전교생 Python 코딩 교육 시스템

Design specification

2022.05.14

Introduction to Software Engineering 41

TEAM 3

팀장 김지회

팀원 강신규

박상현

박상훈

엄지용

목차

1. Preface

1.1 Readership

본 문서는 개발팀이 열람하는 것을 상정하여 작성하였다. 사용자가 따르게 될 시스템의 구성 요소를 자연어 중심으로 표현하여, 개발팀이 이 시스템의 요구사항과 구성 요소를 쉽게 이해할 수 있도록 작성한다.

1.2 Version History

2022-5-12, Version 1.0, 팀원 회의 후 최초 버전 발행, Jihoi Kim, et al.

1.3 Document Structure

(1) Intruduction

이 장에서는 본 서비스의 설계에 사용된 다이어그램과 도구를 소개하고, 시스템의 개발 배경이 되는 문제의식과 이 시스템이 해당 문제들을 어떻게 해결할 것인지 기술한다.

(2) Overall System Architecture

이 장에서는 시스템과 각 서브시스템의 구조를 개략적으로 서술하고 서브시스템과의 연결과 전체적인 기능을 보여주는데 초점을 맞추었다.

(3) System Architecture – Frontend

이 장에서는 본 서비스에서 사용되는 언어에 대한 기술이 이뤄진다. NodeJS를 통해서 개발하기 때문에, NodeJS에 기반한 서버에서 정보를 받아와 프론트 엔드의 여러 함수들을 구현할 것인지 사용자에게 보여줄 내용이 담겨있다.

(4) System Architecture – Backend

이 장에서는 본 서비스에서 사용자와 직접적인 접촉을 하지 않는 백 엔드에 대한 내용을 담는다. Web-server, Docker를 포함한 백 엔드 시스템과 각 서브 시스템들이 연결된 구조를 보여준다. 주로 서버가 돌아가는 모형과 여러 서브 시스템을 표현한다.

(5) Protocol Design

프론트 엔드와 서버 간 상호작용이 어떤 방식으로 구동될 지에 대한 내용을 담았다. 본 서비스에서 어떤 방식으로 서버와 내용을 주고 받을지에 대한 내용을 보여준다. 또한 통신 상황에서 동일한 아이디 및 다른 권한으로 인해 발생하는 에러를 방지하기 위해 각 서브 시스템이 어떤 양식을 가지는지, URL과 관련된 내용은 어떻게 처리하는지, 에러 시 어떤 메시지를 발생시킬지를 보여준다.

(6) Database / SQL

관계형 데이터베이스 다이어그램과 이를 표현한 SQL을 서술한다. 또한, 각 데이터베이스의 값들의 속성과 자료형을 나타낸다.

(7) 읽기에 앞서

개발문서

추후 웹 서비스를 개발하기 위해 추가적인 함수 및 오브젝트가 생길 가능성이 있다.

2. Introduction

2.1 Applied Diagram

sequence diagram, entity-relation diagram, class diagram을 중점으로 사용하여 시스템을 기술하였다.

Sequence Diagram: Sequence Diagram 은 오브젝트 간 상호작용을 시간순으로 나타낸 도표이다. 사전에 작성한 시나리오를 바탕으로 다이어그램을 작성해 필요한 오브젝트와 기능을 식별하였다. 작성한 Sequence Diagram은 이후 Relation Diagram과 Class Diagram을 작성할 때에도 사용되며, 다른 다이어그램에서 보여주기 힘든 상호작용의 시간 순 진행을 보여주고, 추상적인 하나의 기능을 여러 단계로 나누어 생각할 수 있게 돕는다.

Entity-Relation Diagram: Entity-Relation Diagram은 여러 Entity 간의 관계와 각 Entity의 간단한 세부사항을 보여주는 다이어그램으로, 시스템에 어떤 Entity가 필요하며, 여러 Entity들이 어떤 관계를 가질지를 식별하기 위해 작성한다. Entity는 이전에 작성한 시나리오와 다른 다이어그램을 통해 식별해 나가며 다이어그램에 기술하며, 식별된 Entity는 필요한만큼 세분화하여 작성한다.

Class Diagram: Class Diagram은 시스템의 class, attribute, method와 object 간의 포함관계를 보여주는 다이어그램으로 시스템의 전체적인 구조를 서술하기 위해 작성한다. 가장 상위 Entity인 시스템을 위에 기술하며, 아래로 갈수록 직접적으로 실행되는 각각의 operation들을 기술한다.

2.2 System Overview

본 서비스는 Python 프로그래밍을 배우고자 하는 학생들을 위해 개발하였다. 프로그래밍에 대한 관심도와 그와 관련한 분야가 급격하게 성장하고 있으며, 그에 관련한 교육 시장도 급격하게 커지는 추세이다. 학생들의 흥미를 얻기 위해 쉬우면서도 널리 사용되는 python을 선정하였으며, 카카오톡 등 메신저 프로그램의 기초인 실시간 채팅 구현을 서비스 내부 최종 목표로 선정했다.

본 시스템은 브라우저에서 Python 프로그래밍을 연습할 수 있는 환경을 제공하여 컴퓨터 내

python 환경이 갖춰지지 않은 사람들, python 프로그래밍에 대한 기술이 없는 사람들까지 python 프로그래밍을 할 수 있도록 제공한다. 본 시스템은 기존 시스템에 비해 다운로드 및 환경 설정이 따로 필요 없다는 장점을 가짐으로써 시장 경쟁력을 높였다.

본 서비스는 새로운 사용자가 쉽게 적응할 수 있도록 직관적인 사용자 경험을 목표로 기획하였다. 이를 위해 브라우저 기반으로 사용자가 Python 프로그래밍을 연습할 수 있는 환경을 제공하며, 사용자가 서비스에서 제공하는 예제들을 풀이하고, 채점하면서 학습한 내용을 검증할 수 있다. 최종적으로 사용자가 python을 통한 실시간 채팅을 구현할 수 있다. 웹 서비스를 제공하기 위한 환경은 NodeJS 프레임워크를 활용하여 구현한다. NodeJS의 sqlite3 플러그인을 활용하여 본 서비스와 데이터베이스를 연결한다.

본 서비스의 공급자는 확장성을 고려한 설계를 바탕으로, 추후 다른 애플리케이션 구현 혹은 다른 언어에 대한 교육 플랫폼으로 활용할 수 있다.

본 시스템의 사용자들은 이를 통해 학습과 동시에 실습을 진행할 수 있기 때문에 타 프로그램을 동시에 돌리는 일 없이 편리한 사용자 경험을 얻을 수 있을 것이며, 이외에도 추후 다른 언어에 대한 교육 플랫폼으로 활용할 시 여러 개의 환경을 구현할 이유 없이 하나의 웹 환경에서 학습 및 실습을 할 수 있는 경험을 할 수 있을 것으로 기대한다.

3. System Architecture - Overall

3.1 System Organization

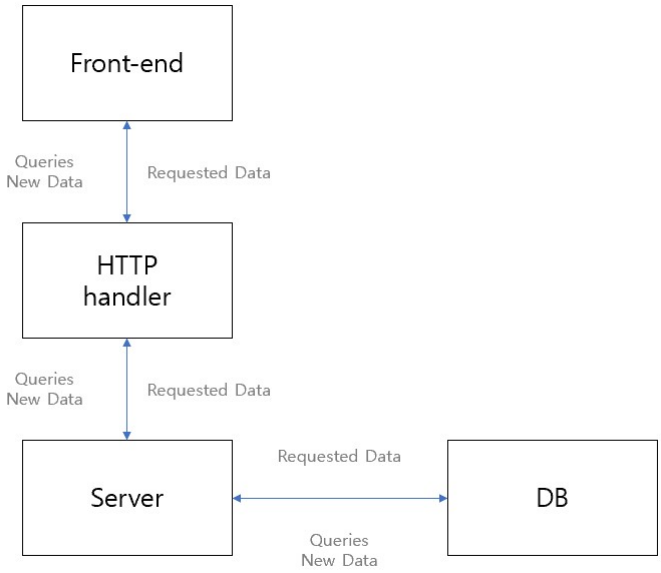

Diagram 3.1 System Organization

본 서비스의 시스템은 크게 프론트 엔드와 백 엔드(Web-server, DB)로 이루어져 있다. 프론트 엔

드는 서버에 정보를 입력하거나 받아오는 역할을 수행한다. Web-server는 Handler와 주요 기능에 대한 Controller를 통해 프론트 엔드와 DB의 상호작용을 돕는다. DB는 로그인, 문제 풀이 등 시스템의 주요 기능 수행 시 호출되어 사용된다.

3.2 System Architecture - Front-end Application

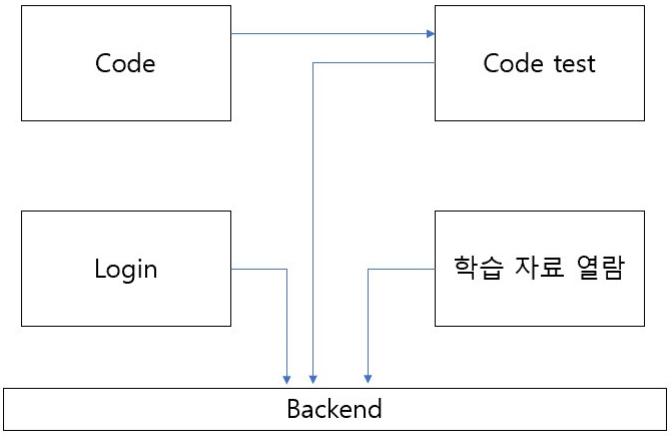

Diagram 1 System Architecture – Front-end

프론트 엔드는 서버에 정보를 입력하고 받아오는 방식으로 작동한다. 사용자가 Chrome 웹 브라우저를 이용해 HTML, CSS, JavaScript로 작성된 내용물을 읽고 로그인(회원가입), 학습자료 열람, 코드 테스트 등 여러 가지 기능을 요청할 수 있으며 최종적으로 Server에게 데이터를 전달한다.

3.3 System Architecture – Back-end

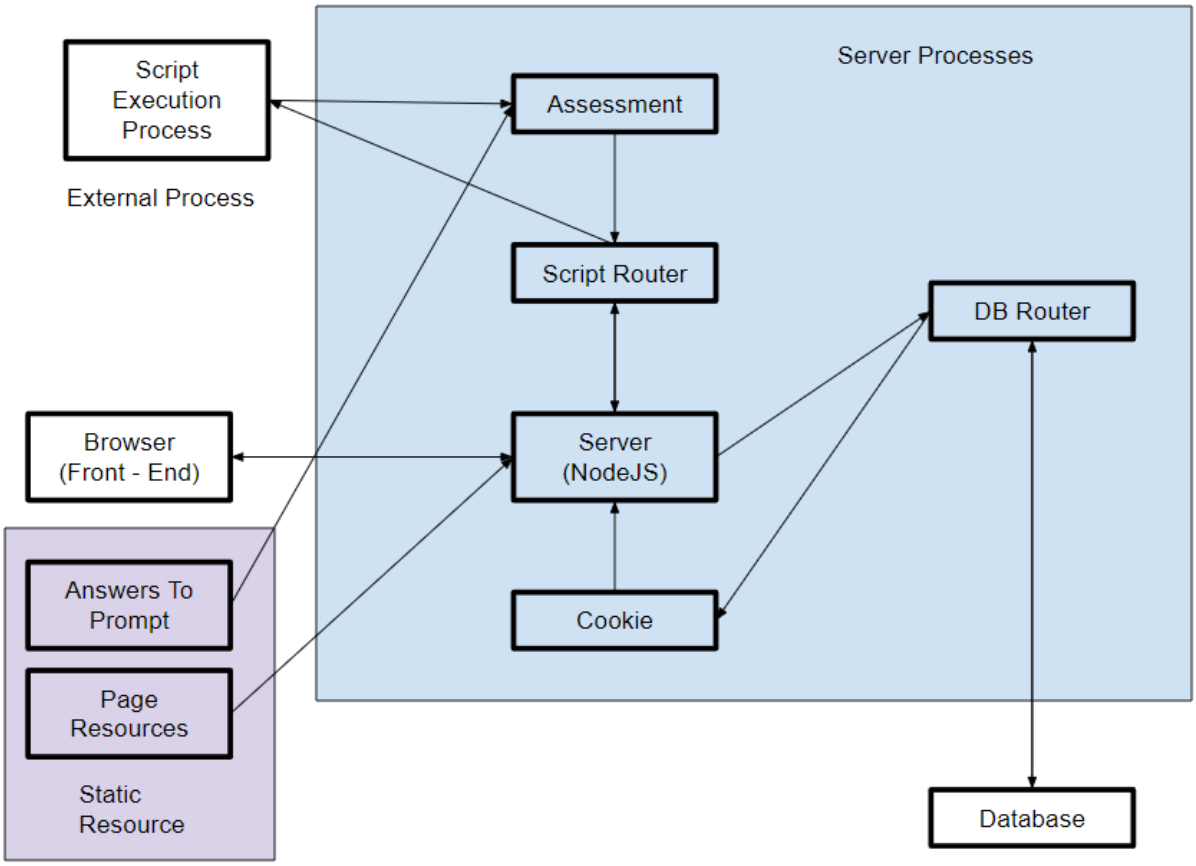

Diagram 2 System Architecture – Back-end

본 시스템은 웹 서비스로 작용하기 때문에 HTTP Request Handler가 필요하며, 사용자 정보, 학습 자료 열람, 코드 채점 등의 주요 기능에 대한 controller가 포함된다. 이와 같이 시스템 내부에서의 Controller, Handler를 정의할 수 있고 이는 NodeJS로 구현된다. 웹 서비스를 사용할 경우 각 사용자에 의해 데이터 유출 공격의 대상이 될 수 있기 때문에 격리된 실행 환경에서 설계해야 하고, 이 시스템 내부 server는 Docker 프로그램을 통해 자연스럽게 구축한다.

사용자 정보 데이터베이스로부터 사용자에 대한 개인 정보, 로그인 관련 정보, 과거 학습 자료 열람 현황, 과거 학습 자료의 정오와 같은 정보를 담는다. 본 서비스와 데이터베이스의 연결은 DB Router를 통하여 SQL Query 및 정보 처리 작업을 진행하며, 이 결과값은 server를 통해서 프론트엔드에 반영시킨다.

4. System Architecture - Front-end

사용자의 요청과 서버의 반응에 따른 state diagram과 각 component들의 class diagram으로 Front-end Architecture를 설명한다.

4-1. Overall Front-end architecture

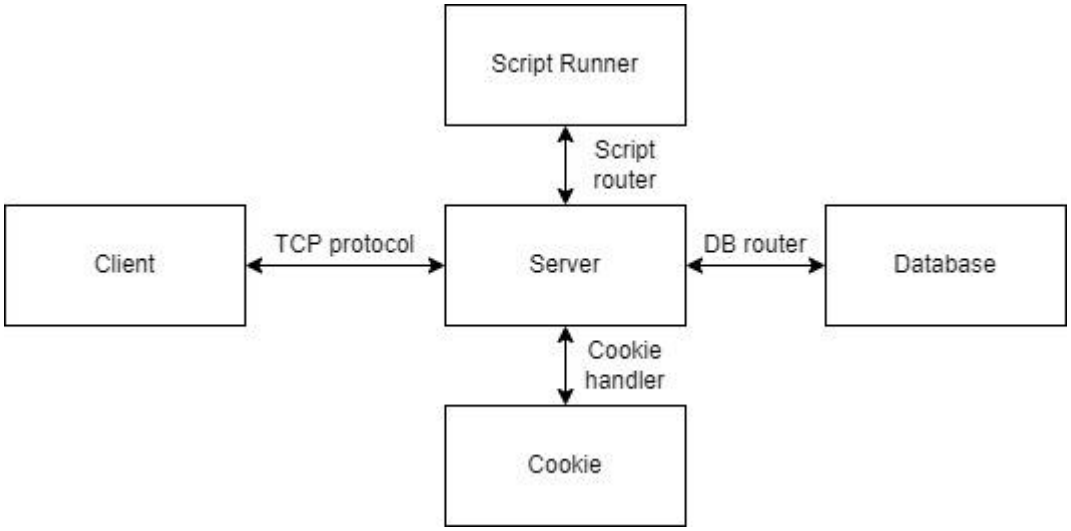

Diagram 3 Overall Front-end Architecture

Front-end는 Client-Server architecture로, TCP protocol을 통해 사용자와 서버 간 데이터를 송, 수신한다. Server는 Client에게 다양한 기능을 제공하기 위해 back-end에서 구현된 Script router, DB router, Cookie handler와 통신하여 데이터베이스, 스크립트 실행 결과, cookie 세션 정보를 제공받는다.

4-2. Class diagram

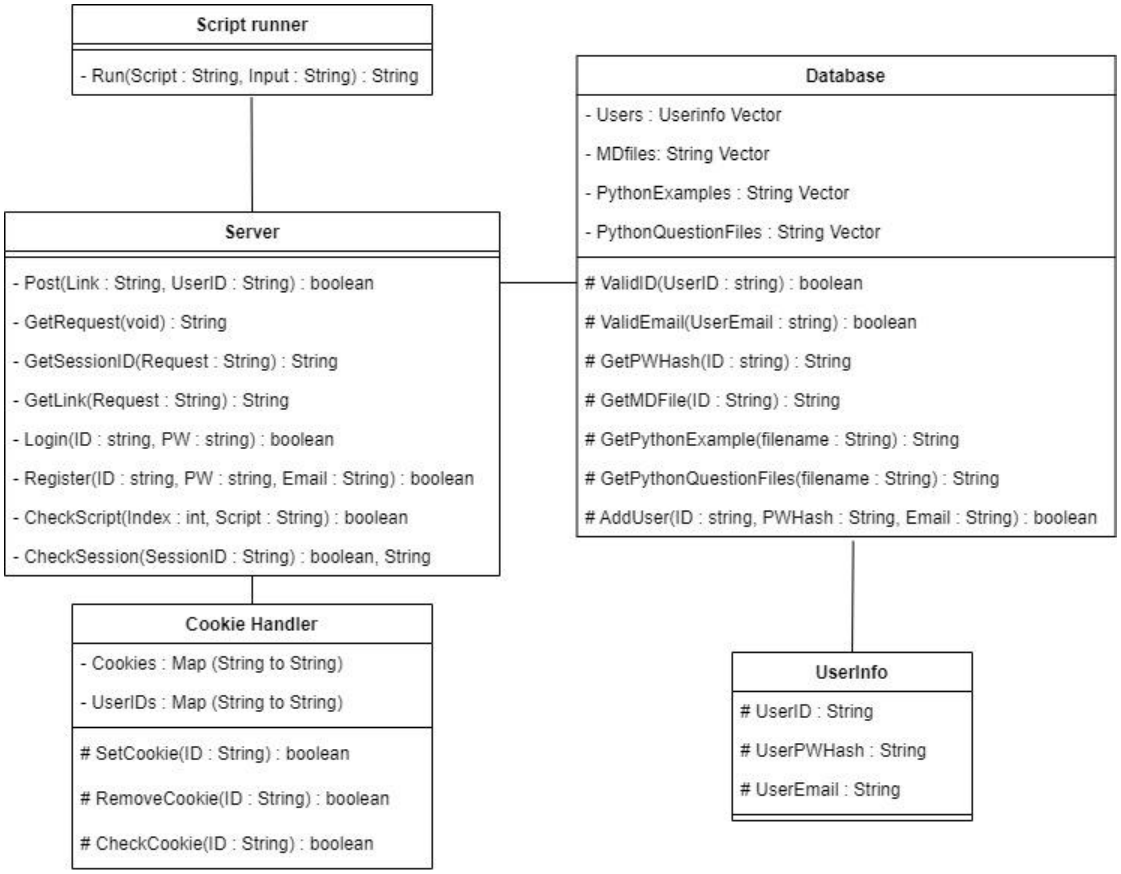

Diagram 4 Class diagram

Script Runner

파이썬 스크립트를 실행하는 객체이다.

Run() : Run() 함수를 통해 파이썬 스크립트를 실행하고, Input을 넣어 출력 결과를 반환한다.

Server

사용자와 직접적으로 통신하는 객체이다.

Post() : 사용자에게 HTML 파일(페이지)을 보낸다. HTML 파일을 보낼 때, Link 값을 사용한다. 만약, CheckSession() 의 반환 값이 False일 경우, 다른 페이지로의 접근을 막는다.

GetRequest() : 사용자의 Request를 입력 받는다. Request로부터 ID, PW, Email, Script와 같은 정보를 파싱한다.

Login() : 사용자로부터 입력 받은 ID와 PW가 DB에 존재하는지 확인한다.

이때, PW는 Hash 값을 이용한다. 만약 존재한다면 True를, 존재하지 않는다면 False를 반환한다.

Register() : 사용자로부터 입력받은 ID, PW, Email 정보를 등록한 후, True를 반환한다. 이 정보는 DB에 저장되며, 추후 로그인 시에 사용된다. 만약 DB에서 ID 혹은 Email이 중복될 경우 False를 반환하며, 정보가 DB에 저장되지 않는다.

CheckScript() : DB로부터 입력 값을 받아온 뒤, 이 값으로 Script를 실행하여 얻은 출력 값을 DB로부터 얻은 정답 값과 비교한다. 만약 정답이면 True를, 정답이 아니면 False를 반환한다.

CheckSession() : 사용자로부터 받은 패킷의 쿠키 SessionID가 CookieHandler에 존재하는지, 또 존재한다면 해당 Cookie가 Timeout 되었는지 판별한다. 만약 그러할 경우 True와 UserID를, 그러하지 않을 경우 False와 Null String을 반환한다.

Cookie Handler

쿠키를 생성, 조작, 접근하는 개체이다.

Cookies : SessionID로부터 Cookie로의 Map이다.

UserIDs : SessionID로부터 UserID로의 Map이다.

SetCookie() : User의 ID로부터 쿠키를 생성한다. UserIDs와 Cookies에 원소가 추가된다.

RemoveCookie() : User의 ID에 해당하는 쿠키를 삭제한다. UserIDs와 Cookies에 원소가 삭제된다.

CheckCookie() : SessionID에 해당하는 쿠키가 존재하는지 판별한다. 존재할 경우 True를, 존재하지 않을 경우 False를 반환한다.

UserInfo

사용자의 정보를 한 묶음으로 저장하는 객체이다.

UserID : 사용자의 ID이다.

UserPWHash : 사용자의 PW를 Hash한 값이다.

UserEmail : 사용자의 Email이다.

Database

서버에 필요한 정보들을 저장하며 조작할 수 있는 저장소이다.

Users : UserInfo의 Vector로, 사용자들의 정보를 저장한다.

MDfiles : 페이지에 사용될 Mark Down 파일을 저장한다.

PythonExamples : 페이지에 사용될 파이썬 예제들을 저장한다.

PythonQuestionFiles : 페이지에 사용될 파이썬 문제, 문제에 해당하는 입력값, 정답을 저장한다.

ValidID() : 사용자의 ID가 DB의 Users 내의 UserInfo의 UserID에 존재하는지를 판별한다. 존재할 경우 True를, 존재하지 않을 경우 False를 반환한다.

ValidEmail() : 사용자의 Email이 DB에 Users 내의 UserInfo의 UserEmail에 존재하는지를 판별한다. 존재할 경우 True를, 존재하지 않을 경우 False를 반환한다.

GetPWHash() : 해당 ID에 해당하는 사용자의 UserInfo 객체에서 UserPWHash 값을 얻어 반환한다.

GetMDFile() : 파일 이름에 해당하는 MDfiles내의 객체에 접근하여 값을 반환한다.

GetPythonExample() : 파일 이름에 해당하는 PythonExamples내의 객체에 접근하여 값을 반환한다.

GetPythonQuestionFIles() :파일 이름에 해당하는 PythonQuestionFiles내의 객체에 접근하여 값을 반환한다.

AddUser() : 해당 ID, PWHash, Email에 해당하는 UserInfo를 추가한다. 추가에 성공할 경우 True를, 실패할 경우 False를 반환한다.

4-3. Client-Server log-in state diagram

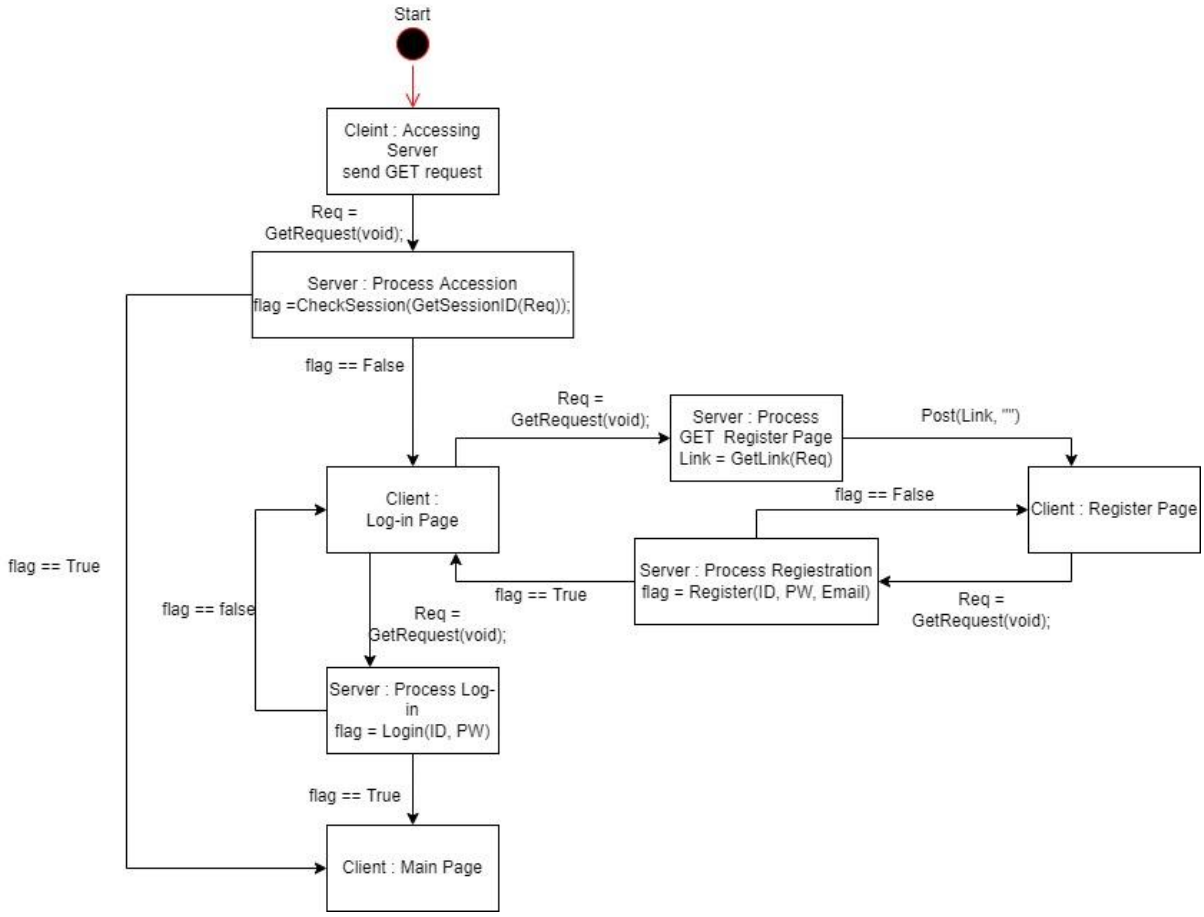

Diagram 5 Client-Server log-in state diagram

사용자는 서버에 GET 요청을 하여 메인 페이지에 접속을 시도한다. 만약 쿠키 세션이 존재할 경우, 메인 페이지에 바로 접근이 가능하며, 그렇지 않을 경우 Log-in 페이지에 접속이 되어, 로그인을 하여야 한다. 로그인을 할 수 없을 경우엔 Registration 페이지에 접속하여 Register한 후, 다시 Log-in 페이지에 접속하여 Log-in을 하여야 한다. 서버에서는 이러한 사용자의 요청을 파싱하고, DB에서 정보를 조작하여 처리한다.

4-4. Use state diagram

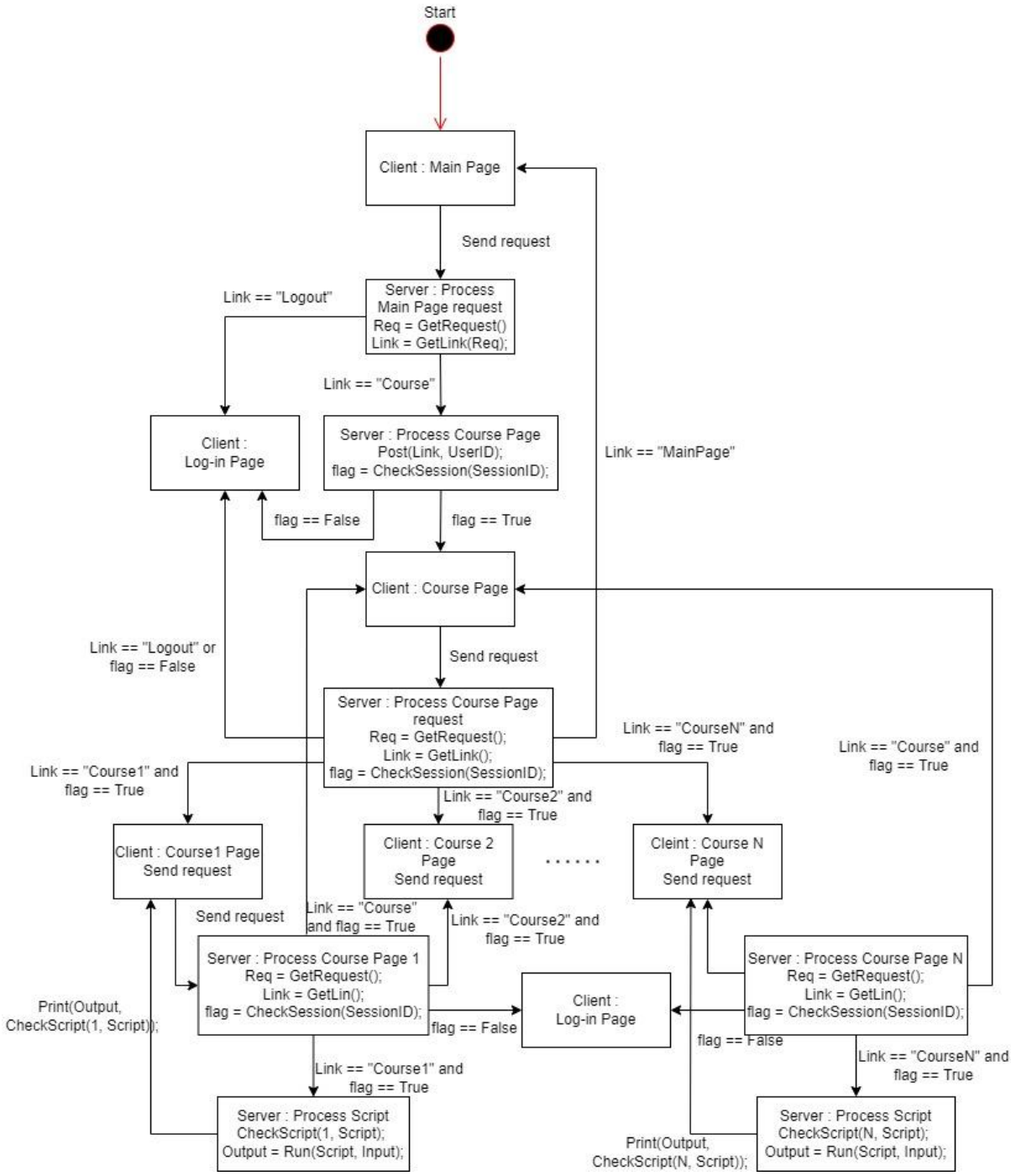

Diagram 6 Use state diagram

사용자는 메인 페이지에 들어간 후, Session을 부여받는다. 서버는 계속 Session을 체크하며 만료됐을 경우 사용자를 Log-in 페이지로 보낸다. 사용자는 메인 페이지에서 로그아웃을 하거나 학습 페이지로 이동할 수 있으며, 학습 페이지에서는 1번 학습부터 N번 학습까지 선택할 수 있다. 각

학습 창에서는 파이썬 코드를 입력하여 실습할 수 있고, 코드를 서버로 보내어 실행 결과를 확인할 수 있다. 또한, 코드가 주어진 문제의 정답인지 판별할 수 있다. 각 학습창에서의 학습이 끝나면, 사용자는 다음 학습으로 이동하거나 학습 페이지로 되돌아갈 수 있다.

5. Back-end Architecture

5.1 Objectives

데이터베이스, 파이썬 스크립트 실행, 사용자 페이지 렌더링 등 백엔드의 전반적인 구조와 데이터베이스, 스크립트 실행 등의 세부 시스템 구조를 설명한다.

5.2 Overall Architecture

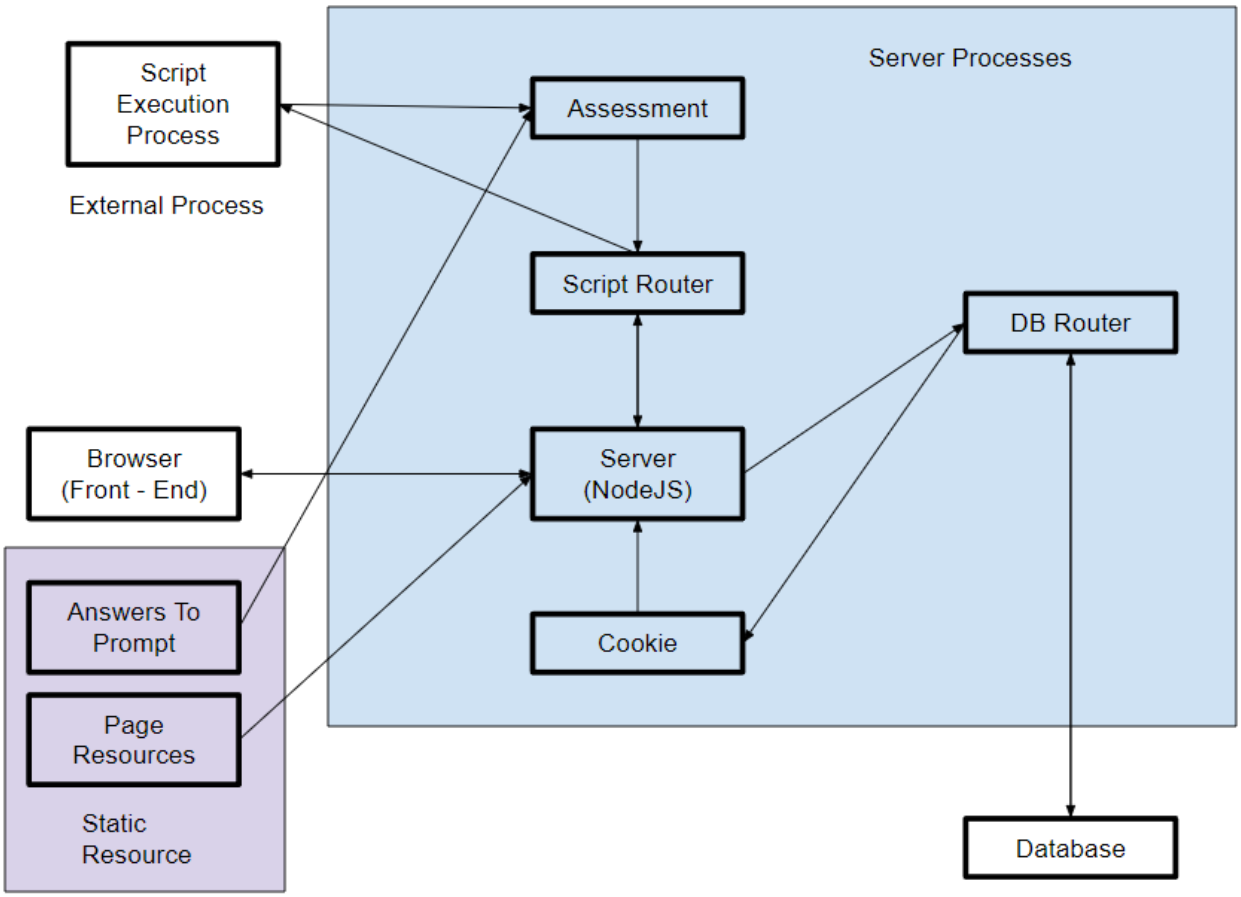

Diagram 3 Overall Front-end Architecture

백엔드의 역할은 간단하다. 웹서버 사용자의 페이지 렌더링 및 작성한 스크립트의 실행과 결과 분석 그리고 사용자 로그인 정보를 페이지 렌더링에 적용하는 것이다. 크게 나누면 해당 서버의 프로세스와 외부 프로세스 그리고 정적 자원들로 나눌 수 있다. 스크립트에 사용될 테스트케이스와 그 정답들은 정적 자원으로 저장되어 있으며 프런트엔드에서는 접근 할 수 없다. 스크립트를

실행할 프로세스는 서버 프로세스와 별개의 외부 프로세스이며 출력 값을 서버에게 제공하도록 설계된다. 마지막으로 데이터베이스 프로그램을 활용하여 로그인 정보를 저장하고 로그인시 비교하여 사용자 맞춤 페이지를 렌더링하도록 한다.

메인 서버는 NodeJS로 작성되며 Router를 이용하여 2가지 주 헨들러들을 나눈다. DB Router는 SQL Query 및 정보 불러오기와 처리하는 작업을 진행하며 결과값을 쿠키로 만들어 브라우저에 전송하여 로그인 데이터를 저장한다. Script Router는 사용자가 작성한 스크립트로 외부 프로세스를 사용하여 실행한 후 출력 값을 분석하여 결과값을 돌려준다. 주 서버는 입력 받는 쿼리를 알맞은 라우터로 전송하는 작업을 한다.

해당 방식으로 구조를 작성하면 외부 스크립트 실행 방식을 변경하더라도 메인서버와 통신하는 방식만 유지하면서 다른 시스템의 변경을 필요로 하지 않는다. 현재 고려하는 외부 스크립트 실행방식은 웹 API를 활용한 실행과 서버를 돌리는 컴퓨터내 새로운 프로세스 생성이 있다. 두 방식 장단점이 있지만 전반적인 구조에는 영향이 없으므로 여기서는 자세하게 다루지 않는다.

5.3 Internal Functions

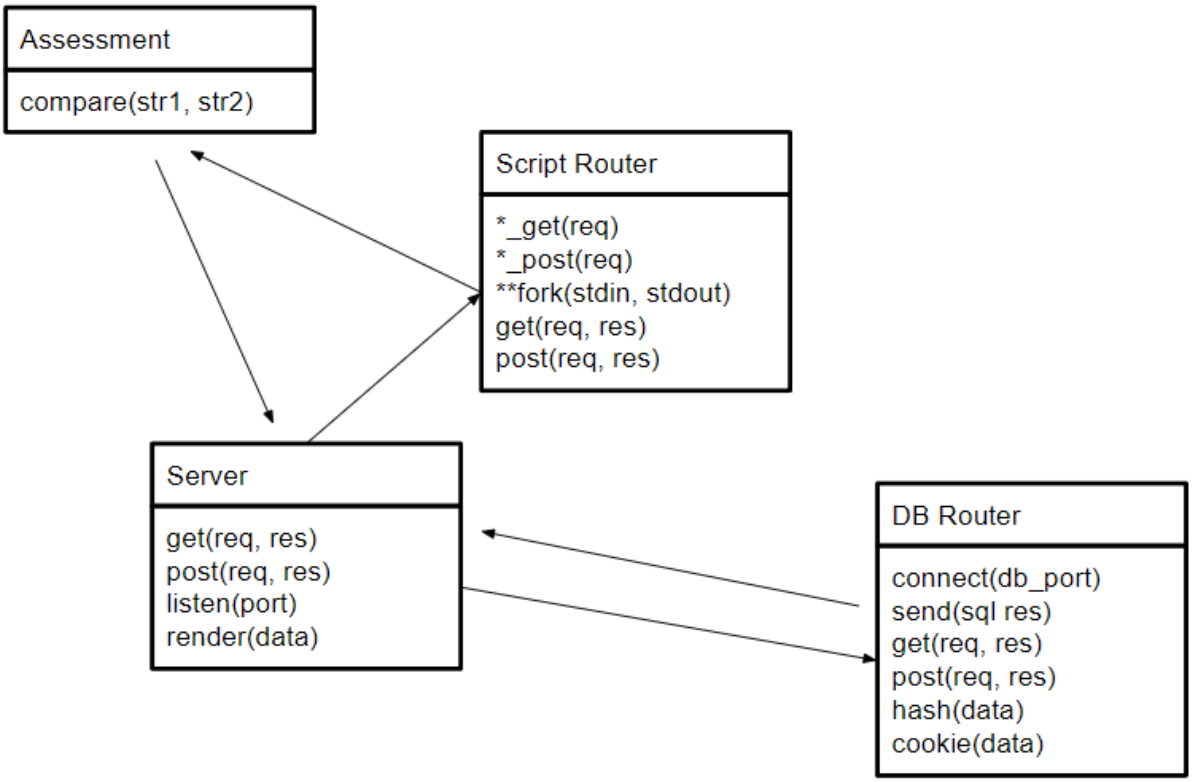

Diagram 8 Internal Functions

Server

Listen(port): 메인 서버는 특정 포트 값으로 서버를 열어 해당 포트로 오는 네트워크 쿼리를 듣고 있는다 입력 받을 시 알맞은 주소라면 해당 라우터로 request를 전송하는 작업을 한다.

Get(req, res)/ post(req,res): 입력 받는 모든 request는 서버가 가장 먼저 보게 된다. 이때 서버는 주소를 보고 알맞게 처리한다. Request에 따라 서버가 처리할 수 있고 다른 라우터에게 정보를 보낼 수도 있다.

Render(data): 주어진 data를 가지고 사용자 페이지를 렌더링하는 작업을 한다.

DB Router

Connect(db_port): 정해진 포트번호를 사용하여 데이터베이스에 접속한다.

Send(sql res): sql문 형식으로 request와 돌려받은 res(result)를 처리한다.

Get(req, res), post(req, res): 라우터의 작동은 메인 서버로부터 받은 해당 request에서 시작된다.

Hash(data): 로그인 정보를 처리하기에 해싱을 한다.

Cookie(data): 로그인 성공시 브라우저에 저장할 쿠키를 생성한다.

Script Router

*_get(req), *_post(req): 웹 API에 request를 보내 받은 답변을 처리한다

**fork(stdin, stdout): 새로운 프로세스를 생성하고 출력 값을 처리한다.

*과 **의 경우 구현 방향에 따라 실제 시스템에서는 다를 수 있다.

Get(req, res), post(req, res): 라우터의 작동은 메인 서버로부터 받은 해당 request에서 시작된다.

Assessment

Compare(str1, str2): 답안과 출력 값을 비교하여 결과값을 출력한다

5.4 Script execution server

스크립트의 실제 실행은 외부 프로세스가 진행해야 한다. 이경우 해당 기능을 수행하는 웹 API가 있으므로 해당 API에게 request를 보내는 형식으로 작성이 가능하다. Figure 5.2의 * 이 해당 경우의 함수를 설명한다. 해당 방식의 장점은 외부 프로그램을 활용하므로 실제 구현을 할 필요가 없어진다. 또한 서버와 스크립트 실행간 적절한 분리가 가능하다. 하지만 외부 API를 사용하여 다소 느릴 수도 있으며 채팅프로그램과 같은 포트를 열어 실행하는 방식의 스크립트는 평가할 수 없다.

해당 약점을 보안하기 위해 실제로 프로세스를 생성하여 활용하는 방식도 고려했다. Figure 5.2의 **이 바로 이 경우이다. 새로운 Child 프로세스를 생성하여 출력 값을 분석하는 방식으로 실행한다면 두 채팅프로그램간 통신도 평가할 수 있다. 하지만 외부 작성 스크립트를 돌리는 작업에 있어 보안관련 고려해야할 사항이 많아 해당 스크립트를 돌린다면 완전히 차단된 환경에서 진행해야 할 것이다.

5.5 Login Process – Database

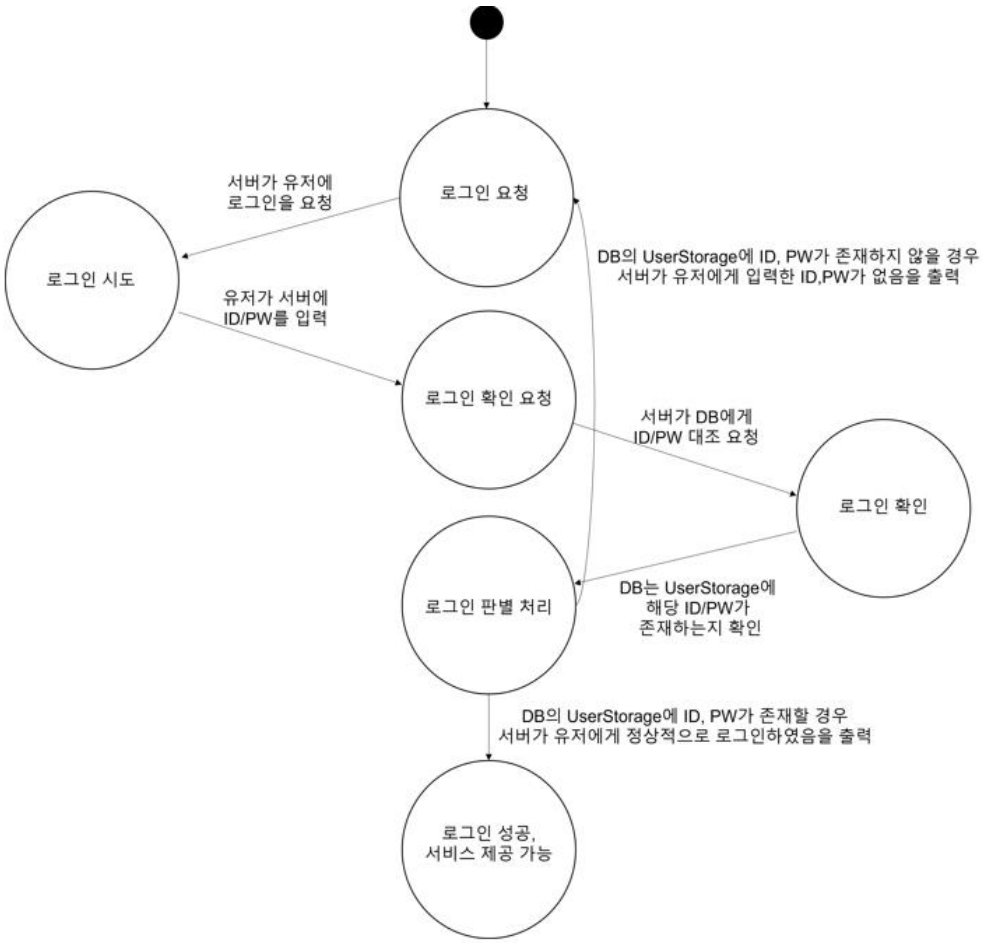

Diagram 9 Login Process - Database

데이터베이스는 RDBMS를 활용하며 로그인 관련 정보는 데이터베이스가 처리한다. 요청을 받은 후 서버는 SQL문을 작성하여 DB에 제출한 후에 DB의 답변을 활용하여 사용자에게 결과를 전해준다. 요청이 거절된다면 서버는 사용자에게 이를 알리고 기존 페이지로 되돌린다. 요청이 수락된다면 서버는 사용자 페이지를 렌더링한다.

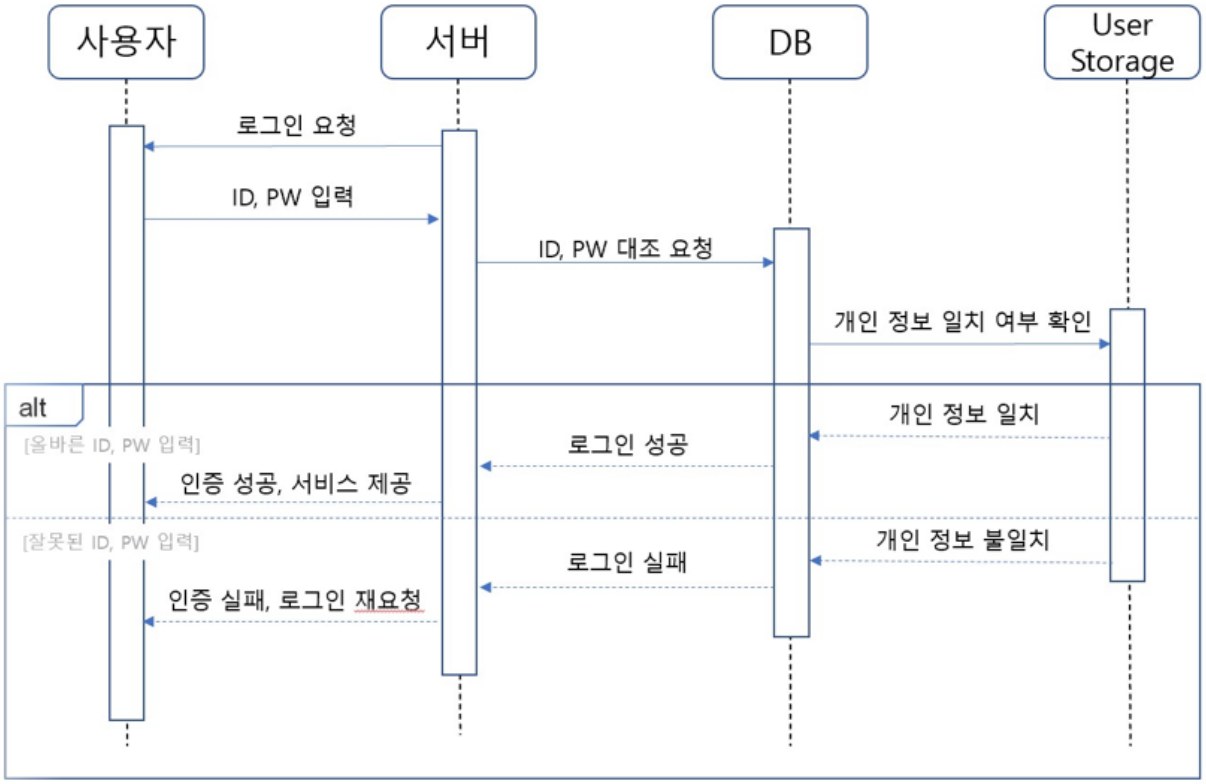

Diagram 9 Login Process Sequence diagram

로그인 성공후에 서버는 사용자에게 쿠키를 발급한다. 해당 쿠키를 이용하여 이후 사용자의 로그인 정보를 저장한다. 서버는 로그인정보를 활용하여 페이지를 렌더링하며 발급한 쿠키가 유효기간이 지나면 새로운 쿠키를 발급받기 위해서는 재로그인이 필요하다. 보안을 위해 DB에 저장되고 불러오는 비밀번호 역시 해싱을 진행하여 활용한다. 고로 사용자가 입력한 ID PW를 이용하여 생성한 해싱된 값을 DB와 대조하여 로그인 성공여부를 결정한다.

6 Protocol

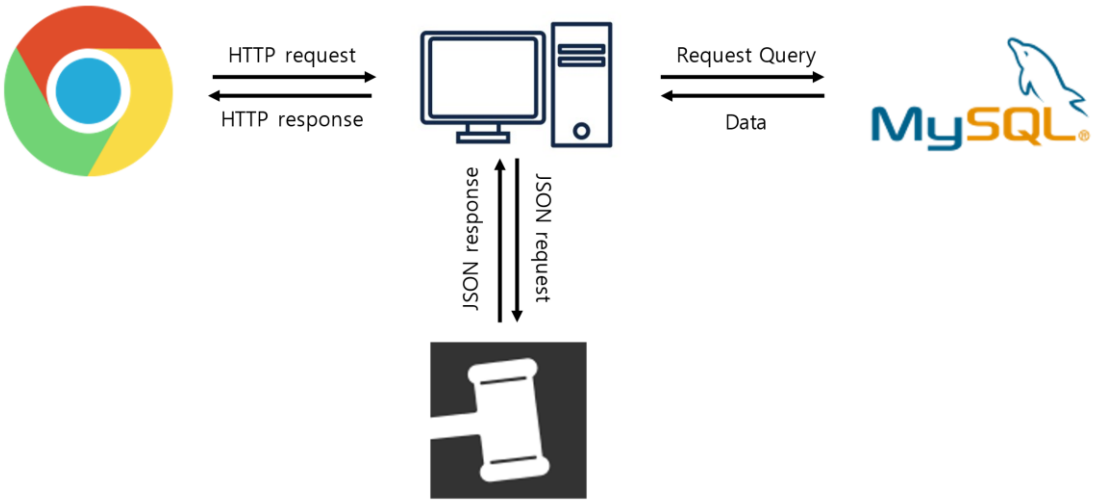

Figure 1 Overall Protocol Structure

프론트엔드와 백엔드는 HTTP를 통해서 이루어진다. 웹 서비스를 제공하기 위한 환경은 NodeJS 프레임워크를 활용해서 제공되며, EJS가 이를 돕는다. 회원의 ID,비밀번호,학습진도에 관한 정보는 MySQL database안에 저장되며, 회원의 테스트 코드는 JSON의 형태로 주어지고, Judge0의 환경에서 테스트된다.

6.1. HTTP

HTTP(HyperText Transfer Protocol)은 Hypertext 형식의 데이터를 빠르게 교환하기 위한 프로토콜의 일종으로 클라이언트가 요청을 하면 서버가 응답하는 구조로 되어있다. 요청(request)과 응답(response)으로 구성되어 있으며, 서버와 클라이언트 사이의 이루어지는 연결과 통신이 HTTP를 통해서 이루어진다. 이 때 주고받는 문서는 HTML(Hyper Text Markup Language)의 형식을 따르며 해당 시스템에서는 EJS를 통해서 HTML 형식을 구현해 내고 있다.

6.2. NodeJS

NodeJS는 네트워크 애플리케이션 개발에 사용되는 소프트웨어 플랫폼으로, 자바스크립트를 작성 언어로 사용하며 Non-blocking I/O와 단일 스레드 이벤트 루프를 통한 높은 처리 성능을 가지고 있다. 내장 HTTP 서버 라이브러리를 포함하고 있어서 웹 서버를 동작하기에 알맞으며 이벤트 기반으로 시스템이 작동한다.

또한 Express라는 프레임워크(Frame)를 제공하고 있어서 웹 애플리케이션을 작성하기가 용이하고, 따라서 해당 시스템에서는 NodeJS와 Express 프레임워크를 사용한다.

6.3. EJS

EJS(Embedded JavaScript)는 자바스크립트를 HTML내에서 구현할 수 있도록 도와주는 템플릿 엔진으로, 클라이언트의 요청에 따라서 달라지는 동적인 결과를 파일에 담을 수 있도록 도와준다. 자바스크립트로 연산된 결과를 손쉽게 HTML 파일로 넣을 수 있다는 장점이 있다.

6.4. JSON

JSON은 JavaScript Object Notation의 약자로 Name/Value 쌍으로 이루어진 경량의 data 교환 형식이다. 사람이 읽을 수 있는 텍스트라서 간단하고, 자료의 종류에 큰 제한이 없으며, 플랫폼이나, 프로그래밍 언어에 독립적이므로 다양한 언어에서 쉽게 사용할 수 있다. 따라서, 서버와 웹 애플리케이션 사이에 data를 교환할 때 자주 사용되며, 해당 시스템에서 서버와 서브시스템인 Judge0와의 통신에 사용된다.

7 Database Design

7.1. Objectives

Requirement specification에서 작성된 내용을 근거로 database design을 기술한다. 변경사항으로 불필요하다고 여겨지는 Table 'Complete_Log' 가 삭제되고, 'Class' Table에서 수강생의 진도를 기록한다. 또한 'Class' Table 에서 'User' Table의 ID 값을 Foreign key로 참조해서 ID 값을 primary key로 지정한다. 이러한 관계와 Entity 들을 ER Diagram 으로 그려주었고, 이를 통해서 relational schema, SQL DDL을 작성한다.

7.2. ER Diagram

해당 시스템에는 User, Class 두개의 Entity가 존재한다. ER diagram 을 위해서 Entity는 직사각형, Entity간의 관계는 마름모, 전체참여/부분참여 여부를 이중선, 단일선으로 표현했다. Entity가 가지는 Attribute들은 타원형으로 표현되었으며, Primary Key는 밑줄을 그어 표시하고, Foreign Key의 참조 여부를 Crow Foot의 형태로 나타내었다.

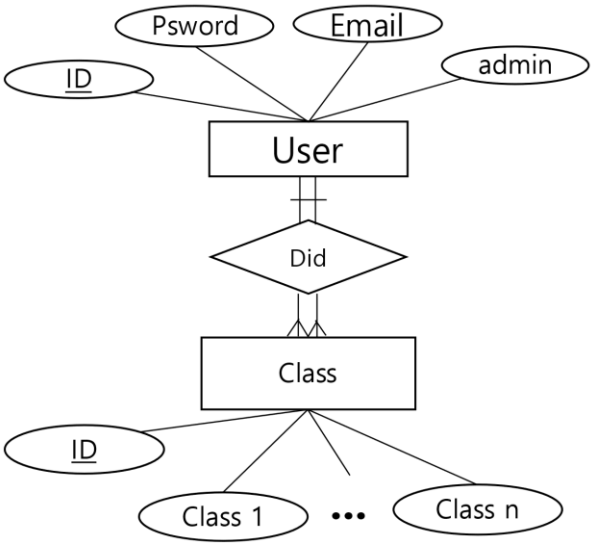

Diagram 10 ER Diagram

A) Entity - User

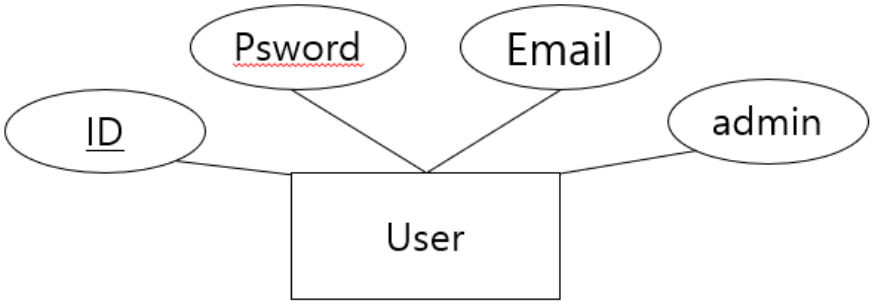

Diagram 11 User Entity

User Entity는 사용자의 ID, password, E-mail 추가로 관리자인지 아닌지 판단하기 위한 admin attributes를 가지고 있으며, ID를 primary key로 가지고 있다.

B) Entity - Class

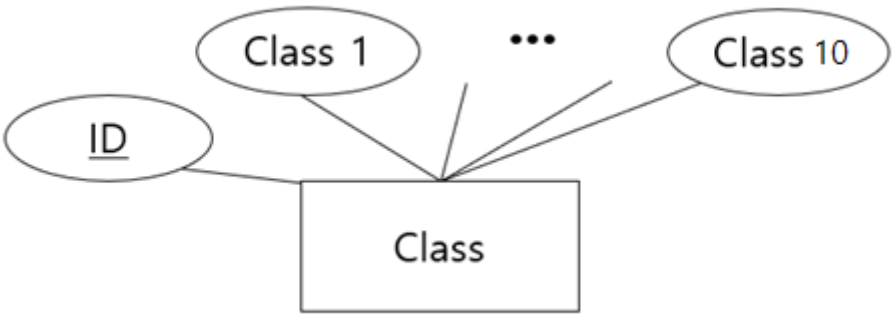

Diagram 12 Class Entity

Class Entity는 사용자의 ID, 특정 강의를 수강했는지에 대한 정보를 가지고 있으며, ID를 foreign key 이자 primary key로 가지고 있다.

7.3. Relational Schema

JSON은 JavaScript

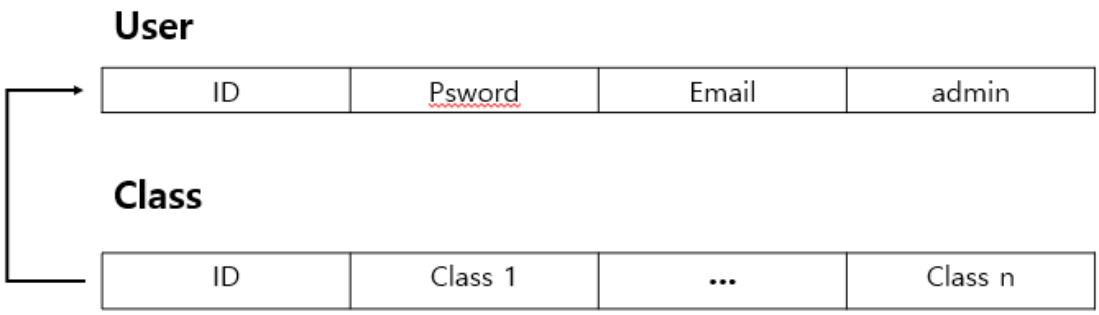

Diagram 13 Relational Schema

7.4. SQL DDL

Entity – User

```
CREATE TABLE IF NOT EXISTS `mydb`.`user`
(
  `id` varchar(30) NOT NULL,
  `Email` varchar(30) NULL,
  `psword` varchar(30) NULL,
  `admin`  tinyint(1) NULL,
  PRIMARY KEY (`id`));
);
```

Entity – Class

```
CREATE TABLE IF NOT EXISTS `mydb`.`users` (
  `id` varchar(30) NOT NULL,
  `Class 2`  tinyint(1) NULL,
  `Class 3`  tinyint(1) NULL,
  `Class 4`  tinyint(1) NULL,
  `Class 5`  tinyint(1) NULL,
  `Class 6`  tinyint(1) NULL,
  `Class 7`  tinyint(1) NULL,
  `Class 8`  tinyint(1) NULL,
  `Class 9`  tinyint(1) NULL,
  `Class 10`  tinyint(1) NULL,
  PRIMARY KEY (`id`));
  FOREIGN KEY (`id`) REFERENCES user(id) ON UPDATE CASCADE ON DELETE
RESTRICT
);
```

8 Index