Design Specification

Online Tutoring Web Application

Team #3

이름	학번
나종명	2014311352
박은찬	2017312838
임성규	2015310908
최아영	2015312791

Table of Contents

1. Preface

1.1 Intended Audience / Readership

A. Project Managers / System Developers

This document is mainly intended for people with knowledge in software systems, i.e. software engineers, designers, managers. System requirements are specifically mainly intended for developers for detailed specification for user requirements.

B. Users

User requirements, intended for users and customers, explains the system requirements in natural language. It limits using professional jargon, and defines requirements easily with simplified explanation in natural language.

2. Introduction

2.1 Objectives

본 문서가 다루는 시스템인 QAHub를 설계할 때 이용하고, 이용할 예정인 다이어그램 모델들과 프레임워크, 툴들에 대해 설명한다.

2.2 Applied Diagrams

A. Activity Diagram

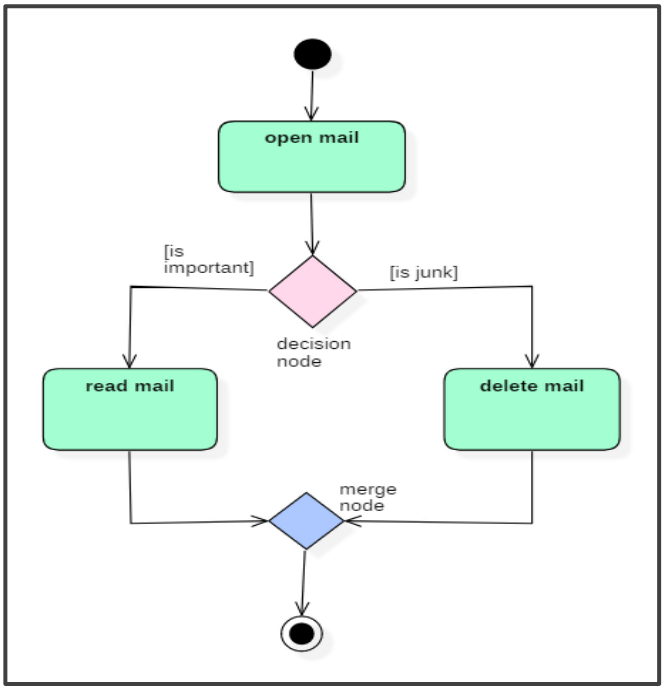

Figure 1 Applied Diagrams: Activity Diagram

Activity diagram이란 작업의 흐름이나 데이터를 처리하는 과정에서 일어나는 Activity(시스템에 의해 수행되어지는 업무, 혹은 Class의 Method)들을 순서에 따라 정의한 다이어그램이다. Business process를 정의하는 데 그 목적을 두며, Activity와 그 흐름을 state의 변화에 따라 시각적으로 표현함으로써 시스템의 operation을 이해하는 데 도움을 준다. 또한 Activity diagram은 실제 시스템이 수행하는 작업과 그 흐름을 나타내기 때문에, 이를 참고함으로써 추후에 불필요하다고 생각되는 작업들을 제거하여 프로세스를 최적화할 수 있다.

B. Use-case Diagram

Use-case diagram은 Actor (사람, 시스템, 혹은 작업을 수행하는 entity)와 Use-case(시스템이 제공하는 서비스) 사이의 상호작용을 표현하는 다이어그램이다. 시스템이 제공하는 서비스의 기본적인 내용을 설명하며, 서비스와 그 환경적 요소들을 사용자의 관점에서 확인할 수 있다. Use-case diagram을 상호 작용을 시각적으로 표현하여 시스템을 이해하는 데 도움을 주지만, 상세한 내용을 표현하기에는 부족함이 있기 때문에 주로 표와 같은 부가적인 정보를 제공할 수 있는 템플릿과 함께 작성된다.

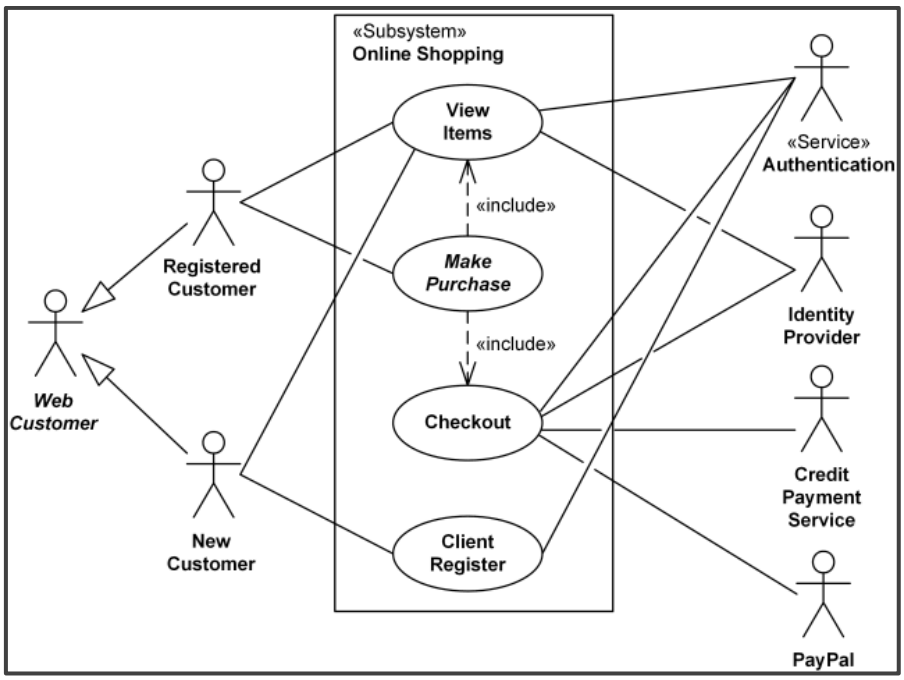

Figure 2 Applied Diagrams: Use-case Diagram

C. Sequence Diagram

Use-case diagram이 사용자와의 상호작용을 표현하는 데 집중했다면, Sequence diagram은 시스템과 시스템, 그리고 시스템 내부의 component간의 상호작용을 표현하는 데 무게를 둔다. 특정한 Use-case에 대한 상호작용을 시간 순서로 나타내며, Event와 작업 흐름에 따른 객체 간 상호작용의 내용들을 확인할 수 있다. 이러한 특성 덕분에 각 component간의 관계와 그들의 속성, 행동들을 구체화시키는 데 도움을 준다.

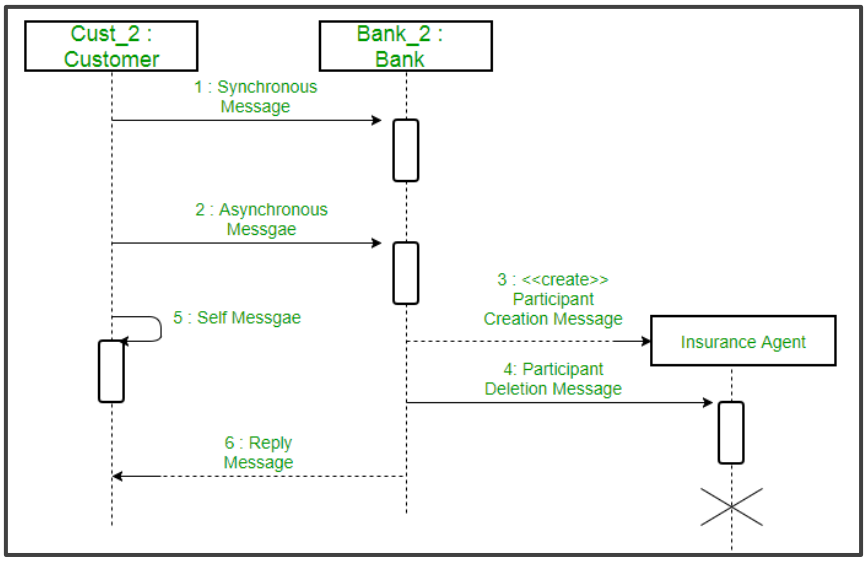

Figure 3 Applied Diagrams: Sequence Diagram

D. Class Diagram

Class diagram은 객체 지향형 시스템 모델을 설계할 때 사용되는 다이어그램이다. 시스템을 여러 개의 객체 클래스의 집합으로 간주하고, 시스템에 존재하는 클래스, 클래스의 속성, 동작 방식, 그리고 그들 간의 관계를 정의함으로써 시스템의 특정 모듈이나 일부, 혹은 전체의 구조를 표현한다. Class diagram은 시간이 경과해도 변하지 않는 시스템의 정적인 면을 나타낸다.

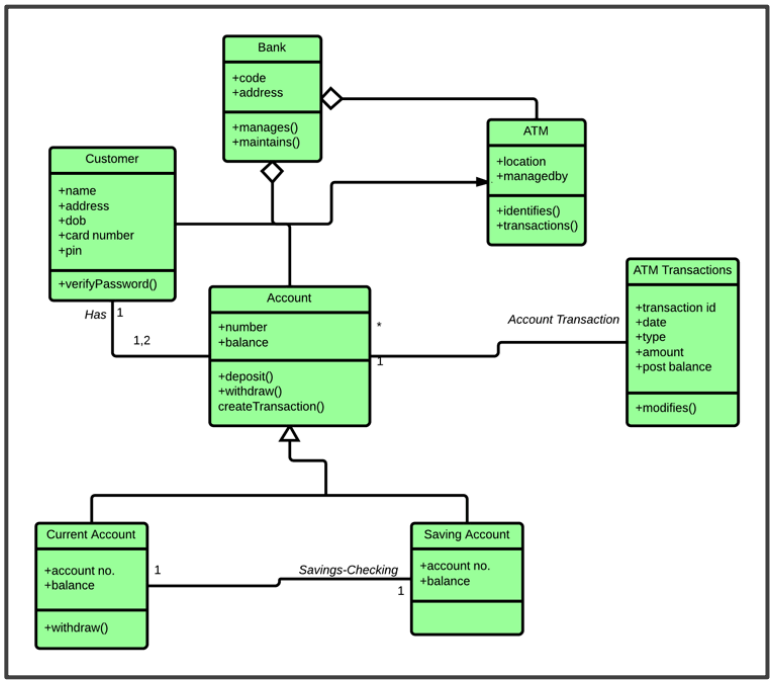

Figure 4 Applied Diagrams: UML Diagram

E. State Diagram

State diagram은 상태 변화를 일으키는 특정한 Event들에 기반하여 시스템의 상태 변화와 시스템이 가질 수 있는 모든 상태를 표현한다. 이를 통해 시스템 내에서의 객체들의 동작 원리를 쉽게 이해할 수 있다. 시스템의 정적인 면을 표현한 Class diagram과 다르게, State diagram은 시스템의 동적인 면을 표현한다. 개발자는 State diagram을 통해 객체들이 어떻게 행동하는지 정확하게 알 수 있기 때문에, UI를 설계하는 데 도움을 받을 수 있다.

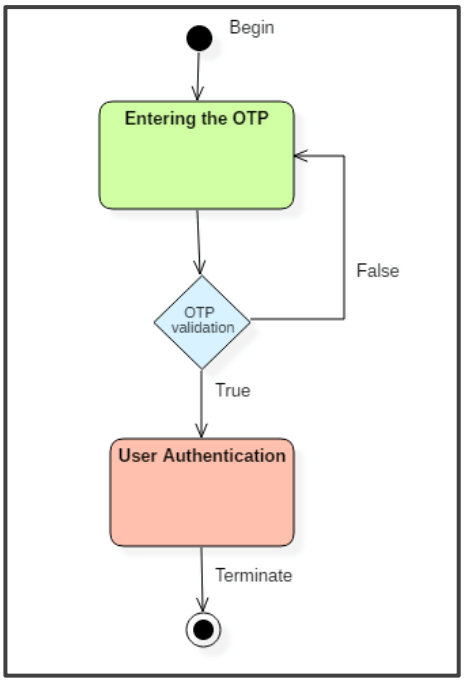

Figure 5 Applied Diagrams: State Diagram

2.3 Applied Tools

A. Vue.js (Front-end)

Figure 6 Applied Tools: Vue.js

Vue.js는 웹 애플리케이션의 UI를 작성하기 위해 사용하는 컴포넌트 구조 기반 오픈 소스 자바스크립트 프레임워크이다 (MIT License).
가볍고, 빠르며, 익히기 쉽다는 장점을 가지고 있어 웹 기반 애플리케이션인 QAHub를 개발하는 데 적절하다고 고려되어 선정되었다.

B. Bootstrap (Front-end)

Figure 7 Applied Tools: Bootstrap

Bootstrap은 웹 사이트의 제작을 돕는 오픈 소스 HTML, CSS, 자바스크립트 프레임워크이다 (MIT License).
웹 페이지의 쉬운 디자인과 반응형 웹 디자인 (RWD, Responsive Web Design)을 위하여 개발에 사용될 프레임워크로 선정하였다.

C. Node.js (Back-end)

Figure 8 Applied Tools: Node.js

Node.js는 Google Chrome의 자바스크립트 엔진 V8로 빌드 된 이벤트 기반 오픈 소스 자바스크립트 런타임이다 (MIT License).
QAHub의 웹 서버를 구축하기 위한 도구로 채택되었다.

D. MySQL

Figure 9 Applied Tools: MySQL

MySQL은 오픈 소스 관계형 데이터베이스 관리 시스템이다 (GPL 2.0 License).
멘티들이 QAHub에 등록한 질문들과 멘토들이 그 질문에 등록한 답변들의 데이터, 그리고 사용자들이 서비스를 이용하기 위해 사이트에 등록한 사용자 정보들을 관리하기 위해 사용될 시스템이다.

E. WebRTC (Back-end)

Figure 10 Applied Tools: Web RTC

WebRTC는 Web Real-Time Communication의 약자로서, 웹 브라우저 간에 별도의 플러그인의 도움 없이 서로 실시간으로 통신할 수 있도록 설계된 오픈 소스 API이다 (BSD License).
QAHub의 멘티-멘토 간의 1:1 튜터링에 사용될 일반 라이브 채팅과 화면 공유를 포함하는 라이브 채팅 기능을 구현하기 위한 도구로 채택되었다.

F. Redis (Back-end)

Figure 11 Applied Tools: redis

Redis는 Remote Dictionary Server의 약자로서, Key-Value 구조의 데이터를 저장하고 관리하기 위한 오픈 소스 비관계형 데이터베이스 관리 시스템이다 (BSD License).

G. Postman

Figure 12 Applied Tools: Postman

Postman은 REST API, URL request, HTTP protocol을 테스트할 수 있게 설계된 도구이다. HTTP 주소에 대해 GET/POST의 응답을 확인함으로써, 서버와 데이터베이스에 신호가 전달되는 지 확인하는 데 사용된다.

H. Draw.io

Figure 13 Applied Tools: draw.io

Draw.io는 구글에서 제공하는, 사이트 상에서 다이어그램을 쉽게 그릴 수 있도록 도와주는 무료 온라인 다이어그램 작성 도구이다.

I. Visual Paradigm

Figure 14 Applied Tools: VisualParadigm

Visual Paradigm은 UML2, SysML 및 비즈니스 프로세스 모델링 표기법을 지원하는 UML Case 도구이다. 또한, 모델링 지원 외에도 보고서 생성 및 코드 생성을 포함한 코드 엔지니어링 기능을 제공하기도 한다.

3. System Architecture

3.1 Objectives

QAHub 시스템의 전체적인 구조를 MVVM 패턴을 중심으로 서술한다. 또한 Use Case Diagram, UML Diagram, Entity Diagram으로 시스템의 내용을 보다 효과적으로 나타낼 수 있도록 한다.

3.2 System Organization

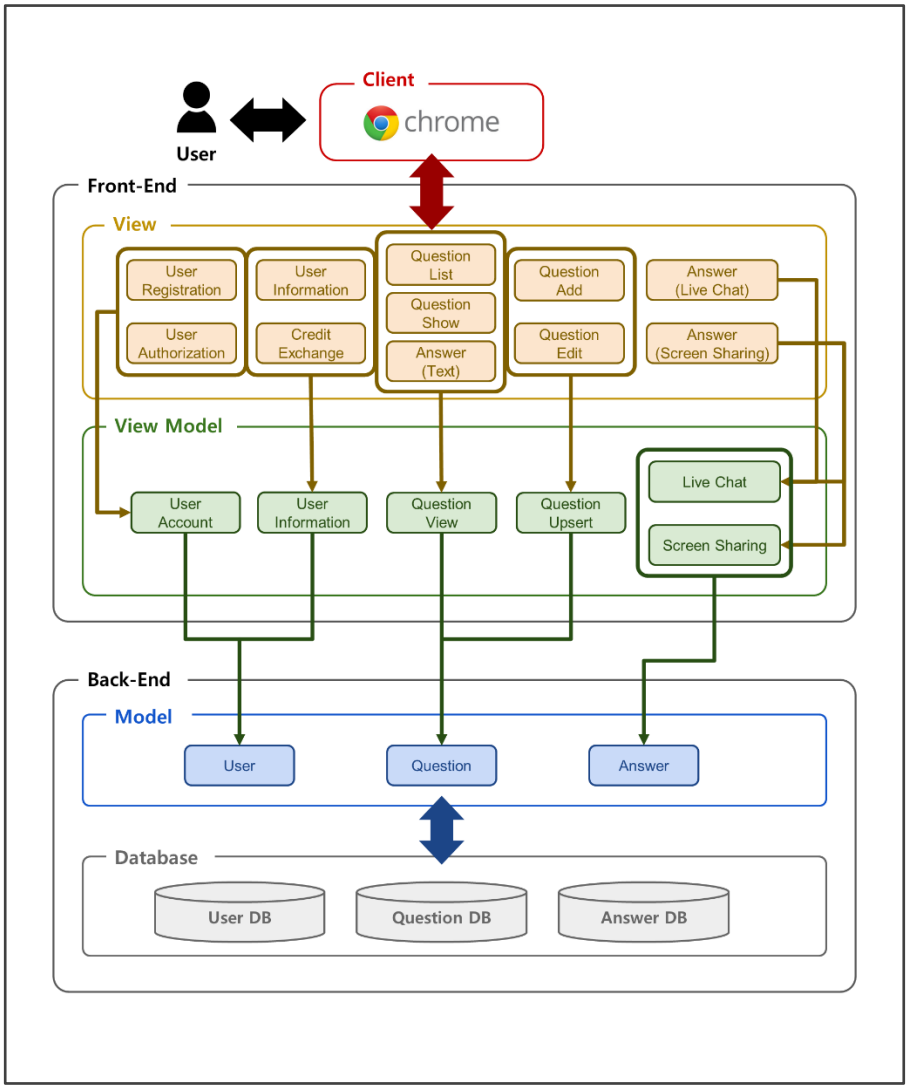

Figure 15 Overall Architecture

QAHub의 전체 아키텍처는 크게 Front-end, Back-end, Live-chatting System, Screen-sharing System으로 나뉘어져 있다.

전체 아키텍처는 MVVM(Model - View - View Model) 패턴을 따른다. MVVM 패턴은 UI와 로직/데이터의 분리를 용이케 하는 소프트웨어 아키텍처 패턴이다. View, View Model, Model 세가지의 구조로 나뉘어 View Model이 Model과 View 사이에서 상호작용하며 데이터를 가져오고 노출하는 역할을 맡게 된다.

A. Front-end Architecture

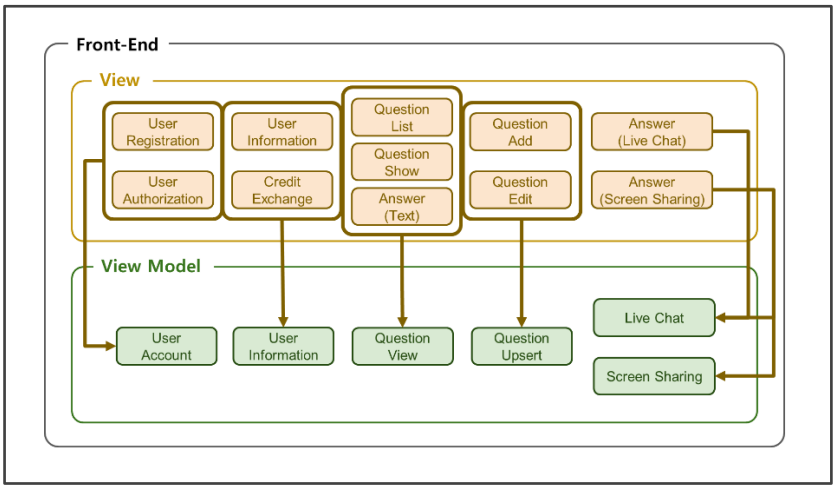

Figure 16 Front-end Architecture

사용자와 상호작용하는 Front-end 아키텍처이다. MVVM 패턴의 핵심인 View Model은 Data Binding을 통해 View가 데이터를 표시할 수 있게 한다. 이는 vue.js로 구현할 것이다.

B. Back-end Architecture

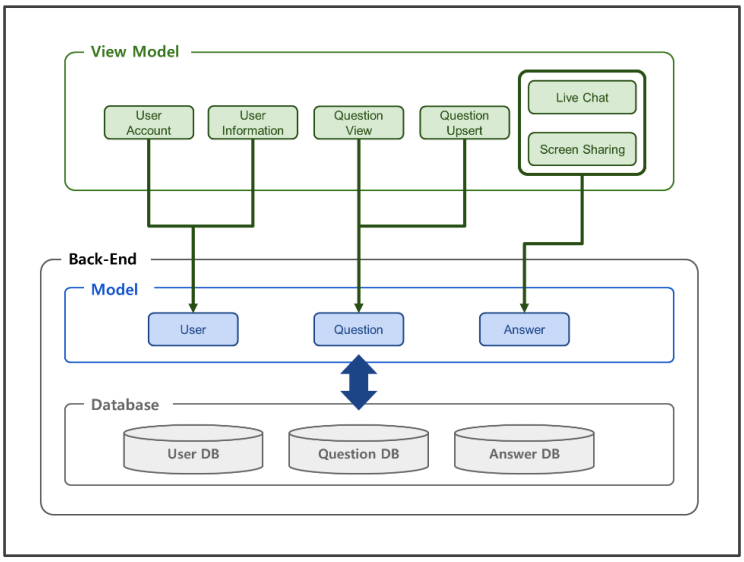

Figure 17 Back-end Architecture

프론트에서 보내는 REST API 요청을 핸들링 하고 데이터베이스와 통신하는 Back-end 아키텍처이다. 모델은 크게 User, Question, Answer 로 나뉘며 각각 해당 데이터베이스 엔티티와 연결이 되어 Query를 보내게 된다. Node.js 로 구현이 될 예정이다.

C. Live-chatting Architrecture

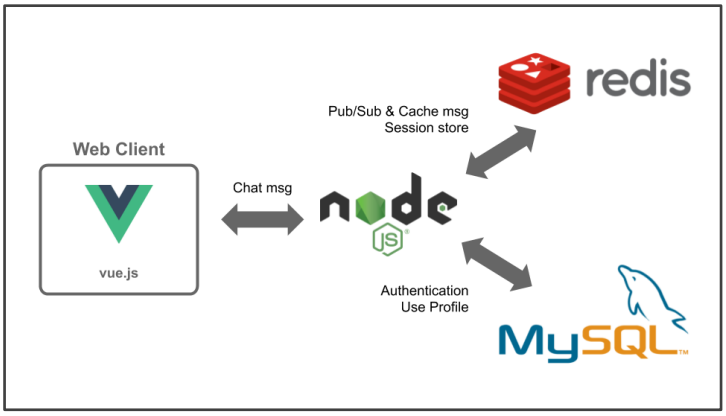

Figure 18 Live-chatting Architecture

실시간 채팅을 위한 아키텍처로 vue.js와 socket.io를 이용하여 node.js 웹 서버와 통신하며 웹 서버에서 요청한 유저에 대한 정보를 MySQL 데이터베이스에서 확인하고 redis를 이용해 채팅 내역 저장과 메시지 캐싱을 하는 구조이다.

D. Screen-sharing Architecture

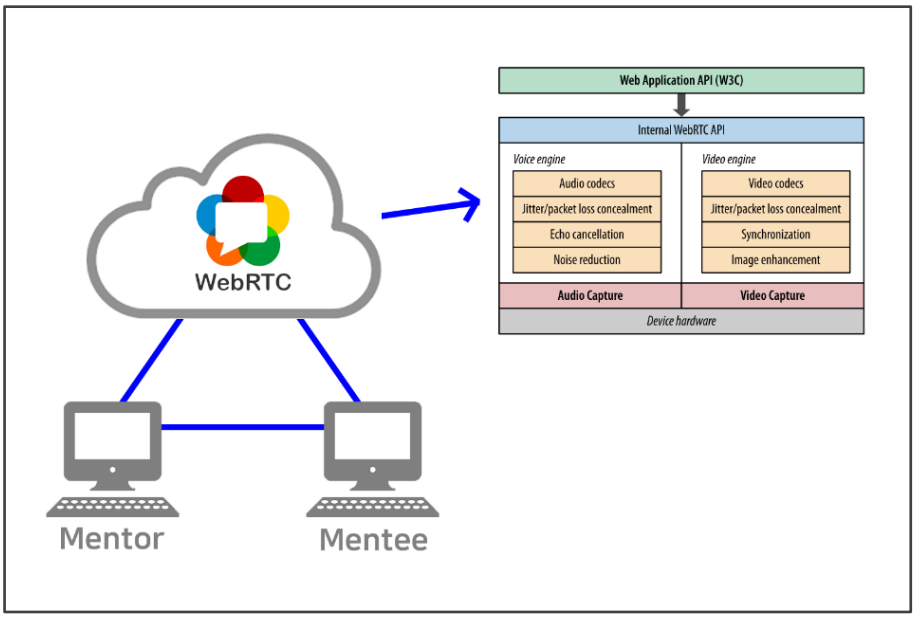

Figure 19 Screen-sharing Architecture

화면 공유를 위한 아키텍처로 각 유저가 브라우저에서 WebRTC API를 통해 통신하게 된다. 이때 WebRTC API 에서는 유저의 디바이스로부터 오디오 및 화면을 캡처하여 상대방에게 전달하게 된다.

3.3 UML Diagram

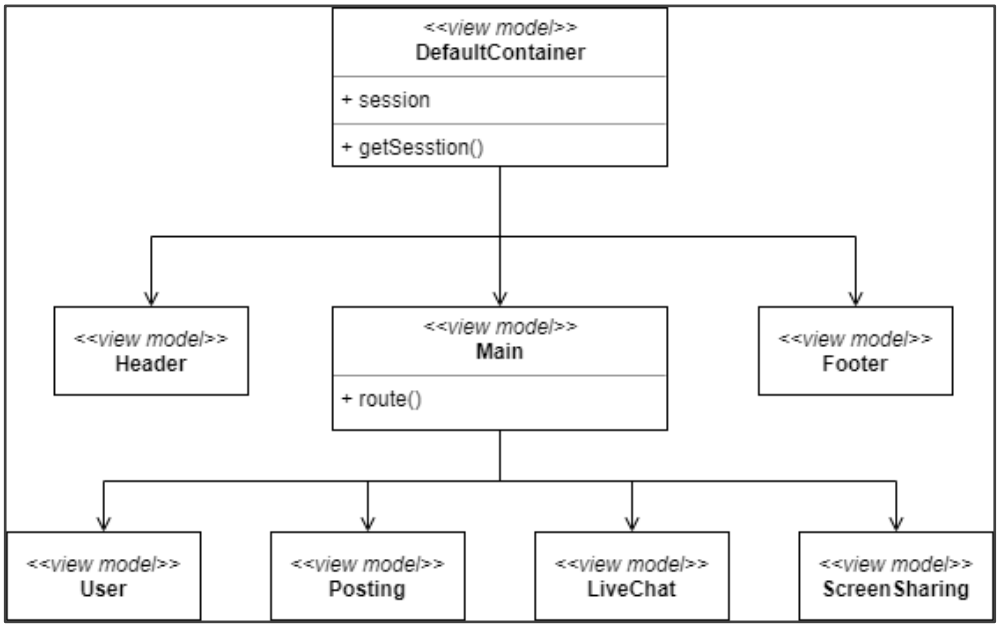

Figure 20 Overall UML Diagram

위 UML은 각 View Model들이 어떻게 구성되는 지 나타낸다.

DefaultContainer는 모든 화면이 공통적으로 표시하는 내용으로써, Header와 Footer가 포함된다. 이때 Main은 URL 주소에 따라 다른 View Model을 라우팅하여, 각각에 해당하는 콘텐츠를 보여준다.

3.4 ER Diagram

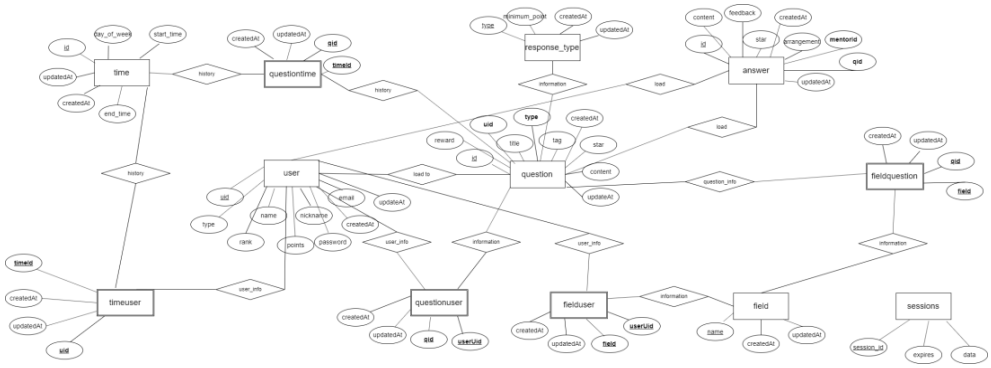

Figure 21 ER Diagram

3.5 Use-case Diagram

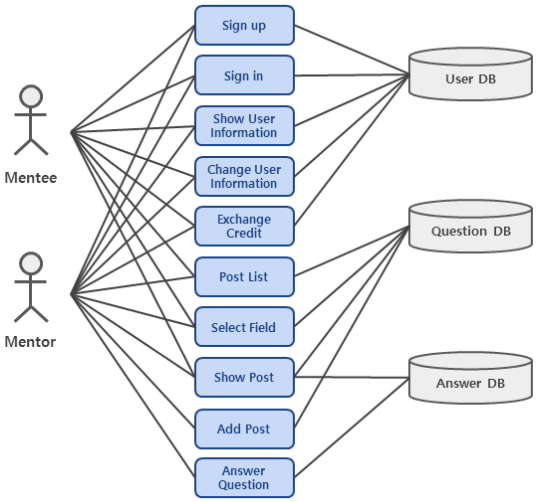

Figure 22 Use-case Diagram

3.6 Activity Diagram

A. Mentor

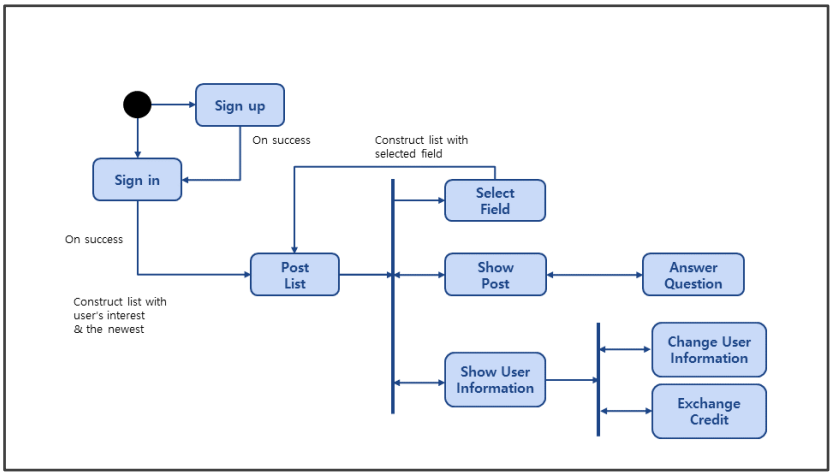

Figure 23 Activity Diagram: Mentor

B. Mentee

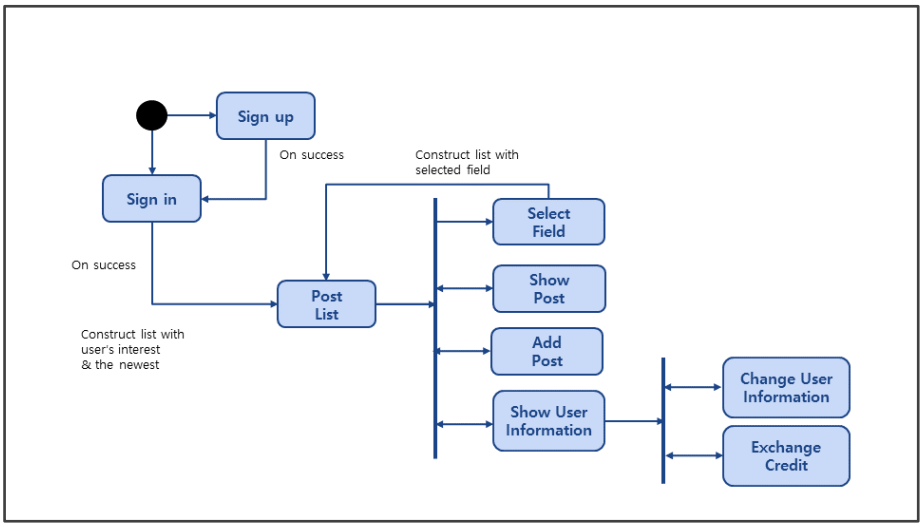

Figure 24 Activity Diagram: Mentee

4. User Management System

4.1 UML Diagram

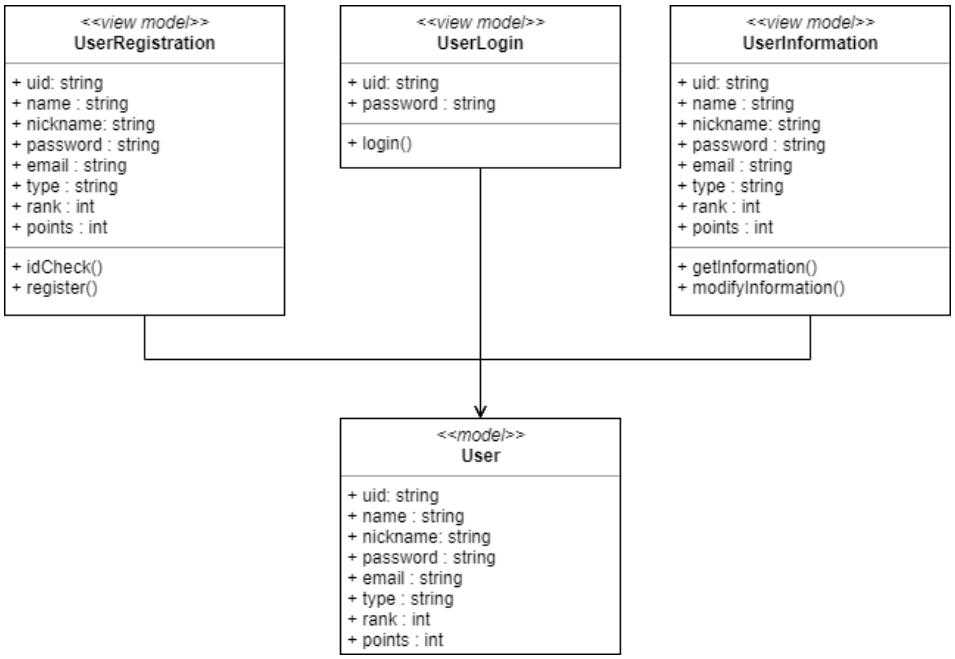

Figure 25 User Managerment System UML Diagram

A. <<view model>> UserRegistration

a. Attributes

i. + uid: string
사용자 ID

ii. + name : string
사용자 이름

iii. + nickname: string
사용자 닉네임

iv. + password : string
사용자 패스워드

v. + email : string
사용자 이메일

vi. + type : string

사용자 타입(멘토/멘티)

vii. + rank : int
사용자 랭크

viii. + points : int
사용자 크레딧

b. Methods

i. + idCheck()
중복된 ID인지 체크한다.

ii. + register()
입력된 사용자 정보를 바탕으로 가입 요청을 한다.

B. <<view model>> UserLogin

a. Attributes

i. + uid: string
사용자 ID

ii. + password : string
사용자 패스워드

b. Methods

i. + login()
입력된 사용자 정보를 바탕으로 로그인 요청을 한다.

C. <<view model>> UserInformation

a. Attributes

i. + uid: string
사용자 ID

ii. + name : string
사용자 이름

iii. + nickname: string
사용자 닉네임

iv. + password : string
사용자 패스워드

v. + email : string
사용자 이메일

vi. + type : string
사용자 타입(멘토/멘티)

vii. + rank : int
사용자 랭크

viii. + points : int
사용자 크레딧

b. Methods

i. + getInformation()
사용자 정보를 가져온다.

ii. + modifyInformation()
사용자 정보를 수정한다.

D. << model>> User

a. Attributes

i. + uid: string
사용자 ID

ii. + name : string

사용자 이름

iii. + nickname: string
사용자 닉네임

iv. + password : string
사용자 패스워드

v. + email : string
사용자 이메일

vi. + type : string
사용자 타입(멘토/멘티)

vii. + rank : int
사용자 랭크

viii. + points : int
사용자 크레딧

4.2 Sequence Diagram

A. Register

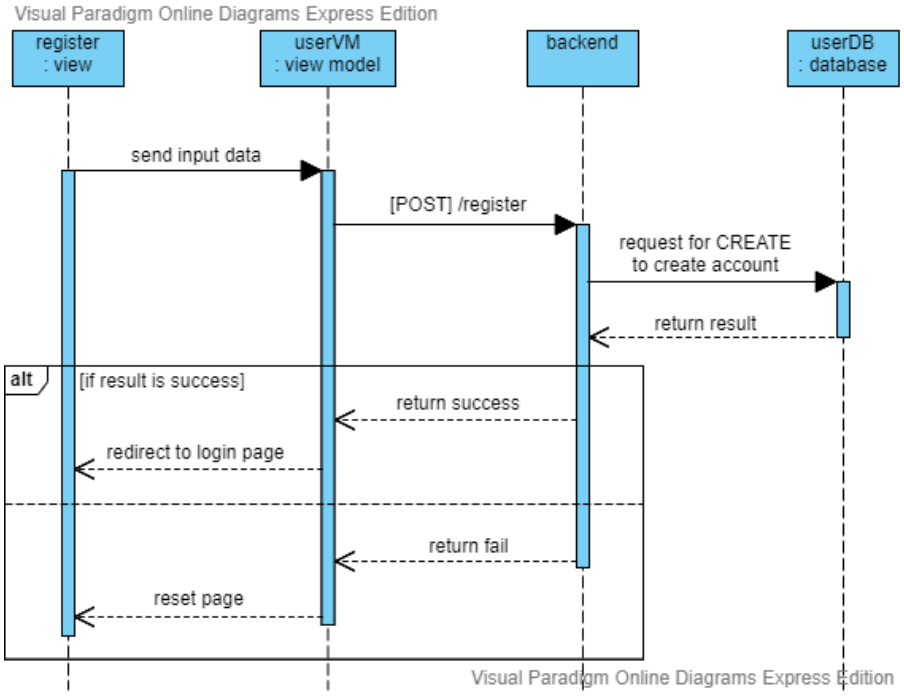

Figure 26 User Management System Sequence Diagram : Register

B. Login

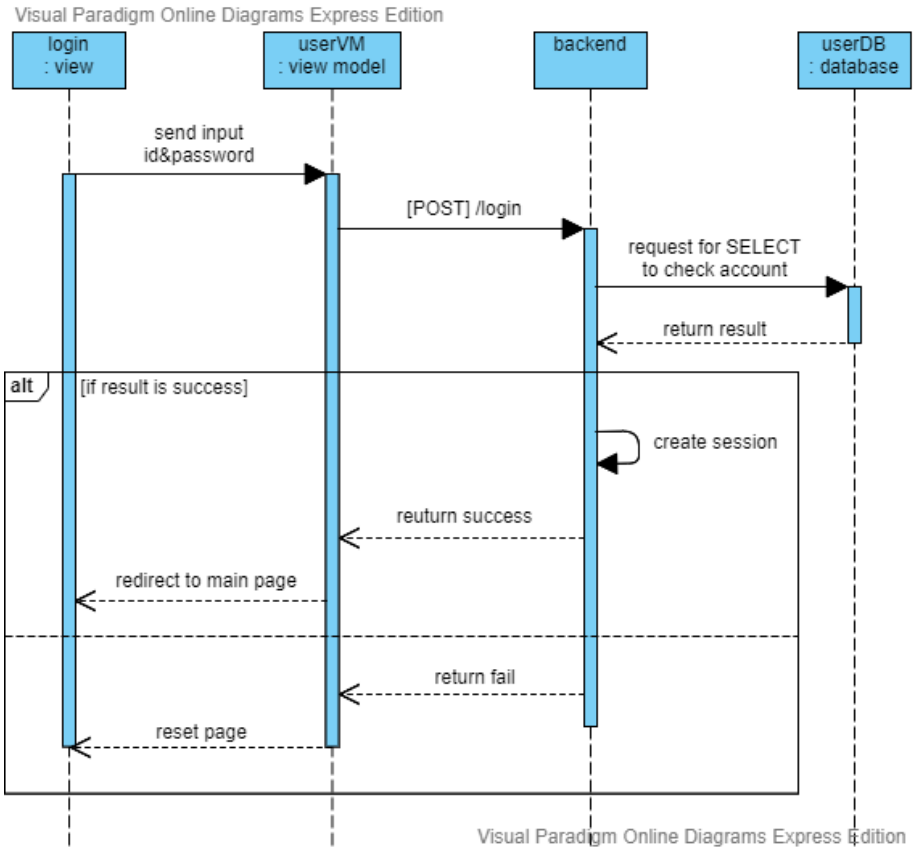

Figure 27 User Management System Sequence Diagram : Login

4.3 State Diagram

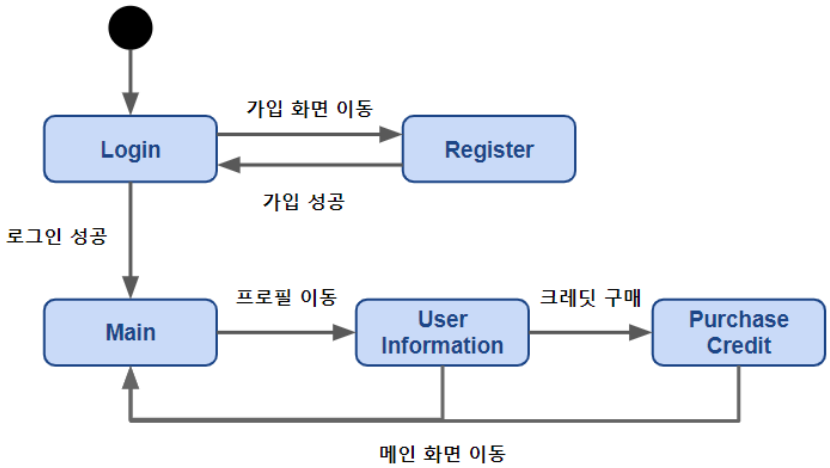

Figure 28 User Management System State Diagram

5. Posting System

5.1 UML Diagram

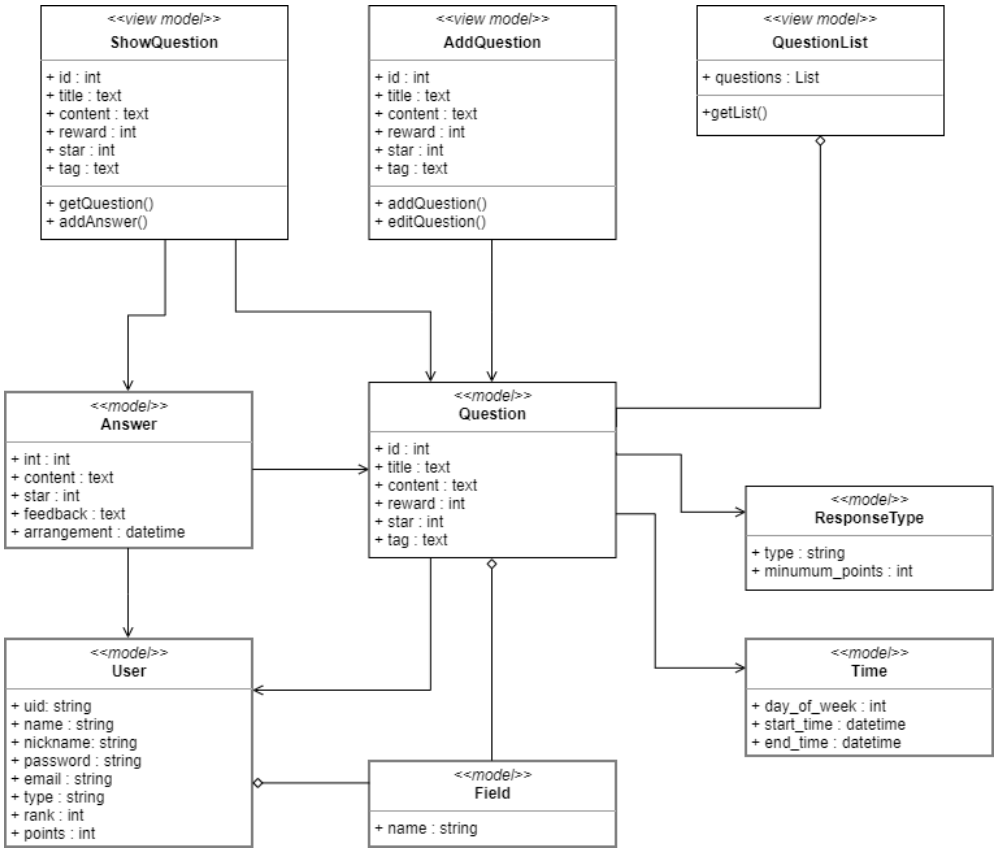

Figure 29 Posting System UML Diagram

A. <<view model>> ShowQuestion

a. Attributes

i. + id : int
문제의 ID

ii. + title : text
문제의 제목

iii. + content : text
문제의 내용

iv. + reward : int
문제의 보상 크레딧

v. + star : int
문제의 star(선호 포인트) 수

vi. + tag : text
문제의 tag 내용

b. Methods

i. + getQuestion()
문제 정보를 가져온다.

ii. + addAnswer()
입력된 내용을 바탕으로 답변을 생성한다.

B. <<view model>> AddQuestion

a. Attributes

i. + id : int
문제의 ID

ii. + title : text
문제의 제목

iii. + content : text
문제의 내용

iv. + reward : int
문제의 보상 크레딧

v. + star : int
문제의 star(선호 포인트) 수

vi. + tag : text
문제의 tag 내용

b. Methods

i. + addQuestion()
입력된 내용을 바탕으로 문제를 생성한다.

ii. + editQuestion()
입력된 내용으로 문제를 수정한다.

C. <<view model>> QuestionList

a. Attributes

i. + questions : List
문제 리스트

b. Methods

i. + getList()
문제 리스트 정보를 가져온다.

D. << model>> Answer

a. Attributes

i. + id : int
답변의 ID

ii. + content : text
답변의 내용

iii. + star : int
답변의 star(선호 포인트) 수

iv. + feedback : text
답변의 피드백

v. + arrangement : datetime
답변 예약 시간

E. << model>> Question

a. Attributes

i. + id : int
문제의 ID

ii. + title : text
문제의 제목

iii. + content : text
문제의 내용

iv. + reward : int
문제의 보상 크레딧

v. + star : int
문제의 star(선호 포인트) 수

vi. + tag : text
문제의 tag 내용

F. << model>> ResopnseType

a. Attributes

i. + type : string
답변 타입명

ii. + minumum_points : int
해당 답변 타입의 최소 보상 크레딧 값

G. << model>> Time

a. Attributes

i. + day_of_week : int
주차 수

ii. + start_time : datetime
시작 시간

iii. + end_time : datetime
종료 시간

H. << model>> Field

a. Attributes

i. + name : string
필드명

I. << model>> User

5.2 Sequence Diagram

A. Question List

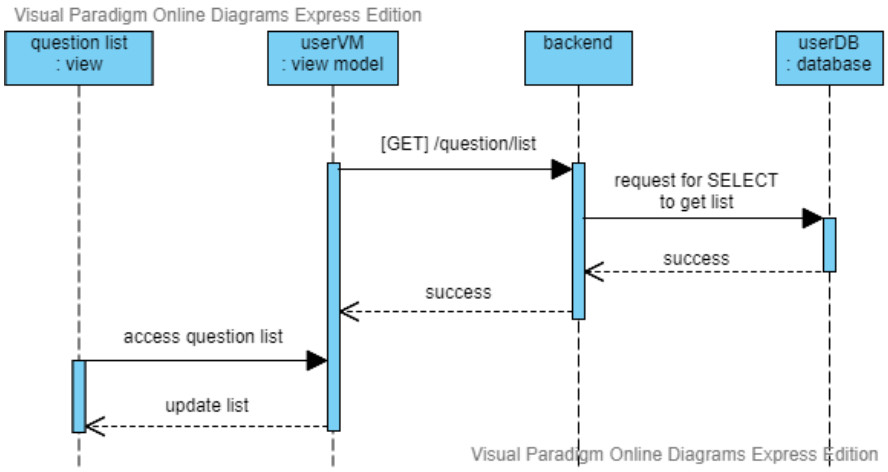

Figure 30 Posting System Sequence Diagram: Question List

B. Add Question

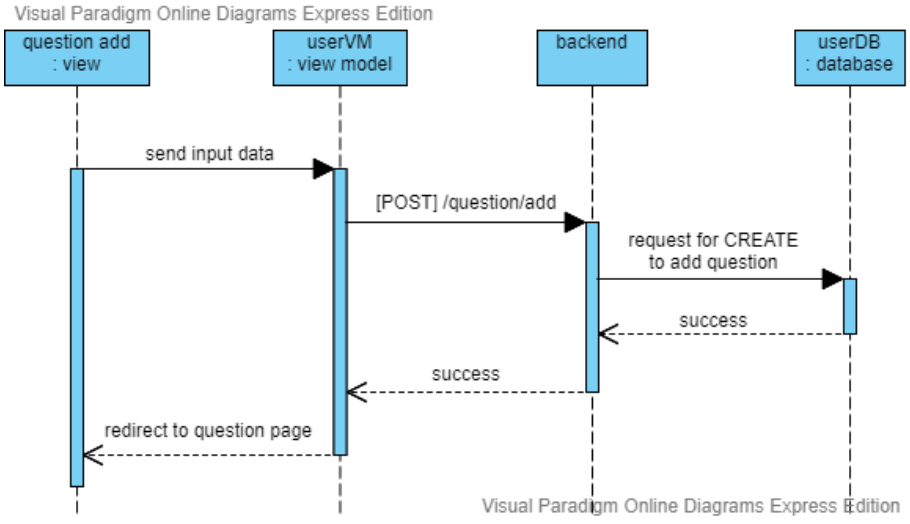

Figure 31 Posting System Sequence Diagram: Add Answer

5.3 State Diagram

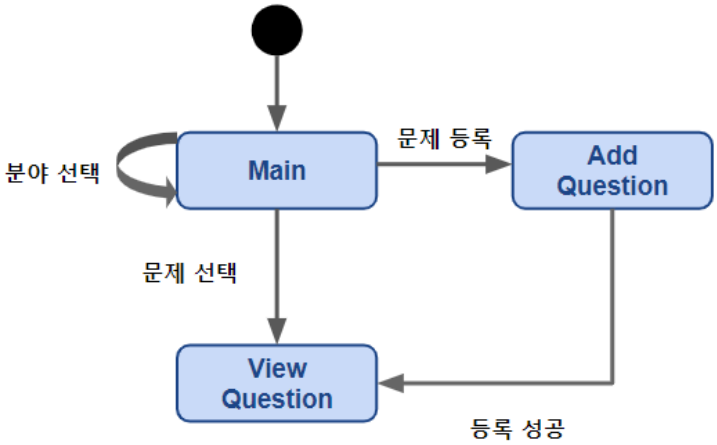

Figure 32 Posting System State Diagram

6. Live-chatting System

6.1 UML Diagram

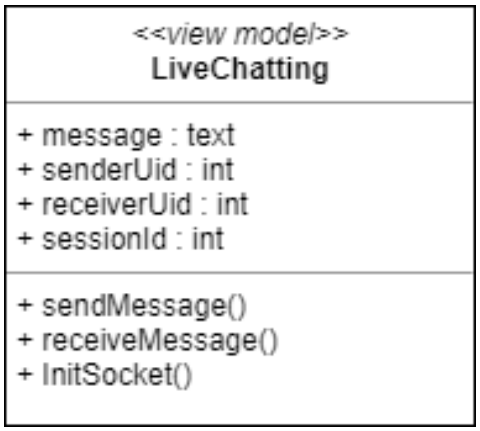

Figure 33 Live-chatting System UML Diagram

A. <<view model>>

a. Attributes

i. + message : text
텍스트 메시지

ii. + senderUid : int
송신자 ID

iii. + receiverUid : int
수신자 ID

iv. + sessionId : int
세션 ID

b. Methods

i. + sendMessage()
메시지를 전송한다.

ii. + receiveMessage()
메시지를 수신한다.

iii. + InitSocket()
소켓을 초기화한다.

7. Screen-sharing System

7.1 UML Diagram

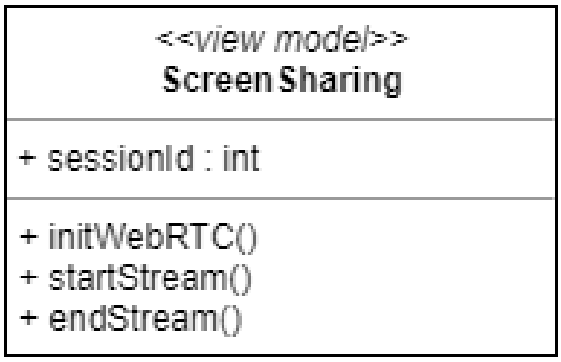

Figure 34 Screen-sharing System UML Diagram

A. <<view model>> ScreenSharing

a. Attributes

i. + sessionId : int
세션의 ID

b. Methods

i. + initWebRTC()
WebRTC를 초기화한다.

ii. + startStream
스트리밍을 시작한다.

iii. + endStream
스트리밍을 종료한다.

8. Protocol Design

8.1 Rest API

A. User Management

a. Register

i. Request

Method	POST	
URI	/rest/register	
Request Body	id (String)	personal identification unique to each user
	password (String)	user password
	name (String)	name of the user
	nickname (String)	nickname of the user, defaults to name if not given
	email (String)	email address of the user, used for id / password recovery
	type (String)	specify whether user is mentor or mentee
	field(s) (String)	fields of interest (mentee) or fields of expertise (mentor)
	available_time(s)	(mentor only) available

	(day of week, time)	time for live help

Table 1 REST API: Register (Request)

ii. Response

Success Code	200 OK	
Failure code	400 Bad Request	
Success Response Body	result	true
Failure Response Body	result	false
	msg	Error message

Table 2 REST API: Register (Response)

b. Login

i. Request

Method	POST	
URI	/rest/login	
Request Body	id (String)	registered id
	password (String)	corresponding user password

Table 3 REST API: Login (Request)

ii. Response

Success Code	200 OK	
Failure code	401 Unauthorized	
Success Response Body	result	true
Failure Response Body	result	false
	msg	Error message

Table 4 REST API: Login (Response)

c. Find ID

i. Request

Method	GET	
URI	/rest/user/id	
Request Body	name (String)	name of the registered user
	email (String)	email of the registered user

Table 5 REST API: Find ID (Request)

ii. Response

Success Code	200 OK	
Failure code	404 Not Found	
Success Response Body	uid	user id of the searched user
Failure Response Body	result	false
	msg	Error message

Table 6 REST API: Find ID (Response)

d. Find Password

i. Request

Method	PUT	
URI	/rest/user/password	
Request Body	id (String)	registered ID
	name (String)	registered name
	email (String)	email of the registered user

Table 7 REST API: Find Password (Request)

ii. Response

Success Code	200 OK	
Failure code	404 Not Found	
Success Response Body	result	true
Failure Response Body	result	false
	msg	Error message

Table 8 REST API: Find Password (Response)

e. Modify User Information

i. Request

Method	PUT	
URI	/rest/user	
Request Body	password (String)	user password
	name (String)	name of the user
	nickname (String)	nickname of the user, defaults to name if not given

Table 9 REST API: Modify User Information (Request)

ii. Response

Success Code	200 OK	
Failure code	401 Unauthorized	
Success Response Body	result	true
Failure Response Body	result	false
	msg	Error message

Table 10 REST API: Modify User Information (Response)

f. Get session

i. Request

Method	GET	
URI	/rest/user/session	
Request Body	None	

Table 11 REST API: Get Session (Request)

ii. Response

Success Code	200 OK		
Failure code	404 Not Found		
Success Response Body	result	true	
	user	uid	user ID
		name	name of the user
		email	email of the user
		nickname	nickname of the user
		rank	rank of the user
Failure Response Body	result	false	

B. Credit

a. Purchase Credit

i. Request

Method	POST	
URI	/rest/credits	
Request Body	amount (int)	amount of credits to purchase

Table 12 REST API: Purchase Credit (Request)

ii. Response

Success Code	200 OK	
Failure code	401 Unauthorized	
Success Response Body	result	true
	amount	amount added
	total	total amount of credits that the user owns
Failure Response Body	result	false
	msg	Error message

Table 13 REST API: Purchase Credit (Response)

b. Exchange Credit to Currency

i. Request

Method	PUT	
URI	/rest/credits	
Request Body	amount (int)	amount of credits to exchange

Table 14 REST API: Exchange Credit to Currency (Request)

ii. Response

Success Code	200 OK	
Failure code	401 Unauthorized	
Success Response Body	result	true
	amount	amount exchanged
	total	total amount of credits left
Failure Response Body	result	false
	msg	Error message

Table 15 REST API: Exchange Credit to Currency (Response)

C. Question & Answer

a. View Question List

i. Request

Method	GET	
URI	/rest/question/list	
Request Body	None	

Table 16 REST API: View Question List (Request)

ii. Response

Success Code	200 OK	
Failure code	401 Unauthorized	
Success Response Body	result	true
	list	json arary of questions
Failure Response Body	result	false
	msg	Error message

Table 17 REST API: View Question List (Response)

b. View Question

i. Request

Method	GET	
URI	/rest/question/:id	
Request Body	None	

Table 18 REST API: View Question (Request)

ii. Response

Success Code	200 OK		
Failure code	401 Unauthorized		
Success Response Body	result	true	
	question	title	title of the question
		content	content of the question
		reward	reward for the question
		tag	question tag
		type	answer type
		uid	user who registered the question
	Answer	content	content of the answer

	(only authorized users)	star	star rating of the answer
		feedback	Mentee feedback of the answer
		mentor	uid of the mentor
Failure Response Body	result	false	
	msg	Error message	

Table 19 REST API: View Question (Response)

c. Register Question

i. Request

Method	POST	
URI	/rest/question	
Request Body	title	title of the question
	content	content of the question
	reward	reward for the question
	tag	question tag
	type	answer type

Table 20 REST API: Register Question (Request)

ii. Response

Success Code	200 OK	
Failure code	401 Unauthorized	
Success Response Body	result	true
Failure Response Body	result	false
	msg	Error message

Table 21 REST API: Register Question (Response)

d. Answer Question

i. Request

Method	POST	
URI	/rest/answer/:qid	
Request Body	content	content of the answer

Table 22 REST API: Answer Question (Request)

ii. Response

Success Code	200 OK	
Failure code	401 Unauthorized	
Success Response Body	result	true
Failure Response Body	result	false
	msg	Error message

Table 23 REST API: Answer Question (Response)

e. Make Arrangement

i. Request

Method	POST	
URI	/rest/question/arrange/:qid	
Request Body	time	time of arrangement

Table 24 REST API: Make Arrangement (Request)

ii. Response

Success Code	200 OK	
Failure code	401 Unauthorized	
Success Response Body	result	true
Failure Response Body	result	false
	msg	Error message

Table 25 REST API: Make Arrangement (Response)

D. Live-chatting

a. Get Chatlog by User

i. Request

Method	GET	
URI	/rest/chatlog/user/:uid	
Request Body	None	

Table 26 REST API: Get Chatlog by User (Request)

ii. Response

Success Code	200 OK	
Failure code	401 Unauthorized	
Success Response Body	result	true
	log	json array of chat logs
Failure Response Body	result	false
	msg	Error message

Table 27 REST API: Get Chatlog by User (Response)

b. Get Chatlog by Question

i. Request

Method	GET	
URI	/rest/chatlog/question/:qid	
Request Body	None	

Table 28 REST API: Get Chatlog by Question (Request)

ii. Response

Failure code	401 Unauthorized	
Success Response Body	result	true
	log	json array of chat logs
Failure Response Body	result	false
	msg	Error message

Table 29 REST API: Get Chatlog by Question (Response)

8.2 Socket Communication and Web RTC

A. Live-chatting

WebSocket and Socket.io is used for socket communication (live-chatting) between clients.

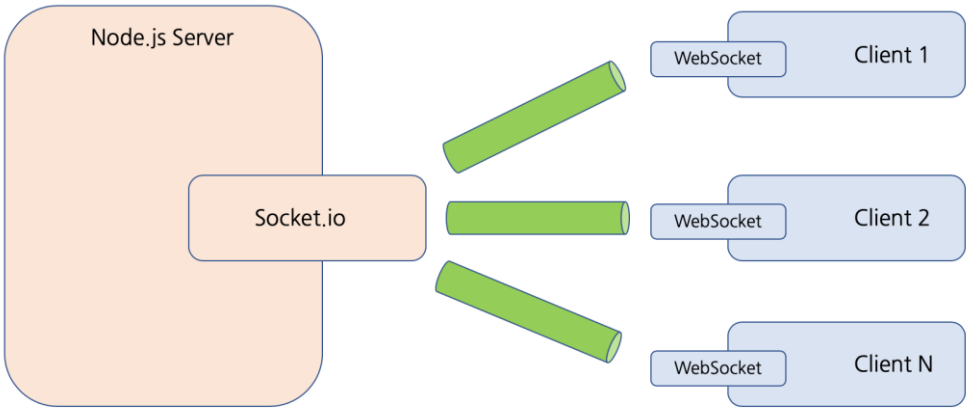

Figure 35 Live-chatting Communication

a. Client

i. on('submit')

send message to another client in the following format:
"/w "{uid}" {message content}"

ii. on('receive message')

append displayed message to client

b. Server

i. on('receive message')

save message content to DB and emit content to the receiver of the message

B. Screen-sharing

Screen-sharing uses WebRTC for media streaming.

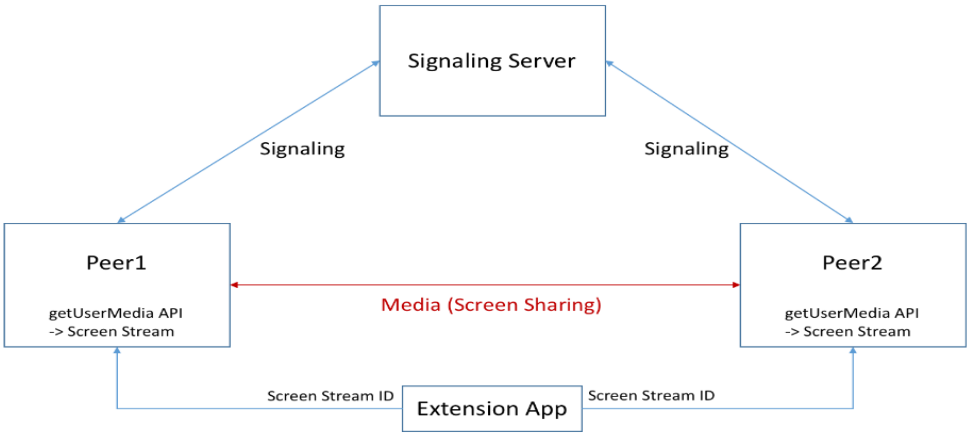

Figure 36 Screen-sharing Communication

9. Database Design

9.1 Objectives

이번 장에서 이전 요구사항 명세서에서 기술했던 데이터베이스에 대하여 보다 구체적으로 데이터베이스를 디자인하였다. 요구사항 명세서의 데이터베이스 요구사항들과 데이터베이스 구조 모델을 바탕으로 ER Diagram (Entity Relationship Diagram) 을 작성 하였고 이를 바탕으로 Relational Schema를 작성하였다. 이후 Normalization을 거쳐 SQL DDL을 작성하였다.

9.2 ER Diagram

ER Diagram(개체-관계 모델)에서 개체(Entity)는 분리된 물체 하나를 표현한다. 개체는 명사 하나에 해당한다고 생각하며 직사각형으로 표현된다. 관계(Relationship)는 두 개 이상의 개체들이 어떻게 서로 연관되어 있는 지를 기록하며 다이아몬드로 표현된다. 각각의 개체와 관계는 속성(Attribute)을 가질 수 있으며 관계집합이나 개체집합에 선으로 연결시킨 타원형들로 그린다. 모든 개체는 고유하게 식별되는 속성 집합을 가지고 있어야 하며 고유 식별 속성 집합은 개체의 기본 키(Primary key)라 부르며 밑줄을 그어 표시한다.

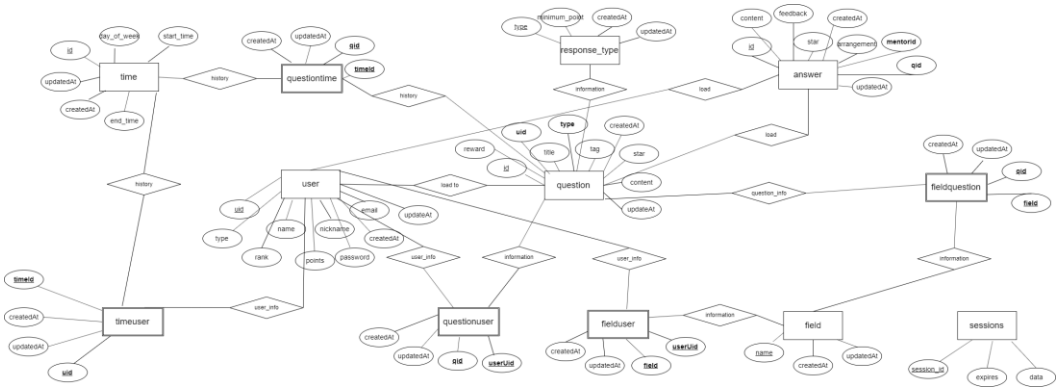

Figure 37 Overall ER Diagram

9.3 Entitiy

A. user

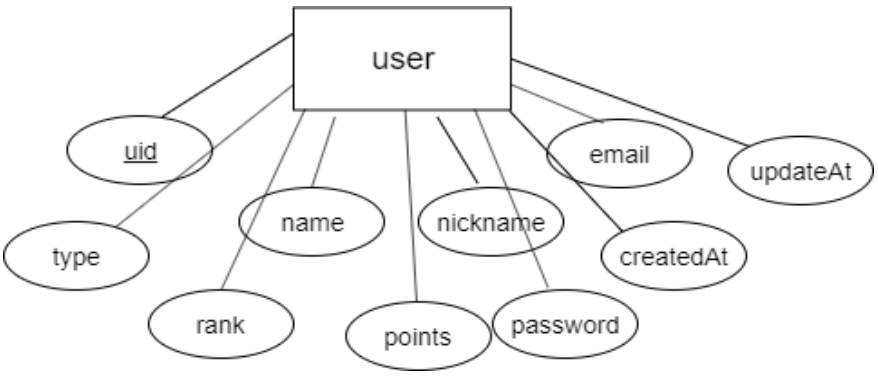

Figure 38 Entitiy: user

User Entity는 사용자의 정보를 나타낸다. uid, name, nickname, email, type, rank, points, password, createdAt, updatedAt 의 속성을 가지고 있으며 uid의 속성은 primary key 이다.

B. time

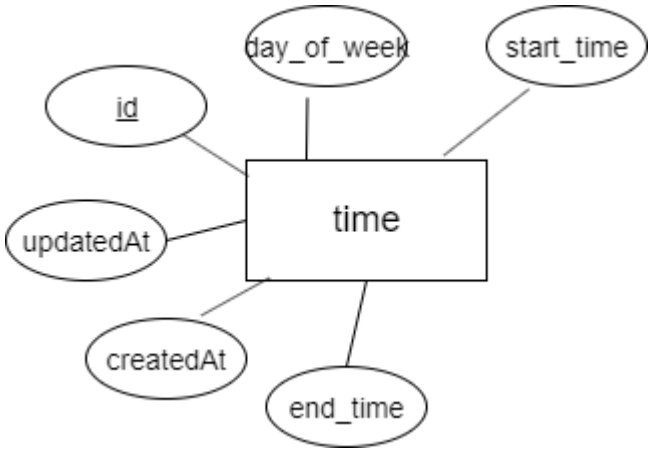

Figure 39 Entitiy: time

Time Entity는 site 내 시간 정보를 나타낸다. id, day_of_week, start_time, updatedAt, createdAt, end_time의 속성을 가지고 있으며, id의 속성은 primary key 이다.

C. questiontime

Figure 40 Entitiy: questiontime

Question Entity는 question service 이용 가능 시간을 나타낸다. qid, timeId, createdAt, updatedAt의 속성을 가지고 있으며, weak entity로 qid와 timeId의 속성은 foreign key이다.

D. timeuser

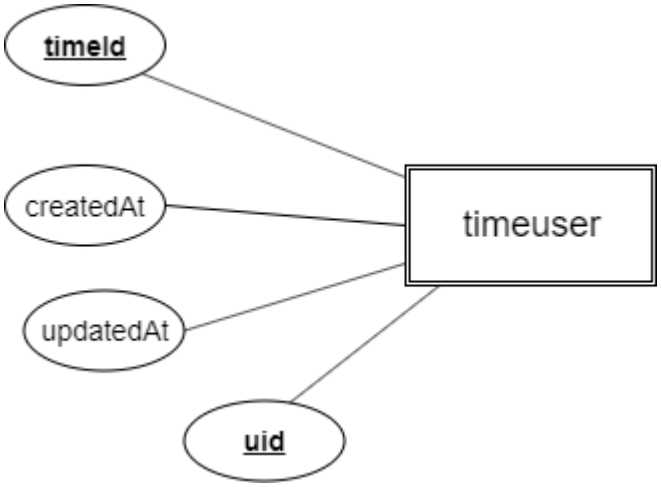

Figure 41 Entitiy: timeuser

Timeuser Entity는 멘토가 답변가능한 시간들을 나타낸다. timeId, uid, createdAt, updatedAt의 속성을 가지고 있으며, weak entity로 timeId, uid의 속성은 foreign key 이다.

E. question

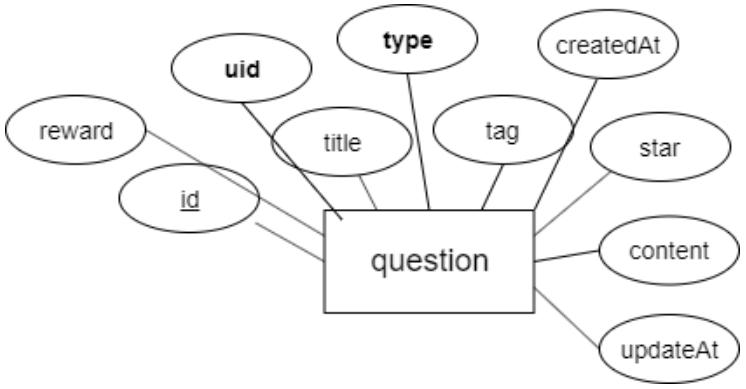

Figure 42 Entitiy: question

Question Entity는 사용자가 업로드한 질문에 대한 정보들을 나타낸다. id, title, tag, star, reward, uid, type, content, createdAt, updatedAt의 속성을 가지고 있으며, id의 속성은 primary key 이다.

F. questionuser

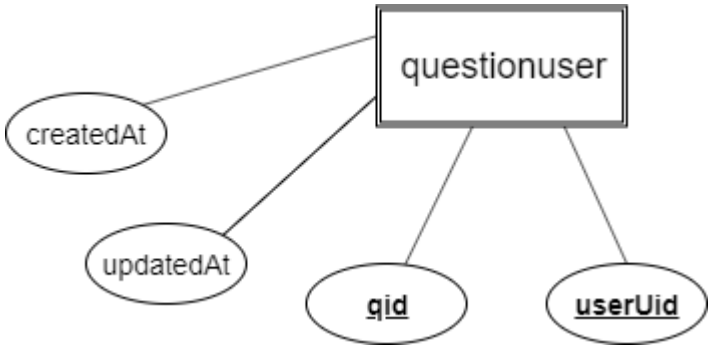

Figure 43 Entitiy: questionuser

Questionuser Entity는 질문을 올린 유저의 정보를 나타낸다. qid, userUid, updatedAt, createdAt의 속성을 가지고 있으며 weak entity로 qid, userUid의 속성은 foreign key 이다.

G. response_type

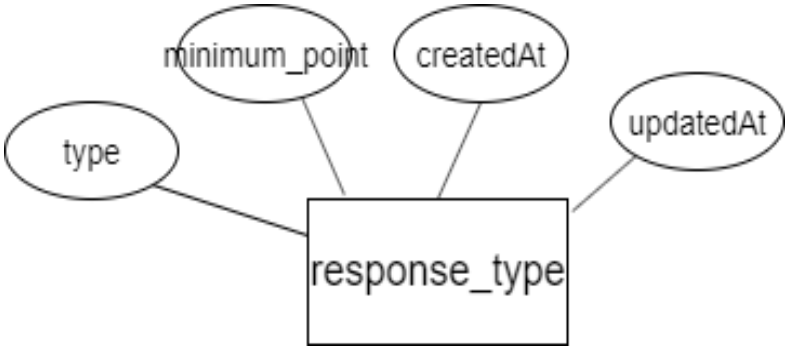

Figure 44 Entitiy: response_type

type, minimum_point, createdAt, updatedAt의 속성을 가지고 있으며 type의 속성은 primary key이다.

H. fielduser

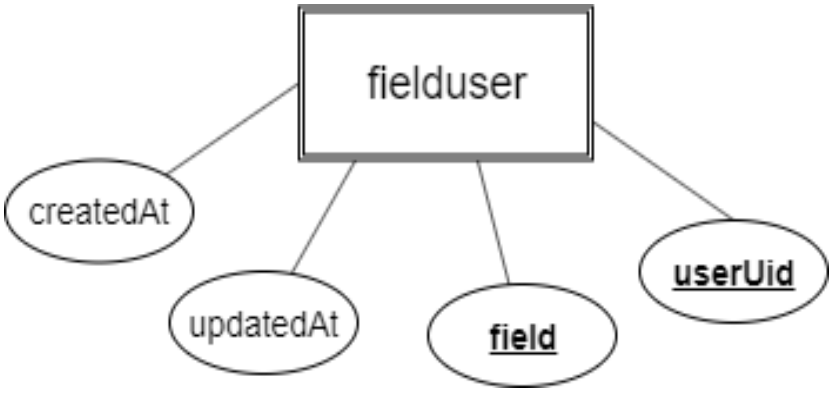

Figure 45 Entitiy: fielduser

Fielduser Entity는 user가 이용하는 분야의 정보들을 나타낸다. field, userUid, createdAt, updatedAt의 속성을 가지고 있다. weak entity로 userUid, field의 속성은 foreign key이다.

I. answer

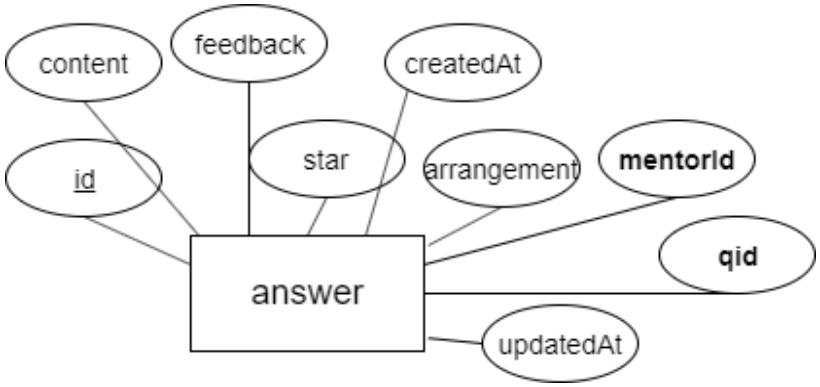

Figure 46 Entitiy: answer

Answer Entity는 질문에 대한 답변에 대한 정보들을 나타낸다. id, star, arrangement, content, feedback, qid, mentorId, createdAt, updatedAt의 속성을 가지고 있으며, id의 속성은 primary key이다.

J. field

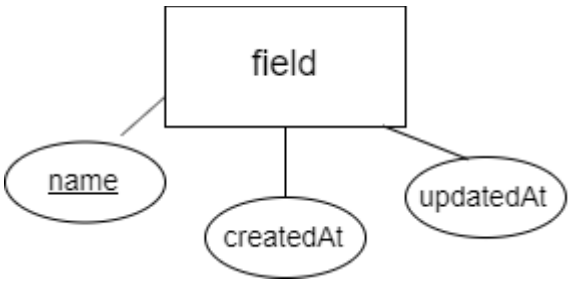

Figure 47 Entitiy: field

Field Entity 는 분야의 종류에 대한 정보를 가지고 있는 entity 이다. name. createdAt, updatedAt의 속성을 가지고 있으며, name의 속성은 primary key 이다.

K. fieldquestion

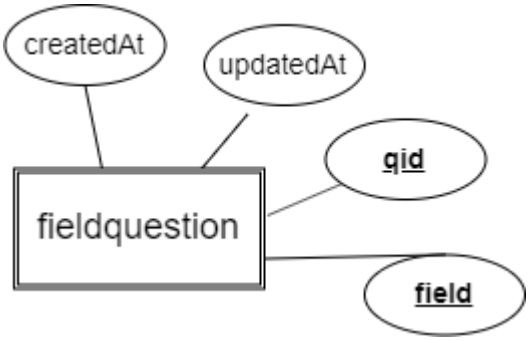

Figure 48 Entitiy: fieldquestion

Fieldquestion Entity는 field의 종류와 해당 field의 question에 대한 정보를 나타낸다. field, qid, createdAt, updatedAt의 속성을 가지고 있다. weak entity로 qid, field의 속성은 foreign key이다.

L. sessions

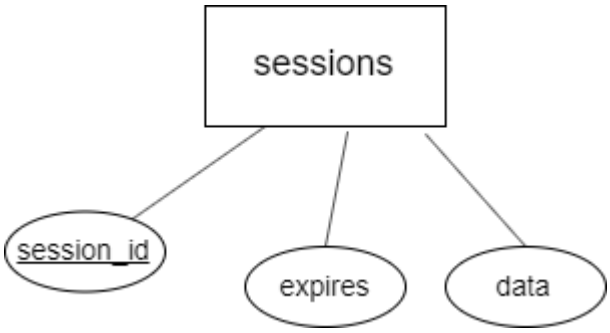

Figure 49 Entitiy: sessions

session_id, data, expires의 속성을 가지고 있으며, session_id의 속성은 primary key이다.

9.4 SQL DDL

A. user

```
CREATE TABLE `user` (
   `uid` varchar(255) NOT NULL,
   `name` varchar(255) NOT NULL,
   `nickname` varchar(255) NOT NULL,
   `password` varchar(255) NOT NULL,
   `email` varchar(255) NOT NULL,
   `type` varchar(255) NOT NULL,
   `rank` int(11) NOT NULL,
   `points` int(11) DEFAULT NULL,
   `createdAt` datetime NOT NULL,
   `updatedAt` datetime NOT NULL
)
```

```
ALTER TABLE `user`
   ADD PRIMARY KEY (`uid`),
   ADD UNIQUE KEY `uid` (`uid`);
```

Table 30 SQL DDL: user

B. time

```
CREATE TABLE `time` (
   `id` int(11) NOT NULL,
   `day_of_week` int(11) NOT NULL,
   `start_time` datetime NOT NULL,
   `end_time` datetime NOT NULL,
   `createdAt` datetime NOT NULL,
   `updatedAt` datetime NOT NULL
)
```

```
ALTER TABLE `time`
   ADD PRIMARY KEY (`id`);
```

Table 31 SQL DDL: time

C. questiontime

```
CREATE TABLE `questiontime` (
  `createdAt` datetime NOT NULL,
  `updatedAt` datetime NOT NULL,
  `qid` int(11) NOT NULL,
  `timeId` int(11) NOT NULL
)
```

```
ALTER TABLE `questiontime`
  ADD PRIMARY KEY (`qid`,`timeId`),
  ADD KEY `timeId` (`timeId`);
```

```
ALTER TABLE `questiontime`
  ADD CONSTRAINT `questiontime_ibfk_1` FOREIGN KEY (`qid`) REFERENCES `question` (`id`) ON DELETE CASCADE ON UPDATE CASCADE,
  ADD CONSTRAINT `questiontime_ibfk_2` FOREIGN KEY (`timeId`) REFERENCES `time` (`id`) ON DELETE CASCADE ON UPDATE CASCADE;
```

Table 32 SQL DDL: questiontime

D. timeuser

```
CREATE TABLE `timeuser` (
  `createdAt` datetime NOT NULL,
  `updatedAt` datetime NOT NULL,
  `uid` varchar(255) NOT NULL,
  `timeId` int(11) NOT NULL
)
```

```
ALTER TABLE `timeuser`
  ADD PRIMARY KEY (`uid`,`timeId`),
  ADD KEY `timeId` (`timeId`);
```

```
ALTER TABLE `timeuser`
  ADD CONSTRAINT `timeuser_ibfk_1` FOREIGN KEY (`uid`) REFERENCES `user` (`uid`) ON DELETE CASCADE ON UPDATE CASCADE,
  ADD CONSTRAINT `timeuser_ibfk_2` FOREIGN KEY (`timeId`) REFERENCES `time` (`id`) ON DELETE CASCADE ON UPDATE CASCADE;
```

Table 33 SQL DDL: timeuser

E. question

```
CREATE TABLE `question` (
  `id` int(11) NOT NULL,
  `title` text NOT NULL,
  `content` text NOT NULL,
  `reward` int(11) NOT NULL,
  `star` int(11) NOT NULL,
  `tag` text,
  `createdAt` datetime NOT NULL,
  `updatedAt` datetime NOT NULL,
  `type` varchar(255) DEFAULT NULL,
  `uid` varchar(255) DEFAULT NULL
)
```

```
ALTER TABLE `question`
  ADD PRIMARY KEY (`id`),
  ADD KEY `type` (`type`),
  ADD KEY `uid` (`uid`);
```

```
ALTER TABLE `question`
  ADD CONSTRAINT `question_ibfk_1` FOREIGN KEY (`type`) REFERENCES `response_type` (`type`)
```

```
ON DELETE SET NULL ON UPDATE CASCADE,

  ADD CONSTRAINT `question_ibfk_2` FOREIGN KEY (`uid`) REFERENCES `user` (`uid`) ON DELETE SET NULL ON UPDATE CASCADE;
```

Table 34 SQL DDL: question

F. questionuser

```
CREATE TABLE `questionuser` (
  `createdAt` datetime NOT NULL,
  `updatedAt` datetime NOT NULL,
  `qid` int(11) NOT NULL,
  `userUid` varchar(255) NOT NULL
)
```

```
ALTER TABLE `questionuser`
  ADD PRIMARY KEY (`qid`,`userUid`),
  ADD KEY `userUid` (`userUid`);
```

```
ALTER TABLE `questionuser`
  ADD CONSTRAINT `questionuser_ibfk_1` FOREIGN KEY (`qid`) REFERENCES `question` (`id`) ON DELETE CASCADE ON UPDATE CASCADE,
  ADD CONSTRAINT `questionuser_ibfk_2` FOREIGN KEY (`userUid`) REFERENCES `user` (`uid`) ON DELETE CASCADE ON UPDATE CASCADE;
```

Table 35 SQL DDL: questionuser

G. response_type

```
CREATE TABLE `response_type` (
  `type` varchar(255) NOT NULL,
  `minimum_point` int(11) NOT NULL,
  `createdAt` datetime NOT NULL,
  `updatedAt` datetime NOT NULL
)
```

```
ALTER TABLE `response_type`
  ADD PRIMARY KEY (`type`);
```

Table 36 SQL DDL: response_type

H. fielduser

```
CREATE TABLE `fielduser` (
  `createdAt` datetime NOT NULL,
  `updatedAt` datetime NOT NULL,
  `field` varchar(255) NOT NULL,
  `userUid` varchar(255) NOT NULL
)
```

```
ALTER TABLE `fielduser`
  ADD PRIMARY KEY (`field`,`userUid`),
  ADD KEY `userUid` (`userUid`);
```

```
ALTER TABLE `fielduser`
  ADD CONSTRAINT `fielduser_ibfk_1` FOREIGN KEY (`field`) REFERENCES `field` (`name`) ON DELETE CASCADE ON UPDATE CASCADE,
  ADD CONSTRAINT `fielduser_ibfk_2` FOREIGN KEY (`userUid`) REFERENCES `user` (`uid`) ON DELETE CASCADE ON UPDATE CASCADE;
```

Table 37 SQL DDL: fielduser

l. answer

```
CREATE TABLE `answer` (
  `id` int(11) NOT NULL,
  `content` text NOT NULL,
  `star` int(11) NOT NULL,
  `feedback` text,
  `arrangement` datetime DEFAULT NULL,
  `createdAt` datetime NOT NULL,
  `updatedAt` datetime NOT NULL,
  `qid` int(11) DEFAULT NULL,
  `mentorId` varchar(255) DEFAULT NULL
)
```

```
ALTER TABLE `answer`
  ADD PRIMARY KEY (`id`),
  ADD KEY `qid` (`qid`),
  ADD KEY `mentorId` (`mentorId`);
```

```
ALTER TABLE `answer`
  ADD CONSTRAINT `answer_ibfk_1` FOREIGN KEY (`qid`) REFERENCES `question` (`id`) ON DELETE SET NULL ON UPDATE CASCADE,
```

```
  ADD CONSTRAINT `answer_ibfk_2` FOREIGN KEY (`mentorId`) REFERENCES `user` (`uid`) ON DELETE SET NULL ON UPDATE CASCADE;
```

Table 38 SQL DDL: answer

J. field

```
CREATE TABLE `field` (
  `name` varchar(255) NOT NULL,
  `createdAt` datetime NOT NULL,
  `updatedAt` datetime NOT NULL
)
```

```
ALTER TABLE `field`
  ADD PRIMARY KEY (`name`);
```

Table 39 SQL DDL: field

K. fieldquestion

```
CREATE TABLE `fieldquestion` (
  `createdAt` datetime NOT NULL,
  `updatedAt` datetime NOT NULL,
  `qid` int(11) NOT NULL,
  `field` varchar(255) NOT NULL
)
```

```
ALTER TABLE `fieldquestion`
  ADD PRIMARY KEY (`qid`,`field`),
  ADD KEY `field` (`field`);
```

```
ALTER TABLE `fieldquestion`
  ADD CONSTRAINT `fieldquestion_ibfk_1` FOREIGN KEY (`qid`) REFERENCES `question` (`id`) ON DELETE CASCADE ON UPDATE CASCADE,
  ADD CONSTRAINT `fieldquestion_ibfk_2` FOREIGN KEY (`field`) REFERENCES `field` (`name`) ON DELETE CASCADE ON UPDATE CASCADE;
```

Table 40 SQL DDL: fieldquestion

L. sessions

```
CREATE TABLE `sessions` (
  `session_id` varchar(128) CHARACTER SET utf8mb4 COLLATE utf8mb4_bin NOT NULL,
  `expires` int(11) UNSIGNED NOT NULL,
  `data` text CHARACTER SET utf8mb4 COLLATE utf8mb4_bin
)
```

```
ALTER TABLE `sessions`
  ADD PRIMARY KEY (`session_id`);
```

Table 41 SQL DDL: sessions

10. Testing Plan

10.1 Objectives

본 문서에서 다루는 시스템인 QAHub가 Requirement specification에 표기된 고객의 requirement들을만족시킬 수 있는 지 확인함과 동시에 시스템이 본 문서인 Design specification에 기술된 설계대로 작성되었고, 설계한 의도대로 작업을 수행하는 지 확인하며, 작업을 수행하는 과정에서 작업 수행 실패나 의도되지 않은 결함이 발생하지 않는지 테스트하기 위한 계획을 수립한다.

이하 항목에서는 계획한 테스트들의 Policy와, 실제로 테스트를 수행하는 데 필요한 Test case들이 어떤 기준으로 작성될지에 대해 기술한다.

10.2 Testing Process

A. Development Testing

Development test는 시스템을 실제로 개발하는 과정에서 시행하는 테스트이다. 개별 Component들을 일일히 테스트하는 Component testing, Component들을 통합한 후 통합한 시스템 전체를 테스트하는 System testing, 마지막으로 실제 고객의 데이터를 이용하여 시스템이 고객의 Needs를 만족시킬 수 있는가를 테스트하는 Acceptance testing으로 이루어진다.

B. Release Testing

Release test는 사용자에게 시스템을 배포하기 전에 시행하는 테스트로써, Requirement specification에 기록된 Requirement들이 모두 충족되었는지, 시스템이 의도된 대로 작동하는지, 치명적인 결함은 없는지 등을 확인한다.

C. User Testing

앞의 테스트들이 개발자들의 개발, 사용 환경 아래에서 시행되었다면, User test는 사용자의 실제 사용 환경에서 시스템이 어떻게 작동하는지, 의도되지 않은 오작동은 없는지 확인하는 테스트이다.

10.3 Test Cases

A. User Registration

항목	설명
User's Action	회원 가입 폼에 사용자 정보를 입력하여 회원 가입 요청을 보낸다.
System's Action	사용자가 제출한 정보를 User DB와 비교하여, 중복되거나 작성되지 않은 항목, 허용되지 않는 입력이 없다면 요청을 승인하고, 사용자 정보를 User DB에 저장하고 그 결과를 사용자에게 고지한다.
Restriction	1. 중복된 아이디는 허용하지 않는다. 2. '필수'로 표시된 항목은 반드시 작성해야 한다. 3. 비밀번호는 8자리 이상의 알파벳 대소문자, 숫자의 조합으로 이루어져야 한다. 4. 사용자가 비밀번호를 정확히 입력했는지 확인하기 위해 '비밀번호 확인' 필드를 만들어 사용자의 입력을 한 번 더 확인해야 한다.
Expected Result	사용자는 등록한 정보를 이용하여 QAHub에 로그인할 수 있게 된다.

Table 42 Test Cases: User Registration

B. Login

항목	설명
User's Action	로그인 폼에 사용자가 사전에 등록했던 ID와 Password를 입력하여 로그인 요청을 보낸다.
System's Action	사용자가 입력한 ID와 Password를 User DB에 저장된, 사용자가 사전에 제출했던 정보와 비교하여 결과가 일치한다면 로그인 요청을 승인하고 사용자를 사이트의 메인 페이지로 리디렉션시킨다.
Restriction	1. ID와 Password 항목은 반드시 모두 작성되어야 한다. 2. 로그인에 실패했을 경우, 반드시 ID와 Password에 대한 내용 모두를 포함한 실패 메시지를 표시하여, 사용자의 정보를 탈취하려는 공격자에게 유의미한 정보를 주지 않도록 한다. Ex) 'ID 또는 Password 정보가 일치하지 않습니다'
Expected Result	사용자는 QAHub 사이트에 로그인하여, QAHub의 서비스를 이용할 수 있게 된다.

Table 43 Test Cases: Login

C. Recovery of ID/Password

항목	설명
User's Action	사용자는 그/그녀가 잊어버린 ID/Password를 복구하기 위해 자신이 사전에 등록했던 정보를 복구 폼에 입력하여 인증을 요청한다.
System's Action	사용자가 입력한 정보가 User DB에 저장된 정보가 일치한다면, 요청을 승인하고 사용자에게 ID를 보여주거나, 사용자를 Password reset 페이지로 리디렉션시킨다.
Restriction	1. 복구 폼의 항목은 반드시 모두 작성되어야 한다. 2. 사용자가 입력한 정보는 사용자가 User registration 과정에서 입력했던 정보와 일치해야 한다.
Expected Result	사용자는 잊어버린 자신의 ID를 알거나, Password를 재설정함으로써 갱신된 정보로 로그인할 수 있게 된다.

Table 44 Test Cases: Recovery of ID/Password

D. Credit Purchase/Exchange

항목	설명
User's Action	사용자는 서비스를 위한 크레딧을 구매하거나, 서비스를 이용하며 얻은 크레딧을 현금으로 환전하기 위해 원하는 금액을 적어 구매/환전 요청을 보낸다.
System's Action	명시된 금액에 따라 크레딧 구매/환전 요청을 승인하고, User DB에 저장된 Credit 항목의 값을 수정한다.
Restriction	1. 크레딧 구매/환전을 위해서는 사용자는 반드시 사이트에 로그인 된 상태여야 한다. 2. 구매/환전이 가능한 최소 금액의 단위는 1,000원이며, 금액의 단위는 1000의 배수여야 한다. 3. 사용자가 소유한 Credit 이상의 금액은 환전할 수 없다. 4. 요청을 처리하기 전에 사용자에게 다시 한 번 금액과 요청에 대한 확인을 요청해야 한다.
Expected Result	사용자는 구매한 크레딧을 이용하여 서비스를 이용하거나, 획득한 크래딧을 현금으로 환전하여 실생활에서 사용할 수 있다.

Table 45 Test Cases: Credit Purchase/Exchange

E. Question Search

항목	설명
User's Action	멘티는 답변을 원하는 질문이 이미 사이트에 등록되어 있는 지 확인하기 위해 키워드를 이용하여 질문을 검색한다. 멘토는 자신이 답변할 수 있는 질문이 있는지 찾아보기 위해 키워드를 이용하여 답변을 기다리는 질문을 검색한다.
System's Action	사용자가 제공한 키워드를 이용하여 Question DB에서 쿼리를 수행한 후, 그 검색 결과를 사용자에게 제공한다.
Restriction	1. 검색을 수행하고, 그 결과를 확인하기 위해서 사용자는 사이트에 로그인 된 상태여야 한다. 2. 사용자가 검색 결과로 나온 질문들에 답변이 등록되었는지, 그렇지 않은 지 확인할 수 있어야 한다.
Expected Result	사용자는 제공한 키워드와 관련된 질문들의 리스트를 확인할 수 있다.

Table 46 Test Cases: Question Search

F. Question Registration

항목	설명
User's Action	멘티는 답변을 원하는 질문의 내용과 답변한 멘토에게 제공할 Reward의 양을 작성하여, 질문 등록 요청을 보낸다.
System's Action	멘티가 제공한 질문의 데이터를 Question DB에 저장하고, 멘티를 지금까지 자신이 등록했던 질문들을 확인할 수 있는 페이지로 리디렉션시킨다.
Restriction	1. 질문을 등록하기 위해서 멘티는 사이트에 로그인 된 상태여야 한다. 2. 멘티는 자신이 소유한 크레딧보다 더 높은 양의 Reward를 걸 수 없다. 3. Reward의 최소 단위는 100이며, 100의 배수여야 한다. 4. 등록하기 전 멘티에게 다시 한 번 확인을 요청해야 한다.
Expected Result	멘티는 답변을 원하는 질문을 등록함으로써 그 질문이 멘토에게 검색될 수 있고, 멘토가 그 질문에 답변을 등록하는 것을 기대할 수 있다.

Table 47 Test Cases: Question Registration

G. Answer Registration

항목	설명
User's Action	멘토는 답변을 기다리는 질문에 대해 답변을 작성하여 답변 등록 요청을 보낸다.
System's Action	멘토가 제공한 답변의 데이터를 Answer DB에 저장하고, 멘토를 지금까지 자신이 등록했던 답변들을 확인할 수 있는 페이지로 리디렉션시킨다.
Restriction	1. 답변을 등록하기 위해서 멘토는 사이트에 로그인 된 상태여야 한다. 2. 이미 답변이 등록되었고, 그 답변의 평가까지 마쳐진 질문에는 답변을 등록할 수 없다. 3. 등록하기 전 멘토에게 다시 한 번 확인을 요청해야 한다.
Expected Result	멘토는 답변을 등록함으로써 멘티가 그 답변을 확인하고, 그 답변에 평가를 내려주는 것을 기대할 수 있다.

Table 48 Test Cases: Answer Registration

H. Answer Evaluation

항목	설명
User's Action	멘티는 자신이 등록한 질문의 답변을 확인하고, 그 답변에 0-5 사이의 점수와 함께 피드백 메시지를 작성하여 평가 요청을 보낸다.
System's Action	멘티가 제공한 점수의 데이터를 Answer DB에 작성된 답변의 데이터에 추가하고, 점수가 0이 아니라면 질문에 명시된 크레딧을 멘티에게서 차감하여 멘토에게 지급한다. 점수가 0이라면 멘토에게 크레딧을 지급하는 것을 유보하고, 관리자가 확인하여 멘티에게 크레딧을 환불할 지, 멘토에게 크레딧을 지급할 지를 결정한다.
Restriction	1. 답변을 평가하기 위해서 멘티는 사이트에 로그인 된 상태여야 한다. 2. 평가를 요청하기 전 멘티에게 다시 한 번 확인을 요청해야 한다.
Expected Result	멘티는 답변의 유용성을 평가함으로써 멘토에게 크레딧을 지급하거나, 답변일 불만족스러웠을 경우 지급을 유보함으로써 크레딧의 환불을 기대할 수 있다.

Table 49 Test Cases: Answer Evalution

11. Index

11.1 Table Index

11.2 Figure Index