Classtagram

Design and Planning

Rev. 1.1 2019-04-23–first version

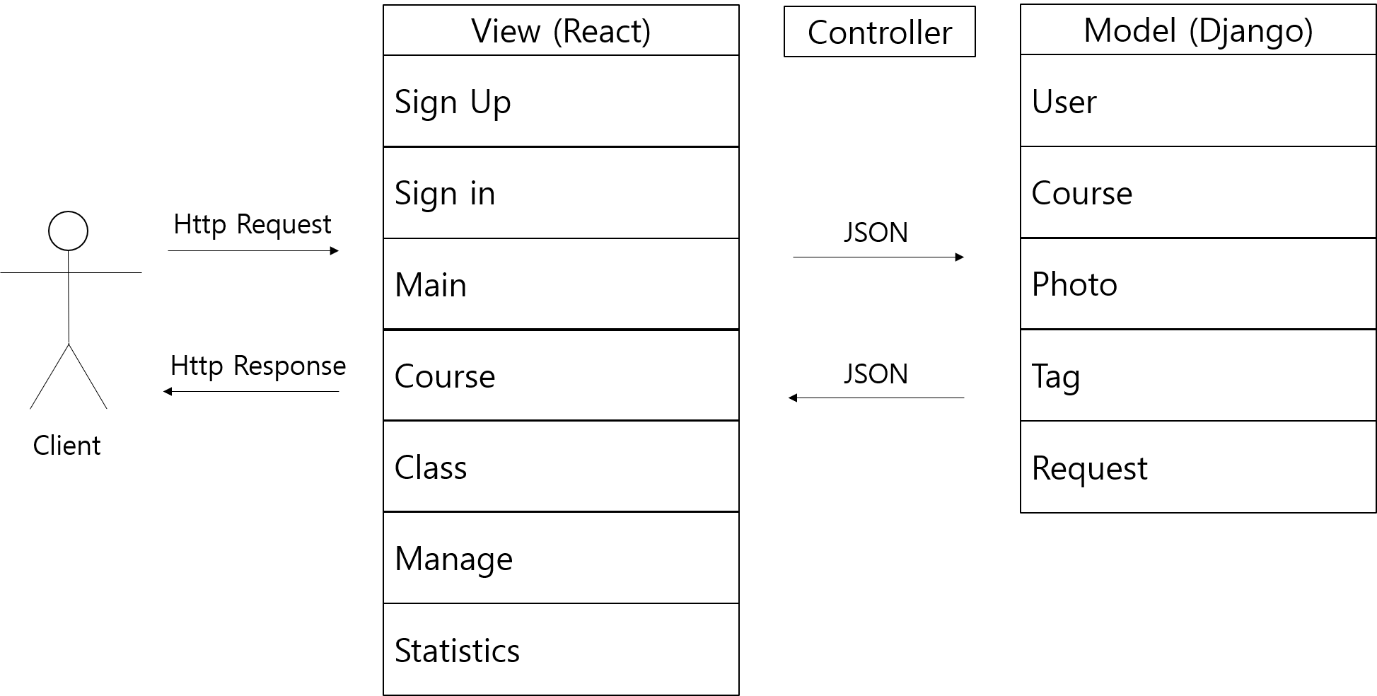
Members

신동민, 신재호, 이효준

System Architecture

MVC

Here is MVC of Classtagram.



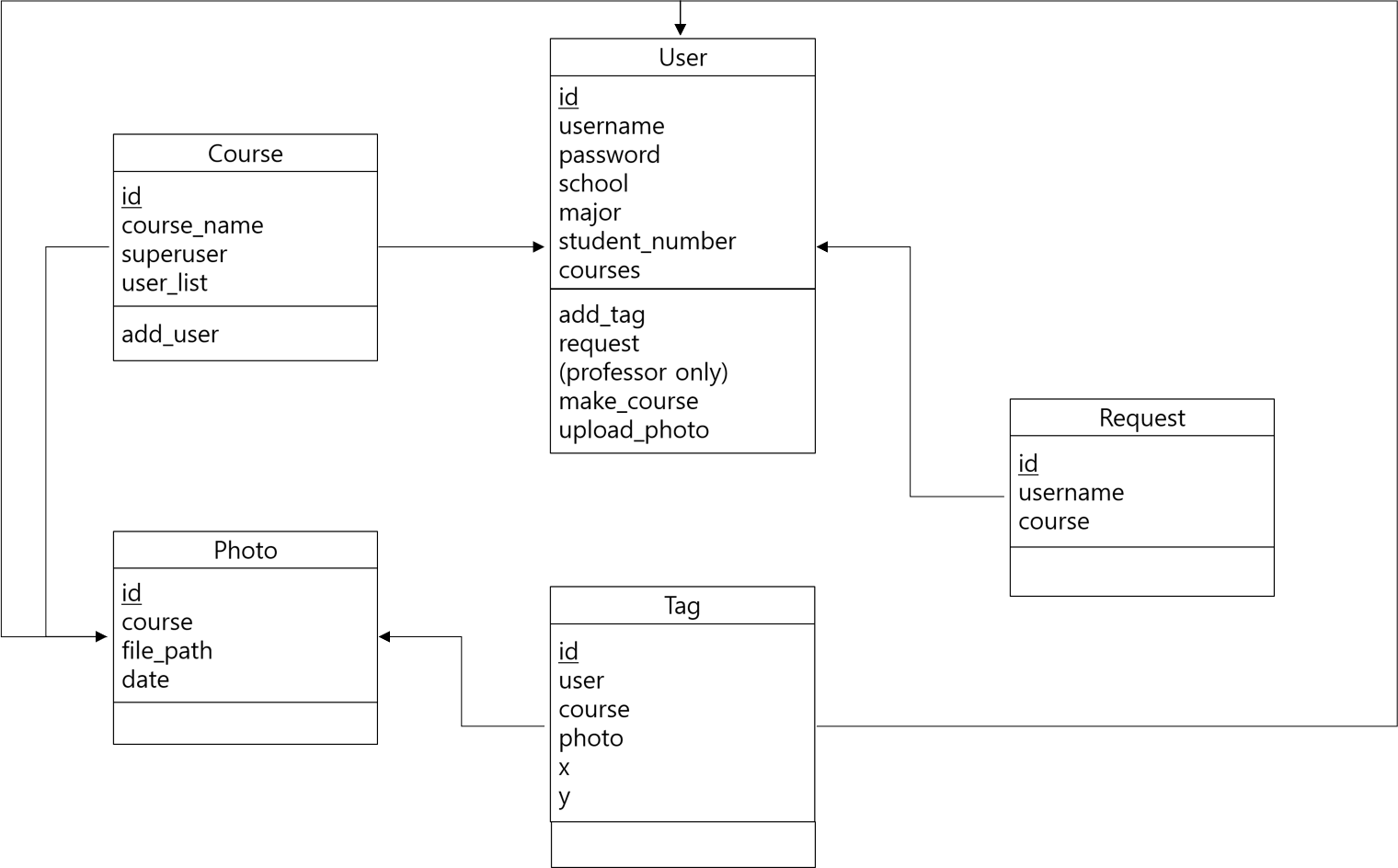
There are 7 views and 5 models. Controller will be described further in Controller part.

Model

Here is E-R (Entity Relationship) diagram for model design.

Rectangle stands for entity set, oval stands for relationship set. Entity attributes are listed inside entity rectangle, and underlined attribute indicates primary key.

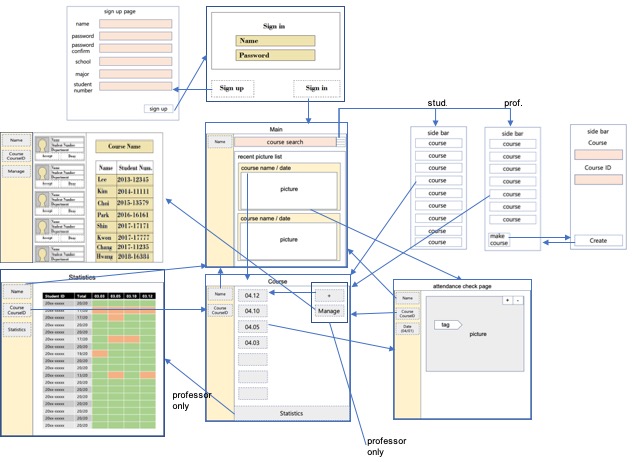
Here is relation schema diagram based on E-R diagram. This is also the structure of Django Models the service will use. It also shows core methods of each model.



Rectangle stands for relation schema. Schema attributes are listed inside relation rectangle, followed by methods for Django models. Underlined attribute indicates primary key, and arrow represents foreign key constraints. Since Django saves whole object for foreign key, this diagram also follows the convention.

View

Here is user interface for view design.



The functionality and the requirement of for each page are described as below.

1. Sign in Page(‘/)

- Sign in

2. Sign up Page(‘/signup)

- Sign up new user

- User can select the type(professor or student)

- (student) Get `name`, `password`, `school`, `major`, `student-id` as user inputs

- (professor) Get `name`, `password`, `school`, `major` as user inputs

3. Main Page(‘/user)

- Show pictures in recent order

- If the user clicks the picture, the user moves to the Attendance check page

- If the user clicks `menu` button, the side bar contains links to course page appears

- If the user types course id that professor gives him/her, the course is added to user’s course

4. Course Page(‘/user/course)

- Show buttons that contain date of class in recent order and (student only) simple statistics

- If the user clicks the button, the user moves to the Attendance check page at that date

- If the user clicks the user button on left side bar, user moves to the Main page

(Professor Only)

- If the user clicks the statistics button, the user moves to the Statistics page

- If the user clicks the manage button, the user moves to the Manage page

- If the user clicks the upload(+) button, show pop-up for uploading picture

5. Class Page(‘/user/course/class)

- Show pictures of that date

- If the user clicks +/- button, the picture is zoomed in/out

- If the user moves the tag to him/her, his/her attendance is checked

- If there is duplicated tag on one person, the page give alert

- If the user clicks the user button on left side bar, user moves to the Main page

- If the user clicks the course button on left side bar, user moves to the Course page

(Professor Only)

6. Course Manage Page(‘/user/course/manage)

- Show participants and candidate for course

- If the users clicks accept/deny the candidate is accepted/denied

- If the user clicks the user button on left side bar, user moves to the Main page

- If the user clicks the course button on left side bar, user moves to the Course page

(Professor Only)

7. Course Statistics Page(‘/user/course/statistics)

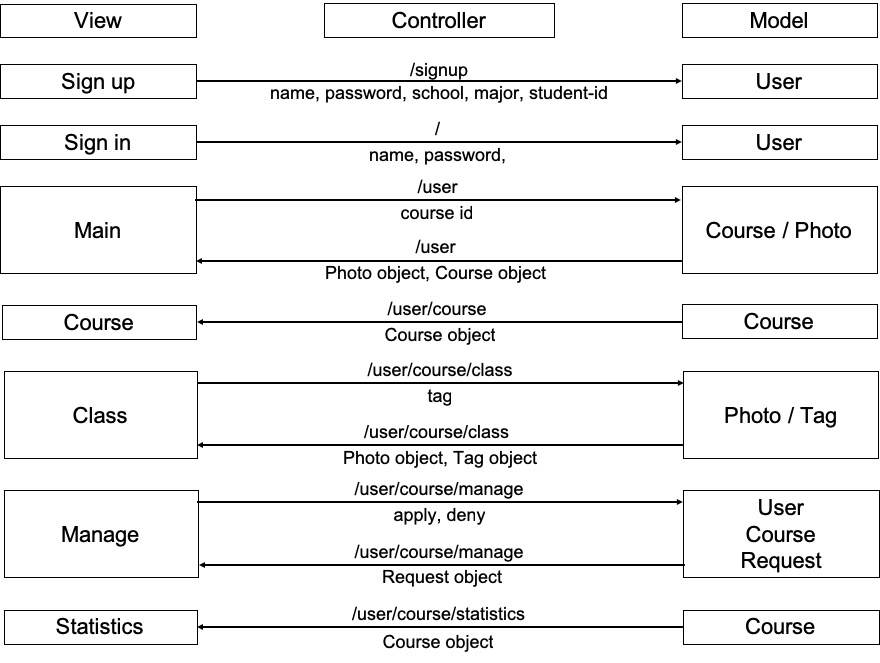
- Show students’ attendance statistics

- If the user clicks the user button on left side bar, user moves to the Main page

- If the user clicks the course button on left side bar, user moves to the Course page

Controller

Here is controller design.

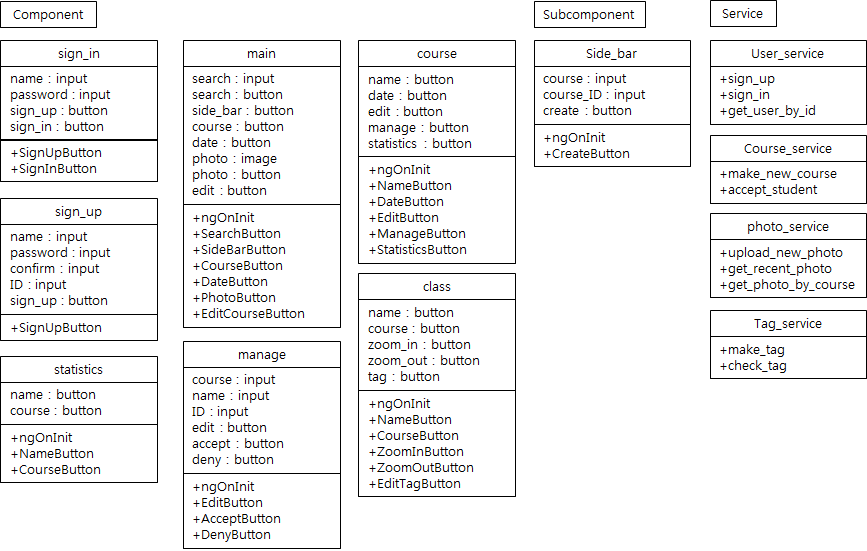


Left side is view part (frontend) and right side is model part (backend). Left-to-right arrow represents http request with user inputs from view, and right-to-left arrow represents http response with data from model. Above the arrow, there is an API that controller uses to transfer JSON data below the arrow.

Design Details

Frontend Design

Here is frontend components, subcomponents, and services. The attributes and the methods of each component are listed in each box.



Frontend Aigorithms

**Components**

1. ngOnInit

- Check the user is logged in by calling backend API, and redirect to sige\_in page if user isn’t signed in.

2. sign\_in

- SignUpButton() : Redirect to sign\_up page.

- SignInButton(name: string, password: string) : Call backend sign-in API. If sign-in is accepted, redirect to the main page. If sign-in is rejected, alert the user.

3. sign\_up

- SignUpButton(name: string, password: string, confirm: string, ID: string) : Make a new user. If password and confirm string isn’t equal, alert the user to check password. If ID isn’t out of form(XXXX-XXXXX), alert the user to change ID formation. If sign-up is successful, redirect to the main page.

4. main

- SearchButton(search: string) : search the class of keyword ‘search’, and send request to class.

- SideBarButton() : Show the side bar from main page.

- CourseButton() : Redirect to course page.

- DateButton() : Redirect to class page.

- PhotoButton() : Redirect to photo page.

- EditCourseButton() : Redirect to side\_bar page.

5. course

- NameButton() : Redirect to main page.

- DateButton() : Redirect to class page.

- EditButton() : Edit new button of new class.

- ManageButton() : Redirect to manage page.

- StatisticsButton() : Redirect to statistics page.

6. manage

- EditButton(course: string, name: string, ID: string) : edit course information of participants.

- AcceptButton() : Accept the request to course.

- DenyButton() : Deny the request to course.

7. class

- NameButton() : Redirect to main page.

- CourseButton() : Redirect to course page.

- ZoomInButton() : Zoom in the photo.

- ZoomOutButton() : Zoom out the photo.

- EditTagButton() : Can make tag on photo by click.

8. statistics

- NameButton() : Redirect to main page.

- CourseButton() : Redirect to course page.

**Subcomponents**

1. side\_bar

- CreateButton(course: string, course\_ID: string) : Create new course page of input name.

**Services**

1. user\_service

- sign\_up(name: string, password: string) : Call backend signup API, and return the result.

- sign\_in(name: string, password: string) : Call backend signin API, and return the result.

- get\_user\_by\_id(userID : number) : Call backend API, and return the user information.

2. course\_service

- make\_new\_course(name: string) : Call backend API, and return the new course.

- accept\_student() : Call backend API to accept student in course.

3. photo\_service

- upload\_new\_photo() : Call backend API to upload new photo, and return the result.

- get\_recent\_photo() : Call backend API to get recent photo, and return the result.

- get\_photo\_by\_course() : Call backend API to get every photo of the course, and return the result.

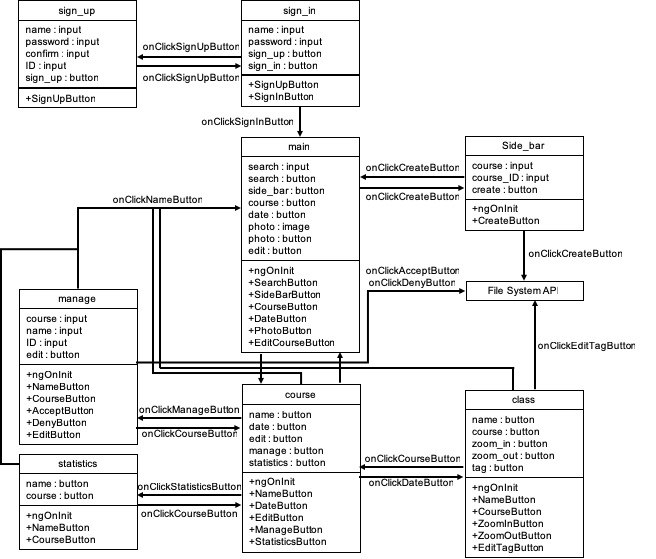
4. tag\_service

- make\_tag() : Call backend API to make new tag, and return the result.

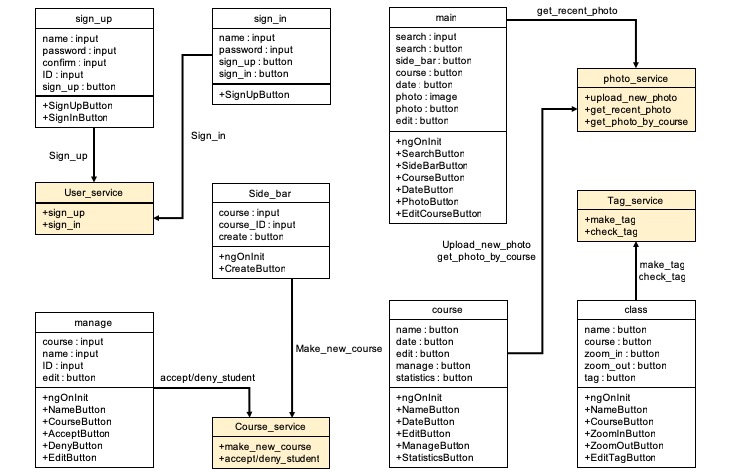
- check\_tag() : Call backend API to check whether tag is valid, and return the result.

Frontend Relations

Here are the relations between components.



Here are the relations between services and components.



Backend Design

In the backend design, we use models which have been discussed in MVC architecture section.

RESTful API

Detailed specifications of RESTful APIs are as following:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Model** | **API** | **GET** | **POST** | **PUT** | **DELETE** |
| User | /signup | X | Create new user | X | X |
|  | /signin | X | Log in | X | X |
|  | /signout | Log out | X | X | X |
|  | /user/:id | Get specific user | X | X | X |
| Course | /course | X | Create new course (professor only) | X | delete |
|  | /course/:id | Get specified course | X | Modify specified course, add/delete user to course | X |
| Photo | /photo | Get photo list of course which the user belongs | Create new photo (professor only) | X | X |
|  | /photo/:id | Get specified photo | X | Modify specified Photo | delete |
| Tag | /tag | X | Create new photo (professor only) | X | X |
|  | /tag/:id | Get specified tag | X | Modify specified tag | delete |
| Request | /request | Get request list of course | Create new request | X | X |
|  | /request/:id | Get specified request | X | X | delete |

Implementation Plan

Our implementation plan gives the division of work into sprints and people. Each user story is broken down into programming tasks. As all user stories are based on the page of the service, so the implementation plan is described based on the page as well.

The dependencies between tasks plays an important part in determining the order of development. We will develop ‘Sign up’ page and ‘Sign in’ page in sprint 2 witch create the user and sign the user in. Then ‘Main’, ‘Side Bar’, ‘Course’, ‘Class’, ‘Manage’, ‘Statistics’ will be developed in order according to their dependencies.



The division is according to the difficulties of tasks. The difficulties of tasks are expressed as numbers between 1 and 5.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Page** | **Feature** | **Difficulties (1-5)** | **Sprints** | **Person** | **Challenge** |
| Sign in | Go to Sign up page | 1 | 2 | 이효준 |  |
| Sign in | Sign in and go to Main page | 2 | 2 | 이효준 |  |
| Sign up | Sign up and go to Main page | 2 | 2 | 신재호 |  |
| Course | Upload photo | 3 | 3 | 신재호, 신동민 | Upload photo |
|  | Show student’s statistics | 3 | 3 | 신재호, 이효준 | Get several user’s statistic of the course |
| Class | Show photo | 1 | 3 | 이효준 |  |
|  | Add tag | 4 | 3 | 신재호, 이효준, 신동민 |  |
|  | Zoom in/out photo | 2 | 3 | 신동민 |  |
| Manage | Show requests | 2 | 4 | 신동민 |  |
|  | Accept/Deny requests | 2 | 4 | 신재호, 신동민 |  |
|  | Add user information of course by professor and add users automatically | 4 | 4 | 신재호, 이효준, 신동민 |  |
| Statistics | Show statistics of course | 3 | 4 | 신재호, 이효준, 신동민 |  |
| Main | Show photos of user’s course | 4 | 5 | 신재호, 이효준, 신동민 | Show photos and scroll photos |
|  | Go to course of a photo | 2 | 5 | 신재호 |  |
|  | Open sidebar | 3 | 5 | 신재호 | Make side bar |
|  | Search a course number and send a request to the course | 3 | 5 | 신재호, 이효준, 신동민 | Send a request |
| Side Bar | Show user’s courses | 2 | 5 | 이효준 |  |
|  | Make course | 3 | 5 | 신재호, 신동민 | Make course |

Testing Plan

Unit Testing

Every components and modules should be tested. In each print, we would test implemented modules by following frameworks. We expect the code coverage is over 85%.

* React & Redux
* Django : Python unit test

Functional Testing

Every APIs should be tested. In sprint 3, we would cover test RESTful API about authentication, course, photo. In sprint 4, we would test API about tag and request.

* React & Redux
* Django : Python unit test

Acceptance & Integration Testing

We would acceptance & integration test in Sprint 5. Since Cucumber automatically maps user stories into tests for user and provides testing without a human in the loop to perform the actions, we would use Cucumber for acceptance testing. For integration testing, we will user Travis CI.

* Acceptance Testing : Cucumber
* Integration Testing : Travis CI