫폼

## Team2

권동민 박수현 양현동  
이균서 이상수 정영준 최준열

# 목차

contents



개요



구현



시스템 구조



그린화 패턴(탐색, 검증)



팀프로젝트 진행

# Overview

About project

## 프로젝트 목표

- ✓ Java code를 입력받아 carbon footprint를 계산해주는 사이트 제작
- ✓ MVP 기능에 집중
- ✓ 다양한 기능보다는 보안성에 초점을 맞춤

## 핵심 기능

Input your code

```
1 public class After {  
2  
3     public static boolean equals(int[] arr, int[]  
4         boolean flag = true;  
5         for(int i=0; i<arr.length; i++) {  
6             if(arr[i] != arr2[i]) {  
7                 flag = false;  
8                 break;  
9             }  
10        }  
11    }  
12 }
```







Code input / formatting

Result

```
carbon emissions (gram) : 0.029476767169716925  
energy needed (kWh) : 0.00007092581128420819  
user time (sec) : 12.34  
cpu (core usage) : 1.09  
memory (KB) : 205804
```

코드 탄소배출량 계산

Result

 0.021682g CO2e Carbon footprint	 0.000052kWh Energy needed
 0.000124km in a passenger car	 0.000000% of a flight Paris-London
 0.000024 tree-	 0.000000%

계산 결과 화면

Output of the process

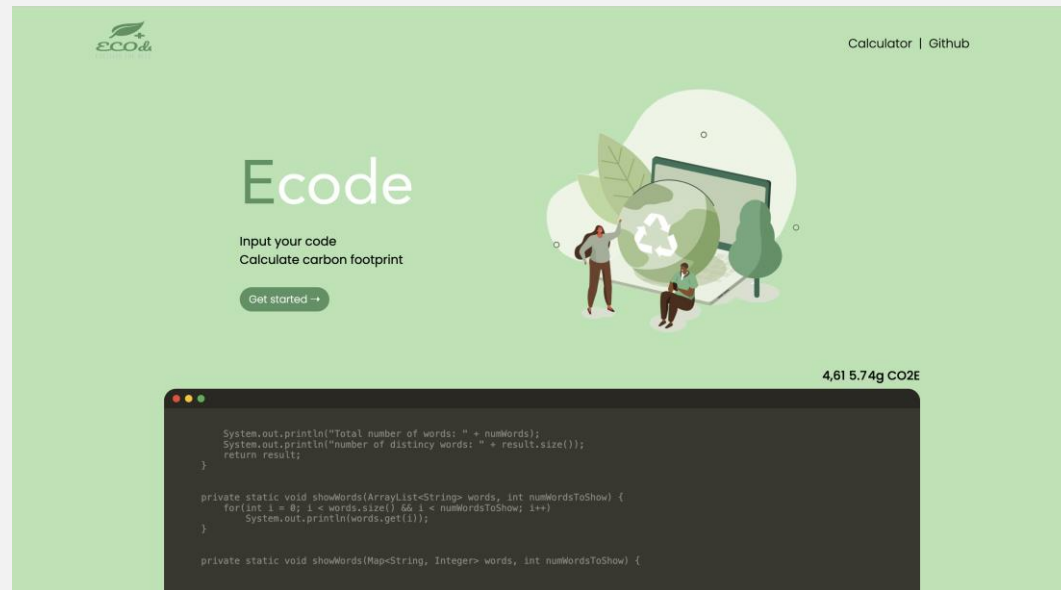
```
fin fin fin fin fin fin fin fin fin fin fin fin fin fin fin fin fin  
fin fin fin fin fin fin fin fin fin fin fin fin fin fin fin fin fin  
fin fin fin fin fin fin fin fin fin fin fin fin fin fin fin fin fin  
fin fin fin fin fin fin fin fin fin fin fin fin fin fin fin fin fin  
fin fin fin fin fin fin fin fin fin fin fin fin fin fin fin fin fin  
fin fin fin fin fin fin fin fin fin fin fin fin fin fin fin fin fin  
fin fin fin fin fin fin fin fin fin fin fin fin fin fin fin fin fin
```

Error of process

stdout, stderr 출력

# Implementation

Main page



Calculate formula

Main banner

```
Scanner fileScanner = new Scanner(new File(titles));
Stopwatch st = new Stopwatch();
st.start();
ArrayList<String> words = countWordsWithArrayList(fileScanner);
st.stop();
times[i] += st.time();
fileScanner.close();
```

## Green Calculate formula

The carbon footprint is calculated by estimating the energy draw of the algorithm and the carbon intensity of producing this energy at a given location:

$$\text{carbon footprint} = \text{energy needed} * \text{carbon intensity}$$

Where the energy needed is:

$$\text{runtime} * (\text{power draw for cores} * \text{usage} + \text{power draw for memory}) * \text{PUE} * \text{PSF}$$

The power draw for the computing cores depends on the model and number of cores, while the memory power draw only depends on the size of memory available. The usage factor corrects for the real core usage (default is 1, i.e. full usage). The PUE (Power Usage Effectiveness) measures how much extra energy is needed to operate the data centre (cooling, lighting etc.). The PSF (Pragmatic Scaling Factor) is used to take into account multiple identical runs (e.g. for testing or optimisation).

The Carbon Intensity depends on the location and the technologies used to produce electricity. But note that the "energy needed" indicated at the top of this page is independent of the location.


Copyright © 2023 Ecode. All rights reserved.

Twitter GitHub



# Implementation

Calculate page









 Calculator | Github

### Input your code

```
1 public class HelloSKKU {  
2  
3     public static void main(String[] args) {  
4         for (int i = 0; i < 10; i++) {  
5             System.out.println("Hello SKKU!");  
6             System.err.println("Bye SKKU!");  
7         }  
8     }  
9 }  
10
```

Format Calculate

### Result

 <b>0.000078g CO2e</b> Carbon footprint	 <b>0.000000kWh</b> Energy needed
 <b>0.000000km</b> in a passenger car	 <b>0.000000%</b> of a flight Paris-London
 <b>0.000000 tree-months</b> Carbon sequestration	 <b>0.04sec</b> user time
 <b>0.89 cores</b> cpu cores usage	 <b>34540KB</b> memory

#### Output of the process

```
Hello SKKU! Hello SKKU! Hello SKKU! Hello SKKU! Hello SKKU! Hello SKKU!  
Hello SKKU! Hello SKKU! Hello SKKU! Hello SKKU!
```

#### Error of process

```
Bye SKKU! Bye SKKU! Bye SKKU! Bye SKKU! Bye SKKU! Bye SKKU! Bye SKKU! Bye SKKU!  
Bye SKKU! Bye SKKU! Bye SKKU!
```

<https://ecode.life>

# System structure

Overall structure – tech stack

## Frontend



React Query



Motion



CodeMirror



Recoil

AXIOS

## Deploy



amazon  
S3



Amazon  
EC2



Route53



AWS  
CloudFront

ACM

gabia.

## Backend

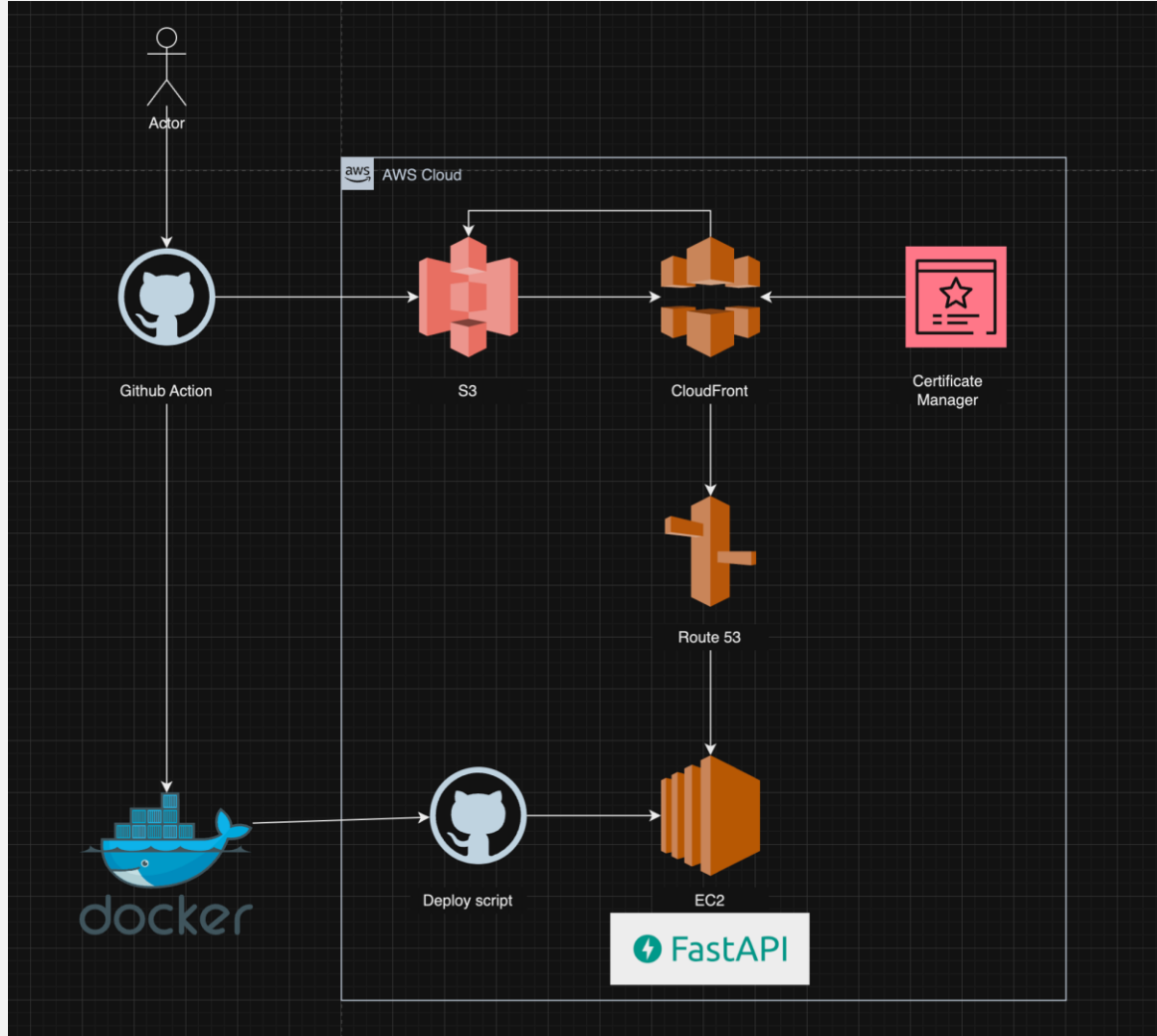


FastAPI



# System structure

Deployment diagram



## Frontend:

- AWS S3
- CloudFront
- Route53

## Backend:

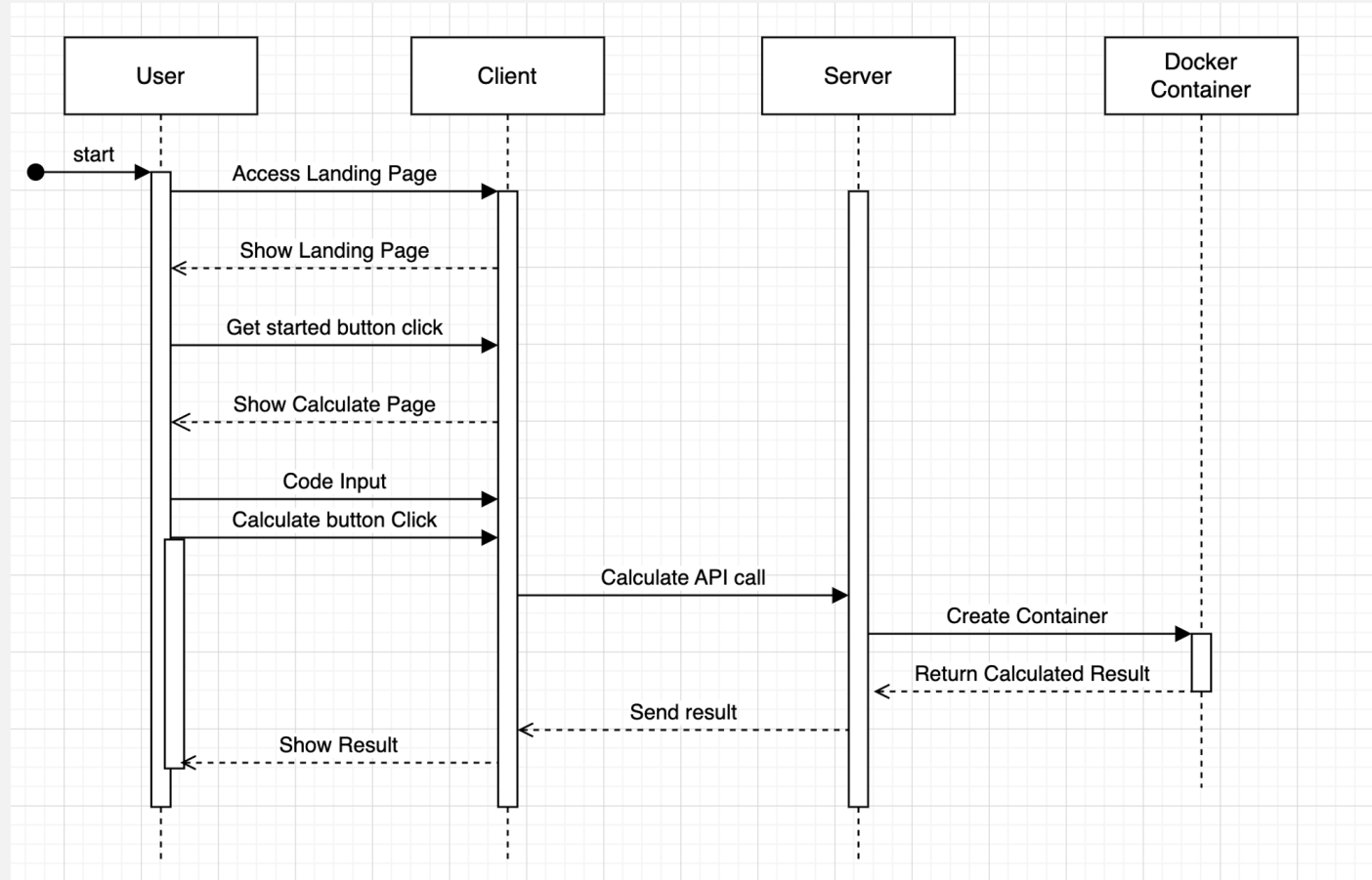
- AWS EC2
- Docker

## CI / CD:

- Github action

# System structure

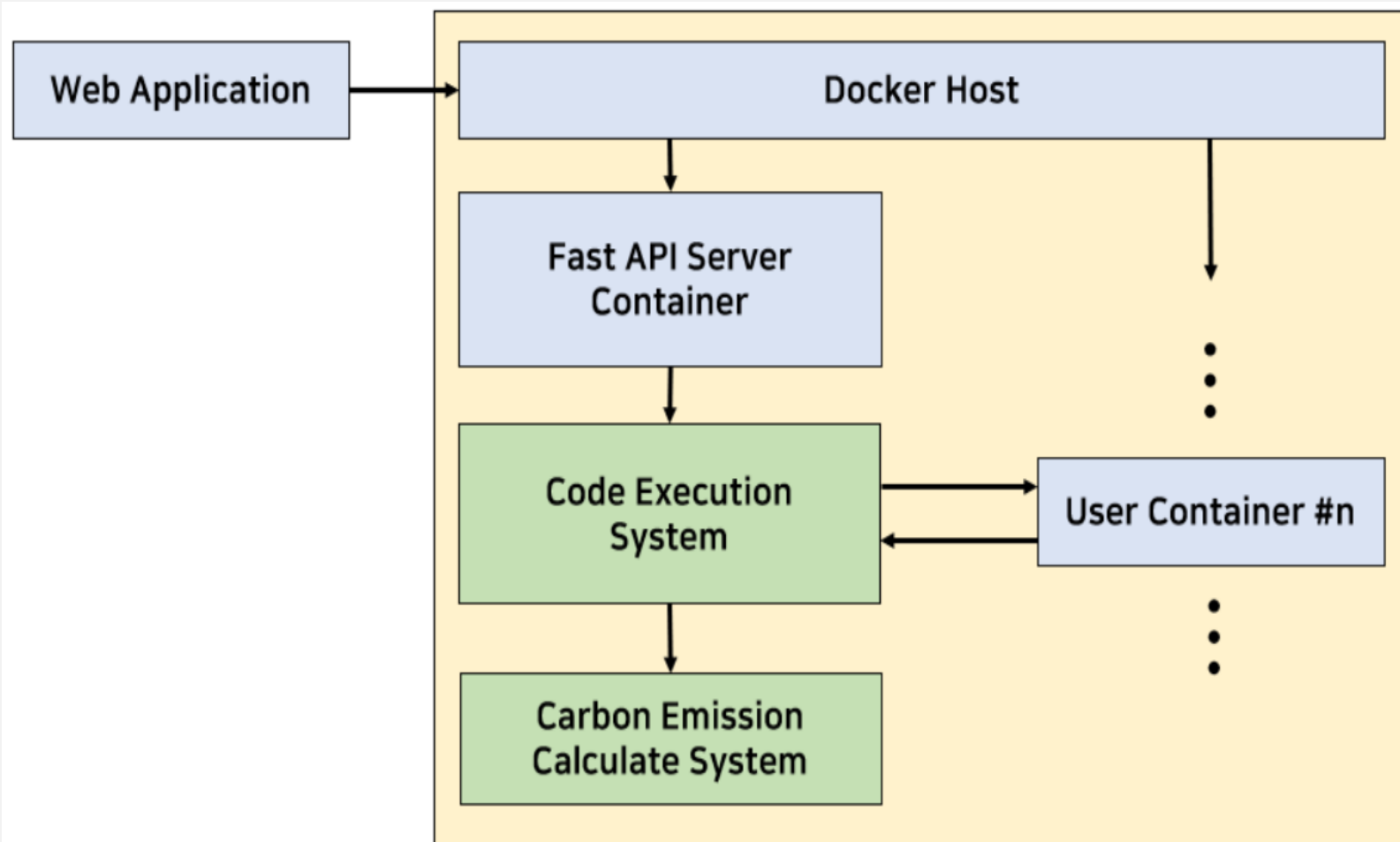
Overall structure – sequence diagram





# Backend structure

Backend diagram



- Main container와 user container로 분리
- 보안을 위한 환경분리
- 고유 id기반 container 생성
- Docker volume을 활용한 결과 저장 / 접근

# API docs

- Success Response:

- Code: 200

- Content:

name	type	unit	example	description
carbon_emissions	Number	gram	0.00007393544761014977	탄소 배출량
energy_needed	Number	kWh	1.7790049954318998e-7	코드 구동에 필요한 전기 에너지
user_time	Number	sec	0.03	구동 시간
cpu_core_use	Number	core usage	1.13	구동에 필요한 CPU 코어
memory_usage	Number	KB	40168	구동에 필요한 최대 메모리 (maximum resident set size 기준)  설명 참조: <a href="https://en.wikipedia.org/wiki/Resident_set_size#:~:text=In%20computing%20resident%20set%20size,in%20main%20memory%20(RAM).">https://en.wikipedia.org/wiki/Resident_set_size#:~:text=In computing%20resident set size,in main memory (RAM)</a>
compile_stderr	String	.	Syntax error, insert ';' to complete Statement.	컴파일 표준 에러 출력
runtime_stdout	String	.	HelloSKKU	런타임 표준 출력
runtime_stderr	String	.	Exception in thread "main" java.lang.ArithmeticException:	런타임 표준 에러 출력

## Carbon Emission Info API

Aa name

≡ URL Patterns

≡ Method



carbon\_emission\_calculate

/carbon\_emission\_calculate

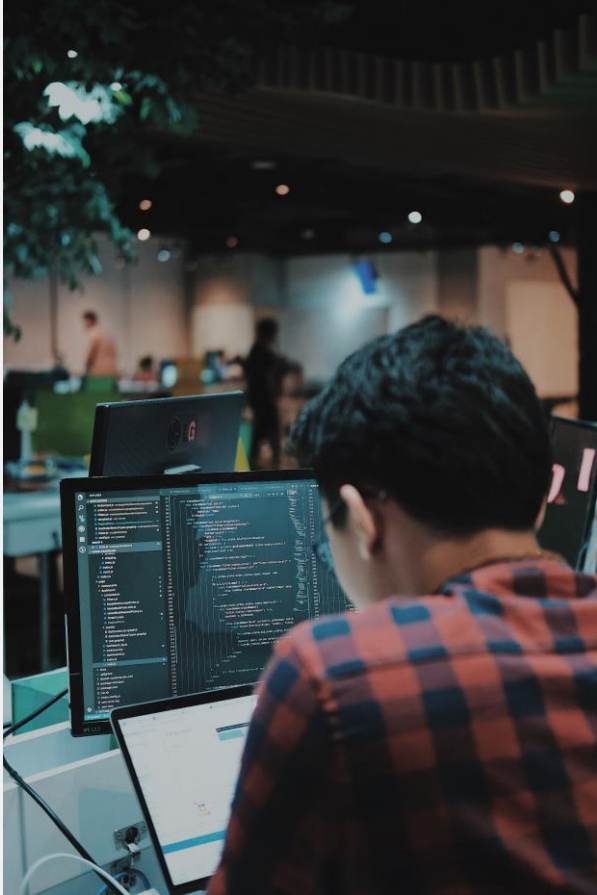
POST

## URL 패턴 및 content 정리

content 각 요소에 대한 name, type, unit 등 기록

# Green Pattern

how to collect? - 인터넷 검색



## Keywords

**코드 최적화 알고리즘**

시간복잡도 감소

공간복잡도 감소

Garbage Collection

## 검색 경로

**포털사이트 (Naver, Google 등)**

**알고리즘 관련 블로그**

**NAVER**

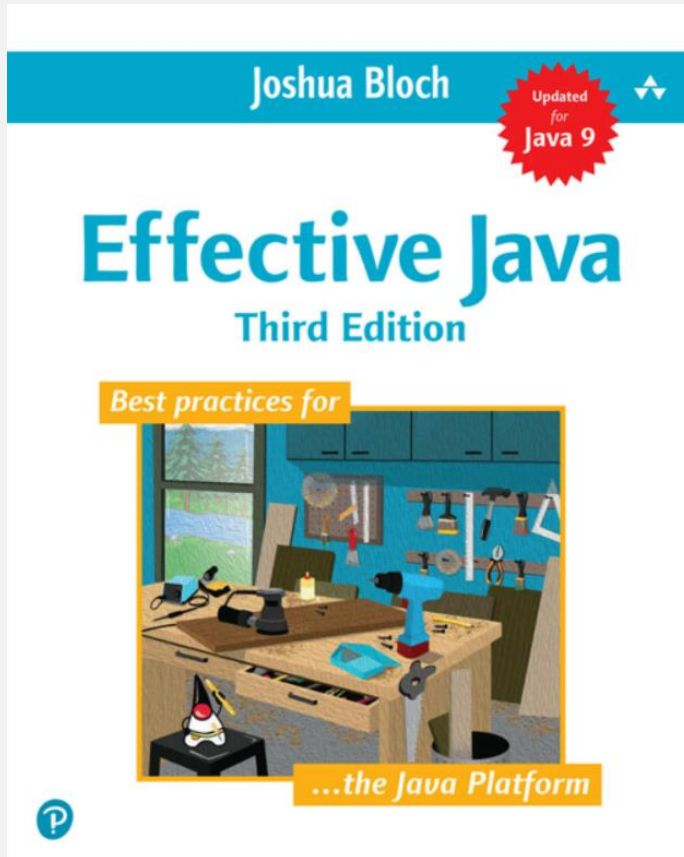
**Google**

 **TISTORY**

 **ChatGPT**

# Green Pattern

how to collect? - 책



For example, the `keySet` method of the `Map` interface returns a `Set` view of the `Map` object, consisting of all the keys in the map. Naively, it would seem that every call to `keySet` would have to create a new `Set` instance, but every call to `keySet` on a given `Map` object may return the same `Set` instance. Although the returned `Set` instance is typically mutable, all of the returned objects are functionally identical: when one of the returned objects changes, so do all the others, because they're all backed by the same `Map` instance. While it is largely harmless to create multiple instances of the `keySet` view object, it is unnecessary and has no benefits.

Another way to create unnecessary objects is *autoboxing*, which allows the programmer to mix primitive and boxed primitive types, boxing and unboxing automatically as needed. **Autoboxing blurs but does not erase the distinction between primitive and boxed primitive types.** There are subtle semantic distinctions and not-so-subtle performance differences (Item 61). Consider the following method, which calculates the sum of all the positive `int` values. To do this, the program has to use `long` arithmetic because an `int` is not big enough to hold the sum of all the positive `int` values:

```
// Hideously slow! Can you spot the object creation?
private static long sum() {
    Long sum = 0L;
    for (long i = 0; i <= Integer.MAX_VALUE; i++)
        sum += i;
    return sum;
}
```

This program gets the right answer, but it is *much* slower than it should be, due to a one-character typographical error. The variable `sum` is declared as a `Long` instead of a `long`, which means that the program constructs about  $2^{31}$  unnecessary `Long` instances (roughly one for each time the `long i` is added to the `Long sum`). Changing the declaration of `sum` from `Long` to `long` reduces the runtime from 6.3 seconds to 0.59 seconds on my machine. The lesson is clear: **prefer primitives to boxed primitives, and watch out for unintentional autoboxing.**

This item should not be misconstrued to imply that object creation is expensive and should be avoided. On the contrary, the creation and reclamation of small objects whose constructors do little explicit work is cheap, especially on modern JVM implementations. Creating additional objects to enhance the clarity, simplicity, or power of a program is generally a good thing.

Conversely, avoiding object creation by maintaining your own *object pool* is a bad idea unless the objects in the pool are extremely heavyweight. The classic

## Effective Java (Joshua Bloch)

자바 플랫폼의 모범적인 코드 가이드

효율적인 리소스 사용을 위한

다양한 코딩 예시 제공

Ex.

Use primitives type rather than boxed

# Green Pattern

how to collect? - PS 문제

코딩테스트 연습 > 2018 KAKAO BLIND RECRUITMENT > [1차] 비밀지도

도움말 컴파일 옵션

[1차] 비밀지도

dark light sublime vim emacs C++

문제 설명

비밀지도

네오는 평소 프로도가 비상금을 숨겨놓는 장소를 알려줄 비밀지도를 손에 넣었다. 그런데 이 비밀지도는 숫자로 암호화되어 있어 위치를 확인하기 위해서는 암호를 해독해야 한다. 다행히 지도 암호를 해독할 방법을 적어놓은 메모도 함께 발견했다.

- 지도는 한 번의 길이가  $n$  인 정사각형 배열 형태로, 각 칸은 "공백"(" ") 또는 "벽"("#") 두 종류로 이루어져 있다.
- 전체 지도는 두 장의 지도를 겹쳐서 얻을 수 있다. 각각 "지도 1"과 "지도 2"라고 하자. 지도 1 또는 지도 2 중 어느 하나라도 벽인 부분은 전체 지도에서도 벽이다. 지도 1과 지도 2에서 모두 공백인 부분은 전체 지도에서도 공백이다.
- "지도 1"과 "지도 2"는 각각 정수 배열로 암호화되어 있다.
- 암호화된 배열은 지도의 각 가로줄에서 벽 부분을 1, 공백 부분을 0 으로 부호화했을 때 얻어지는 이진수에 해당하는 값의 배열이다.

solution.cpp

```
1 #include <string>
2 #include <vector>
3
4 using namespace std;
5
6 vector<string> solution(int n, vector<int> arr1, vector<int> arr2) {
7     vector<string> answer;
8     return answer;
9 }
```

실행 결과

실행 결과가 여기에 표시됩니다.

빠른 A+B

시간 제한	메모리 제한	제출	정답	맞힌 사람	정답 비율
1 초 (여단 참고)	512 MB	290736	131027	106481	45.578%

문제

본격적으로 for문 문제를 풀기 전에 주의해야 할 점이 있다. 입출력 방식이 느리면 여러 줄을 입력받거나 출력할 때 시간초과가 날 수 있다는 점이다.

C++을 사용하고 있고 `cin/cout` 을 사용하고자 한다면, `cin.tie(NULL)` 과 `sync_with_stdio(false)` 를 둘 다 적용해 주고, `endl` 대신 개행문자(`\n`)를 쓰자. 단, 이렇게 하면 더 이상 `scanf / printf / puts / getchar / putchar` 등 C의 입출력 방식을 사용하면 안 된다.

Java를 사용하고 있다면, `Scanner` 와 `System.out.println` 대신 `BufferedReader` 와 `BufferedWriter` 를 사용할 수 있다. `BufferedWriter.flush` 는 맨 마지막에 한 번만 하면 된다.

Python을 사용하고 있다면, `input` 대신 `sys.stdin.readline` 을 사용할 수 있다. 단, 이때는 맨 끝의 개행문자까지 같이 입력받기 때문에 문자열을 저장하고 싶을 경우 `.rstrip()` 을 추가로 해 주는 것이 좋다.

또한 입력과 출력 스트림은 별개이므로, 테스트케이스를 전부 입력받아서 저장한 뒤 전부 출력할 필요는 없다. 테스트케이스를 하나 받은 뒤 하나 출력해도 된다.

자세한 설명 및 다른 언어의 경우는 [이 글](#)에 설명되어 있다.

[이 블로그 글](#)에서 BOJ의 기타 여러 가지 팁을 볼 수 있다.

## Programmers ,백준코딩

다양한 최적화 방법을 활용한 문제 제시

Ex.

비트연산자를 활용한 문제

각 언어별로 빠르게 I/O를 진행하는 방법


# Green Pattern


how to verify?


## Result


carbon emissions (gram) : 0.029476767169716925  
energy needed (kWh) : 0.00007092581128420819  
user time (sec) : 12.34  
cpu (core usage) : 1.09  
memory (KB) : 205804

## Result


 0.008548g CO2e  
Carbon footprint

 0.000021kWh  
Energy needed

 0.000049km  
in a passenger car

 0.000000%  
of a flight Paris-London

 0.000009 tree-months  
Carbon sequestration

 4.02sec  
user time

 0.97 cores  
cpu cores usage

 191332KB  
memory

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

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 memory (Watt).  $PUE$  is the efficiency coefficient of the data centre.

그린 알고리즘 사이트에 적용된 공식을 활용하여  
carbon emission 계산

backend에서 실행하여 runtime, memory사용량 확인



# Green Pattern

how to test

```
import unittest

from green_algorithm import GreenAlgorithm, GreenAlgorithmConstants

class TestGreenAlgorithm(unittest.TestCase):
    def __init__(self, methodName: str = "runTest") -> None:
        super().__init__(methodName=methodName)
        self.ga = GreenAlgorithm(GreenAlgorithmConstants())
        self.data_dict = {
            "constants": self.gc,
            "memory": 64,
            "n_cpu_cores": 4,
            "psf": 1.0,
            "pue_used": 1.67,
            "runtime_hours": 1,
            "runtime_minutes": 0,
            "tdp": 45,
            "tdp_per_core": 11.3,
            "usage_cpu_used": 1.0,
        }
        self.ga = GreenAlgorithm(self.data_dict)
        self.ga_data = self.ga.get_green_algorithm_fields()
        self.ce_example = 47.91735008

        # lazy import for avoid blocking test
        try:
            import pprint
            self.print_func = pprint.pprint
        except ImportError:
            self.print_func = print

    def test_calculate_carbon_emission(self):
        ce = self.ga.calculate_carbon_emissions()
        self.print_func(ce)
        self.assertAlmostEqual(ce["carbon_emissions"], 47, delta=1)

    def test_calculate_tree_months(self):
        tm = self.ga.calculate_tree_months(self.ce_example)
        self.print_func(tm)
        self.assertAlmostEqual(tm, 0, delta=2)

    def test_calculate_driving_kilometers(self):
        dk = self.ga.calculate_driving_kilometers(self.ce_example)
        self.print_func(dk)
        self.assertAlmostEqual(dk["us_nkm"], 0.05, delta=0.001)

if __name__ == "__main__":
    unittest.main()
```

## Python unittest 사용

동일하게 input에 대해 다른 그린 알고리즘 구현체에서의 결과값과 직접 작성한 코드의 결과값의 오차가 허용범위내에 있는지 확인

탄소 배출량 측정, tree month로 배출량 치환, driving kilometer로 배출량 치환 함수로 나누어 테스트

## 잘 측정된건지 어떻게 확인?

unittest의 assertAlmostEqual 함수를 사용하여 허용 오차 범위를 넘으면 Error를 raise하도록 코드 작성

# 팀 프로젝트 진행 방식

역할 분배



## Frontend

권동민, 박수현, 이상수, 최준열



## Backend

양현동, 이균서, 정영준

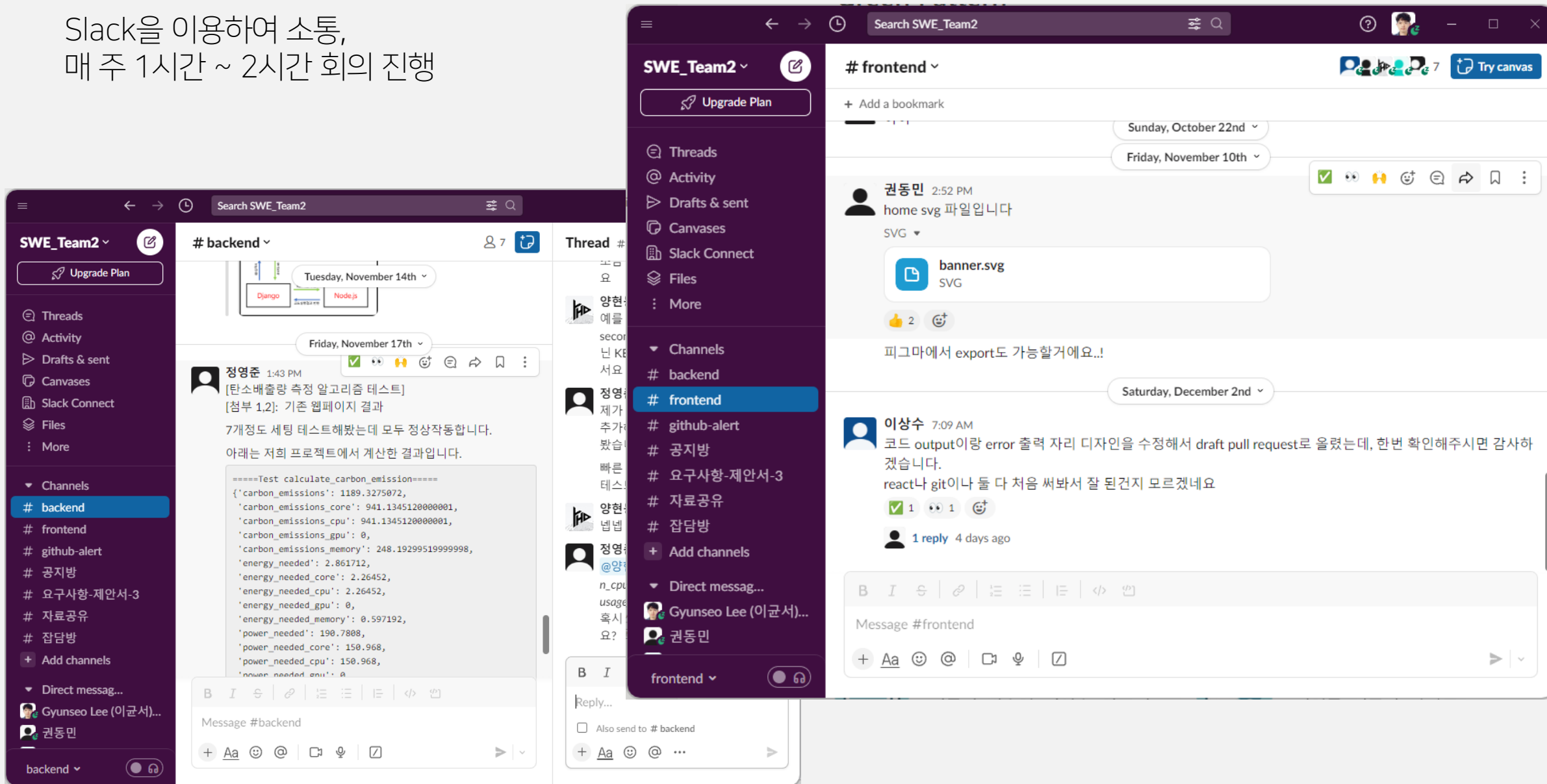


## Green Pattern

Everyone

# 팀 프로젝트 진행 방식

Slack을 이용하여 소통,  
매 주 1시간 ~ 2시간 회의 진행



# 팀 프로젝트 진행 방식

GitHub Issue 및 PR 적극 이용

<input type="checkbox"/>	5 Open	5 Closed	Author	Label	Projects	Milestones	Assignee	Sort
<input type="checkbox"/>	front에서 backend로 java code를 넘겨줄 때, java code를 format해서 넘겨주기	enhancement						
	#30 opened 2 weeks ago by gyunseo							
<input type="checkbox"/>	Docker 컨테이너 user별 볼륨 path 격리 문제	bug enhancement						
	#24 opened 3 weeks ago by Yanghyeondong							
<input type="checkbox"/>	Git wiki, readme 작성	documentation						
	#7 opened on Sep 18 by Fvesta							
<input type="checkbox"/>	Add boilerplate	enhancement						1
	#5 opened on Sep 18 by Fvesta 2 tasks done							
<input type="checkbox"/>	Git init settings	documentation enhancement						
	#1 opened on Sep 17 by Fvesta 5 of 6 tasks							

<input type="checkbox"/>	5 Open	20 Closed	Author	Label	Projects	Milestones	Reviews	Assignee	Sort
<input type="checkbox"/>	Revert "feat: add pipenv run command"	✓							
	#34 opened 2 weeks ago by gyunseo • Review required								
<input type="checkbox"/>	add "public" in regex	✓							3
	#33 opened 2 weeks ago by Yanghyeondong • Approved 4 tasks								
<input type="checkbox"/>	헤더 디자인 변경	✓							
	#27 opened 3 weeks ago by clapsh • Review required 4 tasks done								
<input type="checkbox"/>	Feature/update dockerfile for add carbon emission caculator	✓							
	#17 opened on Oct 12 by gyunseo 4 tasks done								
<input type="checkbox"/>	Add codeowner for auto reviewer	✓							
	#4 opened on Sep 18 by Fvesta • Draft 4 tasks								

# 팀 프로젝트 진행

개발 일정

7주차 | 요구사항 상세 분석

8주차 | 중간고사

9주차 | Backend API 구현

10주차 | Frontend 초안 구현

11주차 | 테스트

12주차 | 그린다 패턴 자료 수집 및 추가 기능 구현

13주차 | 추가 기능 구현(환경분리, animation 추가 등)

14주차 | 그린다 패턴 레포트 작성 및 추가 기능 수정

15주차 | 배포 / 최종 수정 및 발표

16주차 | 기말고사





A photograph showing a pair of hands cupped together, holding a small green plant with many leaves and a mound of dark soil. The background is a blurred forest floor with green foliage and brown leaves. A blue rectangular box with white text is overlaid on the right side of the image.

# 감사합니다

## **Team2**

권동민 박수현 양현동  
이균서 이상수 정영준 최준열