**소프트웨어 디자인 명세서**

**(Software Design Specification)**

탄소 배출량 측정 웹 서비스 개발

(Development of a web service for measuring carbon emissions)

2023년 11월 19일

성균관대학교

소프트웨어공학

Team. 9

2018312081 박태현

2018310169 김한얼

2021313747 이진혁

2019315441 이한준

2018312277 한상안

2020313088 한채윤

**목 차**

**1. 목적 (Purpose)**

1.1 독자층 (Readership)

1.2 범위 (Scope)

1.3 목표 (Objective)

1.4 문서 구조 (Document Structure)

**2. 소개 (Introduction)**

2.1 목표 (Objectives)

2.2 Applied Diagrams

2.2.1 Used Tools

2.2.2 Use Case Diagram

2.2.3 Data Flow Diagram

2.2.4 Class Diagram

2.2.5 Sequence Diagram

2.2.6 Entity Relationship Diagram

2.2.7 Project Scope

2.2.8 References

**3. 시스템 아키텍처 – 전체 (System Architecture - Overall)**

3.1 목표 (Objectives)

3.2 시스템 구성 (System Organization)

3.2.1 System Diagram

3.2.2 Use Case Diagram

3.2.3 Data Flow Diagram

**4. 시스템 아키텍처 – 프론트엔드 (System Architecture - Frontend)**

4.1 목표 (Objectives)

4.2 구성 요소 (Components)

**5. 시스템 아키텍처 – 백엔드 (System Architecture - Backend)**

5.1 목표 (Objectives)

5.2 시스템 구조

5.3 하위 구성 요소 (Subcomponents)

5.3.1 Compilation & Execution System

5.3.2 Carbon Emissions Calculation System

**6. 프로토콜 디자인 (Protocol Design)**

6.1 목표 (Objectives)

6.2 AJAX

6.3 HTTP

6.4 REST

6.5 REST API

6.5.1 Issue session key

6.5.2 Generate Experiment

6.5.3 Load All Experiment

6.5.4 Query Single Experiment

6.5.5 Query Multiple Experiment

**7. 데이터베이스 디자인 (Database Design)**

7.1 목표 (Objectives)

7.2 ER Diagram

7.2.1 System

7.2.2 User

7.2.3 Experiment

**8. 테스팅 계획 (Testing Plan)**

8.1 목표 (Objectives)

8.2 Testing Policy

8.2.1 Development Testing

8.2.2 Release Testing

8.2.3 User Testing

8.3. Test Case

**9. 개발 계획 (Development Plan)**

9.1 목표 (Objectives)

9.2 Frontend Environment

9.2.1

9.3 Backend Environment

9.3.1

9.4 제약 사항 (Constraints)

9.5 전제 조건 및 의존성 (Assumptions and Dependencies)

**1. 목적 (Purpose)**

본 문서의 예상 독자들, 문서의 구조, 그리고 각 단원에 대해 설명한다.

**1.1 독자층 (Readership)**

본 문서는 본 시스템의 개발자들, 소프트웨어 공학개론 수업의 교수, 조교 그리고 참여 학생과 같은 독자들을 위해 만들어졌다.

**1.2 범위 (Scope)**

Python을 이용한 Java 코드의 탄소 배출량 계산 및 웹사이트 개발

**1.3 목표 (Objective)**

이 소프트웨어 설계문서의 주요 목적은 Java 코드의 탄소 배출량 계산 웹사이트 개발에 대한 설명이다. 이 문서는 실습 프로그램의 구현의 기반이 되는 소프트웨어 Front-end, Back-end, Database 측면에서의 설계를 정의한다. 모든 설계는 앞서 제작된 Software Requirements Specification 문서의 요구사항을 기반으로 작성되었다.

**1.4 문서 구조 (Document Structure)**

1) Purpose: 본 문서의 목적, 예상 독자 및 문서의 구조에 대해 설명한다.

2) Introduction: 본 문서를 작성하는데 사용된 도구들과 다이어그램들, 참고 자료들에 대해 설명한다.

3) Overall System Architecture: 시스템의 전체적인 구조를 Context Diagram, Use Case Diagram, Data Flow Diagram을 이용하여 서술한다.

4) System Architecture – Frontend: Frontend 시스템의 구조를 Class Diagram, Sequence Diagram을 이용하여 서술한다.

5) System Architecture – Backend: Backend 시스템의 구조를 Class Diagram, Sequence Diagram을 이용하여 서술한다.

6) Protocol Design: 클라이언트와 서버의 Communication을 프로토콜 디자인으로 서술한다.

7) Database Design: 시스템의 Database Requirements를 기반으로 ER Diagram, Relation Schema를 이용하여 시스템의 데이터베이스 디자인을 서술한다.

8) Testing Plan: 시스템을 위한 테스트 계획을 서술한다.

9) Development Plan: 시스템의 구현 계획 및 구현을 위한 개발 도구, 라이브러리 등 개발 환경을 설명한다.

**2. 소개 (Introduction)**

Design Document는 프로젝트 구현에 있어서 기반이 될 수 있는 설계를 제공한다. 또한 설계는 앞서 제작된 Software Requirements Specification 문서에서 명시된 요구사항을 따른다.

**2.1 목표 (Objectives)**

이번 챕터에서는 제안한 시스템의 설계에 사용된 다이어그램, 툴에 대해 설명하고 개발 범위에 대해 설명한다.

**2.2 Applied Diagrams**

**2.2.1 Used Tools**

Microsoft PowerPoint: 발표용 자료 제작 프로그램인 PowerPoint를 통해 기본적인 다이어그램을 그릴 수 있다.

EdrawMax: UML 다이어그램 툴로, 복잡한 다이어그램을 그릴 수 있다.

**2.2.2 Use Case Diagram**

Use Case Diagram은 시스템에서 제공해야 하는 기능이나 서비스를 명세화한 다이어그램이다. 사용자와 Use Case 간의 관계를 보여주며, 사용자 - 시스템 간 상호작용을 표현한다. User 접속 이후 탄소 배출량 계산까지의 과정을 표현하기 위해 사용되었다.

**2.2.3 Data Flow Diagram**

데이터 플로우 다이어그램은 시스템이나 프로세스 내에서 데이터의 흐름을 시각적으로 표현하는 다이어그램이다. 시스템 구조를 표현하기 위해 사용되었다.

**2.2.4 Class Diagram**

Class Diagram은 시스템을 구성하는 클래스, 그 속성, 기능 및 객체들 간의 관계를 표현하여 시스템의 정적인 부분을 보여준다. 프론트엔드와 백엔드의 구조를 표현하기 위해 사용되었으며, 실제로 구현될 소스코드와는 다를 수 있고 의미나 해석 또한 경우에 따라 달라질 수 있다.

**2.2.5 Sequence Diagram**

Sequence Diagram은 시간 순서로 행동 별로 어떤 객체와 어떻게 상호작용을 하는지 표현하는 Diagram이다. Event Diagram이라고도 부르는데, 시나리오와 관련된 객체와 시나리오의 기능 수행에 필요한 객체 간에 교환되는 메시지를 순서대로 표현한다. 프론트엔드와 백엔드를 표현하기 위해 사용되었다.

**2.2.6 Entity Relationship Diagram**

ER Diagram은 구조화된 데이터와 그들 간의 관계를 사람이 이해할 수 있는 형태로 표현하는 다이어그램이며, 현실에서의 요구사항들을 이용한 데이터베이스 설계과정에서 활용된다. 해당 다이어그램은 Entity, Attribute, Relationship으로 구성된다. 이 다이어그램을 통해 데이터베이스의 논리적 구조를 설명한다.

**2.2.7 Project Scope**

본 문서에서 설계하는 Java 코드 탄소 배출량 계산 웹 서비스는 사용자들이 입력한 코드를 실행하고, 실행 과정에서 발생하는 탄소 배출량을 계산하여 전달하기 위해 만들어졌다.

**2.2.8 References**

• IEEE Std 830-1998 IEEE Recommended Practice for Software Requirements

• Specifications, In IEEEXplore Digital Library

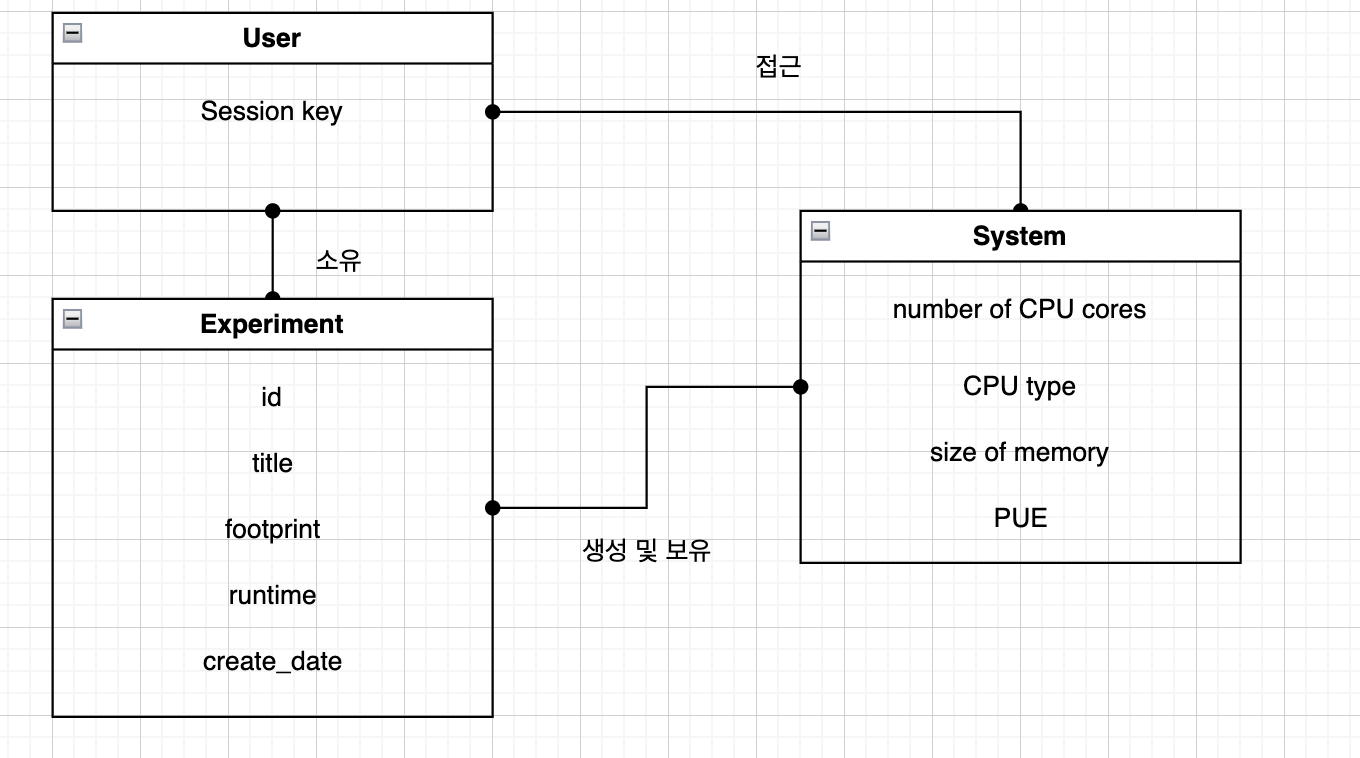
**3. 시스템 아키텍처 – 전체 (System Architecture - Overall)**

**3.1 목표 (Objectives)**

이 챕터에서는 프론트 엔드 설계에서 백 엔드 설계에 이르는 웹 서비스 시스템 구성에 대해 설명한다.

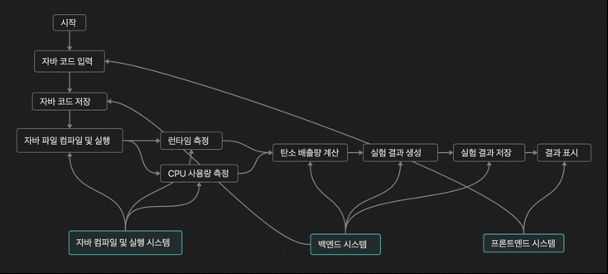
**3.2 시스템 구성 (System Organization)**

이 서비스는 클라이언트-서버 모델을 적용하여 설계되었으며, 프론트 엔드는 사용자와의 모든 상호작용을 담당한다. 프론트 엔드와 백 엔드는 JSON 기반의 HTTP 통신을 통해 데이터를 주고받는다. 백 엔드는 사용자로부터 Java 코드를 전달받고 이를 실행하여 얻은 탄소 배출량 결과값을 다시 내보낸다.



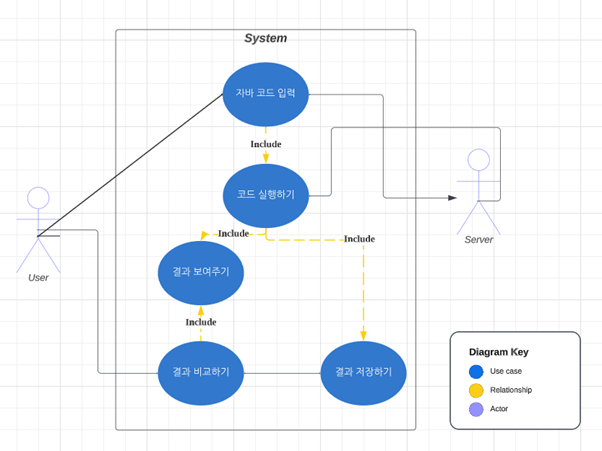
**Figure 3.2**: System Organization Diagram - Carbon Emissions Calculation System

**3.2.1 System Diagram**



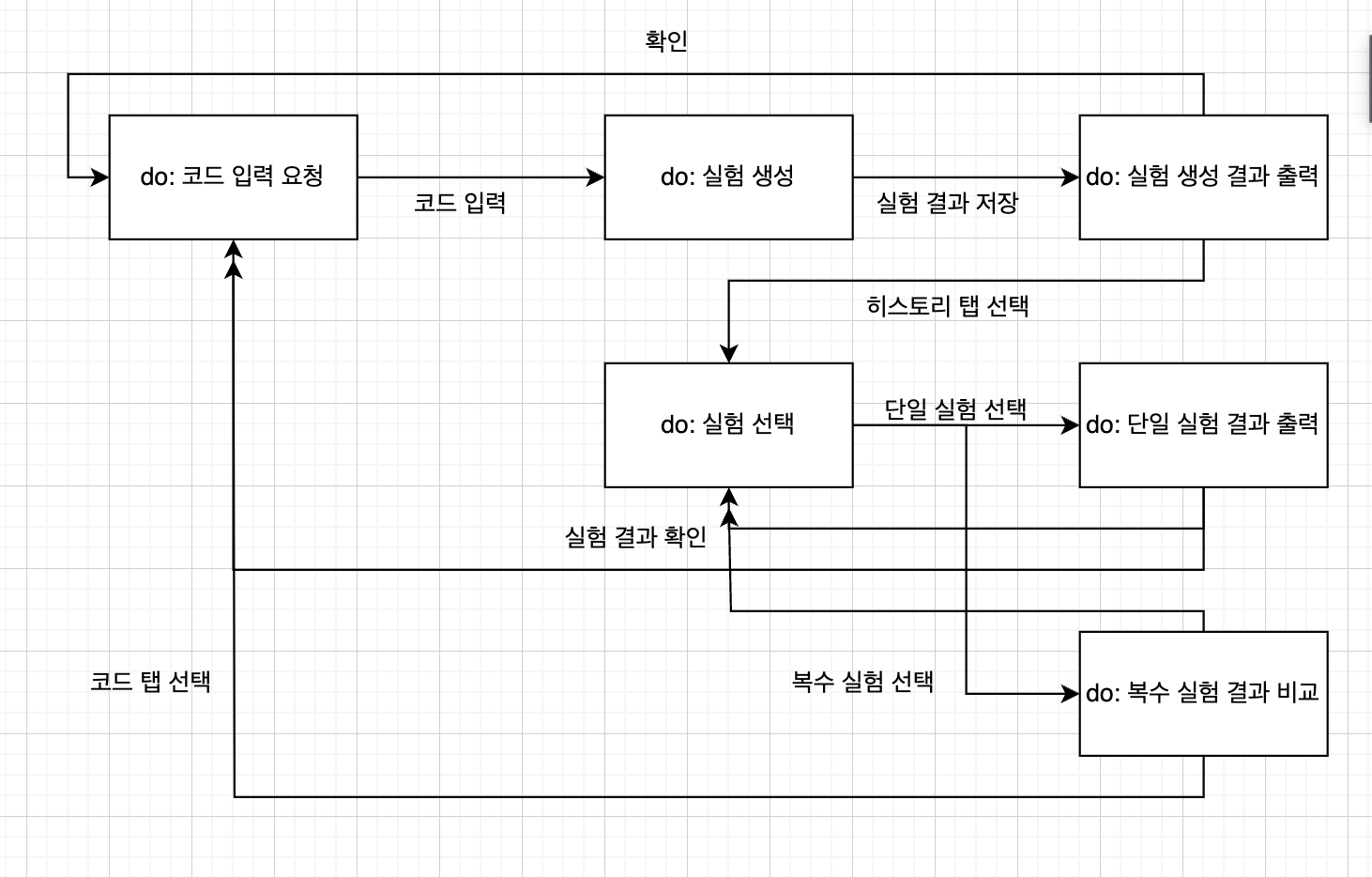
**Figure 3.2.1**: System Diagram - Carbon Emissions Calculation System

**3.2.2 Use Case Diagram**



**Figure 3.2.2**: Use Case Diagram - Carbon Emissions Calculation System

**3.2.3 Data Flow Diagram**



**Figure 3.2.3**: Data Flow Diagram - Carbon Emissions Calculation System

**4. 시스템 아키텍처 – 프론트엔드 (System Architecture - Frontend)**

**4.1 목표 (Objectives)**

이 챕터는 Frontend 시스템 구조에 대해서 설명하고, 각 구성 요소들의 관계를 설명한다.

**4.2 구성 요소 (Components)**

사용자가 Java 코드의 탄소 배출량을 계산하기 위해 시스템에 접속했을 때 Front-end 시스템의 구조이다.

**User Objectives**

Session key: 일시적으로 부여되는 키 값. 유저를 구분하기 위해 사용

Java code: 유저가 입력할 자바 코드

**Frontend Objectives**

Pages: 각 페이지에 대한 정보

**Backend Objectives**

System environments: 시스템 환경에 대한 정보

**Methods**

Send java code(): 자바 코드를 backend 서버에 보내는 함수

Compare experiments(): 실험 결과를 비교하는 함수

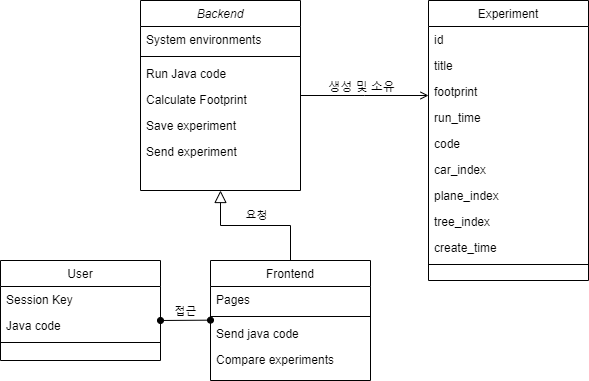
Run java code(): 자바 코드를 실행하는 함수

Calculate footprint(): 자바 코드를 실행한 결과를 토대로 탄소 배출량을 계산하는 함수

Save experiment(): 실험 결과를 저장하는 함수

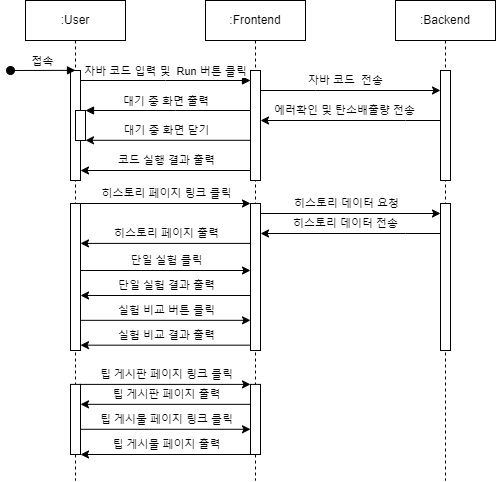
Send experiment(): frontend에 저장된 experiment를 보내는 함수

**Class Diagram**



**Figure 4.2.1**: Class Diagram - System Frontend

**Seqence Diagram**



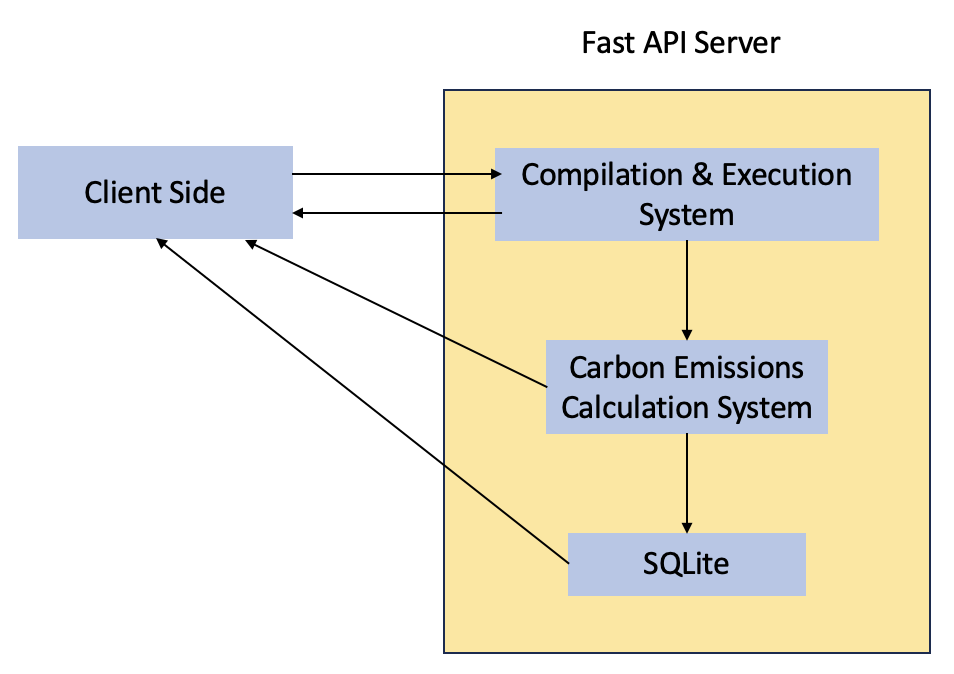
**Figure 4.2.1**: Sequence Diagram - System Frontend

**5. 시스템 아키텍처 – 백엔드 (System Architecture - Backend)**

**5.1 목표 (Objectives)**

이 챕터에서는 Backend 시스템 구조에 대해 기술한다.

**5.2 시스템 구조**



**Figure 5.2**: Overall Architecture

본 시스템의 구조는 다음과 같다. Back-end에서는 Client-side에서 받은 코드를 compilation & execution system에 넣고, 성공 시 carbon emissions calculation system을 통해 입력 코드의 탄소배출량을 출력으로 주고, 실패 시 컴파일 에러인지, 런타임 에러인지 알려준다. 또한 Client-side에서 접속 시 세션키를 발급해주고, 탄소배출량과 세션키를 함께 저장한다. 이후 Client-side에서 세션키를 통해 히스토리 조회가 가능하다.

**5.3 하위 구성 요소 (Subcomponents)**

**5.3.1 Compilation & Execution System**

**Class Diagram**

텍스트, 스크린샷, 폰트, 번호이(가) 표시된 사진

자동 생성된 설명

**Figure 5.3.1**: Class Diagram - Compilation & Execution System

**Client Side**

코드와 코어개수, 코어타입, 코어모델 및 사용가능한 메모리 등의 시스템 정보를 입력하고, 해당 실험의 이름을 Compilation & Execution 시스템으로 보낸다.

**SQLite**

Client-side에서 접속 시 서버에서 세션키를 발급해준다. 이후 탄소배출량과 코드는 세션키와 함께 SQLite에 저장된다.

**Compilation & Execution System**

Client Side의 정보를 받고 코드를 실행시킨다. 코드를 실행하면서 코드의 Runtime을 재고, 코드 컴파일이나 실행이 실패할 경우 Error Type으로 Client Side에 어떤 에러가 발생했는지 알려준다.

**Sequence Diagram**

텍스트, 스크린샷, 폰트, 도표이(가) 표시된 사진

자동 생성된 설명

**Figure 5.3.1**: Sequence Diagram - Compilation & Execution System

**5.3.2 Carbon Emissions Calculation System**

**Class Diagram**

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

**Figure 5.3.2**: Class Diagram - Carbon Emissions Calculation System

**Compilation & Execution System**

Client Side에서 받은 시스템 정보와 runtime 정보를 Carbon Emissions Calculation System과 SQLite에 넘겨준다.

**Carbon Emissions Calculation**

넘겨받은 정보를 토대로 탄소배출량을 계산해서 Client Side에 계산한 실험의 이름과 탄소배출량을 넘겨준다.

**SQLite**

넘겨받은 정보를 접속한 Client의 세션키와 함께 저장해서, 이후 Client-side에서 세션키와 함께 히스토리 요청 시, 히스토리를 제공해준다.

**Client Side**

넘겨받은 탄소배출량을 토대로 사용자에게 Car Carbon Emissions, Airplane Carbon Emissions, Tree Carbon Purification등의 정보 제공과 함께 생성된 시간을 기록한다.

**Sequence Diagram**

텍스트, 스크린샷, 폰트, 라인이(가) 표시된 사진

자동 생성된 설명

**Figure 5.3.2**: Sequence Diagram - Carbon Emissions Calculation System

**6. 프로토콜 디자인 (Protocol Design)**

**6.1 목표 (Objectives)**

이 챕터는 Front-end 애플리케이션과 서버가 상호작용하는 프로토콜에 대해 서술한다. 또한 Protocol을 바탕으로 각 인터페이스가 어떻게 정의되어 있는지 기술한다.

**6.2 AJAX**

Asynchronous JavaScript And Xml(비동기식 자바스크립트와 xml)의 약자로서, 자바스크립트를 이용해 서버와 브라우저가 비동기 방식으로 데이터를 교환할 수 있는 통신 기능이다. 브라우저가 가지고 있는 XMLHttpRequest 객체를 이용해서 전체 페이지를 새로 고치지 않고도 페이지의 일부만을 위한 데이터를 Load하는 기법이다.

**6.3 HTTP**

HTTP란 HyperText Transfer Protocol의 약자로서 하이퍼텍스트 방식의 정보를 교환하기 위한 하나의 규칙이다. 즉 HTML과 같은 문서를 전송하기 위해 사용되면 OSI 7계층에서 응용계층에 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. 이번 프로젝트에서 백엔드와 프론트엔드 사이에서는 JSON 형식의 문서를 주고받는다.

**6.4 REST**

REST(Representational State Transfer)는 월드 와이드 웹과 같은 분산 하이퍼미디어 시스템을 위한 소프트웨어 아키텍처의 한 형식이다. REST는 기본적으로 웹의 기존 기술과 HTTP 프로토콜을 그대로 활용하기 때문에 웹의 장점을 최대한 활용할 수 있는 아키텍처 스타일이다. REST의 구체적인 내용으로는 HTTP URI(Uniform Resource Identifier)를 통해 자원(Resource)을 명시하고, HTTP Method(POST, GET, PUT, DELETE)를 통해 해당 자원에 대한 CRUD Operation을 적용하는 것을 의미한다.

**6.5 REST API**

REST API는 위의 기술된 REST를 기반으로 서비스 API를 구현한 것이다.

**6.5.1 Issue session key**

Request

|  |  |  |
| --- | --- | --- |
| Attribute | Detail | |
| Protocol | HTTP | |
| Method | POST | |
| Request Header | ga-session-key | NULL |
| Request body | None |  |
|  |  |
|  |  |
|  |  |

Response

|  |  |  |
| --- | --- | --- |
| Attribute | Detail | |
| Protocol | HTTP | |
| Success Code | 200 OK | |
| Failure Code | HTTP error code = 400(Bad request) | |
| Response Header | ga-session-key | Session key to access |
| Success  Response Body | status code | success status code |
| message | success message |
| Failure  Response Body | status code | failure status code |
| message | failure message |

**6.5.2 Generate Experiment**

Request

|  |  |  |
| --- | --- | --- |
| Attribute | Detail | |
| Protocol | HTTP | |
| Method | POST | |
| Request Header | ga-session-key | session key to access |
| Request body | code | Java code to calculate carbon footprint |
| title | name of experiment |

Response

|  |  |  |
| --- | --- | --- |
| Attribute | Detail | |
| Protocol | HTTP | |
| Success Code | 200 OK | |
| Failure Code | HTTP error code = 400(Bad request) | |
| Response Header | ga-session-key | Session key to access |
| Success  Response Body | status code | success status code |
| message | success message |
| experiment | result of experiment |
| Failure  Response Body | status code | failure status code |
| message | failure message |

Experiment

|  |  |
| --- | --- |
| id | Experiment ID |
| title | Experiment title |
| footprint | Carbon footprint result |
| run\_time | Running time of java code |
| car\_index | Conversion result of automobile carbon emissions |
| plane\_index | Conversion result of airplane carbon emissions |
| tree\_index | Conversion results of tree carbon purification |
| create\_time | Created time |

**6.5.3 Load All Experiment**

Request

|  |  |  |
| --- | --- | --- |
| Attribute | Detail | |
| Protocol | HTTP | |
| Method | GET | |
| Request Header | ga-session-key | session key to access |
| Request body | None |  |
|  |  |

Response

|  |  |  |
| --- | --- | --- |
| Attribute | Detail | |
| Protocol | HTTP | |
| Success Code | 200 OK | |
| Failure Code | HTTP error code = 400(Bad request) | |
| Response Header | ga-session-key | Session key to access |
| Success  Response Body | status code | success status code |
| message | success message |
| experiment\_list[ ] | list of user's experiment |
| Failure  Response Body | status code | failure status code |
| message | failure message |

Element of experiment\_list

|  |  |
| --- | --- |
| id | Experiment ID |
| title | Experiment title |
| footprint | Carbon footprint result |
| run\_time | Running time of java code |
| create\_time | Created time |

**6.5.4 Query Experiments**

Request

|  |  |  |
| --- | --- | --- |
| Attribute | Detail | |
| Protocol | HTTP | |
| Method | GET | |
| Request Header | ga-session-key | session key to access |
| Request body | exp\_id\_list | ID array of experiments |

Response

|  |  |  |
| --- | --- | --- |
| Attribute | Detail | |
| Protocol | HTTP | |
| Success Code | 200 OK | |
| Failure Code | HTTP error code = 400(Bad request) | |
| Response Header | ga-session-key | Session key to access |
| Success  Response Body | status code | success status code |
| message | success message |
| exp\_data\_list[] | experiment data list |
| Failure  Response Body | status code | failure status code |
| message | failure message |

Element of exp\_data\_list

|  |  |
| --- | --- |
| id | ID of Experiment |
| title | title of experiment |
| code | code of experiment |
| footprint | Carbon footprint result list of experiment |
| run\_time | Running time list |
| car\_index | automobile carbon emissions list |
| plane\_index | airplane carbon emissions list |
| tree\_index | tree carbon purification list |
| Create\_time\_list | Created time list |

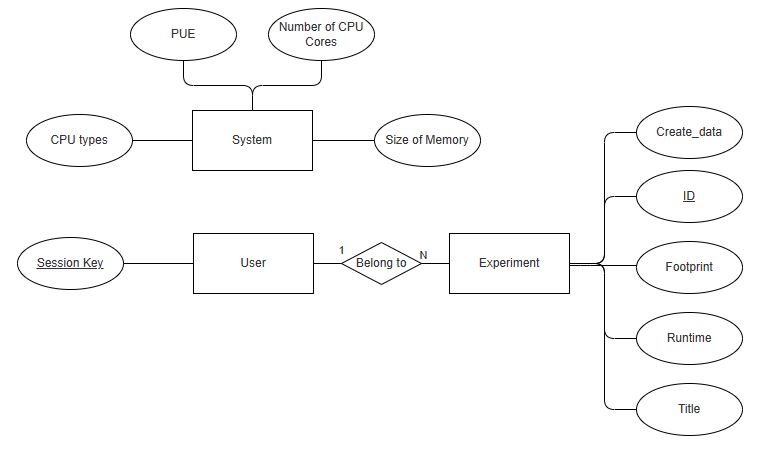
**7. 데이터베이스 디자인 (Database Design)**

7.1 목표 (Objectives)

7장에서는 시스템 데이터 구조와 이러한 구조가 데이터베이스로 어떻게 구현되었는지에 대해 설명한다. ER 다이어그램을 통해 Entity와 그 관계를 식별한다.

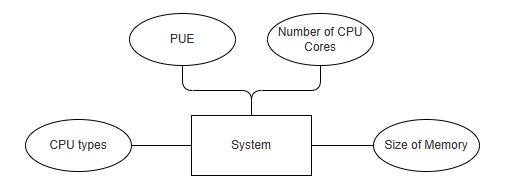
**7.2 ER Diagram**

본 웹 서비스 시스템은 총 3가지 Entity로 이루어져 있다; System, User, Experiment. ER-Diagram은 Entity간의 관계, 그리고 Entity와 Attribute의 관계를 다이어그램으로 설명한다. 각 Entity의 Primary Key는 밑줄로 표시되어 있다. 각 Entity마다 대응되는 개수는 Entity를 연결하는 선 주변에 표기되어 있어 확인할 수 있다.

****

**Figure 7.2**: ER Diagram

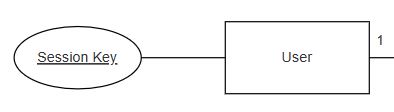
**7.2.1 System**



**Figure 7.2.1**: ER Diagram - System

System Entity는 웹 서버 시스템에 해당한다. System Entity는 CPU types, PUE, Number of CPU Cores, Size of Memory를 Attribute로 가지고 있다. 해당 내용들은 Java 코드의 CPU 사용량 계산을 위해 사용자가 기입한 정보이다.

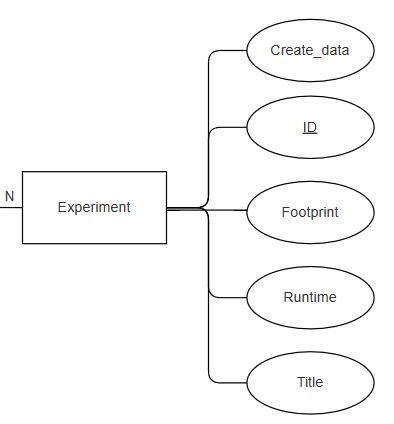
**7.2.2 User**



**Figure 7.2.2**: ER Diagram - User

User Entity는 웹 서비스 사용자에 해당한다. User Entity는 Session Key를 Attribute로 가지며, Session Key가 Primary Key이다. User Entity는 Experiment Entity와 1:N 관계를 갖는데, 이는 사용자가 1개 이상의 실험을 진행할 수 있다는 의미이다.

**7.2.3 Experiment**



**Figure 7.2.1**: ER Diagram - Experiment

Experiment Entity는 사용자의 입력을 받아 실행된 실험에 해당한다. Create\_data, ID, Footprint, Runtime, Title을 Attribute로 가지며, ID를 Primary Key로 가지고 있다. User Entity와 1:N 관계를 가지고 있다.

**8. 테스팅 계획 (Testing Plan)**

**8.1 목표 (Objectives)**

Testing plan에서는 Development Testing, Release Testing 및 User Testing에 대한 계획을 설명한다. 또한, 테스트 환경과 테스트 케이스에 대한 설명을 포함한다.

이러한 테스트의 목적은 프로젝트의 요구사항이 모두 충족되었는지 확인하며, 테스트를 통해 발견된 결함과 문제를 식별하고 해결하기 위하여 Web service의 안정성을 향상시킨다.

결론적으로, 완성도 높은 웹서비스를 User에게 제공하도록 한다.

**8.2 Testing Policy**

**8.2.1 Development Testing**

Development Testing은 목표하고자 하는 시스템의 개발 과정에서 발생 가능한 오류를 찾고 수정하기 위한 목적을 가지고 있다. 이 단계에서는 개발 프로세스 중 기능과 보안 등을 검증하기 때문에 결함을 초기에 발견하고 수정할 수 있다. 따라서, Development Testing을 통해서 개발 프로세스의 효율성과 품질을 향상시킬 수 있다. 이 프로젝트는 Frontend와 Backend의 긴밀한 협업이 요구되기 때문에 명확한 Testing 기준과 계획을 세워서 진행하는 것이 필요하다.

**Performance**

해당 Web service의 performance에 대한 평가 지표는 다음과 같다.

- Web Page 접속은 3초가 넘지 않게 한다.

- 사용자가 입력한 코드를 실행할 때, 에러 발생시 에러 메시지를 나타낼 수 있어야 한다.

- 계산된 결과는 5초 이내에 데이터베이스에 저장되어야 하며 히스토리 페이지에 나타낼 수 있어야 한다.

- 히스토리 페이지에서 선택한 테스트 결과를 비교 분석 페이지로 전달하는 데에는 5초가 넘지 않도록 한다.

**-** 최대 99개의 히스토리를 저장할 수 있도록 한다.

- 비교 분석은 최대 55개를 가지고 진행할 수 있게 한다.

다음과 같은 평가 지표를 만족할 수 있는지는 다양한 환경에서 테스트를 반복적으로 진행하여 확인할 수 있도록 한다.

**Security**

해당 Web Service는 세션이 유지되는 동안 사용자의 코드, 코드 분석 정보, 히스토리가 데이터베이스에 저장되어 있다. 세션이 종료된 후에는 데이터베이스에 저장된 정보를 지움으로써 사용자의 코드에 대한 정보에 대한 보안을 유지한다. 또한, 세션이 유지될 때에도 사용자 외에는 어떤 정보도 열람할 수 없게 데이터베이스 보안 수준을 지켜야 한다.

**8.2.2 Release Testing**

Release Testing은 Web service가 user에게 공개되기 전에 수행하는 테스트로, Web service의 전반적인 품질과 안정성을 확인한다. 배포 전 시스템의 성능, 보안 및 호환성을 평가하여 잠재적인 문제를 식별하고 수정하는 것을 목표로 한다. 우선, Data flow와 명세서를 통해 배포될 Web application의 주요 기능을 식별하고 배포될 소프트웨어 버전과 구성 요소를 명시한다. 그리고 다양한 Web server 환경을 설정하고 테스트 데이터를 준비하여 실제 환경에서 문제없이 Web service가 구동되는지 테스트한다.

테스트는 기본 기능을 우선적으로 배포하여 테스트한 후 단계별로 세부 기능을 배포하는 식으로 진행하여 피드백을 적극적으로 수용할 수 있게 한다. 각 테스트 단계별로 기능 작동에 오류가 없는지 확인하며 요구사항을 만족하는지 확인해야 한다. 단계별 배포가 진행될 때 팀원 모두가 Release Testing에 참여하여 다양한 환경에서 진행될 수 있게 하며 피드백을 받을 예정이다.

**8.2.3 User Testing**

User Testing에서는 Web service를 실제 user에게 제공하여 User의 피드백을 수용한다. User는 팀원을 포함하여 다양한 환경의 사람을 선정할 예정이며 기능과 user 인터페이스에 대한 평가를 거치며 실제 환경에서 발생할 수 있는 버그나 문제를 식별한다.

**8.3. Test Case**

해당 프로젝트는 Web Service인 만큼 Chrome, Firefox, Microsoft Edge 등 다양한 웹 브라우저에서 접속하여 시스템이 문제없이 작동하는지를 확인한다. 또한, 네트워크 환경도 Wi-Fi, 이더넷, LAN 등의 케이스를 통해 시스템의 안정성을 테스트한다. 탄소배출량을 계산하기 위한 코드는 다양한 경우를 시도할 예정이다. 예시로, 코드의 runtime이 지나치게 긴 경우, 코드 자체에 error가 발생하는 경우 등이 있다. 또한, 사용자가 입력한 코드의 탄소배출량을 저장한 히스토리와 분석에 대한 시나리오를 제작하여 테스트한 후 평가 보고서를 작성한다.

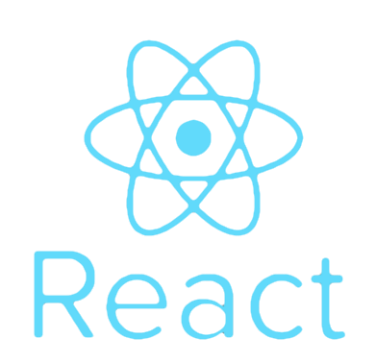
**9. 개발 계획 (Development Plan)**

**9.1 목표 (Objectives)**

해당 챕터에서는 시스템의 개발 환경 및 기술에 대하여 설명한다.

**9.2 Frontend Environment**

9.2.1 React



**Figure 9.2.1**: React Logo

React는 Facebook에서 개발한 JavaScript 라이브러리로, 사용자 인터페이스를 구축하기 위한 선언적이고 효율적인 방법을 제공합니다. 컴포넌트 기반 아키텍처를 기반으로 하며, 가상 DOM을 활용하여 빠른 렌더링을 지원합니다. React는 단일 페이지 애플리케이션(SPA)을 쉽게 구현하고, 재사용 가능한 UI 컴포넌트를 통해 개발 생산성을 높이는 데에 주로 사용됩니다.

9.2.2 JavaScript



**Figure 9.2.2**: JavaScript Logo

JavaScript는 객체 기반의 스크립트 프로그래밍 언어이다. 주로 웹 브라우저 내에서 사용하며, 다른 응용프로그램과의 내장 객체에도 접근할 수 있는 기능을 가지고 있다. 본 소프트웨어에서는 사용자가 입력하는 코드 내용의 송수신과 관련하여 해당 기능을 사용한다.

9.2.3 HyperText Markup Language (HTML)



**Figure 9.2.3**: HTML Logo

HTML은 웹 페이지를 위한 마크업 언어이며, W#C가 HTML과 CSS 표준의 공동 책임자이다. HTML을 통해 제목, 단락, 목록 등 본문을 위한 구조적 의미를 나타내는 것뿐만 아니라 링크, 인용과 그 밖의 항목으로 구조적 문서를 만들 수 있는 방법을 제공받을 수 있다. 이는 태그로 되어있는 HTML 요소 형태로 작성되며 웹 브라우저와 같은 HTML 처리 장치의 행동에 영향을 주는 JavaScript를 포함하거나 불러올 수 있다. 따라서 우리는 웹 서비스의 웹 페이지를 구현하기 위해 HTML을 사용한다.

9.2.4 Cascading Style Sheets (CSS)



**Figure 9.2.4**: CSS Logo

CSS는 마크업 언어가 실제 표시되는 방법을 기술하는 스타일 언어로, HTML과 XHTML에 주로 쓰이며, XML에서도 사용할 수 있다. W3C의 표준이며, 레이아웃과 스타일을 정의할 때의 자유도가 높다. 마크업 언어가 웹사이트의 몸체를 담당한다면, CSS는 옷과 악세서리처럼 꾸미는 역할을 담당한다. 우리는 웹사이트의 가시성과 심미성을 높이기 위하여 이를 사용한다.

**9.3 Backend Environment**

9.3.1 Github



**Figure 9.3.1**: Github Logo

Github는 소프트웨어를 개발하고 Git을 통해 버전을 관리하기 위해 사용되는 툴이다. 이를 통해 여러 명의 개발자가 하나의 프로젝트를 동시에 관리하며 개발할 수 있으며, 각각의 컴포넌트들을 통합하는데 이점이 있다. 따라서 우리는 본 웹 서비스 개발 및 버전 관리를 위해 이를 사용할 것이다.

9.3.2 FastAPI



**Figure 9.3.2**: FastAPI Logo

FastAPI는 Python 기반의 웹 프레임워크로, 빠르고 간편한 방식으로 웹 API를 개발할 수 있는 도구입니다. SQLite와 연동하여 경량 디스크 기반 데이터베이스를 사용할 수 있으며, 개발자들이 협업하고 코드를 관리하기 위해 GitHub을 통한 버전 관리를 지원합니다. FastAPI는 선언적인 형식의 코드 작성과 자동 API 문서 생성 등을 통해 효율적인 웹 서비스 개발을 가능하게 합니다. 우리는 웹 서비스 개발을 위해 이를 사용할 것이다.

9.3.3 SQLite



**Figure 9.3.3**: SQLite Logo

SQLite는 별도의 서버 프로세스가 필요 없고 SQL 질의 언어의 비표준 변형을 사용하여 데이터베이스에 액세스할 수 있는 경량 디스크 기반 데이터베이스를 제공하는 C 라이브러리이다. 일부 응용 프로그램은 내부 데이터 저장을 위해 SQLite를 사용할 수 있으며, SQLite를 사용하여 응용 프로그램을 프로토타입 한 다음 Oracle과 같은 더 큰 데이터베이스로 코드를 이식할 수도 있다. 우리는 본 웹 서비스를 사용하는 사용자의 코드 이력을 저장하기 위해 이를 사용할 것이다.

**9.4 제약 사항 (Constraints)**

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

• 기존에 널리 쓰이고 있는 기술 및 언어들을 사용한다.

• 사용자의 입력 및 코드를 실행하고 결과를 저장하는 시간은 5초를 넘기면 안된다.

• 로열티를 지불하거나 separate license를 요구하는 기술 및 software의 사용을 지양한다.

• 전반적인 시스템 성능을 향상시킬 수 있도록 개발한다.

• 사용자가 편리하게 이용할 수 있도록 개발한다.

• 가능한 오픈소스 소프트웨어를 사용한다.

• 시스템 비용과 유지보수 비용을 고려하여 개발을 진행한다.

• 시스템의 확장을 고려하여 개발을 진행한다.

• 시스템 자원의 낭비를 막을 수 있도록 소스 코드를 최적화한다.

• 소스 코드 작성시 유지보수에 대하여 고려하며 필요한 부분에 대해서는 주석을 통하여 이해하기 쉽도록 한다.

• 개발은 최소 윈도우 7 이상에서 이루어져야 하며 윈도우 10, 11 환경을 타겟으로 한다.

• 윈도우 10, 11 버전에서 실행한다.

**9.5 전제 조건 및 의존성 (Assumptions and Dependencies)**

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